Bilkent University

CS 442

Distributed Systems

Cooperative Routing Protocol

Halil Atmaca
21302459

Burak Gök
21302025

Enes Varol
21604086

Instructor:
Buğra Gedik
Assoc. Prof. of Computer Eng.

Submitted on May 27, 2019
Revised on Aug 22, 2019

1 Introduction

Communication networks are generally not fully connected, which necessitates routing of messages sent among nodes. Routing requires each node in a network to select a next-hop node for every destination. While the selection is trivial for neighbors, it involves the computation of shortest distance for non-neighboring nodes.

If the topology is known prior to the computation, which is the case in link-state routing protocols, a single-source shortest path algorithm, such as the Dijkstra's algorithm, can be used to form the routing table. In such protocols, each node sends its link-state advertisement, which is the collection of its neighbors along with their distances, to every node in the network via flooding. Once a node has the complete set of link-state advertisements, it forms the network graph and its routing table. Since this is a cumulative process, the obtained results may not completely reflect the current state of the network; or, they may contain contradictory information. Usually sequence-numbered messages are used to overcome outdated or contradictory information. Link-state routing protocols are very restrictive for highly dynamic networks, due to their computational complexity and the number of messages they generate in result of flooding.

There is another family of routing protocols, which does not require obtaining the complete network image in order to compute shortest distances. Instead, in such protocols, each node stores the distance to every other node along with the next-hop node associated with it. The array of distances to other nodes is known as the distance vector. When a distance vector is manipulated, it is sent to all neighbors. When a node receives a distance vector from a neighbor, it updates its own vector to reflect changes in network topology and link weights. Since the distance vector of a node functions like its routing table along with the next-hop nodes, the comparison of distance vectors is the only computation done by nodes. For such comparisons, a distributed variant of the Bellman-Ford algorithm is used. The algorithm is slower than the Dijkstra's algorithm, but the former can be decentralized unlike the latter. Distance-vector routing protocols are simpler and has less message overhead compared to link-state routing protocols. However, since they do not try to form the complete network graph, they cannot test the consistency of their distance vectors, which can lead to routing loops and the widely known count-to-infinity problem.

In this project, we try to devise a distance-vector routing protocol that avoids routing loops and the count-to-infinity problem and also has the optimal number of message exchanges. Our protocol is purely theoretical and requires extra work to be implemented in real communication networks. The protocol has the following limitations or relaxations.

1. Network links must have non-negative weights.
2. Reliable message exchange protocol is required such as TCP.
3. Neighboring nodes may have asymmetric weights for the same link.
4. The triangle inequality does not have to be satisfied among network links.
5. Any distance metric can be used as long as from two paths, the more preferable one has the smaller distance.

2 Protocol

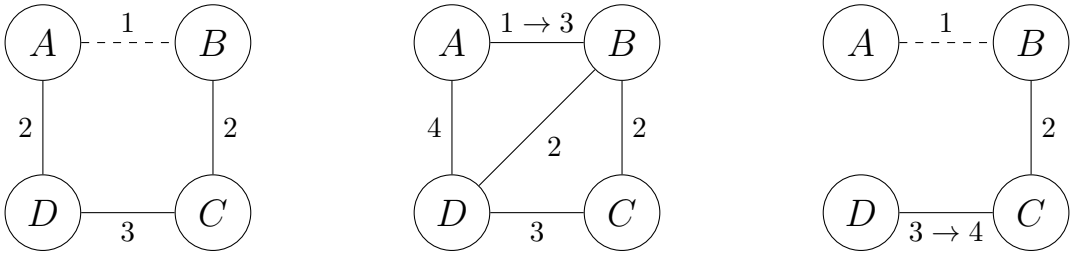
The protocol is based on the Bellman-Ford algorithm as usual. It differs from similar protocols by introducing two properties for nodes: *smartness* and *helpfulness*. A node is smart if it always preserves the consistency of its internal state and does not lose any information that might be helpful while forming the routing table. A node is helpful if it does not spread any information that may mislead its neighbors and informs them if their routing tables are suboptimal.

