The Disciplined Flood Protocol in Sensor Networks

Young-ri Choi and Mohamed G. Gouda
Department of Computer Sciences
The University of Texas at Austin, U.S.A.
{yrchoi, gouda}@cs.utexas.edu

Hussein M. Abdel-Wahab
Department of Computer Science
Old Dominion University, U.S.A.
wahab@cs.odu.edu

Ehab S. Elmallah
Department of Computing Science
University of Alberta, Canada
ehab@cs.ualberta.ca

***Abstract*— Flood is a communication primitive that can be initiated by the base station of a sensor network to send a copy of some message to every sensor in the network. When a flood of some message is initiated, the message is forwarded by every sensor that receives the message until the sensors decide not to forward the message any more. This uncontrolled flood can cause the forwarded messages to collide with one another, with the result that many sensors in the network do not receive any copy of the flooded message. In this paper, we present a flood protocol, called the disciplined flood protocol, that aims to reduce message collisions, avoids redundant message forwarding, and guarantees the termination of a flood without requiring any sensor to maintain any extra information. The disciplined flood protocol is simple and practical for sensor networks that have limited resources.**

I. Introduction

Flood is a communication primitive that can be used by the base station of a sensor network to send a copy of a data message to every sensor in the network. The execution of a flood starts by the base station sending a copy of the data message to everyone of its neighboring sensors. Whenever a sensor receives a data message, it keeps a copy of the message and forwards the message to everyone of its neighboring sensors and the cycle repeats.

To limit the (potentially indefinite) forwarding of a flooded data message within a sensor network, the message is augmented with a field h whose value is in a range $0..hmax$. When the base station sends the data message for the first time, field h in the message has the value $hmax$. Whenever a sensor receives a data(h) message, where $h \geq 1$, then the sensor forwards the message as data($h-1$). Whenever a sensor receives a data(0) message, then the sensor does not forward the message any further.

Flood has several significant uses in sensor networks. In one use, the base station of a sensor network needs to reset the network, and it uses flood to send a reset message to every sensor in the network requesting that each sensor resets itself upon receiving the message. In a second use, the base station needs to pass some data message to some (not necessarily all) sensors in the network. In this case, the base station uses flood to send the data message to all the sensors in the network, but name in the message those sensors that should find the message relevant.

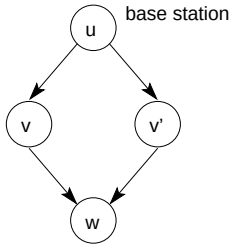

Fig. 1. A sensor network

Unfortunately, flood can cause severe problems in sensor networks.

i) *Collisions within a Single Flood:* Consider the sensor network in Fig. 1. If the base station in this network initiates a flood by sending a data(1) message, then each of the two neighboring sensors, v and v', of the base station receives the message and forwards it as data(0). Because the two data(0) messages are forwarded (by v and v') at the same time, they collide and sensor w never gets a copy of the flooded data message.

ii) *Collisions of Consecutive Floods:* If the base station of a sensor network initiates one flood and shortly after initiates another flood, some forwarded messages from these two floods can "collide" with one another causing many sensors in the network not to receive the message of either flood, or (even worse) not to receive the messages of both floods.

iii) *Redundant Forwarding:* Consider the sensor network in Fig. 2. If the base station in this network initiates a

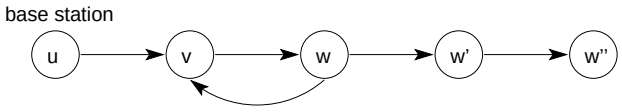

Fig. 2. A sensor network

flood by sending a data(3) message, then
sensor v forwards a data(2) message,
sensor w forwards a data(1) message,
sensor w' forwards a data(0) message, and
sensor w'' forwards no message.
Unfortunately sensor v also receives the data(1) message that is forwarded by sensor w and redundantly forwards it as data(0). Thus, both sensors v and w redundantly receive the message one time each, and sensor v redundantly forwards the message one more time.

