On Independently Finding Converging Paths in Internet [*]

Fei Dai and Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431
{fdai,jie}@cse.fau.edu

Abstract

Two shortest paths $P(u, v)$ and $P'(v, u)$ are called *converging paths* if path $P(u, v)$ is exactly the reversal of path $P'(v, u)$. Independently finding two converging paths from node u and node v, so that a message from u to v is bound to meet a message from v to u, is not trivial even if both nodes have the correct and complete network link state information. This paper presents three different converging path algorithms. The first one is a simple extension to Dijkstra's shortest path algorithm. The rest two make use of the routing information existing in each node. Both theoretical complexity and simulated performance results for these algorithms are given and discussed.

Key words: Converging paths, Dijkstra's algorithm, Internet routing protocol, shortest paths.

[*]This work was supported in part by NSF grant CCR 9900646 and grant ANI 0073736.

1 Introduction

Link-state routing protocols, such as OSPF and IS-IS, are widely used in the Internet nowadays. In link-state routing protocols, global network topology is first collected at each node. A shortest path tree (SPT) is then constructed by applying the Dijkstra's shortest path algorithm at each node. Link-state protocols normally require the flooding of new information to the entire (sub)network after changes in any link state (including link faults).

Narvaez et al.[5] proposed a fault-tolerant link-state routing protocol without flooding. The idea is to construct a shortest restoration path for each unidirectional faulty link uv (as shown in Figure 1 (a)). This is done when the shortest path to a destination contains uv. Faulty link information is distributed only to nodes in the restoration path and only one restoration path is constructed. Wu et al.[6] recently extended the Narvaez' protocol to efficiently handle a bidirectional link fault by making the restoration path bidirectional.

In [6], the restoration path is constructed simultaneously from both end nodes of a faulty link (u, v), node u will send a construction request along a shortest path $P(u, v)$, meanwhile node v will send a similar request along a shortest path $P'(v, u)$. If P and P' are converging paths (i.e., P' is exactly the reversal of P), messages from u and v will meet in midway and make a single restoration path (as shown in Figure 1 (b)). Otherwise, if there are multiple shortest paths between u and v, and P and P' are not converging paths, two fully or partially distinct restoration paths may be constructed, wasting network resources and complicating the restoration process (as shown in Figure 1 (c)).

As a single restoration path involves fewer nodes and is easier to handle than two fully or partially distinct restoration paths, it is preferable to send construction requests along converging paths. However, finding converging paths independently from nodes u and v is not trivial. Traditional shortest path algorithms [1, 4] cannot guarantee finding converging paths. Enumerating all possible shortest paths and selecting one among them according to certain criteria is very inefficient (as shown in Figure 2).

In this paper, we propose three converging path algorithms. The first one is an *asymmetric* extension to traditional shortest path algorithms, where one node, say, node u, finds the *forward* path and the other node v first finds the *backward* path from u to v and then reverses it. The other two are *symmetric* algorithms. One of them uses a divide-and-conquer strategy to determine the converging path through the selection of a series of middle points and the other is its optimization. Both algorithms make use of the distance information, which can be derived from the global link state information collected by each node in a link-state protocol.

The rest of the paper is organized as follows: Section 2 presents three converging path algo-

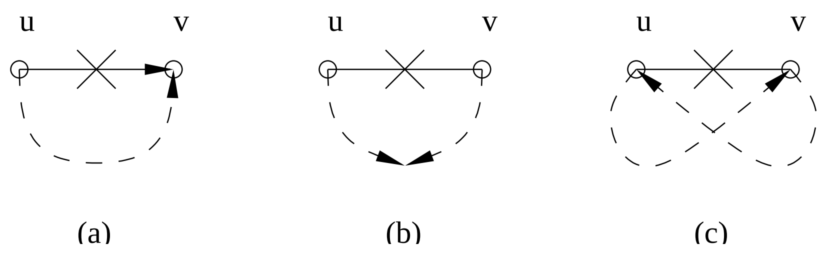

Figure 1: One restoration path is constructed for a faulty unidirectional link (a) and a faulty bidirectional link, if converging pathes are selected (b). Otherwise, two restoration paths are constructed (c).

rithms and compares their asymptotic complexities. Section 3 gives simulated performance results. Section 4 concludes this paper.

