Route Recovery in Vertex-Disjoint Multipath Routing for Many-To-One Sensor Networks

Wei Cheng[1,2], Kai Xing
[1]Dept. of Computer Science
The George Washington University
Washington, DC, USA
wcheng@gwu.edu, kaix@gwu.edu

Xiuzhen Cheng
Dept. of Computer Science
The George Washington University
Washington, DC, USA
cheng@gwu.edu

Xicheng Lu, Zexin Lu
[2]School of Computer
National University of Defense Technology
Changsha, Hunan, China
xclu@nudt.edu.cn, lzx@nudt.edu.cn

Jinshu Su
School of Computer
National University of Defense Technology
Changsha, Hunan, China
sjs@nudt.edu.cn

Baosheng Wang
School of Computer
National University of Defense Technology
Changsha, Hunan, China
bswang@nudt.edu.cn

Yujun Liu
Dept. of Information
Academy of Armored Forces Engineering
Beijing, China
jadecs@126.com

ABSTRACT

Multipath routing is attractive for load-balancing, fault-tolerance, and security enhancement. However, constructing and maintaining a set of node-disjoint paths between the data source and sink is non-trivial in a dynamic environment. In this paper, we study the problem of route recovery in vertex-disjoint multipath routing for sensor networks with many-to-one traffic patterns. We identify the sufficient conditions for multipaths to be recovered when the existing node-disjoint paths are broken, and provide a simple framework for multipath maintenance. This framework is very efficient in time when multipath source routing is employed. Our findings can help to conserve network resource by not launching any route discovery when the data source realizes that a new route may not exist, to guide mobile data sources to relocate themselves in order to reconstruct the new multipaths, and to help newly-deployed data sources quickly determine whether the required number of multipaths exist for sure or not and then compute them. The technique proposed in this paper is a good complement to the classic max-flow algorithm when node-disjoint multipaths are needed.

Categories and Subject Descriptors

C.2 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Architecture and Design—*Wireless Communication*

General Terms

Algorithms, Design, Reliability, Theory.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MobiHoc'08, May 26–30, 2008, Hong Kong SAR, China.
Copyright 2008 ACM 978-1-60558-073-9/08/05 ...$5.00.

Keywords

Multipath routing, route recovery, multipath determinability, Wireless sensor networks

1. INTRODUCTION

In this paper, we consider the problem of route recovery in node-disjoint[1] multipath[2] routing for many-to-one sensor networks. We assume that each data source, a data aggregation node, constructs and maintains N vertex-disjoint paths to the data sink. Note that this is a realistic assumption for many sensor network applications.

There exist a number of vertex-disjoint routing protocols that can be applied to the network we are considering. However, none of them provides answers to the following questions:

- When one of the N node-disjoint paths is broken due to node failures or link breakage, is it possible to reconstruct a new set of N node-disjoint paths? Can we determine the existence of one more vertex-disjoint path before actually looking for it?
- When the number of existing node-disjoint paths can't satisfy the requirement (i.e. the number is $< N$), is it possible to provide information guiding a mobile data source to reposition itself such that N vertex-disjoint paths are available at the new location? Can a newly-deployed data source quickly figure out the existence of N multpaths?
- What information is needed for a data source to recover from a path failure? Under what conditions can we compute a new set of N vertex-disjoint paths based on the information from neighbors only?

Answering these questions plays a significant role in multipath maintenance and reconstruction in sensor networks. Particularly, it can help to save network resource by not launching new route discovery if the data source realizes that no more route might exist; it can guide a mobile data source to reposition itself to recover from

[1]We use node-disjoint and vertex-disjoint interchangeably.
[2]When we say "multipath" we mean "node-disjoint paths". In addition, we use "path" and "route" interchangeably.

a path failure; it can help a newly-deployed data source quickly figure out the existence of N vertex-disjoint paths to the data sink and construct them by exploiting the available paths from its neighbors. Motivated by these considerations and observations, we conduct a determinability study of one more vertex-disjoint path, trying to systematically answer the questions mentioned above for many-to-one sensor networks.

The contributions of this paper are multifold. First, we identify the sufficient conditions when one more vertex-disjoint path exists. Second, we provide a maintenance framework for a data source to recover from route failure when N multipaths are required from each data source to the data sink. This framework is particularly efficient for route recovery in source routing. Third, the technique proposed in this paper is a good complement to the classic max-flow algorithm when node-disjoint multipaths are needed. In addition, we show that $d + 1$ multipaths for a d-dimensional sensor network might be constructed for free when a geometric $(d + 1) - lateration$ localization method is employed to localize the network.

Our network model is sketched below. Each data source transmits packets along N vertex-disjoint paths to M sinks for load-balancing and security enhancement. Note that there is no deterministic relationship between N and M. We only require that each data source maintains N vertex-disjoint paths to a subset of the sinks. Therefore all sinks can be treated as one virtual sink. In this model, data sources could be mobile.

The remaining portion of this paper is organized as follows. Section 2 provides a brief overview on major related works. In Section 3, we study the determinability of one more vertex-disjoint path and report the corresponding sufficient conditions. A N vertex-disjoint path maintenance framework is proposed in Section 4. A couple of relevant problems are discussed in Section 5. We conclude our paper in Section 6 with a discussion on future research.

2. RELATED WORK

In this section, we briefly survey the major related works in vertex-disjoint path construction and maintenance. Multipath routing is very prosperous in providing fault-tolerance, load-balancing [11], and security enhancement [13] in wireless ad hoc and sensor networks. A number of multipath routing protocols have been proposed and evaluated in literature.

The well-known max-flow algorithm computes N vertex-disjoint paths in linear time. Ripphausen-Lipa *et al.* [12] proposes the best known linear-time algorithm for node-disjoint route construction. A polynomial time algorithm ($O(Nn^2)$) to find the shortest N vertex-disjoint paths is reported in [15]. These algorithms are centralized and are exploited as basic building blocks for [14] and [6].

Distributed vertex-disjoint path computation is mainly studied in the context of ad hoc networks. In sensor networks, A node-disjoint parallel multipath algorithm is proposed in [5], where geographic information is employed for disjoint path discovery. Ref. [7] introduces a distributed N-to-1 multipath discovery protocol by employing two phases of flooding for security and reliability enhancement. There exist a number of on-demand node-disjoint multipath routing protocols such as SMR [4], AOMDV [8] and AODV-Multipath [16]. These protocols are the extensions of DSR [3] and AODV [10], the two well-studied routing algorithms for ad hoc networks.

Split Multipath Routing (SMR) [4] is an on-demand source routing protocol, in which an intermediate node broadcasts the same RREQ when it is received from a different neighbor the first time. No route cache is maintained and therefore intermediate nodes are not allowed to send RREP back to the source since otherwise the discovery of maximally disjoint paths at the destination will be prohibited.

AOMDV [8] is an extension to the AODV protocol for computing multiple loop-free and link-disjoint or node-disjoint paths. This protocol explores an important property of flooding: If each node is allowed to broadcast only the first copy of the RREQ, then two copies of the same RREQ arriving at a node from different neighbors define a set of node-disjoint paths from the source to this node. AOMDV introduces additional field in RREQ to facilitate nodes (intermediate and desnation) to compute the node/link-disjoint paths to the source after receiving copies of RREQ from different neighbors.

AODV-Multipath [16] is another extension to AODV for finding multiple node-disjoint paths. Intermediate nodes are not allowed to send a RREP back to the source. Also, duplicate RREQ packets are not discarded at intermediate nodes. Instead, all received RREQ packets are recorded in a RREQ table. The destination sends a RREP for all the received RREQ packets. An intermediate node forwards a received RREP packet to the neighbor if it is in the RREQ table. To ensure that nodes do not participate in more than one route, whenever a node overhears one of its neighbors broadcasting a RREP packet, it deletes that neighbor from its RREQ table. Because a node cannot participate in more than one route, the discovered paths must be node-disjoint.

A qualitative comparison of SMR, AOMDV, and AODV-Multipath is reported in [9]. Simulation results indicate that AOMDV achieves the best performance in high mobility, while AODV-Multipath performs better at lower mobility but higher node density, and SMR performs the best at low node density. The limitations of these protocols are studied in [6], which states that the on-demand multipath routing protocols mentioned above can not find out all existing node-disjoint paths. A theoretical framework showing the equivalence of on-demand multipath discovery and flow network assignment is established to guarantee the on-demand discovery of an arbitrary number of node-disjoint paths between a pair of nodes as long as they exist. A protocol integrating this theoretical framework with DSR to find out node-disjoint paths is also proposed in [6]. Note that Multipath maintenance in these protocols is equivalent to a new multipath discovery, which means that a RREQ from the source is initiated whenever needed.