In this paper, we present a flood protocol, called the disciplined flood protocol, where the above three problems do not occur. Also this protocol does not require sensors in a network to maintain any extra information for terminating a flood. The disciplined flood protocol has the following important features or properties:

i) *Few Collisions within a Single Flood:* When a sensor u receives a data(h) message and checks that $h \geq 1$ (which means that u needs to forward the message), then u selects a random time period, called the forwarding period, and forwards the message only at the end of that period. (Recall that u forwards the message as data($h-1$).)

ii) *No Collisions of Consecutive Floods:* After the base station initiates a flood by sending a data($hmax$) message, it abstains from initiating a second flood for a long enough time period until it is certain that the sensors in the network are no longer forwarding data messages that belong to the first flood. In this paper, we refer to the time period between two consecutive floods as the flood period, and compute a lower bound on the flood period that can be used in our disciplined flood protocols.

iii) *No Redundant Forwarding:* When a sensor u receives a data(h) message and decides that it needs to forward the message as data($h-1$) after a random forwarding period, u computes a time period called the deafness period. If u receives any data(h') message during the computed deafness period, u concludes that the data(h') message belongs to the same flood as that of the earlier data(h) message, and discards the data(h') message without keeping a copy of it and without forwarding it.

There are several common methods to terminate a flood. The first method is to attach a time-to-live (TTL) value to each flood message (e.g. [1]). In this method, the sensors do not maintain any extra information. When a sensor receives a flood message with TTL>1, the sensor decreases the TTL value by one, and forwards the message. However, this method causes a sensor to forward the same message multiple times. The second method is to attach a sequence number to each flood message (e.g. [5], [7]). In this method, the sensors remember the sequence numbers of received flood messages, and so the sensors can recognize redundantly forwarded messages. However, the sensors need to maintain a bit array for the message sequence numbers. The third method is to use a spanning tree. In this method, a sensor forwards a flood message only when the sensor receives the message from its parent. However, the sensors need to build and maintain a spanning tree.

Several flood protocols have been proposed to reduce the total number of forwarded messages in a flood based on probability, location, or neighbor information [5], [6], [4], [3]. Unlike these protocols, our disciplined flood protocols control the activities within a single flood or across consecutive floods in order to satisfy the above three properties.

II. Sensor Network Execution

In this section, we present a formal model of the execution of a sensor network. We use this model to specify the disciplined flood protocol in the next section.

The *topology* of a sensor network is a directed graph that satisfies the following two conditions. First, each node in the topology represents a distinct sensor in the sensor network. Second, each directed edge (u, v) from node u to node v in the topology indicates that every message sent by sensor u can be received by sensor v (provided that neither sensor v nor any "neighboring sensor" of v sends a message at the same time when sensor u sends its message).

If the topology of a sensor network has a directed edge from a sensor u to a sensor v, then u is called an *in-neighbor* of v and v is called an *out-neighbor* of u. Note that a sensor can be both an in-neighbor and an out-neighbor of another sensor in the sensor network.

As an example, Figure 3 shows the topology of a sensor network. This network has six sensors, and sensor u in this network has three out-neighbors, namely sensors v, v', and v''.

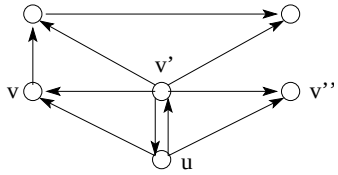

Fig. 3. Topology of a sensor network

Thus, if sensor u sends a message, then this message can be received simultaneously by the three sensors v, and v', and v''. Note that sensor u is both an in-neighbor and an out-neighbor of sensor v' in this network.

We assume that during the execution of a sensor network, the real-time passes through discrete instants: instant 1, instant 2, instant 3, and so on. The time periods between consecutive instants are equal. The different activities that constitute the execution of a sensor network occur only at the time instants, and not in the time periods between the instants. We refer to the time period between two consecutive instants t and $t+1$ as a *time unit* (t, $t+1$). (The value of a time unit is not critical to our current presentation of a sensor network model, but we estimate that the value of the time unit is around 100 milliseconds.)

At a time instant, the time-out of a sensor u may expire causing u to execute its timeout action. Executing the timeout action of sensor u causes u to update its own local variables and to send at most one message. It may also cause u to execute the statement "timeout-after <expression>" which causes the time-out of u to expire (again) after k time units, where k is the current value of <expression>. The timeout action of sensor u is of the following form:

```
timeout-expires ->
   <update local variables of u>;
   <send at most one message>;
   <may execute timeout-after <expression>>
```

To keep track of its time-out, each sensor u has an implicit variable named "timer.u". In each time unit between two consecutive instants, the implicit variable timer.u is either "present" or "not-present". Moreover, if variable timer.u is present in a time unit, then it has a positive integer value in that time unit. Otherwise, it is not-present and has no value in the time unit.

If sensor u executes a statement "timeout-after <expression>" at instant t, then timer.u becomes present in the time unit (t, $t+1$) and its value is the value of <expression> in this time unit.

