Protected Packet Routing: Achieving Fast Failure Recovery in the IP Layer

Kang Xi and H. Jonathan Chao

April 2006
Protected Packet Routing: Achieving Fast Failure Recovery in the IP Layer

Kang Xi
Polytechnic University
Brooklyn, New York
kang@poly.edu

H. Jonathan Chao
Polytechnic University
Brooklyn, New York
chao@poly.edu

ABSTRACT
Failure recovery in IP networks has gained much attention from both academic and industrial circles. The main challenge of this issue is how to achieve fast recovery while keep the implementation complexity and resource requirement at low level. Current IP networks rely on either routing table recalculation or lower layer protection for failure recovery. Although resource-efficient, routing recalculation takes seconds even minutes to complete. On the other hand, lower layer protection requires much redundant resource to achieves the speed of millisecond recovery.

This paper proposes a pure IP layer scheme to achieve fast failure recovery with neither redundant resource reservation nor routing recalculation on the fly, called protected packet routing (PPR). The idea is to include a protection output port in each entry of the routing tables. Once a failure occurs, a number of routers switch to their protection ports for packet forwarding, thus bypass the failed equipment automatically. The challenges of this approach are: 1. Is a single protection port always sufficient to guarantee failure recovery? 2. How to find such ports? 3. How to coordinate the routers for failure recovery without little signaling? 4. How to realize distributed implementation? Focusing on single-link and single-node failure scenarios, this paper provides answers to the above questions and presents the performance evaluation of the proposed algorithms. Based on this work, it is promising to study double-failure scenario such that the survivability of IP networks can be intensified to tolerate the majority of network failures.

1. INTRODUCTION
The Internet has become a global information platform supporting numerous applications ranging from online shopping, banking to worldwide business and science activities. For such a critical infrastructure as Internet, survivability becomes a stringent requirement in that services interrupted by equipment failures must be recovered as quickly as possible. Typically, a recovery time of tens of milliseconds satisfies most quality of service requirements (e.g., SDH/SONET automatic protection switching can be completed within 50 ms). At the same time, it is expected that the failure recovery scheme could have low complexity and requires few redundant resources such as backup channels.

Network failures are caused by a number of reasons such as fiber cut, interface malfunctioning, software bugs, attacks, etc. Despite of continuous technological advances, such failures cannot be completely avoided. Figure 1 shows the statistics of link failures in Sprint backbone network in 2002, where each dot at \((t, l)\) indicates a failure on link \(l\) at time \(t\) [6]. It can be seen that even in such a well-maintained backbone failures still take place frequently. Therefore, failure recovery will remain as a significant issue in the foreseeable future.

Most network failures can be classified into two types: link failure (such as a fiber cut or a malfunctioning linecard) and node failure (typically a router is down). The fundamental
issue of failure recovery is: If the path from node A to B is interrupted by a failure, how to find and setup a new path to resume the communication. Generally speaking, there exist two categories of recovery schemes: protection and restoration. Protection setup a backup path for each primary path in advance and switch the affected traffic to the backup path in case the primary one is disconnected by the failure. Although the switch-over is fast enough, this approach needs to reserve additional bandwidth for the backup path, which increases the overall cost since no traffic is carried on backup paths during normal operation. On the other hand, restoration performs routing recalculation and path reconfiguration only after a failure is detected. Since no backup path is required in advance, this approach is more resource-efficient. However, the calculation-upon-failure usually brings considerable delay before the recovery is completed.

Today’s Internet mainly relies on IP layer restoration and/or lower layer protection for failure recovery. All the existing routing protocols (such as open shortest path first (OSPF) [7], intermediate system to intermediate system intra-domain routing (IS-IS) [4]) are designed to perform failure advertisement and routing table recalculation on network failures. Although this mechanism is able to deal with any type of failures, the time used by the whole procedure can easily reach seconds or even minutes, which is not acceptable for critical applications, especially in case of high speed networks. For example, a 30-second interruption on a 10 Gbps link results in up to 300 Gbits of data loss. On the other hand, lower layer protection provides much faster recovery. For instance, SONET automatic protection switching (APS) guarantees a recovery time less than 50 ms. With such a mechanism in the lower layer, IP routing does not require any change. However, this protection scheme introduces considerable high cost due to the redundant resource requirement. It also increases increases the investment cost and maintenance complexity since a survivable IP network cannot be built without a robust link and/or physical layer. 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—is still yet to be achieved [1].