Our work is different from all these surveyed in this section. we intend to provide a low-cost method to figure out whether there exists one more vertex-disjoint path, to provide information guiding the relocation of a mobile node such that more vertex-disjoint paths can be established, and to identify sufficient conditions for the existence of more vertex-disjoint paths. To our best knowledge, no existing work considers any of these challenges. Our technique can be employed for the recovery of vertex-disjoint paths, and therefore, can be applied together with any of the protocols for multipath construction. In addition, this paper provides sufficient conditions for a node to figure out whether one more vertex-disjoint path can be computed when one of the existing vertex-disjoint paths is broken.

3. DETERMINABILITY STUDY

In this section, we derive the sufficient conditions when one more vertex-disjoint path exists for a node S. For this purpose we introduce the **Flip** and the **Omvdp** algorithms first.

3.1 Induced Path

DEFINITION 3.1. *Given a source node and a destination node, a* **paired vertex-disjoint path set** *is the set of vertex-disjoint paths between these two nodes.*

DEFINITION 3.2. *An* **induced path** *is a path derived from two paired vertex-disjoint path sets, noted as the* **basic set** *and the* **reference set***, respectively, with each starting at different source node and both ending at the same destination node. The induced path starts from the source node of the basic set, partly overlaps one and only one path in the basic set first, then overlaps at most one path in the reference set, and finally ends at the common destination node.*

In the following, we introduce an algorithm termed as ***Flip(First Level Induced Path)*** to compute an induced path. The mathematic notations utilized by Flip are listed in Table 1:

S	S is the source node of the basic set.
P	P is the source node of the reference set.
T	T is the destination node.
C_S	$C_S = \{S_{T_1}, S_{T_2}, \cdots, S_{T_i}, \cdots, S_{T_{N_B}}\}$, the basic set, where N_B is the number of paths and $1 \leq i \leq N_B$.
C_P	$C_P = \{P_{T_1}, P_{T_2}, \cdots, P_{T_j}, \cdots, P_{T_{N_R}}\}$, the reference set, where N_R is the number of paths and $1 \leq j \leq N_R$.
S_{T_i}	$S_{T_i} = \{s_{i_0}, s_{i_1}, \cdots, s_{i_{B_i}}\}$, the ith path in the basic set, where $s_{i_0} = S$, $s_{i_{B_i}} = T$ and B_i is the number of vertices.
P_{T_j}	$P_{T_j} = \{p_{j_0}, p_{j_1}, \cdots, p_{j_{R_j}}\}$, the jth path in the reference set, where $p_{j_0} = P$, $p_{j_{R_j}} = T$ and R_j is the number of vertices.

Table 1: Notations adopted by Flip.

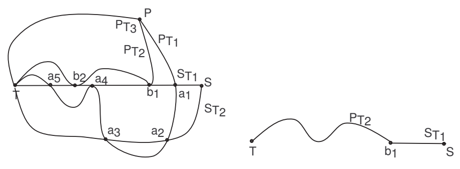
DEFINITION 3.3. $X_i = \{s_{i_k} | \exists j, s.t. s_{i_k} \in P_{T_j}, 0 \leq k \leq B_i\}$ *is S_{T_i}'s* **ordered intersection set***, where a node s_{i_m} is before s_{i_n} in X_i if and only if s_{i_m} is closer to S than s_{i_n} in hop-count.*

Note that the ordered intersection set X_i of S_{T_i} contains the nodes at which S_{T_i} intersects some paths in C_P. All nodes in X_i are ordered based on their distances in hop-count to S. Similarly we define P_{T_j}'s ordered intersection set, denoted by Y_j, which contains all the nodes at which P_{T_j} crosses some paths in C_S. All nodes in Y_j are ordered with increasing distances to P.

DEFINITION 3.4. *If a node is the lth element of X_i, then S_{T_i}'s subpath from S to this node is called the* **lth level subpath***.*

DEFINITION 3.5. *A node is P_{T_j}'s* **effective partition node** *if and only if it is in both P_{T_j} and one of the paths in the basic set, such that P_{T_j}'s subpath from this node to T vertex-disjoints all the other paths in the basic set.*

DEFINITION 3.6. *P_{T_j}'s subpath from an effective partition node to T is called an* **effective residual path***.*



(a) (b)

Figure 1: (a) A example to illustrate the definitions of ordered intersection set and effective partition node. (b) An induced path found by Flip for (a).

Note that an effective partition node of P_{T_j} must belong to one and only one ordered intersection set X_i. Each intersection node belongs to two paths, with one from the basic set, and one from the reference set. Fig. 1(a) gives an example to illustrate these definitions. We have $X_1 = \{a_1, b_1, a_4, b_2, a_5\}$, $X_2 = \{a_2, a_3\}$, $Y_1 = \{a_1, a_2, a_3, a_4, a_5\}$, $Y_2 = \{b_1, b_2\}$, and $Y_3 = \emptyset$. The nodes a_4 and a_5 are P_{T_1}'s effective partition nodes while b_1 and b_2 are P_{T_2}'s effective partition nodes. Note that P_{T_3} does not have any effective partition node.

Now we are ready to propose the Flip algorithm. The inputs to Flip are the basic set C_S and the reference set C_P. The output is an induced path that is either a path in the basic set, or a path composed of a minimum level subpath of the basic set and an effective residual path of the reference set.

Algorithm Flip

```
(1) Construct the ordered intersection set for each path in $C_S$;
(2)
if $\exists i$, s.t. $X_i = \emptyset$ then
    $S_{T_i}$ is the induced path
    STOP
end if
(3) Find the effective partition nodes for each path in $C_P$;
(4) Set $l = 1$
(5)
if $\exists i$, s.t. $S_{T_i}$'s $l$th level subpath joints an effective residual path of $P_{T_j}$ then
    The induced path is the concatenation of these two subpaths
    STOP
end if
(6) $l = l + 1$
(7) Go to (5)
```

The basic idea of Flip is sketched as follows. If there exists a path $S_{T_i} \in C_S$ that vertex-disjoints all paths in C_P, output this path. Otherwise, construct a path from C_S and C_P in the following way: find the smallest l such that the lth element of the ordered intersection set of some S_{T_i}, denoted by s_i^l, is an effective partition node of some P_{T_j}. Then output the concatenation of the lth level subpath of S_{T_i} and the effective residual path of P_{T_j} starting from s_i^l. For example, the induced path found by Flip for Fig. 1(a) is shown in Fig. 1(b). Note that if a connectivity matrix is employed to store the graph derived from the basic set and the reference set, the time complexity of Flip is $O(n)$, where n is the total number of vertices in the graph.

Note that an induced path outputted from Flip contains nodes and edges in C_S and C_P only. The following theorem indicates that there must exist such a path. It also ensures that Flip will stop and return an induced path.

THEOREM 3.1. *There must exist at least one induced path for the given basic set C_S and reference set C_P when P is not in any of the paths in C_S and S is not in any of the paths in C_P.*

PROOF. If $\exists S_{T_i} \in C_S$, s.t. S_{T_i} vertex-disjoints all paths in C_P, then S_{T_i} is an induced path according to the Def. 3.2.

Otherwise, for $\forall S_{T_i}$, there must exist a P_{T_j}, s.t. $S_{T_i} \bigcap P_{T_j} \neq \emptyset$. According to Def. 3.5 and Def. 3.6, there must exist an effective partition node a_m and an effective residual path P_m for P_{T_j} starting from a_m because the number of vertices in P_{T_j} is finite. Without loss of generality, we assume that a_m is in S_{T_i}. Thus, a path, which is composed of P_m and S_{T_i}'s subpath from S to a_m, is an induced path according to Def. 3.2. $\square$

3.2 Constructing One More Vertex-Disjoint Path

In Subsection 3.1, we have proposed the Flip algorithm to find out an induced path given the basic set and reference set. In this subsection, we will study the condition under which the source node can find out one more vertex-disjoint path to the sink node, and will report the method to construct these paths.

DEFINITION 3.7. *If a node has N known vertex-disjoint paths to the sink node, it is called a N-***credit node***.*

We will study the condition to guarantee the existence of the Nth path for a $(N-1)$-credit node. An algorithm termed as ***One More Vertex-Disjoint Path (Omvdp)*** is proposed to compute N vertex-disjoint paths for the $(N-1)$-credit node S when there exist a N-credit node P, such that P is not in any of S's paths and S is not in any of P's paths, and a path S_P between S and P that vertex-disjoints all the known vertex-disjoint paths from S and P to the sink node T. The mathematic notations for Omvdp are listed in Table 2:

A	$A = \{S_{T_1}, S_{T_2}, \cdots, S_{T_i}, \cdots, S_{T_{N-1}}\}$, the set of vertex-disjoint paths for the $(N-1)$-credit node S.
B	$B = \{P_{T_1}, P_{T_2}, \cdots, P_{T_j}, \cdots, P_{T_N}\}$, the set of vertex-disjoint paths for the N-credit node P.
New_A	$New_A = \{new_1, new_2, \cdots, new_N\}$, the set of vertex-disjoint paths constructed by Omvdp for S.

