

# An Efficient Distributed Protocol for Online Gossiping Problem \*

Zhengnan Shi and Pradip K Srimani  
Dept of Computer Science  
Clemson University  
Clemson, South Carolina 29634

## Abstract

*In this paper, we propose an efficient distributed protocol for online gossiping problem in any types of networks, especially for mobile networks and fault-tolerant networks. The nodes in the networks have limited information of the entire network. Each node knows its neighboring nodes. The proposed gossiping protocol is fully distributed and tolerates node or link failures as well as topology changes (mobility). We show that the protocol completes in  $O(n^2)$  time in any types of networks, even when the networks contain multiple failures.*

**Keywords:** Online gossiping protocol, distributed computing, mobile ad hoc networks, fault tolerance

## 1 Introduction

The *broadcasting* problem (sending a message from one node to all other nodes) over an arbitrary communication network is computationally difficult; there are efficient algorithms to construct optimal communication schedules for restricted networks under some communication models [3, 4, 5, 6]. In this paper, we propose an online protocol which generates a live gossiping schedule for each node  $P_i$  based on the information of its neighboring nodes. We show that the protocol is fault tolerant. The proposed protocol completes correctly in  $O(n^2)$  time with multiple faults in the network. We prove that the faults do not significantly increase execution time. We compare our algorithm with the recent EGM algorithm [8], an efficient algorithm in the multicasting communication environment; unlike our protocol, EGM protocol uses the root node heavily (all messages between subtrees go via the root); in case of root failure the gossiping schedule becomes invalid.

---

\*The research reported in this paper is supported by NSF grants # ANI-0073409 and # ANI-0218495.

## 2 System and Communication Model

Let  $P_i$ ,  $1 \leq i \leq n$ , be a node in the communication network of  $n$  nodes. Each node  $P_i$  holds one unique message. The objective of gossiping is that each node becomes aware of the messages of every other node in the network. The total time of data transmissions of the network is divided into *time units* or *rounds*. A round is a short time period consisting of three consequent time slots - one for control messages and two for data messages.

The objective of the proposed scheme is to minimize the total time needed for the process to complete. The protocol ends when each processor has all  $n$  unique messages. The communication model in the network must satisfy the following two restrictions:

1. In a round  $t$ , each node may receive at most one data message provided such message was sent during the previous round or the previous time slot of the same round. The data message that node  $P_i$  receives (if any) at time  $t$  is added to its hold data structure  $h_i$  in the same round.
2. During each data message time slot (denoted as  $s_d$ ), each node  $P_i$  may transmit one of the data messages in its hold  $h_i$  to its neighbors. The data transmission is one hop. The message will remain in  $h_i$ . A message that is sent to  $P_i$  at time slot  $s-1$  arrives at  $P_i$  at time slot  $s$ . The node  $P_i$  may send it to its neighbors in the same time slot.
3. During the control message time slot (denoted as  $s_c$ ), each node  $P_i$  may send and receive multiple control messages.

### 2.1 Proposed Approach

In our model, each node  $P_i$  has the list of neighboring nodes but not the complete knowledge of the entire network. Therefore our protocol is distributed and online. The list of neighbors is maintained by the underlying data link

layer protocol. The basic idea of the distributed protocol is: (1) Each processor  $P_i$  sends to its neighbors a control message of what data messages are in its hold  $h_i$ . The originate processor ID uniquely identifies each data message. In the implementation section, this control message is represented by a bit map with each bit represent a unique message. If the bit is 0 then the message is missing in  $h_i$ ; (2) Because the neighboring nodes send the same control messages,  $P_i$  has knowledge of each  $h_j$  of node  $P_j$ ,  $P_j$  is a neighbor of  $P_i$ .  $P_i$  then sends a request message to one neighbor of the lowest ID whose hold contains a needed message. This is the end of control time slot  $s_c$  in each round; (3) Node  $P_i$  checks all the incoming requests. It selects one or more neighbors, then sends or multicasts one data message to the selected neighbor(s). In case node  $P_i$  and its neighbor  $P_j$  request data messages from each other: if  $P_i > P_j$ , then  $P_i$  sends the data message to  $P_j$  in the first data time slot and  $P_j$  sends to  $P_i$  in the second data time slot.

### 3 Proposed Protocol

#### 3.1 Data Structure

As noted in the introduction,  $n$  is the total number of nodes in the network.

1.  $Neighbors[1 : n]$  is a Boolean array. The value  $Neighbors_i[j]$  is true if node  $j$  is within one hop communication distance from node  $i$  in the network. This array is maintained by data link layer protocol.
2.  $h[1 : n]$  is an array of data messages. A NULL entry in the array  $h_i$  means the data message of the corresponding index ID has not arrived at node  $i$ .
3.  $b[1 : n]$  is an array of binary bits. If  $h_i[k]$ ,  $1 \leq k \leq n$ , is not NULL, then  $b_i[k]$  is 1, otherwise  $b_i[k]$  is 0.
4.  $r[1 : \Delta]$  is a receiver buffer. Each entry of the buffer contains the source node ID (r.ID) and the message received (r.m). It may contain either multiple control messages or one data message. At the end of each round, the content of the buffer is flushed.