This paper presents a pure IP layer approach for failure recovery, called protected packet routing (PPR). The basic idea is to include an additional output port (which we call protection ports) in each entry of a router’s routing table. Once a failure takes place, by letting a number of routers use their protection ports the affected packets can be forwarded along an alternative path bypassing the failure(s). Compared to traditional IP layer restoration, PPR does all the calculations in advance, thus greatly reduces the recovery delay. Compared to lower layer protection, this scheme does not reserve any redundant channels, thus is cost-effective. As a pure IP layer scheme, PPR can significantly facilitate the building of survivable IP networks since a highly survivable and complex link layer is no longer required.

This paper focuses on the most common failure scenarios: single-link and single-node failure. We systematically study each scenario and present the related theorems, mathematical formulations, heuristic algorithms, distributed implementation schemes as well as performance evaluation. It is worth noting that an important issue related to failure recovery is failure detection [2,3,9], which, however, is beyond the discussions of this paper.

2. OVERVIEW

2.1 Problem Description

Each router in an IP network maintains a routing table that tells the next hop and the corresponding output port for a particular destination. Such output ports are calculated using a certain routing protocol (such as Open Shortest Path First (OSPF) [7]) and function properly when no link/node failure occurs. We use the term primary ports to represent such ports.

In the problem addressed by PPR, we are interested in finding one or more protection ports for each primary port. Such protection ports are not used in case there is no failure. Once a failure occurs, some affected routers only need to switch the output of the packet forwarding from the primary ports to the protection ones to achieve failure recovery. This procedure is illustrated in Figure 2 using node 1 as the destination. Figure 2(a) shows the topology of the network and Figure 2(b) shows the primary port of each node with a thick arrow line. The protection ports are presented using thick dashed arrow lines in Figure 2(c). By configuring one protection port for each node, this network can easily recover from any single-link failure. An example is shown in Figure 2(d) where the failure of link 1–3 is handled by letting node 3 switch to its protection port while keeping the other nodes unchanged.

In this paper we consider three failure scenarios respectively: single-link failure, single-node failure and double link failures. To apply PPR to the general topologies, we answer the following questions for each scenario.

1. How many ports are sufficient to deal with a certain type of failure scenario? Since each entry in the routing table needs to be extended to include its protection port(s), it becomes important in terms of implementation to quantitatively study the addition storage introduced by PPR. The above example shows that one protection port is required by each router for a specific destination, we prove that this is also sufficient to deal with any kind of single-link and single-node failures.

![Figure 2: Failure recovery using protection ports.](image-url)
We also prove that two protection ports are sufficient to double link failures.

2. How to find the protection ports? Each node usually has multiple choices on its protection ports. To guarantee the effectiveness of PPR, all the nodes must be coordinated in the decision making. With more than one feasible solutions, it is also valuable to find the optimal one based on a certain objective. We present several heuristic algorithms to answer this question. The algorithms are also independent of the primary ports, which guarantees the compatibility of PPR with existing routing protocols such as OSPF.

3. How to perform configuration in case of failure? In the example shown in Figure 2, only node 3 switches to its protection port in case of failure. For the general case, it is possible that more than one nodes need to perform switch over. Therefore, it is necessary to develop schemes to find out the nodes that needs switch over according to the failure pattern. It is extremely important to pay attention to the time complexity of this decision making.

4. How to implement PPR in a distributed way? To deal with a distributed system like IP network, it is naturally a necessary step to consider the distributed implementation of PPR. Under the environment of link-state routing (such as OSPF, IS-IS [4]), we discuss how to let each router independently calculate its own protection ports and how to perform failure recovery without any centralized control.

5. Performance evaluation. We study the performance of PPR from several aspects including the memory complexity, calculation complexity, failure recovery time, routing loop, packet mis-sequence and the length of recovered paths.

2.2 Related Work

2.3 Assumptions and Notations

Definition 1. Survivable Topology: A topology is said to be survivable to a category of failures if it remains as a connected graph after any such failure occurs.

Based on this definition, a ring is a survivable topology to single-link/node failure but is not survivable to double failures. The discussions on failure recovery in this paper are always based on survivable topologies to the according failure scenarios. It is easy to understand that considering failure recovery in a non-survivable topology (such as double failure recovery in a ring) is theoretically meaningless.