Table 2: Notations utilized by Omvdp.

The inputs to Omvdp are A and B, and the output is New_A.

An example to illustrate Algorithm Omvdp is shown in Fig. 2, where Fig. 2(a) is the input graph containing A, B, and S_P, and Fig. 2(h) is the resultant graph showing the N (here $N = 4$) vertex-disjoint paths found by Omvdp. The procedure of constructing the $N = 4$ vertex-disjoint paths for S in Fig. 2(a) is detailed as follows. Note that we use $\{S_{T_i}, x, P_{T_j}\}$ to represent an induced path derived from S_{T_i} and P_{T_j} with x being the switching point. In other words, the induced path $\{S_{T_i}, x, P_{T_j}\}$ contains the subpath from S to x in S_{T_i} and the subpath from x to T in P_{T_j}.

Algorithm Omvdp

```
(1)
$C_S = A$
$C_P = B$
$New_A = \emptyset$
(2)
**if** $\exists j$, s.t. $P_{T_j} \in C_P$ vertex-disjoints all the paths in $C_S$ and $New_A$ **then**
    (2.1) $New_A = New_A \bigcup C_S \bigcup \{S_P || P_{T_j}\}$, where $||$ stands for concatenation
    (2.2) STOP
**end if**
(3)
**if** $C_S = \emptyset$ **then**
    (3.1) $P_{Flip} = S_P || P_{T_j}, \forall P_{T_j} \in C_P$
**else**
    (3.2) Employing ***Flip*** to find an induced path $P_{Flip}$ with $C_S$ and $C_P$ being the basic set and the reference set, respectively. Without loss of generality, we assume that $P_{Flip}$ is derived from $S_{T_i} \in C_S$ and $P_{T_j} \in C_P$. If $P_{Flip}$ is derived from $C_S$ only, which means that $P_{Flip} \in C_S$, set $P_{T_j} = \emptyset$.
**end if**
(4)
**if** $P_{Flip}$ vertex-disjoints all the paths in $New_A$ **then**
    (4.1) $New_A = New_A \bigcup \{P_{Flip}\}$
    **if** $C_S \neq \emptyset$ **then**
        (4.2) $C_S = C_S \setminus \{S_{T_i}\}$
    **end if**
    (4.3) $C_P = C_P \setminus \{P_{T_j}\}$
**else**
    (4.4) $Temp = \{new_h | new_h \bigcap P_{Flip} \neq \emptyset, new_h \in New_A, 1 \leq h \leq N_{New_A}\}$, where $N_{New_A}$ is the number of elements in $New_A$.
    (4.5) Employing ***Flip*** to find an induced path $P_{temp}$ with $Temp$ and $\{P_{Flip}\}$ being the basic set and the reference set, respectively.
    (4.6) $New_A = New_A \bigcup \{P_{temp}\} \setminus \{new_t\}$, where $new_t \in Temp$ is the path employed to derive $P_{temp}$.
    (4.7) $C_S = C_S$
    (4.8) $C_P = C_P \bigcup \{P_{T_k}\} \setminus \{P_{T_j}\}$, where $P_{T_k} \in B$ is the path employed to derive $new_t$, and $P_{T_j} \in B$ is the path employed to derived $P_{Flip}$.
    (4.9) Go to (3)
**end if**
(5)
**if** $S_P$ is included in one of the paths in $New_A$ **then**
    (5.1) STOP
**else**
    (5.2) Go to (2)
**end if**
```

(a) Fig. 2(a) is the input to Omvdp, where $A = \{S_{T_1}, S_{T_2}, S_{T_3}\}$, $B = \{P_{T_1}, P_{T_2}, P_{T_3}, P_{T_4}\}$, and S_P vertex-disjoints all paths in A and B. Initially $New_A = \emptyset$, $C_S = A$ and $C_P = B$.

(b) Go to step (2) then (3). Compute $P_{Flip} = \{S_{T_1}, b_3, P_{T_2}\}$ (step (3.2)). This P_{Flip} vertex-disjoints all the paths in the current New_A. In step (4), update New_A, C_S, and C_P: $New_A = \{\{S_{T_1}, b_3, P_{T_2}\}\}$ (shown in Fig. 2(b)), $C_S = \{S_{T_2}, S_{T_3}\}$ and $C_P = \{P_{T_1}, P_{T_3}, P_{T_4}\}$.

(c) Go to step (2) then (3). Compute $P_{Flip} = \{S_{T_2}, c_2, P_{T_3}\}$ (step (3.2)). This P_{Flip} vertex-disjoints all the paths in the current New_A. In step (4), update New_A, C_S, and C_P: $New_A = \{\{S_{T_1}, b_3, P_{T_2}\}, \{S_{T_2}, c_2, P_{T_3}\}\}$ (shown in Fig. 2(c)), $C_S = \{S_{T_3}\}$ and $C_P = \{P_{T_1}, P_{T_4}\}$.

(d) Go to step (2) then (3). Compute $P_{Flip} = \{S_{T_3}, d_1, P_{T_4}\}$ (step (3.2)). This P_{Flip} vertex-disjoints all the paths in the current New_A. In step (4), update New_A, C_S, and C_P: $New_A = \{\{S_{T_1}, b_3, P_{T_2}\}, \{S_{T_2}, c_2, P_{T_3}\}, \{S_{T_3}, d_1, P_{T_4}\}\}$ (shown in Fig. 2(d)), $C_S = \emptyset$ and $C_P = \{P_{T_1}\}$.

(e) Go to step (2) then (3). Since $C_S = \emptyset$, $P_{Flip} = S_P || P_{T_1}$ (step (3.1)). This P_{Flip} joints all the paths in the current New_A at a_1, a_2, and a_3, as shown in Fig. 2(e). Therefore in step (4), $Temp = \{\{S_{T_1}, b_3, P_{T_2}\}, \{S_{T_2}, c_2, P_{T_3}\}, \{S_{T_3}, d_1, P_{T_4}\}\}$. Compute $P_{temp} = \{S_{T_1}, a_3, P_{T_1}\}$ (step (4.5)), which is derived from $new_t = \{S_{T_1}, b_3, P_{T_2}\}$, as shown in Fig. 2(f). Update New_A, C_S, and C_P accordingly: $New_A = \{\{S_{T_1}, a_3, P_{T_1}\}, \{S_{T_2}, c_2, P_{T_3}\}, \{S_{T_3}, d_1, P_{T_4}\}\}$, $C_S = \emptyset$ and $C_P = \{P_{T_2}\}$.

(f) Go to step (3). Since $C_S = \emptyset$, $P_{Flip} = S_P || P_{T_2}$ (step (3.1)). This P_{Flip} joints two paths in the current New_A at b_1 and b_2, as shown in Fig. 2(f). Go to step (4), we have $Temp = \{\{S_{T_2}, c_2, P_{T_3}\}, \{S_{T_3}, d_1, P_{T_4}\}\}$. Compute $P_{temp} = \{S_{T_2}, b_2, P_{T_2}\}$ (step (4.5)), which is derived from $new_t = \{S_{T_2}, c_2, P_{T_3}\}$, as shown in Fig. 2(g). Update New_A, C_S, and C_P accordingly: $New_A = \{\{S_{T_1}, a_3, P_{T_1}\}, \{S_{T_2}, b_2, P_{T_2}\}, \{S_{T_3}, d_1, P_{T_4}\}\}$, $C_S = \emptyset$ and $C_P = \{P_{T_3}\}$.

(g) Go to step (3). Since $C_S = \emptyset$, $P_{Flip} = S_P || P_{T_3}$ (step (3.1)).