The protocol utilizes a reachability protocol, to obtain information about neighboring nodes such as link weight changes and mobility events. This is done by sending echo requests and collecting echo replies at each node. The time measured between a request-reply pair is used to assign a weight to the respective link. If no reply is received after the request within the timeout period, the link weight is set to infinity. In that case, the link is said to be congested and for a while, echo requests are continued to send. If the respective neighbor does not respond for an unusual amount of time, it is forgotten. Receiving an echo request from an unknown or non-neighboring node means that a new link is discovered.

When a node joins the network for the first time or a disconnected node becomes connected again, it notifies its neighbors by sending echo requests during a usual echo process. When a node A receives an echo request from an unknown node B, it does not add node B to the distance vector because the distance to node B is not measured yet. After node A initiates an echo process, the distance to node B can be measured after receiving the echo reply. However, node B may send a distance vector to node A before sending an echo reply since it has node A in its distance vector. It is impossible to compare a distance vector without knowing the distance to its sender. That's why the protocol enforces node B to put its distance to node A in the distance vector it sends to node A. This is called *the reported distance* and is only used in such a situation. For all cases, the distance obtained by echoing is preferred.

Since the triangle inequality does not have to be satisfied among links, a neighboring node may have a next-hop node other than itself—or NIL, which is denoted as $\varnothing$. This causes such a node to have a distance that is different from its link weight. Thus, the node distance and the link weight should be stored separately to satisfy the second requirement of smartness property because there may be an update, in the future, which causes switching between two quantities. We will use the following symbols in the rest of the report for the sake of simplicity and formalism.

L	Links (All neighbors)
$W : L \to \mathbb{R}^+$	Link weight
$N : \{x \in L \mid W(x) \neq \infty\}$	Responsive neighbors
$T : L \to \mathbb{R}^+$	Time of the last echo request/reply received
$T(\text{REQ})$	Time of the last echo request sent
A	All nodes in the network
K	Known nodes
$P : K \to N \cup \{\varnothing\}$	Predecessor (Next-hop node)
$D : K \to \mathbb{R}^+$	Node distance
$U : R \to \mathbb{R}^+$, $R \subseteq A$	Distance vector received
M	Myself
S	Sender



(a) Scenario #1 (b) Scenario #2 (c) Scenario #3

Fig. 1. Topology change scenarios

2.1 Smartness

As the first requirement of smartness, nodes should preserve the consistency of their internal state, i.e. distance vector, after an update. For example, in Fig. 1a, assume that the network is stabilized. After the link between node A and node B is broken and node A realizes that node B is unreachable, it sets $D(B)$ to infinity. However, node C is reached via node B, so $D(C)$ should also be set to infinity. In general, when the distance to a neighbor is updated, the distances to all descendants of that neighbor should also be updated. Such an update is denoted as $D(x) \rightsquigarrow y$ and defined as the following.

definition $D(x) \rightsquigarrow y$: ▷ used if $x \in N$
1 **for each** $k \in K$ such that $P(k) = x$ **do**
2 $P(k) = P(x) \neq \varnothing \ ? \ P(x) : x$
3 $D(k) \leftarrow D(k) + (y - D(x))$
4 $D(x) = y$

As stated before, for the second requirement of smartness, nodes should keep information that might be useful while forming the routing table. Assume that the network depicted in Fig. 1b is stabilized. Node A reaches node D via node B. After the weight of the link between node A and node B becomes 3, node A should switch to the link between node D and itself to reach node D. In general, when the distance to a neighbor that is reached indirectly changes, the link weights should be checked before the update. Such an update is denoted as $D(x) \leftarrow y$ and defined as the following. Note that the update function that checks descendants, which is defined above, calls this function.

definition $D(x) \leftarrow y$: ▷ used if $P(x) \neq \varnothing$
1 **if** $x \in N$ **and** $W(x) < y$ **then**
2 $P(x) = \varnothing$
3 $D(x) = W(x)$
4 **else**
5 $D(x) = y$

