Stabilization of Flood Sequencing Protocols in Sensor Networks

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

TR-06-58

Abstract

Flood is a communication primitive that can be used by the base station of a sensor network to send a copy of a message to every sensor in the network. When a sensor receives a flood message, the sensor needs to check whether it has received the message for the first time and so the message is fresh, or it has received the same message earlier and so the message is redundant. In this paper, we discuss a family of four flood sequencing protocols that use sequence numbers to distinguish between fresh flood messages and redundant flood messages. They are a sequencing free protocol, a linear sequencing protocol, a circular sequencing protocol, and a differentiated sequencing protocol. We analyze the self-stabilization properties of these four flood sequencing protocols. We also compare the performance of these flood sequencing protocols, using simulation, over various settings of sensor networks. We conclude that the differentiated sequencing protocol has better stabilization property and provides better performance than those of the other three protocols.

Key words*: Self-stabilization, Flood sequencing protocol, Sequence numbers, Sensor networks*

Stabilization of Flood Sequencing Protocols in Sensor Networks

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

Abstract

Flood is a communication primitive that can be used by the base station of a sensor network to send a copy of a message to every sensor in the network. When a sensor receives a flood message, the sensor needs to check whether it has received the message for the first time and so the message is fresh, or it has received the same message earlier and so the message is redundant. In this paper, we discuss a family of four flood sequencing protocols that use sequence numbers to distinguish between fresh flood messages and redundant flood messages. They are a sequencing free protocol, a linear sequencing protocol, a circular sequencing protocol, and a differentiated sequencing protocol. We analyze the self-stabilization properties of these four flood sequencing protocols. We also compare the performance of these flood sequencing protocols, using simulation, over various settings of sensor networks. We conclude that the differentiated sequencing protocol has better stabilization property and provides better performance than those of the other three protocols.

Key words*: Self-stabilization, Flood sequencing protocol, Sequence numbers, Sensor networks*

1 Introduction

Flood is a communication primitive that can be used by the base station of a sensor network to send a copy of a message to every sensor in the network. The execution of a flood starts by the base station sending a message to all its neighbors. When a sensor receives a message, the sensor needs to check whether it has received this message for the first time or not. Only if the sensor has received the message for the first time, the sensor keeps a copy of the message and may forward the message to all its neighbors. Otherwise, the sensor discards the message.

To distinguish between "fresh" flood messages that a sensor should keep and "redundant" flood messages that a sensor should discard, the base station selects a sequence number and attaches it to a flood message before the base station broadcasts the message. When a sensor receives a flood message, the sensor determines based on the sequence number in the received message if the message is fresh or redundant. The sensor accepts the message if it is fresh and discards the message if it is redundant. We call a protocol that uses sequence numbers to distinguish between fresh flood messages and redundant flood messages a *flood sequencing protocol*.

In a flood sequencing protocol, when a fault corrupts the sequence numbers stored in some sensors in a sensor network, the network can become in an illegitimate state where the sensors discard fresh flood messages and accept redundant flood messages. Therefore, a flood sequencing protocol should be designed such that if the protocol ever reaches an illegitimate state due to some fault, the protocol is guaranteed to converge back to its legitimate states where every sensor accepts every fresh flood message and discards every redundant flood message.

In this paper, we discuss a family of four flood sequencing protocols. They are a *sequencing free* protocol, a *linear sequencing* protocol, a *circular sequencing* protocol, and a *differentiated sequencing* protocol. We analyze the stabilization properties of these four protocols. For each of the protocols, we first compute an upper bound on the convergence time of the protocol from an illegitimate state to legitimate states. Second, we compute an upper bound on the number of fresh flood messages that can be discarded by each sensor during the convergence. Third, we compute an upper bound on the number of redundant flood messages that can be accepted by each sensor during the convergence.

The rest of the paper is organized as follows. In Section 2, we discuss related work and motivation of the flood sequencing protocols. In Section 3, we present a model of the execution of a sensor network. In Section 4, we give an overview of a flood protocol. We present the four flood sequencing protocols and analyze their stabilization properties in Sections 5, 6, 7, and 8. In Section 9, we show the simulation results of these protocols. We finally make concluding remarks in Section 10.

2 Related Work and Motivation

The practice of using sequence numbers to distinguish between fresh and redundant flood messages has been adopted by most flood protocols in the literature. In other words, most flood protocols "employ" some flood sequencing protocols to distinguish between fresh and redundant flood messages. A flood sequencing protocol can be designed in various ways, depending on several design decisions such as how the next sequence number is selected by the base station, how each sensor determines based on the sequence number in a received message if the received message is fresh or redundant, and what information the base station and each sensor stores in its local memory. Unfortunately, flood sequencing protocols have so far not been studied well in the literature. They have been used without full investigation of their design decisions.

The flood protocols discussed in [11, 10, 7, 2] assume that when a sensor receives a flood message, the sensor can figure out whether the sensor has received this message for the first time or not, without specifying any mechanism to achieve this. In [9, 4], it was suggested to associate a sequence number with each flood message, but any details on how sequence numbers are used by sensors (i.e. the design decisions of their flood sequencing protocols) were not specified. The flood protocols discussed in [13, 5] propose to attach a unique identifier to each flood message and make each sensor maintain a list of identifiers that the sensor has received recently. Similarly, it was suggested in [12] that each sensor maintains a list of flood messages received by the sensor recently. However, any details such as how many identifiers or messages each sensor maintains and when a sensor deletes an identifier or a message from the list were not discussed.