2 Proposed Algorithms

2.1 Preliminaries

A network can be viewed as an undirected graph $G = (V, E)$, where V is the vertex (node) set and E is the edge (link) set. $(u, v) \in E$ is a bidirectional link where u and v are two nodes in V. Each (u, v) is associated with a positive *cost*, $d_{u,v}$, representing the cost of travelling from u to v (or from v to u).

A *path* $P(u, v)$ in graph G is a sequence of distinct nodes $(v_1(= u), v_2, v_3, ..., v_k(= v))$ where $(v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k) \in E$. We say a path $P'(v, u)$ is the *reversal* of path $P(u, v)$ if it contains exactly the same set of nodes in reverse sequence $(v_k, v_{k-1}, ...v_1)$. The cost of a path is the sum of the costs of its links. A *shortest path* between two nodes u and v is a path with the minimum cost. We denote this minimum cost as the *distance* between u and v, or $D(u, v)$.

Definition 1: *Paths $P(u, v)$ and $P'(v, u)$ are converging paths if (1) $P(u, v)$ is a shortest path, and (2) $P'(v, u)$ is the reversal of $P(u, v)$.*

Nodes in a shortest path between nodes u and v can only be selected from a subset of V. The distance value $D(u, w)$, if available, can be used to determine whether node w belongs to this subset.

Definition 2: *The intermediate node set $V_s(u, v) \subseteq V$ is a set of nodes that appear in shortest paths between u and v.*

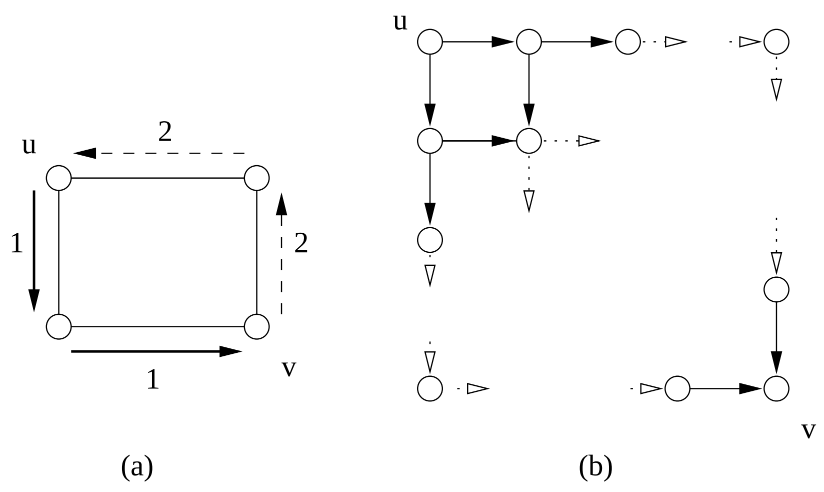

Figure 2: Problems in finding converging paths: (a) different paths (1 and 2) may be selected by nodes u and v, and (b) to enumerate all shortest paths may be a huge task.

According to the above definition, $u, v \in V_s(u,v)$, $V_s(u,v) = V_s(v,u)$ and $V_s(u,w) \subseteq V_s(u,v)$, $\forall w \in V_s(u,v)$. Apparently, $w \in V_s(u,v)$ if and only if $D(u,w) + D(w,v) = D(u,v)$ or $w' \in V_s(u,v) \wedge D(u,w) + d_{w,w'} = D(u,w')$.

A total order relation *less than* ($<$) on V is essential in determining an unique path among all possible shortest paths. '$<$' can be defined based on the lexigraphic order of node id's, or on a positional relation between nodes. Similarly, we say a node v is the *minimum node* in a subset $V' \subseteq V$, if $v < w, \forall w \neq v \in V'$.

In the following discussion, we assume that the network is connected, i.e., at least one path exists between any pair of nodes, thus the converging paths can always be found. We also assume, same as [5], that each node has the global link state information. In addition, each node knows its distance to every other node, as long as its routing table is established and well maintained.

