Finding Shortest MultiPaths with O(N^2) Message Complexity

Jerrolyn Brees and Sukhamay Kundu
Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803, USA

Abstract

This paper presents a distributed algorithm for discovering multiple shortest paths in an ad hoc network from a source s to a destination d. For each neighbor x of s which lies on a loop-free sd-path, we find one shortest sd-path via node x. The algorithm requires only O(N^2) many messages.
Keywords: shortest multipath, distributed algorithm

1 Introduction

In this paper, we introduce and define a distributed algorithm for finding multiple shortest paths in an ad hoc network from a source s to a known destination d. These shortest multipaths could be used by s to route its data to d. Shortest paths are ideal for routing since they typically use the fewest number of resources to send data. Discovering and updating single shortest paths distributively has been an active topic for many years, and [6] is one of the earlier papers which have discussed this.

The benefit of finding multiple sd-paths is that if one of the neighbors of s stops functioning due to failure or congestion s may still know a working sd-path through one of its other neighbors. Several algorithms have been proposed to find multipaths in distributed environments, as [1, 2, 3, 5, 7]. In particular, [5] showed that finding multipaths instead of single paths in a mobile ad hoc network can increase the network performance by reducing the frequency of path discovery.

2 Assumptions and Definitions

Given a graph G with bidirectional links, the source node s is the node that needs to communicate with a particular destination node d; interior nodes are all other nodes i such that $i \neq s$, d. The hop count h_i is used in this paper to determine a shortest path from node i to node d. If a more general distance measure is to be used, then the algorithm developed by Ramarao and Venkatesan in [6] should be used instead.

We assume that each node i has a unique id and knows of its neighbors N_i, where $N_i = \{j\colon\ (i,j) \in G\}$. Additionally, we assume that there is at least one sd-path in the network. Messages that are sent to a node's neighbors are eventually received in the order they were sent.

We use $S(k)$ to denote the set of nodes that are k hops from d, i.e., $S(k) = \{j\colon\ h_j{=}k\}$. Namely, $S(0) = \{d\}$ and $S(1) = N_d$. For an example, see Figure 2. K is the maximum hop distance from d, i.e., $K{=}\max\{k{:}S(k){\neq}\emptyset\}$.

T is the tree of shortest-paths from d that is built by the algorithm. The links from G used in T will depend on the order that messages are received at various nodes. The set $children_i$ is the set of children of i in T, i.e., $children_i{\subset}N_i$. The parent of node i in T is known as $parent_i$. $S_i(k)$ is the set of descendants of i in T which are k hops away from d. In particular, $S_d(k) = S(k)$; for $i{\in}S(k-1)$, $S_i(k) = children_i$; and for $i{\in}S(k')$ where $k'{<}k-1$, $S_i(k) = \bigcup\{S_j(k)\colon\ j \in children_i\}$. Note that except for $S_d(k)$, each $S_i(k)$ may depend on the structure of T.

$T(k)$ is the part of T from d up to level k. Thus, $S(k)$ equals the terminal nodes of $T(k)$ and $T(K){=}T$. We build T by successively extending $T(k)$ to $T(k{+}1)$, starting with $T(0){=}\{T\}$.

3 Algorithm Basics

The discovery of multiple shortest sd-paths is initiated by node s when it has information to transmit to d; s begins by sending a find path message ($fp()$) to each of its neighbors. The first time an intermediate node receives an fp-message, it forwards the message to each of its neighbors. If an sd-path exists, then d eventually receives an fp-message and proceeds with the second phase, ignoring any other fp-messages.

In the second phase of the algorithm, d constructs a tree (rooted at d) of shortest multipaths to all nodes reachable from d without going through s. Once d informs s that this is complete, s can query its neighbors

to find out which of them are on a path to d and what their hop-distance from d is. Once all the queries and responses have been received, s knows of the multiple shortest sd-paths that exist in G.

We introduce two similar algorithms for the second phase: Algorithm 1 and Algorithm 2. The algorithms are based on the same principle with Algorithm 2 being a simplification of Algorithm 1 that improves the message count for sparse networks.