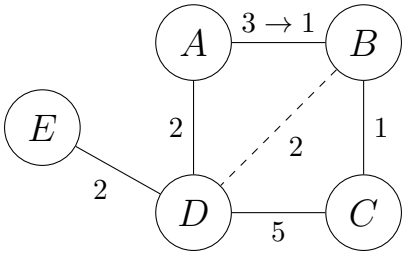


Fig. 2. A topology change scenario that results in a suboptimal state

Pitfalls

When a neighbor is decided to reach indirectly as a result of an update, its descendants might be processed twice: once due to the first requirement of smartness and once due to their probable inclusion in the distance vector received. If the sender node has an optimal state, the distances that are processed for the same node will be equal and the order of processing does not matter. Otherwise, the smaller one should be selected and in some cases, the update may result in a suboptimal state regardless of the order of processing. In Fig. 2, assume that the link between node B and node D is not established yet and the network is stabilized. In this topology, node C reaches node E via node D. If the aforementioned link is established and the weight of the link between node A and node B becomes 1 simultaneously, node C will receive an update from node B, which claims that $D(D) = 2$ and $D(E) = 5$. Two possible scenarios for the order of processing are as follows.

1. If the former is processed first, node D will be decided to reach via node B and consequently the distance of node E will be set to 5. When the latter is processed, the distance of node E will be updated again and set to 6 since it is now reached via node B.

2. If the latter is processed first, node E will be decided to reach via node B and its distance will be set to 6. When the former is processed, the distance of node E will not be updated since it is not reached via node D anymore.

It seems easier to devise a way to avoid this pitfall for the first scenario since the distance of node E is set to the desired value at least temporarily. The first scenario can be ensured by processing the neighbors which are directly reached first. The solution may be as simple as checking if the predecessor of a node is updated before trying to update its distance while processing a distance vector received and this is the adopted solution. A more elegant and symmetric solution, which does not depend on the order of processing, is to construct a set of candidate paths for each node in order to compute the shortest paths.

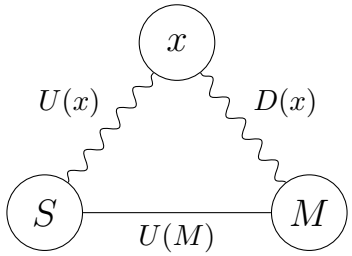


Fig. 3. Visualization of an update

2.2 Helpfulness

As the first requirement of helpfulness, nodes should not spread information that can be misleading. In Fig. 1c, if the link between node A and node B is broken and the weight of the link between node C and node D becomes 4 simultaneously, node B will receive an update from node C, which claims that $D(A) = 3$, after setting node A as unreachable. Since node B has no way of knowing that node C actually reaches node A via itself, it will apply the update. This will eventually lead to the count-to-infinity problem. If node A did not become disconnected after the link is broken; for example, if there was a link between node A and node D as in Fig. 1a, the network would still converge. However, the convergence time and the message overhead would increase. To overcome these problems, when a node sends its distance vector to another node, only the distances of the nodes that are not reached via the destination node are sent. This is expressed more formally as "send $\{(k, D(k)) \mid k \in K,\ P(k) \neq n\}$ to each $n \in N$."

As the second requirement of helpfulness, nodes should inform their neighbors if their routing tables are suboptimal. Since one of the main objectives of the protocol is to decrease the number of messages exchanged, distance vectors are only broadcast if there is an update. Furthermore, when they are broadcast, the message-originator neighbor that causes the update is skipped because it cannot learn new information from an update it originated. Because of these two reasons, sometimes nodes may stuck in a suboptimal, or possibly faulty state.

In Fig. 1a, assume that node B realizes that the link between node A and itself is broken before node A. Node B will set $D(A)$ to infinity and send its distance vector to node C. Since node C reaches node A via node B, it will also set $D(A)$ to infinity and send its distance vector to node D. However, node D will not update its distance vector since it reaches node A via a direct link. After node A realizes that node B is unreachable, it will set $D(B)$ to infinity and sends its distance vector to node B. Since node D reaches node B via node A, it will also set $D(B)$ to infinity and send its distance vector to node C. Since node C reaches node B via a direct link, it will not update $D(B)$. However, it will update $D(A)$ since it realizes that node A is reachable via node D. Note that node D still believes that node B is unreachable, which cannot be corrected since the network is now stabilized.