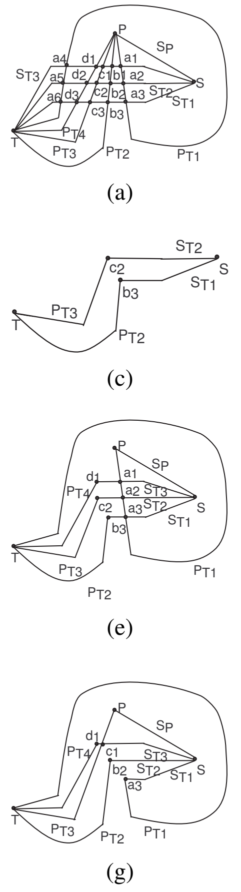
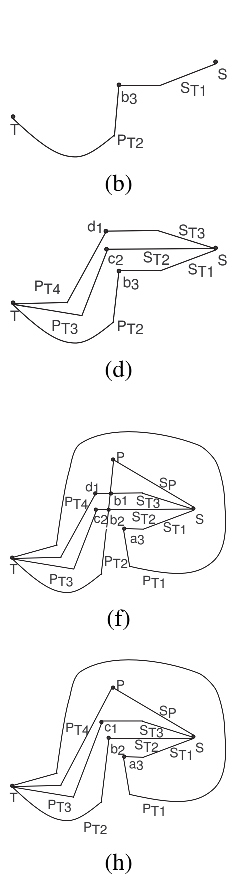



Figure 2: An example to illustrate Algorithm Omvdp for constructing N vertex-disjoint paths

This P_{Flip} joints one path in the current New_A at c_1, as shown in the Fig.2(g). Go to step (4). We have $Temp = \{\{S_{T_3}, d_1, P_{T_4}\}\}$. Compute $P_{temp} = \{S_{T_3}, c_1, P_{T_3}\}$ (step (4.5)). This P_{temp} is derived from $new_t = \{S_{T_3}, d_1, P_{T_4}\}$, as shown in Fig. 2(h). Update New_A, C_S, and C_P accordingly: $New_A = \{\{S_{T_1}, a_3, P_{T_1}\}, \{S_{T_2}, b_2, P_{T_2}\}, \{S_{T_3}, c_1, P_{T_3}\}\}$, $C_S = \emptyset$ and $C_P = \{P_{T_4}\}$.

(h) Go to step (3). Since $C_S = \emptyset$, $P_{Flip} = S_P||P_{T_4}$ (step (3.1)). This P_{Flip} vertex-disjoints all the paths in the current New_A, as shown in Fig. 2(h). In step (4), update New_A, C_S, and C_P: $New_A = \{\{S_{T_1}, a_3, P_{T_1}\}, \{S_{T_2}, b_2, P_{T_2}\}, \{S_{T_3}, c_1, P_{T_3}\}, S_P||P_{T_4}\}$, $C_S = \emptyset$ and $C_P = \emptyset$. Go to step (5) and STOP.

In the following, we will prove that Algorithm Omvdp stops in finite number of steps, and it computes exactly N vertex-disjoint paths.

LEMMA 3.1. *If P_{temp} is found in step (4.5) of Omvdp, then P_{temp} is derived from P_{Flip}.*

PROOF. P_{Flip} joints all the paths in $Temp$. Therefore, there dose not exist an induced path that is derived from a path in $Temp$ only. □

LEMMA 3.2. *Assuming that P_{Flip} is derived from $P_{T_j} \in C_P$, then the intersections of P_{Flip} and the paths in $Temp$ are all in P_{T_j}.*

PROOF. If P_{Flip} is derived from Omvdp's step (3.1), the claim is true. Now we assume that P_{Flip} is derived from $S_{T_i} \in C_S$ in step (3.2). For $\forall\ new_t \in Temp$, without loss of generality, we assume that new_t is derived from $S'_{T_i} \in C_S$ and $P'_{T_j} \in C_P$.

We claim that S_{T_i} does not intersect new_t. This can be proved by the following two observations. First, $S_{T_i} \bigcap S'_{T_i} = \emptyset$. Second, if $\exists x \in S_{T_i}$ s. t. $x \in (P'_{T_j} \bigcap new_t)$, which means x is in the effective residual path of P'_{T_j} employed to construct new_t. However, based on the definition of effective residual path, $(P'_{T_j} \bigcap new_t)$ intersects S'_{T_i} only. Therefore S_{T_i} can not intersect new_t. □

LEMMA 3.3. *All the paths in New_A vertex-disjoint all the paths in C_S in step (4).*

PROOF. Any path in New_A obtained by step (4.1) vertex-disjoints C_S since the path employed to derive the path in New_A is removed from C_S according to step (4.2).

C_S does not change when P_{temp} is derived in step (4.5), as shown in step (4.7). P_{temp} is derived from P_{Flip}, according to the Lemma 3.1. Therefore P_{temp} vertex-disjoints all the paths in C_S except S_{T_i}, which is employed to construct P_{Flip}. According to Lemma 3.2, $P_{temp} \bigcap S_{T_i} = \emptyset$. Therefore, P_{temp} vertex-disjoints all the paths in C_S. □

From Lemma 3.3 and steps (4.1) and (4.6) in Omvdp, we conclude that the number of vertex-disjoint paths, which are found from steps (3) and (4), is determinable. This number is increased by one when step (4.1) is applied, and it remains unchanged when other steps are employed. Next we will prove that the loops containing steps (3) and (4) must stop in finite number of steps.

LEMMA 3.4. *Any loop containing steps (3) and (4) in Algorithm Omvdp will exit in finite number of steps.*

PROOF. We assume that P_{Flip} is derived from $P_{T_j} \in C_P$, and therefore P_{temp} is also derived from P_{T_j} according to Lemma 3.2 and Lemma 3.1. We further assume that P_{temp} is derived from $new_t \in New_A$, and new_t is derived from $S'_{T_i} \in C_S$ and $P'_{T_j} \in C_P$. Then, P_{temp} is derived from S'_{T_i} and P_{T_j}. Assuming the effective partition node, which was employed to construct new_t, is the lth element in X'_i, the ordered intersection set of S'_{T_i}, and the effective partition node employed to construct P_{temp}, is the l'th element in X'_i, then $l > l'$. Since the total number of nodes in New_A and A are finite, the number of P_{temp}s, which are found by step (4.5), must be finite. Therefore any loop containing steps (3)and (4) must exit after finite numbers of steps. □

In the next, we will prove that Omvdp can return exactly N vertex-disjoint paths.

THEOREM 3.2. *Algorithm Omvdp can return exactly N vertex-disjoint paths for a $(N-1)$-credit node S if there exists a N-credit node P and a path S_P between S and P such that S_P vertex-disjoints all the known $(N-1)$ paths from S and the N paths from P to the sink node T.*

PROOF. When Algorithm Omvdp terminates, New_A contains the vertex-disjoint paths for S. Initially $|New_A| = 0$. We will study how New_A is updated in Algorithm Omvdp.

First, if step (2.1) is executed before the algorithm goes to steps (3) and (4), Omvdp terminates with $|New_A| = N$.

Now assume the algorithm does not execute step (2.1) initially. Then it will go to step (3). Notice from step (4) that when C_S becomes empty, it will remain to be empty. Therefore the algorithm will execute steps (3.2) and (4) first, and it will loop to execute steps (3.2) and (4) until the condition in step (2) holds true or $C_S = \emptyset$. Notice from steps (4.1) and (4.2), step (4) maintains the following invariant: $|New_A|+|C_S| = |A| = N-1$ when looping. Therefore if the algorithm goes to step (2.1), it will terminate with $|New_A| = N$.

If the algorithm does not go to step (2.1) to terminate, it will loop to execute steps (3.2) and (4) until $C_S = \emptyset$. From Lemma 3.4, this loop must exit in finite number of steps. When this loop exists, $|New_A| = N - 1$ based on the invariant mentioned above. Now the algorithm will go to steps (3.1) and (4). Note that the algorithm will never go to steps (3.2) and (4) again since step (4) does not place any path back to C_S. Therefore it will loop to execute steps (3.1) and (4) until the condition at step (5) holds true. Again from Lemma 3.4, this loop must exit in finite number of steps. Note that this loop terminates only when step (4.1) is executed, and it terminates at step (5) since S_P is already included in one of the paths in New_A (from step (4.1)). Also note that before the loop exits, $|New_A| = N - 1$ (see step 4.6). Therefore $|New_A| = N$ when the algorithm terminates. □

THEOREM 3.3. *Algorithm Omvdp stops in finite number of steps.*

PROOF. From Lemma 3.4 and the proof of Theorem 3.2, Algorithm Omvpd must stop in finite number of steps. □

In the worst case, Algorithm Omvdp stops in $O(n^2)$ time, where n is the number of vertices that are both in A, B.

3.3 Determinability Study for Constructing One More Vertex-Disjoint Path

As we discussed in section 5, to reduce the complexity, it is important to know the number of vertex-disjoint paths before the discovery of these paths. Here, if we can determine whether there exists one more vertex-disjoint path for a data source, the effort of actually finding the Nth path could be saved when at most $N - 1$ paths exist. In this section, we propose sufficient conditions based on Algorithm Omvdp when one more vertex-disjoint path exists. We are going to adopt the following notations, as illustrated in Table 3.