2.2 An Asymmetric Algorithm

The first converging path algorithm is based on traditional shortest path algorithms. The idea is that the two involving nodes u and v first decide a *primary direction.* If both sides, for example, agree that the primary direction is from u to v, then node u will find an unique shortest path $P(u,v)$, meanwhile node v will first find the same unique path, also from u to v, and then reverse it to obtain the converging path $P'(v,u)$.

The unique path is determined in two phases. In phase one, an arbitrary traditional shortest path algorithm is used to compute the distance from node u to every other node $w \in V$, thus any possible shortest path can be constructed from node v via the selection of a series of preceding nodes. In phase two, a shortest path is determined according to a predefined total order '$<$'. The two-phase division makes the algorithm more flexible, and the overhead of the second phase is negligible compared with the total execution time.

```
UNIQUE-PATH $(u, v)$
1: $P \leftarrow [v],\ w \leftarrow v$
2: while $w \neq u$ do
3:     $w' \leftarrow \min\{x | (x, w) \in E \wedge D(u, x) + d_{x,w} = D(u, w)\}$
4:     $P \leftarrow [w'] \| P,\ w \leftarrow w'$
5: end while
6: return $P$

ACP $(u, v)$ {Asymmetric Converging Path}
1: if $u < v$ then
2:     compute $D(u, w), \forall w \in V$ {using any shortest path algorithm}
3:     return UNIQUE-PATH $(u, v)$
4: else
5:     compute $D(v, w), \forall w \in V$
6:     return reverse (UNIQUE-PATH$(v, u)$)
7: end if
```

The critical part of the asymmetric algorithm is UNIQUE-PATH, which guarantees that the same shortest path will be found independently by both node u and node v.

Lemma 1: *If $D(u, w), \forall w \in V_s(u, v)$ are known,* UNIQUE-PATH*(u, v) will terminate and return an unique shortest path.*

Assuming ACP$_u$ denotes the execution of procedure ACP on node u and ACP$_v$ denotes the execution of the same procedure on node v, we have the following theorem:

Theorem 1: *The asymmetric converging path (*ACP*) algorithm will terminate and* $\{$ACP$_u(u, v)$, ACP$_v(v, u)\}$ *are converging paths.*

Proof: First for termination, the procedure of computing one-to-all distance values is known to terminate; from Lemma 1, UNIQUE-PATH will also terminate. So will ACP. Then for correctness, without loss of generality, we can assume that $u < v$, which means ACP$_u(u, v) =$ UNIQUE-PATH$_u(u, v)$ and ACP$_v(v, u) =$ **reverse**(UNIQUE-PATH$_v(u, v)$). From Lemma 1, UNIQUE-PATH$_u(u, v) =$ UNIQUE-PATH$_v(u, v)$, and hence, ACP$_u(u, v)$ and ACP$_v(v, u)$ are converging paths.

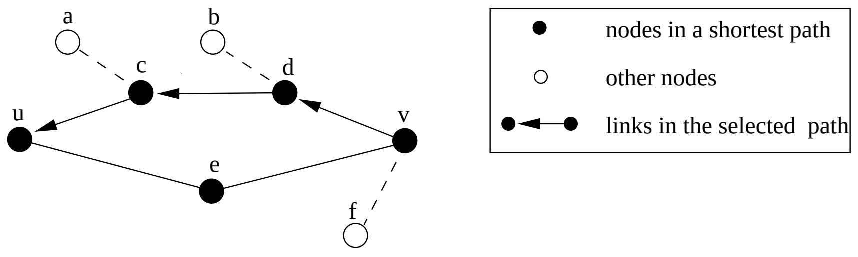

Figure 3: UNIQUE-PATH(u, v) selects a shortest path (u, c, d, v). Node v selects node d because $d < e$ and $f \notin V_s(u, v)$. Node d in turn selects c, which is the only qualified node.