A node's state can be corrected if its neighbors detect possible improvements in its distance vector and informs it. Fig. 3 visualizes an update, where S and M are the sender and receiver nodes respectively. If $U(M) + D(x) < U(x)$ and $P(x) \neq S$, that means that node S can reach node x via node M, which is shorter than the path it currently uses.

The complete protocol is as follows. The following operators are defined to manipulate sets.

$$x \mathrel{+}= y \equiv x = x \cup \mathbb{S}(y)$$
$$x \mathrel{-}= y \equiv x = x \setminus \mathbb{S}(y)$$
$$\text{where} \quad \mathbb{S}(x) = \begin{cases} x & \text{if } x \text{ is a set,} \\ \{x\} & \text{otherwise} \end{cases}$$

procedure RECEIVE(Echo Request)

```
1   if S ∉ L then
2       L += S
3       W(S) = ∞
4   T(S) = NOW
5   send echo request to S
```

procedure RECEIVE(Echo Reply)

```
1   if S ∉ L then
2       L += S
3       W(S) = ∞
4   T(S) = NOW
```

procedure ECHO()

```
1   T(REQ) = NOW
2   send echo request to each x ∈ L
3   wait for echo timeout
4   updated = FALSE
5   for each x ∈ L do
6       W(x) = T(x) > T(REQ) ? T(x) − T(REQ) : ∞
7       if x ∉ K then
8           if W(x) ≠ ∞ then
9               K += x
10              P(x) = ∅
11              D(x) = W(x)
12              updated = TRUE
13          else
14              if P(x) ≠ ∅ and W(x) ≤ D(x) then
15                  P(x) = ∅
16                  D(x) = W(x)
17                  updated = TRUE
18              else if P(x) = ∅ and W(x) ≠ D(x) then
19                  D(x) ⇜ W(x)
20                  updated = TRUE
21      if W(x) = ∞ and T(REQ) − T(x) ≥ link life then
22          L −= x
23      if updated then
24          send {(k, D(k)) | k ∈ K, P(k) ≠ n} to each n ∈ N
25          K −= {k ∈ K | D(k) = ∞}
```

procedure RECEIVE(U)

updated = FALSE
inform = FALSE
if $S \notin L$ **or** $W(S) = \infty$ **then**
    $L \mathrel{+}= S$
    $W(S) = U(M)$
if $S \notin K$ **then**
    $K \mathrel{+}= S$
    $P(S) = \varnothing$
    $D(S) = U(M)$
    updated = TRUE

$N' = \{k \in K \mid P(k) = \varnothing\}$
let X be a permutation of $R \setminus \{M\}$
    $\hookrightarrow$ such that $(\forall x_i \in N')(\forall x_j \notin N')[i < j]$
$U' = \{\}$
for each $x \in X$ **do**
    $D' = D(S) + U(x)$
    if $x \notin K$ **then**
        if $D' \neq \infty$ **then**
            $K \mathrel{+}= x$
            $P(x) = S$
            $D(x) = D'$
            updated = TRUE
    else
        if $D' < D(x)$ **then**
            $P(x) = S$
            $D(x) \rightsquigarrow D'$
            $U' \mathrel{+}= \{k \in K \mid P(k) = x\}$
            updated = TRUE
        else if $D(x) \neq D'$ **and** $P(x) = S$ **and** $x \notin U'$ **then**
            $D(x) \leftarrow D'$
            updated = TRUE
        if $U(M) + D(x) < U(x)$ **and** $P(x) \neq S$ **then**
            inform = TRUE

if updated **or** inform **then**
    destination = {}
    if updated **then**
        destination $\mathrel{+}= N \setminus \{S\}$
    if inform **then**
        destination $\mathrel{+}= S$
    send $\{(k, D(k)) \mid k \in K,\ P(k) \neq d\}$ to each $d \in$ destination
    $K \mathrel{-}= \{k \in K \mid D(k) = \infty\}$