A	$A = \{S_{T_1}, S_{T_2}, \cdots, S_{T_i}, \cdots, S_{T_{N-1}}\}$, the set of vertex-disjoint paths for the $(N-1)$-credit node S.
B	$B = \{P_{T_1}, P_{T_2}, \cdots, P_{T_j}, \cdots, P_{T_M}\}$, the set of vertex-disjoint paths for the M-credit node P.
S_P	A path between S and P.
A_B	$A_B = \{S_{T_i} \mid \exists P_{T_j} \in B$, s.t. $S_{T_i} \bigcap P_{T_j} \neq \emptyset, S_{T_i} \in A\}$, the set of paths in A that joint at least one path in B.
B_A	$B_A = \{P_{T_j} \mid \exists S_{T_i} \in A$, s.t. $P_{T_j} \bigcap S_{T_i} \neq \emptyset, P_{T_j} \in B\}$, the set of paths in B that joint at least one path in A.

Table 3: Notations to derive the sufficient condition.

3.3.1 S_P *Vertex-Disjoints All Paths in* $A \cup B$

First, we will propose a sufficient condition for a $(N-1)$-credit node to be augmented to N-credit when S_P vertex-disjoints all paths in $A \cup B$. It also means that P is not in any of S's paths and S is not in any of P's paths

Based on Algorithm Omvdp and Theorem 3.2, we obtain the following theorem:

THEOREM 3.4. *Given a $(N-1)$-credit node S, a M-credit node P, and a path S_P between S and P that vertex-disjoints all the paths in both A and B. If $M > |B_A|$ or $|A_B| < |B_A|$, S is a N-credit node.*

PROOF. If $M > |B_A|$, there exists at least one path in B, denoted by P_{T_j}, that vertex-disjoints all paths in A. Then we can compute the N-th vertex-disjoint path for S by concatenating S_P and P_{T_j}, as shown in Algorithm Omvpd's step (2.1).

When $|A_B| < |B_A|$, the construction procedure of Algorithm Omvdp yields $(|A_B|+1)$ vertex-disjoint paths for S when A_B and B_A are the inputs to Omvdp according to the Theorem 3.2. Any path in $A \setminus A_B$ vertex-disjoints all the paths in both A_B and B. Therefore, the number of vertex-disjoint paths for S is N. □

Note that Algorithm Omvdp will also return N vertex-disjoint paths for S if A and B are used as the inputs because all the paths in $A \setminus A_B$ are the 0th-level induced paths and they will be found earlier than the others.

Theorem 3.4 provides a sufficient condition for a $(N-1)$-credit node to be augmented to a N-credit node. Note that this condition is not necessary. As shown in Fig. 3(b), Algorithm Omvdp yields N vertex-disjoint paths for the S in Fig. 3(a), where $N = 3$, $M = 2$, and $|A_B| = |B_A| = 2$. Another example is reported in Fig. 4, for which Algorithm Omvdp computes N vertex-disjoint paths for the S in Fig 4(a), where $N = 4$, $M = 2$, and $3 = |A_B| > |B_A| = 2$. Two more examples are illustrated in Fig. 5(a) and Fig. 5(b), with $|A_B| = |B_A| = 2$ and $2 = |A_B| > |B_A| = 1$, respectively. For these two examples, it is impossible to find out another vertex-disjoint path for S.

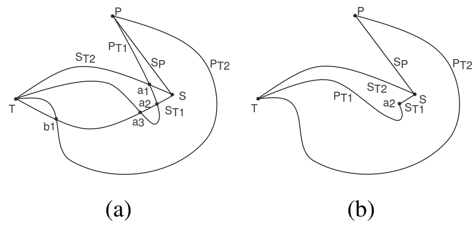


Figure 3: An example of N vertex-disjoint paths found by the Algorithm Omvdp when $|A_B| = |B_A|$, where $N = 3$.

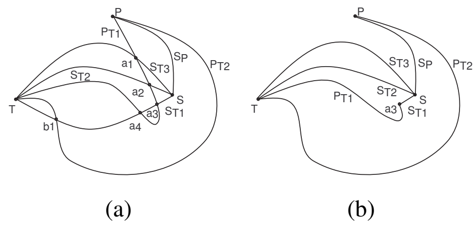


Figure 4: An example of N vertex-disjoint paths found by the Algorithm Omvdp when $|A_B| > |B_A|$, where $N = 4$.

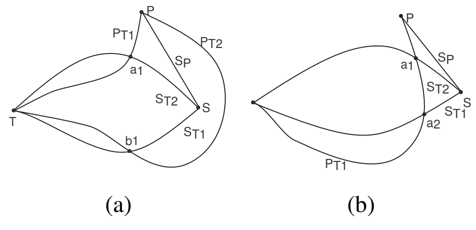


Figure 5: Examples showing that no more vertex-disjoint path can be found for S. (a) $|A_B| = |B_A|$, (b) $|A_B| > |B_A|$.

3.3.2 S_P *Joints At Least One Path In* $A \cup B$

Algorithm Omvdp is dependent on the existence of S_P that vertex-disjoins all the paths in both A and B. As shown in Theorem 3.4, this is suffiecient but not necessary. Therefore a naive question to ask is: can a $(N-1)$-credit node S be augmented to N-credit given a M-credit node P with $M > |B_A|$ or $|A_B| < |B_A|$? In the following, we will study this question by considering different cases.

Case 1. There exists no S_P that does not pass T, as shown in Fig. 6.

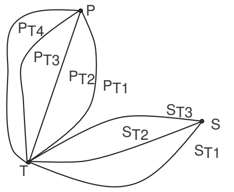


Figure 6: The S_P without the T does not exist.

LEMMA 3.5. *In this case, it is impossible to construct N vertex-disjoint paths for S from A and B.*

PROOF. In this case, S contains only $N-1$ neighbors that could connect to any path in $A \cup B$. Therefore the claim is true. □

Case 2. There exist S_P's without passing T, but each connects to some path in $A \cup B$. We are going to consider the following four sub-cases:

1. S is not in any of P's paths, and P is not in any of S's paths.
2. P is in one of S's paths, but S is not in any of P's paths.
3. S is in one of P's paths, but P is not in any of S's paths.
4. S is in one of P's paths, and P is in one of S's paths.

Subcase 2.1. S is not in any of P's paths, and P is not in any of S's paths.

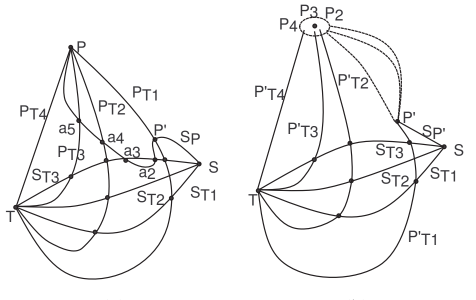
Consider all the nodes from S to P in the path S_P. Let P' be the first node in this path that joints either a path in A, or a path in B. Without loss of generality, we assume the path in A (B) that intersects S_P at P' is S_{T_1} (P_{T_1}). Note that none of the intermediary nodes from S to P' resides in any path in A and B.

LEMMA 3.6. *If P' joints a path P_{T_1} in B, then there exist N vertex-disjoint paths for a $(N-1)$-credit node S if P' is closer to P than any other node through which P_{T_1} intersects a path in A.*

PROOF. We prove this lemma by constructing M virtual vertex-disjoint paths from P' to T. The construction procedure is sketched as follows. We first replace P with $M-1$ virtual nodes $P_2, \cdots, P_{M-1}, P_M$, with P_i connecting to P_{T_i} for $i = 2, 3, \cdots, M$. Then we introduce $M-1$ virtual edges connecting P' to each P_i for $i = 2, 3, \cdots, M$. Now we obtain $M-1$ virtual vertex-disjoint paths from P' to T, denoted by $P'_{T_2}, \cdots, P'_{T_M}$. We construct the Mth vertex-disjoint path from P' to T with the subpath from P' to T in P_{T_1}. Denote this path by P'_{T_1}. Denote the set of M (virtual) vertex-disjoint paths from P' to T by B'. With B', $S_{P'}$ and A as inputs to Algorithm Omvdp, we can obtain a set of N vertex-disjoint paths for S. Denote this set by A'.