4 Finding Shortest Paths from d with Algorithm 1

When d receives the *fp*-message forwarded from s, it begins creating a tree of shortest paths from itself to the nodes i that are taking part in this algorithm. A shortest path from i to d is defined as the path with the smallest number of hops needed for i to reach d without traversing s. As an effect of this definition, nodes that cannot reach d without going through s do not find their distance to d.

Since we assume that all links are of equal cost, we add all nodes found at the same hop count at the same time.

4.1 Message Types

Several different message types are used to find the shortest paths. The message $bp(k)$ is sent from nodes in $S(k-1)$ to nodes in $S(k)$ after all nodes in $S(k-1)$ have been discovered. The sender of the first *bp*-message that node i receives is set as $parent_i$.

The ch(Yes/No) message is used as an acknowledgment of the parent link. Node i sends ch_i(Yes) to $parent_i$ and sends ch_i(No) to any other node from which it receives a *bp*-message.

The node collection count message $ncc(k, |S_i(k)|)$ originates from node i at the $(k-1)$-th level and is sent to d through the parent links. It tells how many new nodes i found at the k-th level. A group of node collection messages $nc(k, j)$ follows the ncc message relating each specific node j found at the k-th level.

Similarly, the node dissemination count message $ndc(k, |S_d(k)|)$ originates from d and is sent through the child links to inform the other nodes that there are $|S_d(k)|$ nodes at the k-th level. A group of $|S_d(k)|$ node dissemination messages $nd(k, i)$ follows the ndc-message letting each child know exactly which nodes have joined the spanning tree at level k.

The $end()$ message halts the algorithm once there are no new nodes to add to the spanning tree.

- When d receives an *fp*-message informing it that an sd-path is wanted:
  - Let k=1
  - Send $bp_d(1)$ to each $i \in N_d$
  - Loop the following until there are no new descendants:
    - If k=1, listen for a ch_i(Yes/No) from each $i \in N_d$
      - For each ch_i(Yes) received, add i to $children_d$ and add i to $S_d(k)$
    - If k>1, listen for $ncc_i(k,c_i)$ from each neighbor i, $i \in children_d$
      - Receive c_i $nc_i(k, j)$ messages. Add j to $S_d(k)$
    - Send $ndc_d(k, |S_d(k)|)$ to each $i \in children_d$
    - For each $j \in S_d(k)$
      - Send $nd_d(k, j)$ to each i, $i \in children_d$
    - Increment k by 1
  - Send $end_d()$ to each $i \in children_d$
- When node i receives $bp_m(k)$:
  - If k_i = null
    - Let $parent_i = m$ and k_i = k
    - Send ch_i(Yes) to m
  - Otherwise,
    - Send ch_i(No) to m
- When node i receives $ndc_m(k, n)$:
  - If i=s
    - Send $ncc_s(k+1,0)$ to $parent_s$
  - If $k_i < k \wedge i \neq s$
    - Send $ndc_i(k, n)$ to each $c \in children_i$
    - Listen for $nd_m(k, j)$ messages and do the following until n such messages have been received
      - Send $nd_i(k, j)$ to each $c \in children_i$
    - Listen for $ncc_j(k+1, t)$ and do the following until $|children_i|$ such messages received.
      - For each ncc-message, receive t $nc_j(k+1, r)$ messages and add each r to $S_i(k+1)$
    - Send $ncc_i(k+1, |S_i(k+1)|)$ to $parent_i$
    - For each $j \in S_i(k+1)$, send $nc_i(k+1, j)$ to $parent_i$
  - If $k_i = k \wedge i \neq s$
    - Listen for $nd_m(k, j)$ messages and do the following until n such messages have been received
      - If k = $k_j \wedge j \in N_i$, remove j from N_i
    - Send $bp_i(k+1)$ to each $j \in N_i$
    - Listen for $ch_j(ans)$ and do the following until $|N_i|$ such messages received.
      - If ans = Yes, add j to $children_i$
    - Send $ncc(k_i+1, |children_i|)$ to $parent_i$
    - For each $j \in children_i$, send $nc(k_i+1, j)$ to $parent_i$
- When node i receives $end_m()$
  - Send $end_i()$ to each $j \in children_i$. End execution at i.