□

Although any shortest path algorithm can be used, our current implementation is based on the Dijkstra's algorithm [3, 4] for the sake of efficiency. The Dijkstra's algorithm can find the shortest path from a source node to every other node. Initially, the distance value of source is set to 0 and the distance value of every other nodes is unknown. Among them, only the distance value of the source is *determined*. At each round, the adjacent nodes to the last determined node update their distance values, and the node with the lowest undetermined distance value is selected as the next determined node. Finally, all the distance values are determined in $n-1$ rounds. In our simulation, the above process stops after the value of $D(u, v)$ is determined. Because the values of $D(u, w)$, $\forall w \in V_s(u, v)$, which are smaller than $D(u, v)$, are determined before $D(u, v)$, this partial distance information is sufficient for determining an unique shortest path.

2.3 A Recursive Algorithm

The drawback of the asymmetric algorithm is that it does not make full use of the distance information available in a real protocol, where each node often computes and keeps its distance to every other node in order to select the shortest route. In ACP, if u is the 'smaller' node, it can use this information and skip line 2. However, when u is 'larger', it has to compute the distances from node v to every other node (line 5).

The following recursive symmetric algorithm assures that both nodes will use the distance information available and avoid unnecessary computation. The problem is solved in a divide-and-conquer manner:

1. Each node traverses $V_s(u, v)$ to find the unique node $w = \min\{V_s(u, v)\}$.
2. Each node recursively finds two sub-paths $P_1(u, w)$ and $P_2(w, v)$, and merges them to obtain the converging path.

UNIQUE-NODE uses a queue Q to traverse $V_s(u,v)$, directed by the distance values from other nodes to u. For each visited node, its distance value to v can be easily computed and reserved for further traversing. After the traversal, the 'minimum' node is determined and recursively divides the problem of finding one path into finding two sub-paths. To distinguish a visited node from an unvisited one, a marking process (M) is used to mark every node in $V_s(u,v)$. Initially, $M(u) = M(v) = T$ and $M(w) = F$ for any other node w.

```
UNIQUE-NODE $(u,v)$ {find $\min(V_s(u,v))$ and compute distance to $v$}
 1: $Q \leftarrow \{v\}$, $min \leftarrow \infty$
 2: **while** $Q \neq \emptyset$ **do**
 3:    extract $w$ from $Q$
 4:    $D(v,w) \leftarrow D(u,v) - D(u,w)$
 5:    **for all** $w' \in \{x | (w,x) \in E \wedge D(u,x) + d_{x,w} = D(u,w) \wedge M(x) = F\}$ **do**
 6:       $M(w') \leftarrow T$, add $w'$ to $Q$
 7:       **if** $w' < min$ **then**
 8:          $min \leftarrow w'$
 9:       **end if**
10:    **end for**
11: **end while**
12: **return** $min$
```

```
RSCP $(u,v)$ {Recursive Symmetric Converging Path}
 1: **if** $u = v$ **then**
 2:    **return** $[u]$
 3: **else if** $(u,v) \in E \wedge d_{u,v} = D(u,v)$ **then**
 4:    **return** $[u,v]$
 5: **else**
 6:    $w \leftarrow$ UNIQUE-NODE$(u,v)$
 7:    **return** RSCP$(u,w)$ $\|$ **reverse**(RSCP$(v,w)$) †
 8: **end if**
```

† a redundant w should be removed.

Lemma 2: *If $D(u,w)$ and $D(v,w)$, $\forall w \in V_s(u,v)$, are known to node u and node v, respectively, then (1) execution of* UNIQUE-NODE*(u,v) on u will terminate and $D(v,w)$, $\forall w \in V_s(u,v)$, are know to u after termination, (2) so will the execution of* UNIQUE-NODE*(v,u) on v, and (3)* UNIQUE-NODE$_u$*(u,v) =* UNIQUE-NODE$_v$*(v,u).*

Theorem 2: *The recursive symmetric converging path (*RSCP*) algorithm will terminate and*

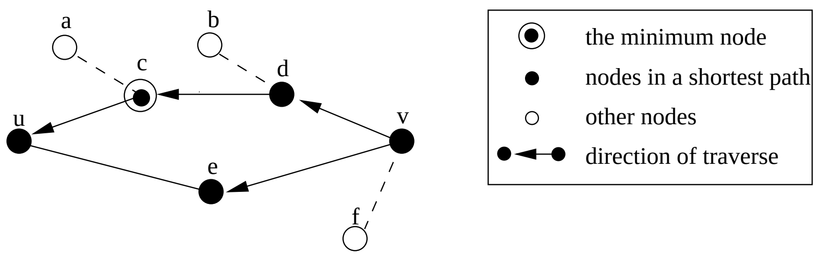

Figure 4: UNIQUE-NODE(u, v) selects a intermediate node c. All the nodes in $V_s(u, v)$ are traversed, and c has the minimum node id.