We claim that at most one path in A' contains a virtual node and virtual edge. This is true because Algorithm Omvdp utilizes $Flip$ to compute an induced path, which starts from a path $S_{T_i} \in A$ and switch to another path $P'_{T_j} \in B'$ at the intersection node of S_{T_i} and P'_{T_j}. And no virtual node and edge will be included in any induced path since the virtual part of any P'_{T_j} always comes before the intersection node of P'_{T_j} and $S_{T_i} \in A$. Therefore the only path, which possibly contains a virtual node and edge, is the path derived from $S_{P'}$. If there exists such path, we replace the virtual node with P and virtual edge with the path from P' to P in P_{T_1}. □

An example to illustrate the construction procedure in the proof of Lemma 3.6 is shown in Fig. 7.

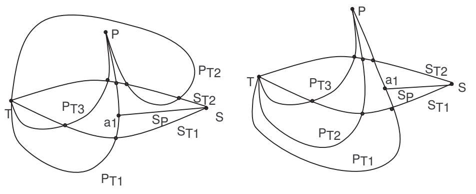


(a) (b)

Figure 7: S_P first joints P_{T_1} at P', and P' is closer to P than any other node of P_{T_1} that intersects with a path in A.

If there exists a node in P_{T_1} that intersects a path in A and this node is before P' in the path P_{T_1}, we may not be able to find out N vertex-disjoint paths for a $(N-1)$-credit node S. An example is illustrated in Fig. 8, where 3 vertex-disjoint paths for S in Fig. 8(a) can be computed, but the 3rd vertex-disjoint path does not exist for the figure shown in Fig. 8(b). This observation helps us to derive the following lemma:

LEMMA 3.7. *The existence of N vertex-disjoint paths for a $(N-1)$-credit node S is not guaranteed if there always exists a node in P_{T_1} that intersects a path in A and is closer to P than P'.*



(a) (b)

Figure 8: S_P joints P_{T_1} first, but there exists a node in P_{T_1} that intersects a path in A and is closer to P than P'.

If all S_P's joint S_{T_1} first, the existence of N vertex-disjoint paths for a $(N-1)$-credit node S is not guaranteed, as illustrated by the examples given in Fig. 9.

LEMMA 3.8. *The existence of N vertex-disjoint paths for the $(N-1)$-credit node S is not guaranteed if all S_P's joint a path in A first.*

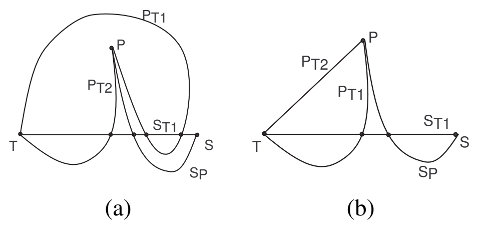


Figure 9: All S_P's joint the paths in A first. (a) There exist $N = 2$ vertex-disjoint paths for S; (b) The second vertex-disjoint path for S does not exist.

Subcase 2.2. P is in one of S's paths, and S is not in any of the P's paths.

In this case, the existence of N vertex-disjoint paths for a $(N-1)$-credit node S can not be guaranteed, as illustrated by the two examples in Fig. 10. Therefore, we have

LEMMA 3.9. *There existence of N vertex-disjoint paths for the* $(N-1)$*-credit node S is not guaranteed when P is in one of the paths in A.*

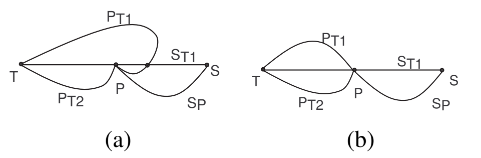


Figure 10: The node P is in one of the S's paths. (a) There exist $N = 2$ vertex-disjoint paths for S; (b) The second vertex-disjoint path for S does not exist.

Subcase 2.3. S is in one of P's paths, and P is not in any of S's paths.

Without loss of generality, we assume that S is in P_{T_1}. Let Y_1 denote the ordered intersection set of P_{T_1}.

LEMMA 3.10. *There exist N vertex-disjoint paths for the* $(N-1)$*-credit node S if S is the first node in* Y_1.

PROOF. Let S_P be the subpath from S to P in P_{T_1}. Let b be the second element in the ordered set Y_1. If there exists a node a such that $a \in P_{T_1}$, $a \notin Y_1$, and a is between S and b in P_{T_1}, then we can create a virtual edge connecting P to a. Let P'_{T_1} be the path concatenated from the virtual edge and the subpath of P_{T_1} from a to T. The construction of P'_{T_1} is illustrated in Fig. 11(a) and Fig. 11(b). Now we use P'_{T_1} to replace P_{T_1} in B to obtain B'. Then S_P vertex disjoints all paths in A and B'. By applying Algorithm Omvdp to A and B', we obtain the set A' containing N vertex-disjoin paths for S. Note that the virtual edge exists in at most one of the paths in A', and if it exists, S_P exists too in the same path. Therefore we can replace the subpath containing the virtual edge and S_P with P_{T_1}'s subpath from S to a.

If such an a does not exist and P_{T_1}'s subpath from S to T completely overlaps a path, denoted as S_{T_1}, in A, then S_{T_1} vertex-disjoints all the paths in B except for P_{T_1}. Now we remove S_{T_1} from A to obtain A' and remove P_{T_1} from B to obtain B'. Since $|A'| = |A| - 1 = N - 2$, $|B'| = |B| - 1 = M - 1$, and S_P vertex-disjoints all paths in A' and B', from Theorem 3.4, there exists $N - 1$ vertex-disjoint paths for S that can be constructed from S_P, A', and B' only. Since the removed S_{T_1} vertex-disjoints all paths in B and A, it vertex-disjoints all the $N - 1$ paths constructed for S. Therefore, we can include this path as the Nth path of S to T.

If such an a does not exist and P_{T_1} partly overlaps a path in A starting with S. Without loss of generality, let S_{T_2} be the path in A that partially overlaps P_{T_1} and the end node of this overlap is c, as shown in Fig. 11(c). We add a virtual edge from c to P that vertex-disjoints all the paths in A and B, as shown in Fig. 11(d). We replace P_{T_1} with the concatenation of the virtual edge and P_{T_1}'s subpath from c to T to obtain B'. Since $|B'| = |B| = N$, and S_P vertex-disjoints all paths in A and B', from Theorem 3.4, there exists N vertex-disjoint paths for S that can be constructed from S_P, A, and B'. We claim that the virtual edge dose not exist in any of the above constructed N paths. The only possible path that contains the virtual edge is derived from S_P, and this possible path must joint the path derived from S_{T_2}, which is inconsistent with the fact that the N newly-constructed paths are vertex-disjoint. □

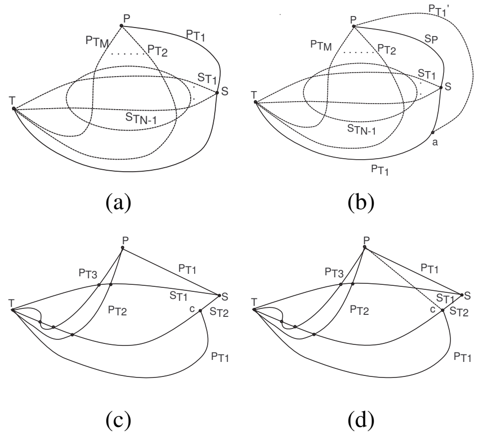


Figure 11: S is closer to P than all nodes in Y_1, the ordered intersection set of P_{T_1}.

LEMMA 3.11. *The existence of the* $(N+1)$*th vertex-disjoint path of S is not guaranteed if S is the first node in* Y_1.

PROOF. For the figure shown in Fig. 11(a), where S is the only intersection node of P_{T_1}, we divide P_{T_1} into two subpaths, with one from S to P labeled as S_P, and the other from S to T labeled as S_{T_N}. Then the original problem is transformed to the problem where S has N vertex disjoint paths and P has $M - 1$ vertex-disjoint paths. The existence of the $(N+1)$th vertex-disjoint path of S can't be guaranteed based on Theorem 3.4 and the examples following Theorem 3.4. □

When S is not the first node in the ordered set Y_1, the existence of N vertex-disjoint paths for S can't be guaranteed, as illustrated by the two examples in Fig. 12. Therefore we have

LEMMA 3.12. *The existence of N vertex-disjoint paths is not guaranteed when S is not the first node in the ordered intersection set* Y_1.

Note that in Fig. 12(b), there exist $N + 1$ vertex-disjoint paths for S but Fig. 12(a) does not have N vertex-disjoint paths for S for $N = 3$. Therefore we have

LEMMA 3.13. *The existence of the* $(N+1)$*th vertex-disjoint path is not guaranteed when S is not the first node in the ordered intersection set* Y_1.