Given a $N$-node network $(V, E)$, a failure pattern can potentially affect the routing to any of the $N$ destinations. For simplicity, we assume that every node maintains a specific entry in its routing table for each destination node instead of each prefix. To use PPR for recovery under a certain failure scenario, each node needs to find a number of protection ports for each destination and associate that information with the specific entry accordingly. Since the protection ports of a certain entry does not affect the routing to the other destinations, it becomes possible to those ports one destination a time. Without loss of generality, we only discuss the case with node 1 as the destination and node 2 to $N$ as the packet sources. Routing to the other destinations can be solved using the same approach.

Throughout the paper, it is assumed that all the links are bidirectional.

3. SINGLE-LINK FAILURE

3.1 Number of Protection Ports

Theorem 1 given below shows that as long as the topology of a network is survivable to any single-link failure, it is possible to use PPR for failure recovery regardless of the selection of primary ports. Theorem 2 further proves that each node only needs to maintain one protection port for each entry to support PPR. The above two properties make PPR a feasible solution compatible with existing routing protocols and with low implementation complexity.

Theorem 1. Given a set of primary ports in a network survivable to any single-link failure, a set of protection ports can always be found accordingly.

Proof:
Based on the definition of survivable topology, a node is never disconnected by any link failure. Therefore, there must exist a output port other than the primary one that leads to a different path to the destination. Such a port can then be used as a protection port.

Theorem 2. Given a network $(V, E)$ survivable to any single-link failure, each node needs only one protection port per destination to achieve failure recovery.

Proof:
Proof by contradiction. Based on the discussions in Section 2.3, let’s choose node 1 as the destination and node 2 to $N$ as the packet sources. Therefore, the primary ports together with the nodes forms a tree $T(1)$ with node 1 as the root.

We assume node $i$ has a primary port $P$ and two different protection ports $b_1$ and $b_2$ for the recovery of link failure $e_1$ and $e_2$, respectively. We further assume that $b_1$ cannot be used for $e_2$ and $b_2$ cannot be used for $e_1$.

Since either $e_1$ or $e_2$ disconnects $i$ from the tree $T(1)$, the two links must lie along the primary routing path from $i$ to node 1. Without loss of generality, we assume $e_1$ is closer to $i$, as illustrated in Figure 3. Since $b_2$ can be used to recovery $e_2$, the forwarding path through $b_2$ must not merge to the primary path unless it is beyond node $k$. Therefore, the recovery path through $b_2$ also covers the failure $e_1$, contradicting the assumption that $b_2$ cannot be used for $e_1$.

3.2 Mathematical Formulation

Knowing that PPR requires only one protection port in each node for a specific destination, we discuss in this section how
to find such protection ports. While many feasible solutions may exist, our goal is to find the one that minimizes the number of nodes that need to switch over to the protection ports upon failures.

Given a network \((V, E)\) with node 1 as the destination, a primary tree can be derived from the primary ports maintained by node 2-N. Such a tree can be expressed as a vector

\[
P = [1, p_2, p_3, \ldots, p_N],
\]

where \(P(1)\) always equals to 1 and \(P(i) = P_i\) \((i = 2, \ldots, N)\) is the primary port of node \(i\). \(p_i\) also identifies the parent of node \(i\) in the primary tree.

On the other hand, it is easy to verify if a port configuration \(P^* = [1, P_2', P_3', \ldots, P_N']\) is a valid routing scheme by simply checking if each node can reach the root by recursively moving to the parents. We define this verify procedure as a function

\[
\mathcal{R}(P, i) \triangleq P(P(\cdots P(i) \cdots)) \quad \forall i \in V.
\]

For example, the primary tree in Figure 2 is represented as \(P = [1, 1, 1, 3, 6, 1]\). Verifying if node 4 can reach node 1 is done by check the value of \(\mathcal{R}(P, 4) = P(P(4)) = P(3) = P(1) = 1\).

The protection ports of the nodes can be expressed as a vector \(B = [b_2, b_3, \ldots, b_N]\), where \(b_i\) \((i = 2, \ldots, N)\) is the protection port of node \(i\).