$\{$RSCP$_u(u, v)$, RSCP$_v(v, u)\}$ *are converging paths.*

Proof: By using induction on the maximum hop number ($H_{max}(u, v)$) of all shortest paths between u and v. When $H_{max}(u, v) \leq 1$, RSCP returns immediately with the correct result (lines 1–4). Assume that the theorem holds for $H_{max}(u, v) = k$, we prove that it also holds for $H_{max}(u, v) = k + 1$.

From Lemma 2, UNIQUE-NODE will terminate and return the same node for RSCP$_u$ and RSCP$_v$ (line 6). Notice that both $H_{max}(u, w)$ and $H_{max}(v, w)$ are smaller than k, from the assumption that the computation of sub-paths will terminate and return two pairs of converging paths ($P_1(u, w)$, $P_2'(w, u)$) and ($P_2(w, v)$, $P_1'(v, w)$). Therefore, RSCP$_u$ and RSCP$_v$ will both terminate and return two converging paths $P_1(u, w) \| P_2(w, v)$ and $P_1'(v, w) \| P_2'(w, u)$ (line 7). □

Two 'less than' relations can be employed for this algorithm:

1. Comparison of node id: $w' < w$ **iff** $id(w') < id(w)$.
2. Comparison of node position: $w' < w$ **iff** $|D(u, w') - \frac{D(u,v)}{2}| < |D(u, w) - \frac{D(u,v)}{2}|$.

The second criterion may divide the path more evenly, thus reducing the recursion depth.

2.4 A Faster Algorithm

Although the recursive symmetric algorithm performs better than the asymmetric algorithm by making use of the distance information, its performance can be further improved. Notice that the most time consuming part of the recursive algorithm is the procedure of traversing $V_s(u, v)$ to find the minimum node, and this procedure is reiterated until every sub-path is split into one-hop pieces. Unlike RSCP, the following fast symmetric algorithm invokes UNIQUE-NODE only once to determine a middle point w and collects the necessary distance information. Then it uses the asymmetric algorithm to construct two sub-paths $P_1(u, w)$ and $P_2(w, v)$ and merges them to a converging path.

```
FSCP$(u, v)$ {Fast Symmetric Converging Path}
1: if $u = v$ then
2:     return [u]
3: else if $(u, v) \in E \wedge d_{u,v} = D(u, v)$ then
4:     return $[u, v]$
5: else
6:     $w :=$ UNIQUE-NODE $(u, v)$
7:     return UNIQUE-PATH$(u, w)$ $\|$ reverse(UNIQUE-PATH$(v, w)$) ‡
8: end if
```

‡ a redundant w should be removed.

Theorem 3: *The fast symmetric converging path (* FSCP*) algorithm will terminate, and* $\{\text{FSCP}_u(u, v), \text{FSCP}_v(v, u)\}$ *are converging paths.*

Proof: The theory holds when $u = v$ or a one-hop shortest path exists between u and v (lines 1–4). Otherwise, from Lemma 2, we know that UNIQUE-NODE will terminate and return the same w for both FSCP_u and FSCP_v, and after it returns, both $D(u, x)$ and $D(v, x)$, $\forall x \in V_s(u, v)$, are computed (line 6). From Lemma 1, UNIQUE-PATH will terminate and return the same path $P_1(u, w)$ and $P_2(v, w)$. Therefor, both FSCP_u and FSCP_v will terminate and return two converging paths $P_1(u, w)\|\textbf{reverse}(P_2(v, w))$ and $P_2(v, w)\|\textbf{reverse}(P_1(u, w))$ (line 7). □

2.5 Asymptotic Complexity

In the following discussion, we shall use $n = |V|$ to represent the total number of nodes, $n' = |V_s(u, v)|$ the number of nodes between end nodes u and v, $h = H_{max}(u, v)$ the maximum hop number of all shortest paths between u and v, and k the maximum node degree, that is, the maximum number of links adjacent to a single node.