Figure 1: Processing of Messages at a Node in Algorithm One

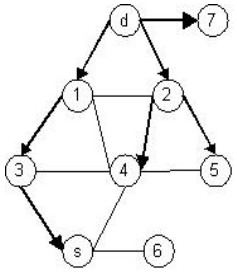

Figure 2: S(k) sets: S(0)={d}, S(1)={1,2,7}, S(2)={3,4,5}, S(3)={s}, S(4)=∅. Node 6 does not participate in this computation. Bold edges in this graph illustrate a possible T.

4.2 Processing Messages in Algorithm 1

Figure 1 illustrates the actions taken by nodes in G when a message is received. In this section, we discuss the use of *ncc* and *ndc*-messages in more detail.

4.2.1 Node Collection Messages: $ncc(k,m)$, $nc(k,j')$

These messages are used to inform d of the nodes at $S(k)$. The node collection count message $ncc(k,m)$ is initiated by a node j that belongs to $S(k-1)$ with $m = |S_j(k)| = |children_j|$ and is sent to $i = parent_j$; it is immediately followed by m many $nc(k,j')$-messages to i, one for each $j' \in S_j(k)$.

Once node i collects all $ncc(k,j)$-message and the associated $nc(k,j')$-messages from each $j \in children_i$, i knows $S_i(k) = \{j' \mid nc(k,j')$ received by $i\}$ and at that point i propagates $S_i(k)$ to $parent_i$ using a combination of one $ncc(k,|S_i(k)|)$-message and $|S_i(k)|$ many $nc(k,.)$-message. This continues until d knows $S_d(k)$.

4.2.2 Node Dissemination Messages: $ndc(k,m)$, $nd(k,j)$

These messages are used to notify each node in $S(k)$ about the whole set $S(k)$. The message $ndc(k,m)$, with $m = |S(k)|$ is initiated by d to each $i \in children_d$, and it moves down the tree from a node i to its children until it reaches the nodes in $S(k)$ and is not forwarded further down the tree. If some node i at level $k_i < k$ does not have a child then the arriving *nd*-message is responded to with an $ncc_i(k+1,0)$ to show that i has no new descendants.

The $ndc(k,m)$ message from d to its children is followed by m many $nd(k,j)$, one for each j in $S(k)$. These *nd*-messages follow the *ndc*-message down the tree. When a node $i \in S(k)$ (where $i \neq s$) receives all of the *nd*-messages, it starts the $k+1$-th step of the algorithm by sending a $bp_i(k+1)$ to each of its neighbors that were not mentioned in an *nd*-message.

4.2.3 Optional Improvement

We can improve this algorithm by having the nodes keep track of which of their children have no more descendants. When node i sends $ncc(k,0)$ to $parent_i$, it will not report any more new descendants for the duration of the algorithm; and thus, it does not need to participate in any more of the tree building activities. Table 1 shows a progression of messages that occur using Algorithm 1 with this improvement. We leave this improvement out of the algorithm so that the message count derived in the next section is not dependent on the specific structure of the tree that is built.

4.3 Message Count

We count the number of *nc*-messages up to the point when d knows that it is done finding a shortest path to all reachable nodes. After that d informs each node i in the tree T of shortest di-paths that it is done by sending the *end* message along T, which will require N-1 messages. where N is the number of nodes in T. We write i_k for a general node in $S(k)$, $k \geq 1$.

4.3.1 Specific Case Where k=1.

If there is no $S(2)$, then each node $i_1 \in S(1)$ will report $S_{i_1}(2) = \emptyset$, and d will know that it is done. This requires d to send the set $S(1)$ to each node $i_1 \in S(1)$ using N messages (one *ndc*-message for the size N-1 $= |S(1)|$ and N-1 *nd*-messages for the elements of $S(1)$). Each node $i_1 \in S(1)$ sends one *ncc*-message indicating $S_{i_1}(2) =$ empty. The total number of messages is $(N-1) * N + (N-1) = O(N^2)$.

