Methods

The multipath search algorithm

The major goal of the search algorithm is to construct a directed acyclic (DA) graph between the source router s and the destination d in a distributed manner, given a network. The topology information of the network can be acquired via the corresponding functions of existing routing protocols, such as open shortest path first (OSPF), routing information protocol (RIP) and intermediate system-to-intermediate system (IS-IS). Each node executes the same algorithm to build up its routing table.

In the search algorithm, the source s is the first router to process. In principle, for a node v, whether an entry is added to the routing table depends on whether it has been confirmed that one of the direct neighbors is connected forward to the destination d or is node d exactly. If yes, then node v is also connected forward to node d and two new routing entries are added to the tables of it and the neighbor, respectively. Unlike general search ideas, where the search process exits once the destination is reached, this algorithm continues to search for other nodes that are also connected forward to the destination node d to obtain new transmission paths, according to the breadth-first strategy. Note, here the search process cannot return back to deal with the previous node until all the direct neighbors have been checked up. For example, as shown in Fig. 1b, if neighbor v_{i+1} has been confirmed to be connected to the destination d, i.e., a packet sent by v_{i+1} can reach node d forward, then two entries are added to the routing tables of v_i and v_{i+1}, respectively. On the other hand, since other neighbors v_{i+2} and v_{i+3} are also connected forward to node d, they also can act as the next hops of v_i to the destination.

Give a node v, the processing algorithm can be divided into two subfunctions for convenience: one is to process the direct neighbors of v that are connected forward to the destination, and the other is to process the neighbors that are not connected to the destination or it is not confirmed whether they are connected to the destination (they are added to the queue L_1 or L_2). If the edge between node v and a direct neighbor n has not been visited yet and the minimum hop number from the source s to v is less than or equal to the minimum hop number to neighbor n, then both routing tables of node v and n should be updated, provided that it is confirmed that node n is connected forward to the destination. This subfunction is shown in Extended Data Fig. 1 (Function F_1), where A is the adjacency matrix of a network, A_0 is a matrix with the same size as A, containing the state information whether a directed edge has been visited, M_0 is a vector containing the minimum hop numbers from the source s to other nodes, and queues L_1 and L_2, indexed by node v, are used to temporarily contain the direct neighbors of v to be processed in later steps.

Clearly, as shown in Extended Data Fig. 1, if it is not confirmed whether neighbor n is connected to the destination d or not, then which queue (L_1 or L_2) the node will be added to depends on the minimum hop numbers from the source s to it and node v. To clarify the difference, two cases are shown in Extended Data Fig. 3, where the black solid lines are the links that have not been visited while the blue ones represent the links that have been visited (the blue arrow refers to the directed edge that not only has been visited but has been selected out to form the DA graph, which means two routing entries also have been added to the routing tables of both sides, respectively), and the red dashed arrow represents a directed edge that the neighbor's minimum hop number is less than that of node v (it is possible to form routing loops).

As shown in Extended Data Fig. 3a, node v_3 is a direct neighbor of v_1 and its minimum hop number from the source node s is larger than the number of v_1. Since at that time node v_3 has not been visited (only the edge between them has been visited), it is impossible to know whether node v_3 is connected forward to node d or not. Thus, node v_3 is added to the end of queue L_1 of node v_1. Similarly, the neighbor v_4 is also added to queue L_1. On the other hand, as shown in Extended Data Fig. 3b, since node v_3 is visited prior to node v_2 according to the depth-first strategy and the minimum hop number of node v_3 is larger than that of node v_2, v_2 is added to queue L_2 of node v_3.

Next, the nodes in L_1 of node v will be processed first because they have larger hop numbers from the source s, which is in accord with the principle of DA graphs. The subfunction is given in Extended Data Fig. 2 (Function $F2$), where vector B_0 contains the state whether a node has been checked (for example, 1 means *yes* while 0 represents *no*). Clearly, for a neighbor n in L_1, if it has not been visited yet, then the state in B_0 is modified and Function F_1 is called first to process its direct neighbors, using the depth-first search idea. If it is confirmed that neighbor n is connected forward to d, then the routing tables of node v and n will be updated. Note, the second loop (line 11 to 15) is used to restore the states because the nodes may be useful in the later steps though they are not connected to d forward. For example, as shown in Extended Data Fig. 3a, though node v_4 is not connected forward to the destination d, it can be connected to node d reversely via node v_1, which allows more nodes to relay packets to the destination.

Only if it is confirmed that node v cannot be connected forward to node d after all the neighbors in L_1 have been checked up, then the nodes in L_2 will be processed by Function F_2. For example, as shown in Extended Data Fig. 3b, since the current node is v_3 and the destination d is exactly a direct neighbor of it, Function F_2 is not called, namely line 20 of Function F_1 is not executed to process the node v_2 in L_2, indexed by node v_3, which means the search algorithm will return back to process the other neighbor node in queue L_1 of node v_1, i.e., node v_4.