#### 3.2 Message Types and System Primitives

The protocol uses two types of control messages and one type of data message.

1.  $Control(i, b[1 : n])$  This is a control message. The message is sent to all the neighbors of node  $P_i$ . It contains bitmap  $b[1 : n]$  indicating the messages of node  $P_i$ .
2.  $Request(i, j, k)$  This is also a control message. The message is sent by node  $P_i$  to node  $P_j$  requesting message number  $k$ .

3.  $Data(i, j, h[k])$  This is a data message. The message is sent by node  $P_i$  to node  $P_j$ . The data message contains message  $k$  from array  $h_i$  of node  $P_i$ .

The following system primitives are used in the pseudo-code given in the next section. They are used to simplify the presentation of the algorithm. Note that both *send* and *receive* primitives are one hop message transfer: (1) **send**(*source, destination, message*), if any link failures or node failures are reported by the underlying data link layer protocol, the corresponding neighboring nodes are simply deleted from the *Neighbors* list. (2) **receive**(*source, message*), the parameter *source* will return the source node ID (address) of the received message; the receive primitive includes a timeout. If no message arrives at the receiver network interface, the receive primitive will terminate at the end of each time slot.

### 4 Algorithm Description

In this section, we present the protocol in pseudo-code. At each node  $P_i$  we run two main processes. One of the processes is for sending messages; the other is for receiving messages. The two processes can communicate with each other. They both have access to the local system clock via system call: `get-current-time()`. As stated in system model section, we assume that the local clocks are approximately synchronized. In the pseudo-code, constant  $t_c$  indicates the time boundary between control time slot and data time slots. Constant  $t_d$  indicates the time boundary of the two data time slots. Constant  $t$  represents the time period of a complete round. The procedures are run on node  $P_i$ .

The procedure *Receive* is the receiving process of the gossiping protocol. Variable  $r$  is the receiving buffer at node  $P_i$ . The variables that are received from or calculated based on the sending process are prefixed with *sd-*. Each round of the *Receive* procedure is divided into two consequent time slots – one for control message and the other for data message. The *Receive* procedure receives all the messages for a node.

---

Procedure *Receive*()

```

begin
  while(true)
    begin
      i = 1;
      tt = get-current-time();
      //control message slot
      while(true)
        begin
          //check control messages
          receive(r[i].ID, r[i].m);
          i ++;
          temp = get-current-time();
          if(temp ≥ tt + tc)

```

```

    then break;
end
//check request messages
while(true)
  begin
    receive(r[i].ID,r[i].m);
    i ++;
    temp =get-current-time();
    if(temp ≥ tt + td)
      then break;
    end
  //data message slot; check data message
  receive(sd-j,h[sd-k]);
end
end

```

Next, we introduce the subroutine *wait-until*. It is implemented for the *Send* procedure. The procedure causes the calling process to idle until an absolute time value. The absolute time value is give by parameter  $t$ .

---

```

Procedure wait-until(t)
begin
  while(true)
    begin
      temp =get-current-time();
      if (temp ≥ t)
        then return;
      end
    end
  end
end

```

The procedure *cal-request* calculates and returns the ID (parameter  $j$ ) of a neighboring node who has a needed message with the smallest ID (parameter  $k$ ) among the neighbors of  $P_i$ .

---

```

Procedure cal-request(j,k)
begin
  j = n + 1; k = n + 1;
  for index = 1 ··· n
    begin
      if r[index].ID=NULL
        then return;
      //pick the smallest node ID with the smallest message ID
      if r[index].m < k
        then k =r[index].m;
        if r[index].ID < j
          then j=r[index].ID;
        end
      end
    end
  end
end

```

The procedure *cal-data* calculates and returns the ID (parameter  $p$ ) of a neighboring node whose request of message ID (parameter  $k$ ) will be responded by node  $P_i$ .

---

```

Procedure cal-data(p,q)
begin
  p = n + 1; q = n + 1;
  for index = 1 ··· n
    begin

```

```

      if r[index].ID=NULL
        then return;
      //pick the smallest node ID with the smallest message ID
      if r[index].m < q
        then q =r[index].m;
        if r[index].ID < p
          then p =r[index].ID;
        end
      end
    end
  end
end

```

The following procedure *Send* is the sending process of the gossiping protocol. The variables that are received from or calculated based on the receiving process are prefixed with *rec-*. Each round of the *Send* procedure is divided into three consequent time slots – one for control messages and two for data messages. The *Send* procedure sends all the messages from a node.