The asymptotic complexity of the Dijkstra's shortest path algorithm is $\Theta(kn \log n)$. Notice that although only the distance values of the nodes in $V_s(u, v)$ are needed, they are determined only after all the nodes with smaller distance values are determined. The asymptotic complexity of UNIQUE-PATH is $\Theta(kh)$ and that of UNIQUE-NODE is $\Theta(kn')$. The recursion depth of RSCP is $\Theta(\log h)$ and the overall complexity is $\Theta(kn' \log h)$ on average. However, in the worst case, the recursion depth is $h - 1$ and the overall complexity is $\Theta(kn'h)$. Now we consider the complexities of these algorithms in two cases:

1. When $D(u, w), \forall w \in V_s(u, v)$, is available.

   - The complexity of ACP is $\Theta(kh)$ for the forward side and $\Theta(kn \log n)$ for the other side, the overall complexity is $\Theta(kn \log n)$.
   - The complexity of RSCP is $\Theta(kn' \log h)$ on average and $\Theta(kn'h)$ in the worst case.
   - The complexity of FSCP is $\Theta(kn')$.

   Because $n' \ll n$ holds in most cases, the performance of both symmetric algorithms shall be much better than the asymmetric algorithm.

2. When no distance information is available. The complexity of ACP is $\Theta(kn \log n)$, the complexity of RSCP is $\Theta(kn \log n + kn'h)$, and the complexity of FSCP is $\Theta(kn \log n + kn') = \Theta(kn \log n)$. Because $n' \ll n$ holds in most cases, the difference shall be very small.

3 Performance Simulation

The three converging path algorithms are simulated to compare their performance. Among them, the recursive symmetric algorithm is implemented with two 'less than' functions (comparing id and position). Two different situations are considered in the simulation: (1) when the distance information is available, and (2) when the distance information is not available.

3.1 Network Generator

Networks used in the simulation are generated by a modified version of the random network generator GRIDGEN [2]. This software can be used to construct a connected graph with n nodes and m links. At first, n nodes are placed into an $x \times y$ grid ($xy = n$). The grid links form a "skeleton" guaranteeing connectivity. Then additional links are added between randomly selected pairs of nodes, until the total number of links reaches m.

All skeleton links are associated with the same cost D_{max}, which is always larger than those random distances associated with the additional links. If $m \gg 2n$, a shortest path will not contain many skeleton links. However, when m is very close to $2n$, those skeleton links will produce many alternative shortest paths.

Two groups of networks are generated: (1) networks with different number of nodes (n) and fixed average node degree (20 or link/node ratio $\frac{m}{n} = 10$), and (2) networks with fix number of nodes ($n = 900$) and different number of links (m). For each network, 100 pairs of nodes are randomly selected for computing the converging paths.

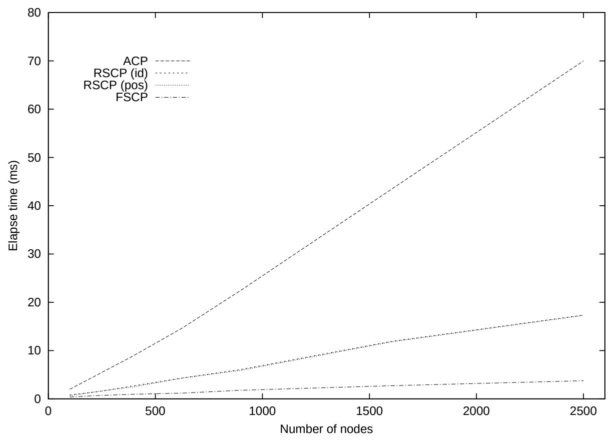

Figure 5: Execution time when distance information is available and $m = 10n$.