Implementation of the search algorithm

An example of the search algorithm is shown in Extended Data Fig. 4, where the black lines are the edges that have not been visited while the blue ones represent the links that have been visited, and the blue arrows refer to the edges that not only have been visited but have been selected out to form a directed acyclic graph between the source router s and the destination d, i.e., two entries have been added in the routing tables of both sides of the directed edge, respectively. Each node of the network, denoted by v for convenience, executes Function F_1, in which the source router s is the first node to process.

Clearly, node s has two direct neighbors v_1 and v_2. Since both of them have not been visited yet, node v does not know whether they can be connected forward to d and has to add them to queue L_1, indexed by node s, which is shown in Extended Data Fig. 4a.

Without loss of generality, assume node v_1 (in queue L_1 of node s) is first checked by Function F_2. Since node v_1 has not been visited, node v jumps to call Function F_1 via line 4 of Function F_2. Since the destination d is a direct neighbor of v_1, so the directed edge from node v_1 to d is redrawn as a blue arrow line, which means that v_1 is connected forward to d and two entries are added to the routing tables of them, respectively. On the other hand, node v_2 is also a direct neighbor of v_1 while both of them have the same minimum hop number from the source, so it is added to queue L_1, indexed by v_1. This is shown in Extended Data Fig. 4b.

The next step is to check the nodes in queue L_1, indexed by v_1. Obviously, as shown in Extended Data Fig. 4c, node v_2 is the only one to be processed by Function F_2. Since the destination d is also a direct neighbor of v_2, the directed edge from v_2 to d is re-drawn as a blue arrow line, which means node v_2 is also connected forward to d and two entries will be added to the routing tables of them, respectively.

Note, for node v_2, after its direct neighbor d having been checked by Function F_2, node v returns back to continue to execute line 6 of Function F_2 (it is executed for the first time). Thus, the directed edge from node v_1 to v_2 is re-drawn as a blue arrow line, i.e., two entries are added to the routing tables of them, respectively, which is shown in Extended Data Fig. 4d. Similar operations are performed to re-draw the directed edge from s to v_1, as shown in Extended Data Fig. 4e.

For the source node s, since the other neighbor v_2 in queue L_1 is connected forward to d, similar operations are performed to re-draw the directed edge from it to v_2, which is shown in Extended Data Fig. 4f.

Finally, a DA graph is constructed from the source s to node d by interconnecting all the selected nodes along the directed edges between them. This is similar to the generation procedure of a SPF tree by the shortest path algorithm.

The setting process of encoding matrices

The procedure is started up by the destination d. After receiving a request from the source router s, it first generates a $|In(d)| \times |In(d)|$ random square matrix $\boldsymbol{R}$ with full rank, and then sends each row vector of $\boldsymbol{R}$ over an incoming edge $e_i \in In(d)$ reversely to the source s, where $In(d)$ refers to the incoming edge set of d.

At an intermediate node v, $|Out(v)|$ row vectors can be received from the outgoing edges and given as a $|Out(v)| \times |In(d)|$ matrix, denoted by $\boldsymbol{Y}'_v$ for convenience, where $Out(v)$ is the outgoing edge set of v. Pre-multiplying this matrix by the local encoding matrix (kernel), we can obtain a new matrix

$$\boldsymbol{X}'_v = \boldsymbol{K}_v \cdot \boldsymbol{Y}'_v$$

where $\boldsymbol{K}_v$ is a $|In(v)| \times |Out(v)|$ random matrix (the rank $r(\boldsymbol{K}_v) = \text{Min}(|In(v)|, |Out(v)|)$), generated by node v itself. Similarly, each row vector of $\boldsymbol{X}'_v$ is sent over an incoming edge $e_i \in In(v)$ reversely to the source.

This procedure is performed until the source node s, where the received matrix is denoted by $\boldsymbol{Y}'_s$ for convenience and the row number is $|Out(s)|$**.** Note, linear row transformation is employed at each intermediate node in the above procedure, namely each row vector of $\boldsymbol{Y}'_s$ is a linear combination of the row vectors of matrix $\boldsymbol{R}$, according to the theory of linear algebra. Thus, the maximum dimension ω of the source messages (the number of edge-disjoint paths from the source s to the destination d) is equal to the row rank of $\boldsymbol{Y}'_s$, which means ω independent column vectors[5] can also be extracted from $\boldsymbol{Y}'_s$ because linear column transformation will be employed in the data transmission phase. The local encoding matrix $\boldsymbol{K}_s$ of the source node s can be obtained by calculating the generalized inverse of a matrix consisting of ω independent column vectors from $\boldsymbol{Y}'_s$, and for node d, the corresponding column vectors with the same order information as the selected independent column vectors of $\boldsymbol{Y}'_s$ can be extracted from $\boldsymbol{R}$ to form its local encoding matrix $\boldsymbol{K}_d$, which is used to decode the received packets in the data transmission phase.