Although there are totally \(|E|\) different types of single-link failures, we only need to consider \(N - 1\) of them that affect the primary tree \(T\). These include the link failures that result in the failure of the primary ports \(p_2\) to \(p_N\). Such failures can be denoted as a binary vector \(F_k = [0, f_2, f_3, \ldots, f_k]\) \((k = 2, \ldots, N)\), which contains only one non-zero component \(f_k = 1\) to indicate the failure on the primary port of node \(k\). Under failure \(F_k\), the primary tree \(P\) is split into two parts and is expressed as \(P \times (1 - F_k)\), where the operator \(\times\) is defined as a component-wise multiplications.

Failure \(F_k\) causes a certain number of nodes to switch to their protection ports for failure recovery. This can switch-over be denoted as a binary reconfiguration vector \(S_k = [s_2^k, s_3^k, \ldots, s_N^k]\), where \(s_i^k = 1\) \((i = 2, \ldots, N)\) means node \(i\) needs to switch to its protection port for the link failure while a 0 indicates no change (\(S_1^k\) is always 0). The norm of the vector \(|S_k| = \sum_{i \in V} s_i^k\) gives the complexity of the reconfiguration.

Given the primary tree \(P\), the protection port vector \(B\) and a reconfiguration vector \(S_k\) for the according link failure \(F_k\), the recovered tree consisting of both primary and protection ports becomes

\[
\mathbf{P}_k = P \times (1 - F_k) \times (1 - S_k) + B \times S_k. \tag{3}
\]

With the above notations, the failure recovery of the example in Figure 2 can be expressed in the following:

\[
\begin{align*}
P & = [1, 1, 1, 3, 6, 1] \\
B & = [1, 3, 2, 5, 4, 5] \\
F_3 & = [0, 0, 1, 0, 0, 0] \\
S_3 & = [0, 0, 1, 0, 0, 0] \\
P_3 & = P \times (1 - F_3) \times (1 - S_3) + B \times S_3 \\
& = [1, 1, 2, 3, 6, 1].
\end{align*}
\]

It is easy to verify that \(P_3\) is a valid routing scheme since \(\mathcal{R}(P_3, i) = 1\) \((\forall i \in V)\).

Based on the above definitions, the problem can be formulated as an integer programming given in the following.

**Given:**
- A network \((V, E)\), and a set of precalculated primary ports \(P\).
- A set of protection ports \(B\).

**Minimize:**

\[
\sum_{k=2}^{N} |S_k|
\]

**Subject to:**

\[
\mathcal{R}(P_k, i) = 1 \quad k, i = 2, \ldots, N. \tag{5}
\]

Solving the above problem achieves the following goals:

1. Find a set of protection ports \(B\) for the routing to node 1.
2. Get the reconfiguration scheme \(S_k\) corresponding to each failure \(F_k\).
3. Achieve the optimality by minimizing the overall failure reconfiguration complexity.

Although a seemingly simple problem, solving the integer programming directly may turn out to have high complexity in large networks. The following section presents a heuristic algorithm that finds the optimal solution with low complexity.

### 3.3 Heuristic Algorithm

#### 3.3.1 Algorithm Description

The optimal solution of the problem given in (4)–(5) can be obtained using a sequential search as described below. We call this algorithm SS_LINK_1.

1. Init: Set \(b_i = 0\) \((i = 2, \ldots, N)\).
2. Find a node \(k\) \((k = 2, \ldots, N)\) using depth-first search, assume there is a link failure on the primary port of node \(k\) (i.e., failure \(F_k\)) and do the following
   - If \(b_i \neq 0\), the protection port of node \(i\) is already found, go back to step 2; otherwise continue the next step.
Figure 4: Finding protection ports for single-link failures.

(b) The failure disconnects the sub-tree \( T(k) \) from the primary tree, where \( k \) is the root node. Dye the nodes in \( T(k) \) black and all the other nodes white.

c) Starting from node \( k \) in \( T(k) \), use breadth-first search to find the first node \( i \) with a link directly connected to a white node \( j \) and set \( b_i = j \). We call this port \( i \rightarrow j \) an exit of the sub-tree \( T(k) \) under failure \( F_j \). If \( k \neq i \), find the path \( k \rightarrow i \) along the links in \( T(k) \). Suppose the path is \( k \rightarrow m_1 \rightarrow m_2 \ldots \rightarrow m_L \rightarrow i \), set the protection ports of the nodes as \( b_k = m_1, b_{m_1} = m_2, \ldots, b_{m_L} = i \). Go back to step 2.