If timer.u is present and its value is k, where $k > 1$, in the time unit ($t-1$, t), then timer.u continues to be present and its value is $k-1$ in the time unit (t, $t+1$).

If timer.u is present and its value is 1 in the time unit ($t-1$, t), then sensor u executes its timeout action at instant t and timer.u becomes not-present in the time unit (t, $t+1$), unless u executes "timeout-after <expression>" as part of its timeout action.

If a sensor u executes its timeout action and sends a message at instant t, then any sensor v, that is an out-neighbor of u, receives a copy of the message at instant t, provided that the following two conditions hold.

i) Sensor v does not send any message at instant t. (This condition indicates that either sensor v does not execute its timeout action at t, or it executes its timeout action at t but this execution of the timeout action does not include sending a message.)

ii) Sensor v has no in-neighbor, other than sensor u, that sends a message at instant t. (If v sends a message at t or if an in-neighbor of v, other than u, sends a message at t, then this message is said to *collide* with the message sent by u at t with the net result that v receives no message at t.)

If a sensor u receives a message at instant t, then u executes its receiving action at t. Executing the receiving action of sensor u causes u to update its own local variables. It may also cause u to execute the statement "timeout-after <expression>" which causes the time-out of u to expire after k time units, where k is the value of <expression> in the time unit (t, $t+1$). The receiving action of sensor u is of the following form:

```
rcv <msg> ->
   <update local variables of u>;
   <may execute timeout-after <expression>>
```

It follows from the above discussion that at a time instant, a sensor u executes exactly one of the following:

i) u sends one message, but receives no message.
ii) u receives one message, but sends no message.
iii) u sends no message and receives no message.

In the next section, we specify the disciplined flood protocol using the formal model of sensor protocols in this section.

III. Disciplined Flood Protocol

Consider a network that has n sensors. In this network, sensor 0 is the base station and can initiate message floods over the network. To initiate the flood of a message, sensor 0 sends a message of the form data($hmax$), where $hmax$ is the number of hops to be made by this data message in the network.

Once sensor 0 broadcasts a message, it needs to wait enough time until this message is no longer forwarded in the network, before broadcasting a next message. The time period that sensor 0 needs to wait after broadcasting a message and before broadcasting a next message is called the *flood period*. The flood period consists of f time units. (A lower bound on the value of f is computed in the next section.) Thus, after sensor 0 broadcasts a message, it sets its timeout to expire after f time units in order to broadcast a next message.

A formal specification for sensor 0 is as follows.

```
sensor 0                  {base station}

const hmax    : integer, {max hop count}
      f       : integer  {flood period}
begin
   timeout-expires -> {generate new msg}
                      send data(hmax);
                      timeout-after f
end
```

Note that sensor 0 does not receive any messages.

When a sensor receives a data(h) message, the sensor accepts the message and forwards it as a data($h-1$) message, provided $h > 0$. To reduce the probability of message collision, any sensor u, that receives a message, chooses a random period whose length is chosen uniformly from the range 1..$tmax$, and sets its timeout to expire after the chosen random period, so that u can forward the received message at the end of the random period. This random time period is called the *forwarding period*.

Once sensor u accepts a received message and decides that it needs to forward the message after a random forwarding period, sensor u discards all subsequently received messages during a time period, called the *deafness period*, until the same message is no longer forwarded in the network. This way, sensor u is guaranteed not to accept or forward the same message multiple times. The deafness period consists of d time units. (A lower bound on the value of d is computed in the next section.) At the end of the deafness period, sensor u times-out and becomes ready to accept and forward the next received message.

A sensor u is in any of three states: an accepting state, a forwarding state and a deafness state. In the accepting state, u is ready to accept and forward any data(h) message it receives, provided $h > 0$. In the forwarding state, u is waiting for its forwarding period to finish, so that it can forward the last data(h) message it has accepted as data($h-1$). In the deafness state, u discards any data(h) message it receives. It stays in this state for the duration of the deafness period (d time units). Fig. 4 shows the three states of sensor u and the different transitions between them. The cycle of an accepting state followed by a forwarding state and then a deafness state (or an accepting state followed by a deafness state) is repeated over and over. Sensor u maintains a state variable st that has three possible values 0, 1, and 2.