Competing interests The authors declare no competing interests.

Function F_1: Processing algorithm of a node v

Input: the source node s, the destination d, a node v;
Output: L_1, L_2, new routing entries;
Data: adjacency matrix A, state matrix A_0, state vector M_0;

for *all neighbors n of node v* **do**
    if $A_0[v][n] == 0$ **then**
        if $M_0[n] \geq M_0[v]$ **then**
            if *node n is connected to d* **then**
                A new entry is added to the routing table of v(src = s, dst = d, next = n);
                A new entry is added to the routing table of n(src = d, dst = s, next = v);
            else
                node n is added to the end of L_1;
            end
        else
            node n is added to the end of L_2;
        end
        $A_0[v][n] = 1$;
    end
end
if L_1 *is not empty* **then**
    Call $F_2(s, d, v, L_1)$;
end
if *v is not connected d and* L_2 *is not empty* **then**
    Call $F_2(s, d, v, L_2)$;
end

Extended Data Fig. 1. Processing algorithm of a router node v.

Function F_2: Processing algorithm of queue L_i of v

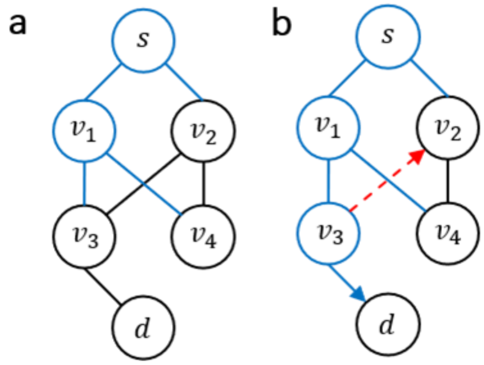
Input: node s, d, current node v, $queue\ L_i (i = 1, 2)$;

Output: new routing entries;

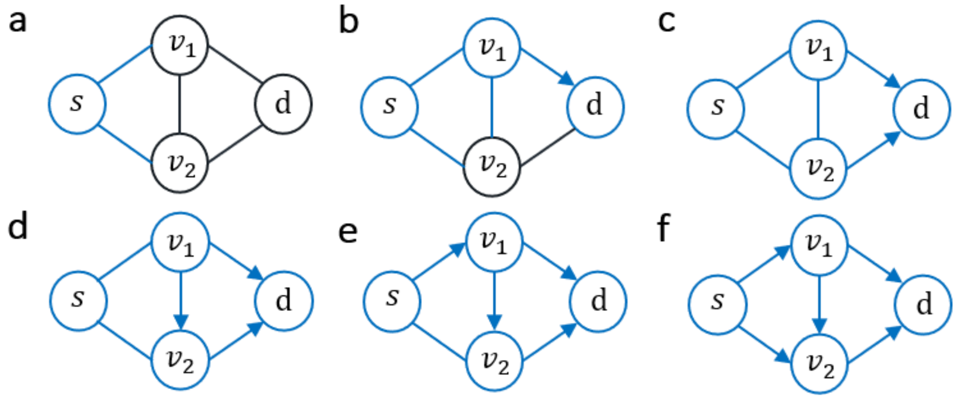
Data: state vector B_0

for $all\ nodes\ n\ in\ L_i$ **do**
    if $B_0[n] == 0$ **then**
        $B_0[n] = 1$;
        Call $F_1(s, d, n)$; //depth-first search
    end
    if $node\ n\ is\ connected\ to\ d$ **then**
        A new entry is added to the routing table of v(src = s, dst = d, next = n);
        A new entry is added to the routing table of n(src = d, dst = s, next = v);
    end
end
for $all\ nodes\ n\ in\ L_i$ **do**
    if $node\ n\ is\ not\ connected\ to\ d$ **then**
        $B_0[n] = 0$;
    end
end

Extended Data Fig. 2. Processing algorithm of neighbors in queue L_1 (or L_2) of node v.



Extended Data Fig. 3. Two cases of the search algorithm. a, Nodes $\boldsymbol{v_3}$ and $\boldsymbol{v_4}$ are added to the end of queue $\boldsymbol{L_1}$ of node $\boldsymbol{v_1}$, because their minimum hop numbers from the source node are larger than the hop number of $\boldsymbol{v_1}$. **b,** Node $\boldsymbol{v_2}$ is added to queue $\boldsymbol{L_2}$ of node $\boldsymbol{v_3}$, because the minimum hop number of node $\boldsymbol{v_3}$ is larger than that of $\boldsymbol{v_2}$.



Extended Data Fig. 4. An example of the search algorithm. According to the sequence of time, **a** to **f** correspond to the six steps of the search algorithm.