Figure 4 shows a primary tree and the procedure of using the heuristic algorithm on the search path 2–5–7–9, which considers four failures sequentially:

1. \( F_2 \) disconnects sub-tree \( T(2) \). Breadth-first search finds the exit is 5 → 6 and the recovery path is 2→5→6. Thus we set \( b_2 = 5 \) and \( b_5 = 6 \), as shown in Figure 4(b).
2. \( F_5 \) affects \( T(5) \), and the recovery can be achieved by using the previously found \( b_5 \).
3. \( F_7 \) affects \( T(7) \), and the search result is \( b_7 = 4 \).
4. \( F_9 \) affects \( T(9) \), which can be recovered handled by setting \( b_9 = 8 \).

3.3.2 Properties
Observations reveal the following properties of SS_NODE_1.

Optimality:

**Theorem 3.** \( SS\_LINK_1 \) minimizes the overall reconfiguration complexity defined in \( \sum_{k=2}^{N} |S_k| \).

**Proof:**
The overall complexity is the aggregation of the complexity under each failure \( F_k \) (\( k = 2, \ldots, N \)). If each \( |S_k| \) reaches its minimum value, the optimality can then be achieved. In \( SS\_LINK_1 \), the exit under each failure \( F_k \) is searched following the breadth-first rule, and the depth of the exit in sub-tree \( T(k) \) is equal to the reconfiguration complexity \( |S_k| \). Breadth-first search minimizes the depth of the exit, thus minimizes the number of nodes requiring reconfiguration the same. This guarantees the overall optimality.

**Algorithm Complexity:** The algorithm has low computation complexity. Although it contains two nested searches in the tree and has a upper bound of \( O(N^2) \), the operation on each node is very efficient: only to check its neighbors without any addition or multiplication. The next section shows that \( SS\_LINK_1 \) can be also implemented in a distributed way with minor complexity.

**Sequence of Search:** The sequence of the failures to consider is determined using depth-first search in step 2 of the algorithm. The same result can also be obtained using breadth-first search. This can be explained by the relationship between two failures.

Given two failures \( F_k \) and \( F_j \), if \( k \notin T(j) \) and \( j \notin T(k) \), the protection ports for the two failures are independent to each other (such as \( F_2 \) and \( F_6 \) in Figure 4).

On the other hand, if \( k \in T(j) \), the recovery of \( F_j \) may need to use the protection port of node \( k \). Therefore, \( F_4 \) cannot be considered until \( F_7 \) is processed. In Figure 4, if \( F_5 \) is considered ahead of \( F_6 \), node 5 may choose 5→4 as the protection port. Clearly, it does work for \( F_5 \) but cannot protect \( F_6 \). Both depth-first and breadth-first search guarantees the above requirement on the sequence.

**Recovery Configuration:** While getting the protection ports, \( SS\_LINK_1 \) also finds the nodes requiring reconfiguration. Under failure \( F_k \), the nodes are indicated by the path from node \( k \) to the corresponding exit. Generally speaking, node \( k \), upon failure detection, needs to notify each of the nodes for a switch-over. Section 3.4 shows that such signaling procedure can be avoided using a special design based on the property of \( SS\_LINK_1 \).

3.4 Distributed Implementation
This section discusses the distributed implementation of PPR in a network using link-state routing (e.g., an autonomous system (AS) running OSPF). The implementation scheme needs to answer three questions: First, how to store the information of the protection ports; second, how to let each router find its protection port for each entry in the routing table; third, how to let each router perform switch-over when a failure occurs or after it is fixed.

3.4.1 Routing Table Extension
Today’s IP routers maintain a routing table where each entry has the structure of Figure 5(a). Upon the arrival of a
IP packet, the router performs longest prefix matching and routing table lookup to get the appropriate next hop and port, which identifies the output of the packet (the primary port). When the proposed PPR is supported, the entry is extended to include the protection port identified by two new fields: pro_next_hop and pro_port.

### 3.4.2 Protection Port Calculation

With link-state routing, each router has the knowledge of the AS topology and is able to get the primary tree to every node in the AS. For the entries that share the same path in the AS, they certainly have identical protection ports and there needs only one calculation for such entries. For simplicity, we only discuss how a router performs protection port calculation and omit the details of mapping such information to each specific prefix.