4.3.2 Specific Case Where k=2.

Now assume that we have $|S(1)| = N_1 > 0$, $|S(2)| = N_2 > 0$, and $S(3) = \emptyset$. In this case, after d sends $N_1 * (N_1 + 1)$ *nd*-messages to inform each $i_1 \in S(1)$ of the set $S(1)$, each node i_1 in $S(1)$ will send 1 +

Table 1: Example Order of Messages for Algorithm 1 using the graph in Figure 2 and optional improvement from Section 4.2.3 with 79 total messages: 9 bp, 9 ch, 11 ndc, 27 nd, 11 ncc, 5 nc, 7 end

S.Time	Message	From	To	R.Time
1	*bp*(1)	*d*	1, 2, 7	2
2	*ch*(Yes)	1, 2, 7	*d*	3
3	*ndc*(1, 3)	*d*	1, 2, 7	4
	nd(1, 1)	*d*	1, 2, 7	5
	nd(1, 2)	*d*	1, 2, 7	6
	nd(1, 7)	*d*	1, 2, 7	7
7	*ncc*(2, 0)	7	*d*	8
	bp(2)	2	4, 5	9
	bp(2)	1	3, 4	10
9	*ch*(Yes)	4, 5	2	11
10	*ch*(Yes)	3	1	12
	ch(No)	4	1	13
11	*ncc*(2, 2)	2	*d*	14
	nc(2, 4)	2	*d*	14
	nc(2, 5)	2	*d*	14
13	*ncc*(2, 1)	1	*d*	15
	nc(2, 3)	1	*d*	15
15	*ndc*(2, 3)	*d*	1, 2	16
	nd(2, 3)	*d*	1, 2	16
	nd(2, 4)	*d*	1, 2	16
	nd(2, 5)	*d*	1, 2	16
16	*ndc*(2, 3)	1	3	17
	nd(2, 3)	1	3	17
	nd(2, 4)	1	3	17
	nd(2, 5)	1	3	17
	ndc(2, 3)	2	4, 5	18
	nd(2, 3)	2	4, 5	18
	nd(2, 4)	2	4, 5	18
	nd(2, 5)	2	4, 5	18
17	*bp*(3)	3	*s*	19
18	*bp*(3)	4	*s*	20
	ncc(3, 0)	5	2	20
19	*ch*(Yes)	*s*	3	21
20	*ch*(No)	*s*	4	22
21	*ncc*(3, 1)	3	1	23
	nc(3, *s*)	3	1	23
22	*ncc*(3, 0)	4	2	24
23	*ncc*(3, 1)	1	*d*	25
	nc(3, *s*)	1	*d*	25
24	*ncc*(3, 0)	2	*d*	26
26	*ndc*(3, 1)	*d*	1	27
	nd(3, *s*)	*d*	1	27
27	*ndc*(3, 1)	1	3	28
	nd(3, *s*)	1	3	28
28	*ndc*(3, 1)	3	*s*	29
	nd(3, *s*)	3	*s*	29
29	*ncc*(4, 0)	*s*	3	30
30	*ncc*(3, 0)	3	1	31
31	*ncc*(3, *s*)	1	*d*	32
32	*end*()	*d*	1, 2, 7	33
33	*end*()	1	3	34
	end()	2	4, 5	34
34	*end*()	3	*s*	35

$|children_{i_1}|$ many *nc*-messages to d to inform it of the set $S_{i_1}(2)$; the total number of these messages is N_1+N_2. Now d will compute the set $S(2)$ and inform each node $i_2 \in S(2)$ about the set $S(2)$ via the links in tree T of shortest-paths. This requires $(1+N_2)$ *nd*-messages along each of N_1+N_2 links of T, a total of $(1+N_2)(N_1+N_2)$. Finally, each node $i_2 \in S(2)$ will inform its parent in $S(1)$ that $S_{i_2}(3)=\emptyset$ and each node $i_1 \in S(1)$ will inform d that $S_{i_1}(3)=\emptyset$, and d will stop since $S(3)=\emptyset$; this part requires N_2+N_1 *ncc*-messages.

$$\begin{aligned} Total &= N_1(N_1+1)+(N_1+N_2)+(1+N_2)(N_1+N_2)+(N_1+N_2) \\ &= N_1(N_1+1)+3(N_1+N_2)+N_2(N_1+N_2) \\ &= N_1(N_1+1)+3(N-1)+(N-1-N_1)(N-1) \\ &= N_1(N_1-(N-2))+3(N-1)+(N-1)^2 \end{aligned}$$

Since $1 \le N_1 \le N-2$, this is maximum when N_1 is maximum, i.e., $N_1 = N-2$ and $N_2 = 1$. The maximum is $3(N-1)+(N-1)^2 = (N-1)(N+2)$.