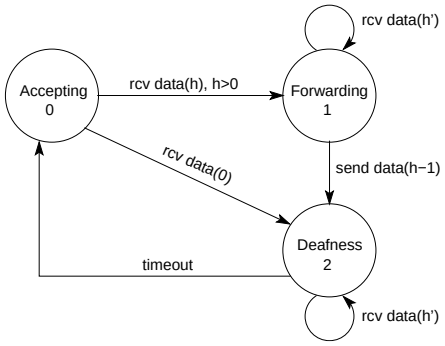

Fig. 4. Three states of a sensor

$$st = \begin{cases} 0 & \text{if } u \text{ is in accepting state} \\ 1 & \text{if } u \text{ is in forwarding state} \\ 2 & \text{if } u \text{ is in deafness state} \end{cases}$$

A formal specification for sensors 1 .. $n-1$ is as follows.

```
sensor 1..n-1

const hmax    : integer, {max hop count}
      tmax    : integer, {max frwrding period}
      d       : integer  {deafness period}
var   st      : 0..2,    {state, init. 0}
      h,hlast : 0..hmax, {rcvd,last hop cnt}
      t       : 1..tmax  {forwarding period}
begin
   timeout-expires ->
           if st!=1 -> st := 0
           [] st=1  -> send data(hlast);
                       st := 2;
                       timeout-after d
           fi

[] rcv data(h) ->
   if st=0 -> {accept msg}
           if h>0 -> st := 1;
                     hlast := h-1;
                     t := random;
                     timeout-after t
           [] h=0 -> st := 2;
                     timeout-after d
           fi
   [] st>0 -> skip
fi
end
```

Next, we estimate the deafness period and the flood period of the above disciplined protocol, and analyze the behavior of this protocol.

Theorem 1 : *(The Deafness Period Theorem)*

$$d \geq hmax * tmax + 1$$

Theorem 2 : *(The Flood Period Theorem)*

$$f \geq hmax * tmax + d$$

The proofs of Theorem 1 and 2 can be found in [2].

To analyze the protocol, we use the minimum possible values for the deafness period d and the flood period f, from Theorem 1 and 2, as follows:

$$d = hmax * tmax + 1$$
$$f = 2 * hmax * tmax + 1$$

Adopting these values of d and f, it is straightforward to show that the disciplined flood protocol satisfies the following three properties discussed in Section I: (i) Few collisions within a single flood. (ii) No collisions of consecutive floods. (iii) No redundant forwarding.

In this protocol, if a sensor receives a message in an accepting state, then the message is new and the sensor accepts it. Moreover, sensor 0 broadcasts a new message only when every sensor in the network is in an accepting state. Therefore, the sensor does not discard any new message, if the sensor receives it.

For the simulation result of this protocol, readers can refer to [2].

IV. Concluding Remarks

In this paper, we presented the disciplined flood protocol. that aims to reduce message collisions, avoids redundant message forwarding, and guarantees the termination of a flood without requiring any sensor to maintain any extra information. The disciplined flood protocol is simple and practical for sensor networks that have limited resources.

In the disciplined flood protocol, sensor 0 may need to wait for a long time before broadcasting a next message, when large values are chosen for $tmax$ and $hmax$. To reduce the flood period of the protocol, one bit sequence number can be attached to each flood message. A flood protocol that can reduce the flood period by a factor of two is discussed in [2].

Acknowledgment

This work was supported by the Defense Advanced Research Projects Agency (DARPA) Contract F33615-01-C-1901. We would like to thank Dr. Mehmet Karaata for interesting discussions.

References

[1] N. Chang and M. Liu. Revisiting the TTL-based controlled flooding search: optimality and randomization. In *Proceedings of the ACM MobiCom 2004*, PA, September 2004.

[2] Y. Choi and M. Gouda. Disciplined Flood Protocols in Sensor Networks. *Technical Report TR-05-03, Department of Computer Sciences, the University of Texas at Austin*, 2003.

[3] R. Gandhi, S. Parthasarathy, and A. Mishra. Minimizing broadcast latency and redundancy in ad hoc networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing*, Maryland, 2003.

[4] H. Lim and C. Kim. Multicast Tree Construction and Flooding in Wireless Ad Hoc Networks. In *Proceedings of the ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*, 2000.

[5] S. Ni, Y. Tseng, Y. Chen, and J. Sheu. The Broadcast Strom Problem in a Mobile Ad Hoc Network. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 151–162, 1999.

[6] M. Sun, W. Feng, and T. Lai. Location Aided Broadcast in Wireless Ad Hoc Networks. In *Proceedings of the IEEE GLOBECOM 2001*, pages 2842–2846, November 2001.

[7] B. Williams and T. Camp. Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing*.