The distributed calculation is based on SS\_LINK\_1. We still use router 1 as the destination node to explain the calculation performed by router \( k \). After routing table calculation (e.g., using Dijkstra’s algorithm), router \( k \) can get the primary tree and its primary path to node 1, denoted as \( k \rightarrow m_2 \rightarrow \ldots \rightarrow m_1 \rightarrow 1 \). Since only the failures along this path may cause the switch-over at router \( k \), the router only needs to run a part of SS\_LINK\_1 rather than the whole algorithm. The details of the algorithm is given in the following.

1. Init: Set \( b_k = 0 \).

2. Sequentially choose a failure from list \( F_{m_1}, \ldots, F_{m_L}, F_k \) and do the following
   (a) The failure disconnects a sub-tree from the primary tree, dye the sub-tree black and denote its root as \( r \),
   (b) Find an exit of the sub-tree based on the description in step 2c of SS\_LINK\_1.
   (c) If router \( k \) is included in the path from \( r \) to the exit, set \( b_k \) accordingly and stop. Otherwise go back to step 2.

For example, node 9 in Figure 4 only needs to check \( F_2, F_5 \) and \( F_7 \) to find its protection port. It is easy to see that the complexity of the above search is bounded by the number routers in the AS and, thus the calculation can be performed very efficiently.

### 3.4.3 Packet Forwarding of PPR

The packet forwarding policy of a router determines how to switch to the protection port upon a failure and how to switch back to the primary port after the failure is fixed.

![Figure 5: Structure of routing tables.](image-url)

![Figure 6: Forwarding Policy of PPR for single-link failure.](image-url)

Upon the occurrence of a link failure \( F_k \), the first router that detects the failure is router \( k \). Therefore, packets destined to node 1 through \( p_k \) are directed to the protection port \( b_k \) accordingly. If \( b_k \) points to a router that is not a child of router \( k \), the packets can then be delivered correctly and the recovery is completed. Otherwise, the recovery requires the switch-over of that child router and possible routers thereafter. The design of SS\_LINK\_1 avoids additional signaling of failure recovery because of the following property of the protection ports.

**Corollary 1.** If the protection port \( b_k \) of router \( k \) points to one of its children \( j \), \( b_k \) must share the same link with the primary port \( p_j \) of router \( j \).

**Proof:**

Based on the description of SS\_LINK\_1 (Section 3.3.1), step 2c always finds the path from a node to the corresponding exit along the topology of the primary tree, and then set the protection ports accordingly. Therefore, the protection port of a router, if pointing to one of its children, must use the link between the two routers in the primary tree.

According to Corollary 1, a router can detect if an incoming packet is deflected back due to a failure by comparing the incoming port with the primary forwarding port. If it is a deflected packet, the router uses the protection port for the forwarding, which realizes the failure recovery. This is the same as the idea presented in [5, 8]. After the failure is fixed, there would be no deflected packets, and all routers automatically use their primary ports without relying on any signaling. In Figure 4, upon the failure of link 2-1, node 2 directs all the packets (going to node 1) through its protection port to node 5. Node 5 detects that the packets are coming from its primary forwarding port, thus learns the failure of the primary path and send the packets to node 6 through its protection port. The detailed forwarding policy of PPR is presented in Figure 6 in the form of pseudo code.

### 3.4.4 Discussions

The above implementation of PPR features the following advantages:

1. The additional memory requirement for the routing table is bounded and is independent of the network size. Only two fields are added to each entry, which can be easily achieved regarding the cost of today’s RAM.
2. The speed of routing table lookup is not affected in that no addition read/write operation is required during the failure recovery.

3. The complexity of the protection port calculation for each destination is bounded by the number of nodes in the network and is trivial compared to the routing calculation.

4. The switch-over when a failure occurs or after it is fixed is adaptive in that no signaling is required at all.

It is worth noting that the forwarding path after failure recovery is usually not the shortest path. For example, upon the failure of link 2–1 in Figure 4, the forwarding path starting from node 4 becomes 4–2–5–6–3–1 while the shortest path is 4–5–6–3–1. It is also possible that a packet may visit a certain number of routers twice along the recovery path. For example, the forwarding path from node 7 under the previous failure would be 7–5–2–5–6–3–1. A simple solution is to set a flag in node 5 to create a cut-through path once the failure is detected. The problem raised by it is how to prevent packet mis-order. How to further optimize the recovery path without introducing much complexity will be investigated in our future work.