4.3.3 General Proof of O(N^2).

Assume that the maximum number of *nc*-messages for the case where $S(k-1)$ is non-empty and *S(k)* is empty is $(N-1)(N+k-1)$. We show by induction that when *S(k)* is non-empty and $S(k+1)$ is empty the maximum number of message is $(N-1)(N+k)$.

For the next case of $S(k)$ non-empty and $S(k+1)$ empty, we have the following additional messages:

- $(k-1)*N_k$: *nc*-messages in communicating $S(k)$ to d via the sets $S_x(k)$ where x are nodes in $S(k-1)$, $S(k-2)$, ..., $S(1)$; previously N_k was zero
- $(1+N_k)(N_1+N_2+...+N_k)$: *nd*-messages in communicating $S(k)$ to each node in $S(k)$; previously d stopped since N_k was zero
- $N_1+N_2+...+N_k$: *ncc*-messages in communicating to d that $S(k+1)$ is empty; previously this part was not present

The total number of extra messages is $E_k = (k-1)*N_k+(1+N_k)(N-1)+(N-1)$.

Thus for a given N_k, the number of messages is

$$\begin{aligned} Total &= (N-N_k-1)(N-N_k+k-1)+E_k \\ &= (N-N_k-1)(N+k-1)+N_k^2+N_k(k-1)+2(N-1) \\ &= (N-1)(N+k-1)-N_k*(N-N_k)+2(N-1) \\ &= (N-1)(N+k+1)-N_k*(N-N_k) \end{aligned}$$

This is maximum when $N_k = 1$.

$$\begin{aligned} Total_{max} &= (N-1)(N+k+1)-(N-1) \\ &= (N-1)(N+k) \end{aligned}$$

Thus, the number of messages required is O(N^2).

5 Finding Shortest Paths from d with Algorithm 2

In this section, we modify Algorithm 1 to reduce the number of messages to compute the tree T of shortest paths. Though Algorithm 2 also takes O(N^2) messages, it tends to reduce the actual number.

In this algorithm, the $S(k)$ sets are not communicated up and down T; instead, when looking for nodes in $S(k+1)$, a node $i \in S(k)$ sends a *bp*-message to each node $j \in N_i$, where $j \in S(k)$ or $j \in S(k+1)$. By doing so, we reduce the number of different types of messages needed. Namely, the *nc*, *ndc*, and *nd* message types are no longer used; *ncc*-messages are still used to tell parents how many new nodes have been found. The details of this algorithm are illustrated in Figure 3.

An important note is that this approach will cause nodes to send *bp*-messages to other nodes in the same k-level. Additionally, since *bp*-messages originate from d, many more *bp*-messages will be used in Algorithm 2 than in Algorithm 1. Thus, for networks that have high connectivity within $S(k)$, Algorithm 1 may actually perform better than Algorithm 2.

- When d has received an *fp*-message:
  - Send $bp_d(1)$ to each $i \in N_d$
  - Listen for a ch_i(Yes/No) from each $i \in N_d$
    - For each ch_i(Yes), add i to $children_d$
  - Let $k_d = 2$
  - Do the following until no new descendants are found
    - Send bp_d(k) to each $i \in children_d$
    - Listen for a $ncc_i(k,c_i)$ message from each $i \in children_d$
    - Let $k = k+1$
  - Send $end_d()$ to each $i \in children_d$
- When node i receives $bp_m(k)$:
  - If k_i = null
    - Let $parent_i = m$ and $k_i = k$
    - Send ch_i(Yes) to m. Add m to $closed_i$.
  - Otherwise if $i = s$
    - Make a note of the new path to d through m
    - If $m \neq parent_s$, send ch_s(No) to m
    - Otherwise, send $ncc(k,0)$ to $parent_s$
  - Otherwise if $k_i = k$
    - Send ch_i(No) to m. Add m to $closed_i$.
  - Otherwise, if k_i = k-1
    - Send $bp_i(k)$ to all $j \in N_i \wedge j \notin closed_i$
    - Listen for $ch_j(ans)$ and do the following until $|N_i|$-$|closed_i|$ such messages received
      - If ans = Yes, add j to $children_i$
    - Send $ncc_i(k,|children_i|)$ to m
  - Otherwise, if $k_i < $ k-1
    - Send $bp_i(k)$ to all $j \in children_i$
    - Let $c_i = 0$
    - Listen for $ncc_j(k, c_j)$ and do the following until $|children_i|$ such messages received
      - Increment c_i by c_j
    - Send $ncc_i(k, c_i)$ to $parent_i$
- When node i receives $end_m()$
  - Send $end_i()$ to each $j \in children_i$