---

```

Procedure Send()
begin
  while(true)
    begin
      tt =get-current-time();
      //control message slot, send control messages
      send(i,neighbors,Control(i,bi));
      wait-until(tt + tc);
      cal-request(j,k);
      //send one request message
      send(i,neighbors,Request(i,rec-j,rec-k));
      wait-until(tt + td);
      //data message slots
      cal-data(p,q);
      if rec-j ≠ rec-p ∪ i < rec-p
        then
          send(i,rec-p,Data(i,rec-p,hi[rec-q]));
        else
          wait-until(tt + t);
          send(i,rec-p,Data(i,rec-p,hi[rec-q]));
        end
      end
    end
  end
end

```

The *Send* and *Receive* procedures transfer messages between the nodes and solve the gossiping problem of the entire network.

## 5 Correctness and Performance Analysis

In this section, we look at the correctness and complexity of our protocol. We take network changes and faults into consideration while conducting the analysis; we omit the proofs for lack of space.

### 5.1 Correctness

**Theorem 1** *Upon stabilization, the proposed protocol completes gossiping in connected networks.*

**Theorem 2** Upon stabilization, the proposed protocol produces a live communication schedule for gossiping in connected mobile networks.

**Theorem 3** In a connected network with node failures and/or link failures, the proposed protocol achieves gossiping upon stabilization.

The proposed protocol achieves gossiping upon stabilization in mobile networks, as well as networks with one or more failures. If the network is no longer connected before a gossiping protocol is finished, no more data messages can be transferred between the disconnected components of the network.

## 5.2 Complexity

To bound the complexity of the protocol, we have the following lemma.

**Lemma 1** In each round  $t$ , there is at least one node sends and one node receives a data message in a connected network.

**Lemma 2** In a connected network of  $n$ , the total number of data messages received for gossiping is no more than  $O(n^2)$ .

**Theorem 4** If the network is connected, the proposed protocol completes in  $O(n^2)$  rounds in the worst case.

The upper bound of the number of data messages for gossiping does not change in a mobile network. We summarize this into the following corollary of theorem 4.

**Corollary 1** In a mobile network, the proposed protocol completes in  $O(n^2)$  rounds in the worst case.

**Theorem 5** In a connected network with any number of node failures and link failures, the proposed protocol completes in  $O(n^2)$  rounds in the worst case.

We have shown that the protocol is fault tolerant. It can adjust to mobile environment and tolerates multiple failures. The complexity of the algorithm does not increase for mobility or network faults.

## 6 Comparison and Discussion

The EGM algorithm [8] consists of two steps. First, it builds a special tree network and, then, it performs all the communications in that tree network. The EGM protocol takes  $O(mn)$  time. The proposed protocol has better time complexity  $O(n^2)$ .

**Online and Distributed:** The EGM algorithm is an offline protocol. A node with full knowledge of the network calculates the communication schedule ahead of time. The proposed algorithm in this paper is an online protocol. Each node in the network only needs to know its neighboring nodes. None of the nodes in the network is required to have the complete knowledge of the network.

**Fault Tolerance:** The EGM algorithm is vulnerable to faults in the network. A node constructs the communication schedule ahead of time. Hence the produced schedule is not guaranteed to work if any of the spanning tree links or nodes fails in the network. In contrast, the proposed protocol is fault tolerant. It is shown to tolerate one or more faults, as well as network topology changes. The proposed protocol is proven to complete in  $O(n^2)$  time even with multiple link and/or node faults in the network.

**Mobility:** As an offline protocol, the EGM algorithm cannot be used in mobile networks. Mobility could invalidate the minimum depth spanning tree of the EGM protocol or the precalculated communication schedule. Our protocol is fully applicable to mobile networks.

## References

- [1] P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. *Discrete Applied Math.*, 53:79–133, 1994.
- [2] S. T. Hedetniemi S. M. Hedetniemi and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, (18):129–134, 1988.
- [3] S. Even and B. Monien. On the number of rounds necessary to disseminate information. In *Proc. Int'l Symp. Algorithms. Parallel Algorithms and Architectures*, pages 318–327, 2001.
- [4] S. Fujita and M. Yamashita. Optimal group gossiping in hypercube under circuit switching rounds. *SIAM J. Computing*, 25(5):1045–1060, 1996.
- [5] K. N. Venkataraman D. W. Krumme and G. Cybenko. Gossiping in minimal times. *SIAM J. Computing*, 21(2):111–139, 1992.
- [6] R. Ravi. Rapid rumor ramification. In *Proc. 35th Ann. Symp. Foundations of Computer Science*, pages 202–213, 1994.
- [7] W. Heintzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Adaptive protocols for information dissemination in wireless sensor networks, Proc. MO-BICOM*, pages 174–185, Seattle, 1999.
- [8] Teofilo F. Gonzalez. An efficient algorithm for gossiping in the multicasting communication environment. *IEEE Trans. Parallel Distrib. Syst.*, 14(7):701–708, 2003.