3.5 Performance Evaluation

4. SINGLE-NODE FAILURE

Single-node failure is different from single-link failure in that a node failure brings down all the directly connected links. When a particular destination node is considered, such a failure may disconnect more than one sub-trees from the primary tree. Nevertheless, the discussions in this section show that a scheme similar to SS_LINK1 can be used to deal with single-node failures. We call it SS_NODE1.

This section assumes that the topology is survivable to any single-node failure and the discussions only consider the case with node 1 as the destination.

4.1 Number of Protection Ports

Similar to the case of single-link failure, it can be proved that PPR can be used to deal with single-node failure by maintaining only one protection port in each node, and such protection ports can be found under any kind of primary port configuration.

**Theorem 4.** Given a set of primary ports in a network survivable to any single-node failure, a set of protection ports can always be found accordingly.

**Proof:**
In such a topology, a node i is never disconnected from the network due to the failure of node k (k ≠ i). Therefore, if the path through the primary port of node i cannot be used to reach node 1, there must exist another port through which a different path can be found to reach node 1. Such a port can then be used as a protection port.

**Theorem 5.** Given a network (V, E) survivable to any single-node failure, each node needs only one protection port per destination to achieve failure recovery.

**Proof:**

4.2 Mathematical Formulation

The formulation of the single-node failure recovery is identical to that of the single-link failure recovery given in (4)–(5) except the definition of the binary failure vector $F_k$ ($k = 2, \ldots , N$). In Section 3, there is only a single non-zero component in the vector. In this section, the failure of node k generates a $F_k = [0, f_2, f_3, \ldots , f_N]$, where the value of $f_i$ is defined according to the primary tree $T$:

$$f_i = \begin{cases} 1 & \text{if } i = k \\ 1 & \text{if } p_i = k \\ 0 & \text{otherwise} \end{cases} \quad i = 2, \ldots , N. \quad (6)$$

It is easy to see that all the disabled ports of the primary tree are reflected in the vector. In Figure 2 the failure of node 3 generates a vector $F_3 = [0, 0, 1, 1, 0, 0]$.

Although there is much similarity between single-node and single-link failure, the protection ports found by SS_LINK1 cannot always work for node failures. This can be explained by the example in Figure 7, where Figure 7(a) shows the protection ports of node 2, 4 and 6 calculated using SS_LINK1. Although such a configuration is able to handle the failure of link 2–1, 4–2 or 6–2, it cannot deal with the failure of node 2. This is because the failure detaches two sub-trees $T(4)$ and $T(6)$, where $T(4)$ must rely on the other to find a protection forwarding path. In contrast, a link failure detaches only one sub-tree. Figure 7(b) provides a scheme for node failures. The next section discusses how to find the protection ports under the assumption of single-node failures.

4.3 Heuristic Algorithm

The heuristic algorithm is still based on sequential search and we call it SS_NODE1.

4.3.1 Algorithm Description

SS_NODE1 takes the following steps to find all the protection ports for the routing to node 1.

1. Init: Set $b_i = 0 \ (i = 2, \ldots , N)$. 

![Figure 7: Compare between link and node failure.](image-url)
2. Find a node \(k (k = 2, \ldots, N)\) using depth-first search, assume there is a failure at node \(k\) and do the following:

   (a) Dye all the node in \(T(k)\) black and the other nodes white.

   (b) Assume node \(k\) has \(m_k\) children, denote the nodes as \(c_1, \ldots, c_{m_k}\).

   (c) Check the protection port of each children, if \(b_i \neq 0 (i = c_1, \ldots, c_{m_k})\), dye all the nodes in \(T(i)\) white.

   (d) Node \(k\) and the black nodes forms a tree, denote it as \(T^*(k)\). If \(T^*(k)\) contains a node other than \(k\), repeat the following steps; otherwise, go back to step 2.

      i. Use breadth-first search on \(T^*(k)\) to find the first node \(j\) that has a white neighbor \(w\), set \(b_j = w\).

      ii. Search \(c_1, \ldots, c_{m_k}\) for the node \(r\) satisfying \(w \in T(r)\).

      iii. Following the links in \(T(r)\), find a path from \(r\) to \(j\). This shows the recovery forwarding path. Set the protection ports of the nodes in that path accordingly.

      iv. Dye all the nodes in \(T(r)\) white, go back to step 2d.