Figure 3: Processing of Messages at a Node in Algorithm 2

5.1 Optional Improvement

Similar to Algorithm 1, Algorithm 2 can also be improved by only sending *bp*-messages to those nodes that may have descendants left. When a node sends

ncc(.,0) to its parent, it no longer needs to participate in this tree building step because it has no more open descendants. Table 2 illustrates a possible message procession for Algorithm 2 using this improvement.

5.2 Message Count

The following section builds a proof showing the maximum number of messages required to successfully complete Algorithm 2. Additionally, we show that the worst case network graph scenario is similar to the one found for the previous algorithm, namely that each of $|N_k|$, $|N_{k-1}|$, ..., $|N_3| = 1$ and either ($|N_1|$=$N-k$ and N_2=1) or (N_1=1 and N_2=$N-k$).

Here are a few notes about the algorithm to explain where these message counts are derived from. Each node $x_k \in S(k)$ will send a bp-message only after it receives a $bp(k{+}1)$ from its parent corresponding to the determination of $S(k+1)$. By that time, each x_k has its unique parent in $S(k-1)$.

In order to avoid sending bp-messages to nodes in $S(k-1)$, each x_k keeps track of all nodes in $S(k-1)$ from which it has received a bp-message through using the $closed_x$ set.

Each node $x_k \in S(k)$ sends bp-message to adjacent nodes that are not in $closed_x$. This includes only nodes in $S(k)$ and nodes in $S(k+1)$. All nodes in $S(k)$ will reject x_k as its parent, and 0 or more nodes in $S(k{+}1)$ will accept x_k as their parent. When x_k has heard from all nodes to which it sent a bp-message, it informs $parent_{x_k}$ that it is ready to proceed with the next stage by sending an ncc-message.

Table 2: Example Order of Messages for Algorithm 2 using the graph in Figure 2 and optional improvement from Section 5.1 with 59 total messages: 26 bp, 15 ch, 11 ncc, 7 end

S.Time	Message	From	To	R.Time
1	bp(1)	d	1, 2, 7	2
2	ch(Yes)	1, 2, 7	d	3
3	bp(2)	d	1, 2, 7	4
4	bp(2)	2	1, 4, 5	5
	bp(2)	1	2, 3, 4	6
	ncc(2,0)	7	d	7
5	ch(Yes)	4, 5	2	7
	ch(No)	1	2	7
6	ch(Yes)	3	1	8
	ch(No)	2, 4	1	9
7	ncc(2,2)	2	d	8
9	ncc(2,1)	1	d	10
10	bp(3)	d	1, 2	11
11	bp(3)	1	3	12
	bp(3)	2	4, 5	13
12	bp(3)	3	4, s	14
13	bp(3)	4	3, 5, s	15
	bp(3)	4	2	16
14	ch(Yes)	s	3	17
	ch(No)	4	3	17
15	ch(No)	3, 5, s	4	18
16	ch(No)	4	5	18
17	ncc(3,1)	3	1	19
18	ncc(3,0)	4, 5	2	20
19	ncc(3,1)	1	d	21
20	ncc(3,0)	2	d	22
22	bp(4)	d	1	23
23	bp(4)	1	3	24
24	bp(4)	3	s	25
25	ncc(4,0)	s	3	26
26	ncc(4,0)	3	1	27
27	ncc(4,0)	1	d	28
28	end()	d	1, 2, 7	29
29	end()	1	3	30
	end()	2	4, 5	30
30	end()	3	s	31