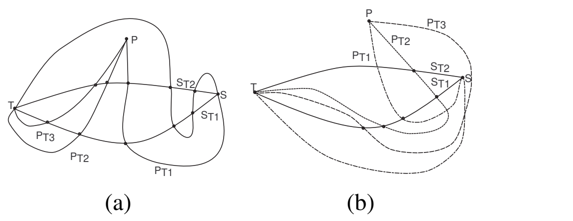


Figure 12: S is not the first node in P_{T_1}'s intersection set Y_1. Here $N = 3$. (a) There does not exist N vertex-disjoint paths for S; (b) There exist N vertex-disjoint paths for S.

Subcase 2.4. S is in one of P's paths, and P is in one of S's paths.

Without loss of generality, we assume that S is in P_{T_1} and P is in S_{T_1}.

LEMMA 3.14. *The existence of N vertex-disjoint paths is not guaranteed if S is in one of P's paths and P is in one of S's paths.*

We use the examples illustrated in Fig. 13 to demonstrate the correctness of this lemma. Note that it is clear that there does not exist N vertex-disjoint paths for S in Fig. 13(a), but there exist N vertex disjoint paths for the S in Fig. 13(b), where $N = 3$. Note that Fig. 13(b) is drawn in 3D, and only the black nodes represent the intersections.

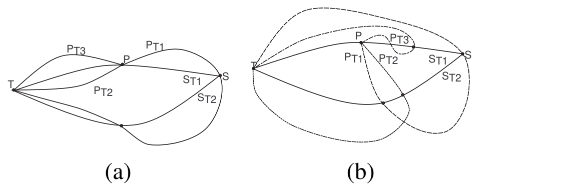


Figure 13: S is in one of P's paths, and P is in one of the S's paths. Here $N = 3$. (a) S does not have N vertex-disjoint paths; (b) S has N vertex disjoint paths.

LEMMA 3.15. *The existence of the $(N + 1)$th vertex-disjoint path is not guaranteed if S is in one of P's paths and P is in one of S's paths.*

PROOF. Again we use the two examples illustrated in Fig. 13 to demonstrate the correctness of this lemma. There does not exist N vertex-disjoint paths for S in Fig. 13(a), but there exist $N + 1$ vertex-disjoint paths for S in Fig. 13(b). Therefore the lemma holds. □

From the above analysis, we realize that under certain conditions, a $(N-1)$-credit node S can be augmented to N-credit when S_P does not vertex-disjoint all paths in A and B. To summarize these conditions, we introduce the concept of *effective information node* first.

DEFINITION 3.8. *Given a $(N - 1)$-credit node S and a M-credit node P, such that P is not in any of S's paths and S is not in any of P's paths, a node D is S's* **effective information node** *if one of the following two conditions hold: (1) D is P; (2) D is in P_{T_j} for $\forall j \in \{1, 2, \cdots, M\}$ such that D is closer to P than any node in Y_j.*

With the definition of the effective information node, the results reported in Theorem 3.4 and Lemmas 3.5-3.15 can be summarized by the following theorem:

THEOREM 3.5. *Given a $(N - 1)$-credit node S, a M-credit node P, such that P is not in any of S's paths and S is not in any of P's paths, and a path S_P. If S_P joints A and B first in an effective information node, and $M > |B_A|$ or $|A_B| < |B_A|$, then S is a N-credit node.*

Note that Theorem 3.5 states that the sufficient conditions for S to be N-credit include: (i) $\exists P$, such that P is not in any of S's paths and S is not in any of P's paths; (ii) $\exists S_P$ that joints A and B first in an effective information node; (iii) $M > |B_A|$ or $|A_B| < |B_A|$.

THEOREM 3.6. *If A, B, and S_P satisfy the sufficient conditions mentioned above, the maximum number of vertex-disjoint paths that can be found for S is N.*

PROOF. The number of paths that starts from S is N under the two conditions. Therefore, the maximum possible number of vertex disjoint paths is N. □

4. VERTEX-DISJOINT ROUTE RECOVERY FRAMEWORK

In a large scale network, it is hard and expensive for an average node to obtain the global information. Therefore, we concentrate on the local and distributed solution. Our route recovery framework contains two phases: checking and reconstructing. In the first phase, whether there exists N vertex-disjoint paths for a node is determined based on some sufficient conditions. In the second phase, the N vertex-disjoint paths to the sink are reconstructed. Here, the motivation and idea is that the complexity of checking phase is less than the complexity of reconstructing phase, such that the complexity of multipaths reconstruction can be reduced comparing with repeating the reconstructing phase.

4.1 Checking Phase

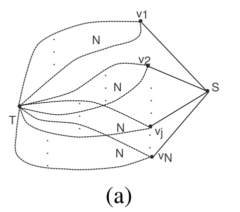
Based on our results studied in section 3, to check whether N vertex-disjoint paths to the sink exist or not, a node needs to know the number of vertex-disjoint paths available to each neighbor. The following theorem summarizes the sufficient condition.

THEOREM 4.1. *The existence of N vertex-disjoint paths is guaranteed if the node has N neighbors with each having at least $N+1$ vertex-disjoint paths to the sink.*

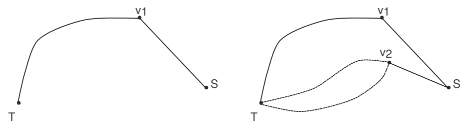
PROOF. Let S be the node that needs to reconstruct its N vertex-disjoint paths to the sink, as shown in Fig. 14(a). Let $v_1, v_2, \cdots, v_j, \cdots, v_N$ be the neighbors of S that have at least $N + 1$ vertex-disjoint paths. Then, for any v_j, there must exist N vertex-disjoint paths that do not pass S as shown in Fig. 14(a).

In the following, we construct N vertex-disjoint paths for S step by step.

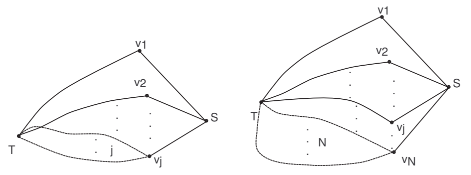
- Pick one of v_1's paths, which does not pass any of $v_2, \cdots, v_j, \cdots, v_N$. There must exist such a path because it is impossible for N vertex-disjoint paths sharing $N - 1$ nodes. The concatenation of this path and the edge between S and v_1 forms the first path for S, as shown in Fig. 14(b). And this path does not pass any of $v_2, \cdots, v_j, \cdots, v_N$.
- Pick two of v_2's paths, which do not pass any of $v_3, \cdots, v_j, \cdots, v_N$, as shown in Fig. 14(c). According to Theorem 3.2, there exist two vertex-disjoint paths between S and T.
- $\cdots$
- Pick j of v_j's paths, which do not pass any of $v_{j+1}, \cdots, v_N$, as shown in Fig. 14(d). Based on Theorem 3.2, there exist j vertex-disjoint paths between S and T.



(a)



(b) (c)



(d) (e)

Figure 14: The procedure of constructing N vertex-disjoint paths.

- $\cdots$
- Pick N paths from v_N's paths, as shown in Fig. 14(e). Based on Theorem 3.2, there exist N vertex-disjoint paths between S and T.

Therefore, the claim is true. □

This checking is simple, and its cost is $O(1)$. If the node has passed the checking, it can go to the reconstruction phase. Otherwise, the node can either move to a new location (for mobile nodes) with the hope of passing the checking, or force an reconstruction immediately because the condition is not necessary. For a node that just deployed to the network, the following corollary can be derived from the proof of Theorem 4.1.

COROLLARY 4.1. *The existence of N vertex-disjoint paths is guaranteed if the newly-deployed node has N neighbors that have at least N vertex-disjoint paths.*

The above two simple checking methods only request the node to be aware of the number of vertex-disjoint paths of its neighbors. It is not necessary for the node to know any other information, such as the details of the vertex-disjoint paths. Therefore, the checking methods can be applied to any protocol mentioned in Section 2.

When source routing is adopted, we can obtain stronger sufficient conditions to check the availability of N vertex-disjoint paths. Assume the node S has M vertex-disjoint paths with $M < N$ already. We can derive the following corollary based on the proof of Theorem 4.1.

COROLLARY 4.2. *The existence of N vertex-disjoint paths is guaranteed if the node S has $N - M$ neighbors with each having at least $N + 1$ vertex-disjoint paths and none of them appearing in the M vertex-disjoint paths of S.*

We can obtain an even stronger condition if the node's neighbors can announce the number of vertex-disjoint paths that do not pass this node.