Figure 8 gives an example of how SS\_NODE\_1 works. The topology and primary tree are shown in Figure 8(a). We explain how to find the protection ports under the failure of node 2 by repeating step 2(d)i—2(d)iv. Under the assumed failure, all the nodes belonging to \(T(2)\) is dyed black before the search starts. Since node 2 has three children, it takes three cycles to turn the three sub-trees \(T(4), T(5)\) and \(T(6)\) to white.

**Cycle 1:** Now the black sub-tree \(T^*(2)\) is the same as \(T(2)\), breadth-first search in the it finds link 10—7 the first exit to a white node. As a result, we set \(b_{10} = 10, b_{10} = 7\) and dye node 6, 10, 14 and 15 white.

**Cycle 2:** After the first cycle, \(T^*(2)\) is updated by removing \(T(6)\). Performing breadth-first search hits node 5. Therefore, we set \(b_5 = 10\) and turn \(T(5)\) into white.

**Cycle 3:** At this moment, \(T^*(2)\) shrinks to include only \(T(4)\). Doing step 2(d)i—2(d)iv yields \(b_4 = 8\) and \(b_9 = 9\). Now all the children of node 2 becomes white and we can move to search the next node failure.

The paths found in the above procedure also indicates the failure recovery configuration. Upon the failure of node 2, each of its child sub-trees switches to the recovery path for packet forwarding. In single-link failure scenario, the exit of a sub-tree always connects to an unaffected node directly. In contrast, single-node failure may generate more than one black sub-trees (disconnected from the primary tree). In this case, a black sub-tree may have to go through other black ones to reach the destination, such as \(T(4)\) and \(T(5)\) in Figure 8. SS\_NODE\_1 automatically coordinate the neighboring sub-trees to guarantee the correctness of the forwarding path. For example, the forwarding path from node 4 becomes 4 → 8 → 9 → 5 → 10 → 7 → 3 → 1.

4.3.2 Properties

Observations reveal the following properties of SS\_LINK\_1.

**Optimality:** It is worth noting that SS\_NODE\_1 does not always gives the optimal solution of the mathematical formulation. Given multiple black sub-trees, there are many combinations to let them go through each other for the recovery path. Using sequential search, SS\_NODE\_1 does not explore all the combinations, thus the optimality of its result is not guaranteed. Figure 4.3.2 gives an example with the protection ports found under the failure of node 3. It can be seen that letting the protection port of node 6 point to node 12 results in less switch-overs than the solution found by the SS\_NODE\_1.

**Complexity:** Compared to SS\_LINK\_1, SS\_NODE\_1 may need to perform more than one breadth-first search for each node failure, which is determined by the number of children of that node. Nevertheless, search of the sub-trees involves in only part of the whole topology, thus the algorithm is still efficient in terms of calculation complexity. There is no universal closed form express of the complexity since it differs in different topologies. In Figure 8, the breath-first search of SS\_NODE\_1 performs totally 18 node checks before finding all the protection ports, resulting in an average 1.125 visits per node.
**Sequence of Search:** SS\_NODE\_1 has the same property as SS\_LINK\_1 in determining the sequence of failure search. Since the protection port of a node is affected only by its parent or indirect parent, either depth-first search or breadth-first search can be used in step 2 of SS\_NODE\_1.

**Recovery Configuration:** The path of each sub-tree to its exit is found by SS\_NODE\_1 and can be used for the recovery configuration. And the signaling can be avoided using the same implementation scheme as in the scenario of single-link failure.

**Deal with Single-Link Failures:** It is easy to see that the protection ports and the configuration schemes founded by SS\_NODE\_1 can be directly used to deal with single-link failures since each single-link failure can always be covered by a node failure. The exception lies at the nodes whose parent is the root of the primary tree. Since SS\_NODE\_1 does not assume the failure of the root node, it does not find any protection port for its children, e.g., node 2 and 3 in Figure 8. This can be easily solved by running SS\_LINK\_1 only for such nodes after SS\_NODE\_1 is completed.

### 4.4 Distributed Implementation

The routing table structure and forwarding policy of PPR for single-node failures is identical to those for single-link failures described in Section 3.4. The slight difference is the way routers perform protection port calculation.

### 5. CONCLUSIONS AND FUTURE WORK

### 6. REFERENCES