5.2.1 Case of $S(1)$ non-empty and $S(2)$ empty.

Let $N_1 = |S(1)|$. The total number of messages are as follows:

- N_1, $bp(1)$ messages from d to nodes in $S(1)$
- N_1, ch(Yes) messages from nodes in $S(2)$ to d
- N_1, $bp(2)$ messages from d to nodes in $S(1)$
- $N_1(N_1-1)$, bp-messages from nodes in $S(1)$ to others in S(1)
- $N_1(N_1-1)$, ch(No) messages among the nodes in $S(1)$
- N_1, $ncc(0)$ messages to d from nodes in $S(1)$

The total number of messages sent when $S(2)$=$\emptyset$ is $2(N{-}1)(N{-}2){+}4(\mathrm{N}{-}1)$.

5.2.2 Case of $S(2)$ non-empty and $S(3)$ empty.

Let $N_2 = |S(2)|$. The total number of messages are as follows:

- N_1, $bp(2)$ messages from d to nodes in $S(1)$
- $N_1(N_1-1)+N_1N_2$, bp-messages from nodes in $S(1)$ to those in S(1)$\cup$S(2)
- $N_1(N_1-1)$, ch(No) messages among the nodes in $S(1)$
- N_1N_2, with N_2 ch(Yes) messages and the remainder being ch(No) messages from nodes in $S(2)$ to $S(1)$)
- N_1, ncc(.) messages to d
- N_1+N_2, $bp(3)$ messages from d to nodes in $S(2)$ via T
- $N_2(N_2-1)$, $bp(3)$ messages among nodes in $S(2)$
- $N_2(N_2-1)$, ch(No) messages among nodes in $S(2)$
- N_2, $ncc(0)$ messages to parents of nodes in $S(2)$
- N_1, $ncc(0)$ messages to d from nodes in $S(1)$

$$\begin{aligned} Total &= 2*(N_1N_1 + N_1N_2 + N_2N_2 + N_1) \\ &= 2*(N_1{+}N_2)(N_1+N_2) + 2N_1 - 2N_1N_2 \\ &= 2(N-1)(N-1) - 2N_1(N_2-1) \end{aligned}$$

which is maximum when $N_1 = N{-}2$ and $N_2 = 1$, and the maximum is $2(N{-}1)(N{-}1)$.

5.2.3 The case of S(3) non-empty and S(4) empty.

The computation of $S(3)$ will have following additional messages, where $N_3 = |S(3)|$.

- N_2N_3, $bp(3)$ messages from nodes in $S(2)$ to nodes in $S(3)$
- N_2N_3, with N_3 ch(Yes) messages from nodes in $S(3)$ to nodes in $S(2)$ and the remainder being ch(No) messages.

There will be following number of messages for determining that $S(4)$ is empty:

- $N_1+N_2+N_3$, $bp(4)$ messages from d via T
- $N_3(N_3-1)$, bp-messages among nodes in $S(3)$
- $N_3(N_3-1)$, ch(No) messages among nodes in $S(3)$
- N_3, $ncc(0)$ messages to parents of nodes in $S(3)$
- N_2, $ncc(0)$ messages to parents of nodes in $S(2)$
- N_1, $ncc(0)$ messages to d from nodes in $S(1)$

The total additional messages here is $2[N_1 + N_2 + N_3(N_3+N_2)]$.

The total number of messages are now:

$$Total = 2N_1N_1 + (N_2N_2{+}N_1N_2{+}N_1) + (N_3N_3{+}N_3N_2{+}N_1{+}N_2)$$

Here N_1 and N_3 appears in a symmetric fashion for the most part (N_1N_1 vs N_3N_3 and N_1N_2 vs N_3N_2) except for $2N_1$ (without a matching $2N_3$). This shows that to maximize for a given N_2, we need to make N_3=1. This gives the total number of messages:

$$\begin{aligned} Total &= 2[N_1N_1 + N_1N_2 + N_2N_2 + 2N_1 + 2N_2 + 1] \\ &= 2[(N_1+N_2)(N_1+N_2) - N_1N_2 + 2.(N_1+N_2) + 1] \end{aligned}$$

Since N_1+N_2=$N-2$, to maximize the number of messages, we make N_1=1 and N_2=N−3 or vice versa and the maximum is

$$\begin{aligned} Total &= 2((N-2)(N-2)+1(N-2)+2) \\ &= 2(N-1)(N-2)+2(2) \end{aligned}$$

In particular, both (N_1=N-3, N_2=1, N_3=1) and (N_1=1, N_2=N-3, N_3=1) give the maximum.