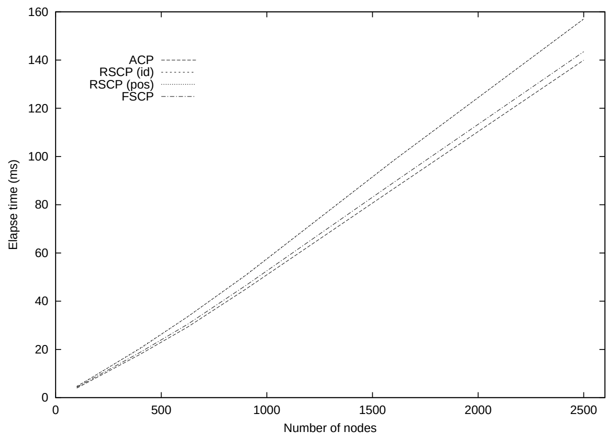

Figure 6: Execution time when there is no distance information, and $m = 10n$.

3.2 Data Interpretation

For the networks with average node degree 20, all symmetric algorithms are much faster than Acp when distance information is available (Figure 5). Among the symmetric algorithms, Fscp is much faster than Rscp. However, the difference between two recursive algorithms is not so obvious. This is because the maximum hop number of the converging paths is relatively small ($h = 10$ on average). When distance information is not available (Figure 6), the asymmetric algorithm is the fastest one, but the difference is relatively small (10–20%).

For networks with different node degrees (Figure 7), data shows that the asymmetric algorithm slows down quickly as the node degree increases. However, the symmetric algorithms are relatively insensitive to the node degree . This is because when the node degree k increases, the hop number

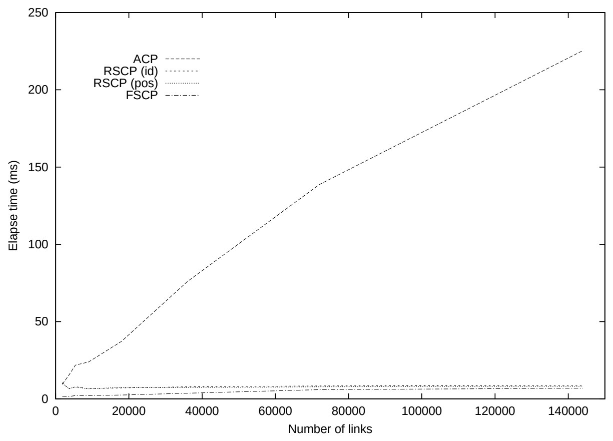

Figure 7: Execution time when distance information is available and $n = 900$, m =1800-144000.

of the converging path h also decreases, thus the overall execution time, which is $\Theta(kn'h)$, still maintains on the same level. Therefor, relatively the symmetric algorithms perform even better when the average node degree is larger.

4 Conclusion

We have proposed three algorithms to solve the converging path problem. All of them have $\Theta(kn \log n)$ complexity if the converging paths are constructed from scratch. However, when the one-to-all distance information is available, as it is in many link-state routing protocols, the two symmetric algorithms performs much better. This conclusion is also supported by the simulation results. Future research works include: (1) using more efficient shortest path algorithm in the asymmetric algorithm, (2) more simulations, especially with larger hop number of converging path, and (3) the effect of outdated distance information caused by network topology changes.

References

[1] R. Bellman. On a routing problem. *Appl. Math.*, 16:87–90, 1958.

[2] D. P. Bertsekas. *Linear Network Optimization: Algorithms and Codes.* The MIT Press, Cambridge, Massachusetts, 1991.

[3] E. V. Denardo and B. L. Fox. Sortest-route methods: 1. reaching, pruning, and buckets. *Operations Res.*, 27:161–186, 1979.

[4] E. W. Dijkstra. A notes on two problems in connection with graphs. *Numer. Math.*, 1:269–271, 1959.

[5] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng. *Fault-tolerant routing in the Internet without flooding*, pages 193–206. Aversky (ed.), Kluwer Academic Publishers, 2000.

[6] J. Wu, X. Lin, J. Cao, and W. Jia. An extended fault-tolerant link-state routing protocol in the internet. *Proc. of 8th International Conf. on Parallel and Distributed Systems (ICPADS)*, June 2001.