COROLLARY 4.3. *The existence of N vertex-disjoint paths is guaranteed for S if S has $N - M$ neighbors with none of them passing the M available paths of S and each having at least N vertex-disjoint paths that do not pass S.*

The nodes that have found N vertex-disjoint paths should know its next-hop neighbors serving as relays. Therefore we obtain the following corollary based on the proof of Theorem 4.1, Lemma 3.10 and Corollary 4.3.

COROLLARY 4.4. *The existence of N vertex-disjoint paths is guaranteed for S if S has $N - M$ neighbors with none of them passing the M available paths of S and each having at least N vertex-disjoint paths that either do not pass S or S is the first relay node in one of the paths.*

This is an interesting result. It indicates that S can get rewarded from others it has helped. This means that each node should help others in order to get support for its own route maintenance. This results can be applied to other applications such as P2P download.

4.2 Reconstructing Phase

In general, all the existing methods mentioned in section 2 can be employed in this phase. When source routing is adopted, there exist two efficient reconstructing methods for a node that passes the checking phase. In the first method, the node collects vertex-disjoint path information from $N - M$ neighbors with each having at least N vertex-disjoint paths to the sink, and then employs either max-flow algorithm or Omvdp algorithm (introduced in section 3 for proofing the theoretical results) to reconstruct its own N vertex-disjoint paths. In the second method, the node first broadcasts its available vertex-disjoint path information to all neighbors when the number of available vertex-disjoint paths is $N - 1$. Then its neighbors check whether N vertex-disjoint paths exist or not based on Theorem 3.5 and Lemma 3.10. If the answer is positive, the neighbors can either compute N vertex-disjoint paths for the node or notify the node of the existence to let the node make the decision on which neighbor it shall rely on. Note that the node does not need to flood RREQs and waits for the sink's response in both reconstruction methods. Therefore network performance could be improved because the information collection is restricted to selected neighbors, which lead to less interference comparing with the flooding of RREQs. In addition, network resources can be conserved because only the selected nodes participate in the reconstruction. Note that a node could still try the two reconstruction methods first instead of flooding RREQs to the network even it fails the checking.

When source routing is not available and the node passes the checking phase, the node can send RREQs to the selected neighbors that are guaranteed to be helpful. This also reduces the resource consumption.

5. DISCUSSION

5.1 Existence Checking Problem

The checking phase we proposed in section 4 can be treated as a complement to the classic max-flow solution. Let's take a look at the following example. Given $m \times m$ grids representing the possible deployment points for a node v, finding a deployment grid for v such that there exist at least N multipaths to the sink nodes from v. For this problem, the best known max-flow algorithm has a time complexity of $O(m^2n)$ since it needs to be fired $m \times m$ times in the worst case. On the other hand, the checking phase reduces the time complexity to $O(m^2) + O(n)$ if a positive answer is returned. Note that the checking may not guarantee to identify a deployment grid that supports N multipaths, even though there exists such a grid for v, since it is based on a sufficient condition.

This example motivates us to study the following general **Existence Checking Problem**.

Given a problem that can be solved or reported as unsolvable for an input in $O(M)$ time, does there exist a method that can sufficiently and necessarily check the existence of a solution for any input in $O(N)$ time with $O(M) > O(N)$, such that the complexity of finding an input that guarantees the solvability of the problem can be reduced?

The above problem has many practical applications. Nevertheless, we conjecture that it is impossible to get a necessary and sufficient checking procedure that can reduce the time complexity.

5.2 Unified Geometric Localization and Multipath Construction

Location information is critical to many network applications []. Generally speaking, a to-be-localized node needs to have range (distance) information toward at least $(d+1)$ beacons in order to be uniquely localized in d-dimensional space via a procedure termed *$(d + 1)$-lateration localization*. A node becomes a beacon whenever its position information is available. In [1], we have identified the following sufficient and necessary condition for a node to be localisable:

LEMMA 5.1. *A node v can be localized by $(d + 1)$-lateration localization method in d-dimensional space, if and only if v has at least $(d + 1)$ vertex disjoint paths to $(d + 1)$ distinct beacons, and each of the nodes in v's paths also has at least $(d + 1)$ vertex disjoint paths to $(d + 1)$ distinct beacons.*

Notice that if we treat a to-be-localized node as a newly-deployed node, and all the relevant beacons as a virtual sink[3], then the following result can be obtained from Corollary 4.1 and Lemma 5.1.

THEOREM 5.1. *If a node v can be localized by a $(d + 1)$-lateration localization method in d-dimensional space, then there exist $(d + 1)$ vertex-disjoint paths from v to the sink node.*

Considering a sensor network whose initial beacons are sinks and/or one-hop neighbors of the sink nodes, based on Theorem 5.1, any $(d+1)$-lateration localization procedure implies the availability of a $(d + 1)$ multipath routing topology. In other words, the procedure of localizing a d-dimensional network via a $(d + 1)$-lateration localization method constructs $(d+1)$ vertex-disjoint paths for any localizable node to the sink. However, the existence of $(d + 1)$ vertex-disjoint paths for a node can not guarantee localizability of the node, as illustrated by the counterexample shown in [2]. This is consistent with Theorem 5.1, which provides a sufficient condition.

6. CONCLUSION AND FUTURE WORKS

In this paper we study the problem of route recovery in node-disjoint multipath routing for many-to-one sensor networks. We identify the sufficient conditions under which multipaths can be recovered. We propose a route recovery framework consisting of a constant-time checking phase and a reconstruction phase. When multipath source routing is adopted, the checking is much stronger and the reconstruction is much more efficient. Our research provides help for route recovery, for data source relocation, and for newly-deployed sensors to construct the required multipaths.

As a future research, we will leverage on the sufficient conditions identified in this paper to design efficient node-disjoint multipath routing protocols for general ad hoc and sensor networks, with the objectives of resource management and security enhancement.

[3]Since all relevant beacons play exactly the same role in localizing the to-be-localized node, they are jointly treated as a virtual sink.

7. ACKNOWLEDGMENTS

This work was completed when Wei Cheng was visiting The George Washington University. The research was partially supported by the US National Science Foundation under grants CNS-0347674 and CNS-0721669, the National Basic Research Program of China (973) under Grant No. 2005CB321801, and the National Natural Science Foundation of China under Grant No. 60673167 and No. 90604006.

8. REFERENCES

[1] W. Cheng, A. Y. Teymorian, L. Ma, X. Cheng, X. Lu, and Z. Lu. Underwater localization in sparse 3d acoustic sensor networks. In *IEEE INFOCOM*, 2008.

[2] D. Goldenberg, A. Krishnamurthy, W. Maness, Y. R. Yang, A. Young, A. S. Morse, A. Savvides, and B. Anderson. Network localization in partially localizable networks. In *IEEE INFOCOM*, 2005.

[3] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing*, pages 153–181, 1996.

[4] S.-J. Lee and M. Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. In *ICC*, 2001.

[5] S. Li and Z. Wu. Node-disjoint parallel multipath routing in wireless sensor networks. In *ICESS*, 2005.

[6] C. Liu, M. Yarvis, W. S. Conner, and X. Guo. Guaranteed on-demand discovery of node-disjoint paths in ad hoc networks. *Computer Communications*, 30:2917–2930, 2007.

[7] W. Lou and Y. Kwon. H-spread: A hybrid multipath scheme for secure and reliable data collection in wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 5, 2006.

[8] M. K. Marina and S. R. Das. On-demand multipath distance vector routing in ad hoc networks. In *ICNP*, 2001.

[9] G. Parissidis, V. Lenders, M. May, and B. Plattner. Multi-path routing protocols in wireless mobile ad hoc networks: A quantitative comparison. In *NEW2AN*, 2006.

[10] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *WMCSA*, 1999.

[11] L. Popa, A. Rostamizadeh, R. Karp, C. Papadimitriou, and I. Stoica. Balancing traffic load in wireless networks with curveball routing. In *MobiHoc*, pages 170–179, 2007.

[12] H. Ripphausen-Lipa, D. Wagner, and K. Weihe. The vertex-disjoint menger problem in planar graphs. *SIAM Journal on Computing*, 26:331–349, 1997.

[13] M. Ruiping, X. Liudong, and M. H. E. A new mechanism for achieving secure and reliable data transmission in wireless sensor networks. In *HST*, pages 274–279, 2007.

[14] A. Srinivas and E. Modiano. Minimum energy disjoint path routing in wireless ad-hoc networks. In *MobiCom*, 2003.

[15] J. W. Suurballe. Disjoint paths in a network. *Networks*, pages 125–145, 1974.

[16] Z. Ye, S. V. Krishnamurthy, and S. K. Tripathi. A framework for reliable routing in mobile ad hoc networks. In *IEEE INFOCOM*, pages 270–280, 2003.