5.2.4 The case S(k) not empty and S(k+1) empty, k ≥ 2.

The maximum is obtained when $N_1 = N-k$ and all other $N_i = 1$ for $0 \leq i \leq k$. If we interchange, N_1 and N_2 we also get the same maximum.

$$\begin{aligned} Total &= 2(N-(k-1))(N-1)+2(2+3+4+...+(k-1)) \\ &= 2(N-(k-1))(N-1)+[k(k-1)-2] \\ &= O(N^2) \end{aligned}$$

6 Finding Multiple Shortest *sd*-Paths

After following Algorithm 1 or Algorithm 2, each viable neighbor of s now has a path to d. If Algorithm 1 is used, then after it receives the *end*-message, s must ask its neighbors for the lengths of their paths. Any of its neighbors that do not have a path length can only reach d through s. With this, s can pick which of its neighbors to use to forward its data to d.

If Algorithm 2 is used, then s has already heard from each of its neighbors that have a valid path to d. As such, no additional steps are needed. Node s can use the information gathered in Algorithm 2 to decide which nodes to use to forward its data to d.

As such, regardless of whether Algorithm 1 or Algorithm 2 is used, node s has discovered multiple shortest paths to d, if such paths exist.

7 Optional Bounding of the Graph

In [4], an algorithm is presented that can determine which neighbors of d have a viable path to s given an s and d node pair. A viable path from $i \in N_d$ to s is one that does not use d as an intermediate node. To do this, the algorithm traverses all of the links in the graph starting at s without going through d and takes linear time based on the number of links.

For a large, sparse network G' where d is likely to be a cut vertex, using this algorithm can help reduce the number of messages needed to find the shortest multipaths by limiting $S(1)$ to only the neighbors of d that have a viable path to s.

8 Conclusion

We have obtained an efficient O(N^2) distributed algorithm for finding multiple shortest paths for a node pair s and d. From an application point of view, introducing node disjointness in the shortest paths would be useful because the more disjoint the paths are the more likely that one *sd*-path will survive through several node failures. Currently, it is possible for a single node to be a part of each shortest path found, and thus if that single node fails, then the entire algorithm would have to be reexecuted to discover new shorter paths. We wish to address this issue in the future.

References

[1] C. Balasubramanian and J.J. Garcia-Luna-Aceves, "Shortest Multipath Routing Using Labeled Distances," *IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pp.314-323, 2004.

[2] M.S. Corson and A. Ephremides, "A Distributed Routing Algorithm for Mobile Wireless Networks," *Wireless Networks*, vol.1(1), pp.61-81, 1995.

[3] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks," *ACM SIGMOBILE Mobile Computing and Communication Review*, vol.5(4), pp.11-25, 2001.

[4] S. Kundu, "A Clean Distributed Algorithm for Building Multiple Paths for a Source-Destination Pair," Technical Report: TR-01-06, Computer Science Department, Louisiana State Univ., 2006.

[5] A. Nasipuri, R. Castaneda, and S.R. Das, "Performance of Multipath Routing for On-Demand Protocols in Mobie Ad Hoc Networks," *Mobile Networks and Applications*, vol.6(4), pp.339-349, 2001.

[6] K.V.S. Ramarao and S. Venkatesan, "On Finding and Updating Shortest Paths Distributively," *Journal of Algorithms*, vol.13(2), pp.235-257, 1992.

[7] W.T.Zaumen and J.J.Garcia-Luna-Aceves, "Loop-Free Multipath Routing Using Generalized Diffusing Computations," *INFOCOM '98*, vol.3, pp.1408-1417, 1998.