A flood sequencing protocol is important, since the fault tolerance property of a sensor network is affected by a flood sequencing protocol used in the network. When a fault corrupts the sequence number stored in some sensor in the network, the sensor may discard fresh flood messages and accept redundant flood messages. The number of fresh flood messages discarded by the sensor and the number of redundant flood messages accepted by the sensor, before the network reaches a legitimate state, are different depending on which flood sequencing protocol is used in the network. Therefore, we need to study various flood sequencing protocols and analyze the stabilization properties of these protocols. The stabilization properties of the flood sequencing protocols are useful for sensor network designers or developers to select a proper flood sequencing protocol that satisfies the needs of a target sensor network.

In practice, a flood sequencing protocol is used with a flood protocol that may use other techniques to improve the performance of flood such as reliability or efficiency. In this paper, each of the flood sequencing protocols is described focusing on how sequence numbers are used by sensors, and it is not described as a specific flood protocol. Note that the stabilization property of a flood protocol is affected by that of a flood sequencing protocol used in the flood protocol. If the flood protocol does not maintain any extra state such that it is based on probability [4, 10], the stabilization property of the flood protocol is the same as that of the used flood sequencing protocol. If the flood protocol maintains extra state such that it is based on neighbor information [8, 11], the stabilization property of the flood protocol also depends on how the extra state in each sensor is stabilized.

3 Model of Sensor Networks

In this section, we describe a formal model of the execution of a sensor network, which was introduced first in [3]. This model of sensor networks is abstract, but it accommodates characteristics of sensor networks such as unavoidable local broadcast, probabilistic message transmission, asymmetric communication, message collision, and timeout actions and randomization steps. We use this model to specify our flood sequencing protocols, verify the stabilization properties of these protocols, and develop our simulation of these protocols.

The *topology* of a sensor network is a directed graph where each node represents a distinct sensor in the network and where each directed edge is labeled with some probability. A directed edge (u,v), from a sensor u to a sensor v, that is labeled with probability p (where $p > 0$) indicates that if sensor u sends a message, then this message arrives at sensor v with probability p (provided that neither sensor v nor any "neighboring sensor" of v sends another message at the same time).

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.

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.)

A sensor is specified as a program that has global constants, local variables, one timeout action, and one receiving action.

At a time instant t, if the timeout of a sensor u expires, then u executes its timeout action at t. Executing the timeout action of sensor u at t causes u to update its local variables, and to send at most one message at t. It also causes u to execute the statement "timeout-after <expression>" which causes the timeout of u to expire (again) after k time units, where k is the value of <expression> at the time unit (t, t+1). The timeout action of sensor u is of the following form:

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

To keep track of its timeout, each sensor u has an implicit variable named "timer.u". In each time unit between two consecutive instants, timer.u has a fixed positive integer value. If the value of timer.u is k, where $k > 1$, in a time unit ($t - 1$, t), then the value of timer.u is $k - 1$ in the time unit (t, $t + 1$). On the other hand, if the value of timer.u is 1 in a time unit ($t - 1$, t), then sensor u executes its timeout action at instant t. Moreover, since sensor u executes the statement "timeout-after <expression>" as part of executing its timeout action, the value of timer.u in the time unit (t, $t + 1$) is the value of <expression> in the same time unit.

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

i. A random integer number is uniformly selected in the range 0 .. 99, and this selected number is less than $100 * p$, where p is the probability label of edge (u,v) in the network topology.

ii. Sensor v does not send any message at instant t.

iii. For each in-neighbor w of v, other than u, if w sends a message at t, then a random integer number is uniformly selected in the range 0 .. 99, and this selected number is at least $100*p'$, where p' is the probability label of edge (w,v) in the network topology.

If v sends a message at t, or if w sends a message at t and for v, selects a random number that is less than $100 * p'$, then this message *collides* with the message sent by u 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>". The receiving action of sensor u is of the following form:

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

A *state* of a sensor network protocol is defined by a value for each variable and timer.u for each sensor u in the protocol.

During the execution of a sensor network protocol, several faults can occur, resulting in corrupting the state of the protocol arbitrarily. Examples of these faults are wrong initialization, memory corruption, message corruption, and sensor failure and recovery. We assume that these faults do not continuously occur in the network.

4 Overview of a Flood Protocol

In this section, we give an overview of a flood protocol that is used with our flood sequencing protocols. Consider a network that has n sensors. In this network, sensor 0 is the base station and can initiate 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 maximum number of hops to be made by this data message in the network.

If sensor 0 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.

To prevent message collision across consecutive flood messages, once sensor 0 broadcasts a message, it needs to wait enough time until this message is no longer forwarded in the network, before broadcasting the next message. The time period that sensor 0 needs to wait after broadcasting a message and before broadcasting the 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 below.) Thus, after sensor 0 broadcasts a message, it sets its timeout to expire after f time units in order to broadcast the next message.

When a sensor receives a data(h) message, the sensor decides whether the sensor accepts the message and forwards it as a data($h - 1$) message, provided $h > 1$. To reduce the probability of message collision, any sensor u, that decides to forward 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*.

Next, we compute the lower bound of the flood period f.

Theorem 0:

$$f \geq (hmax - 1) * tmax + 1$$

Due to space limit, the proofs of the theorems in this paper are discussed in Appendix.

To analyze each of the four flood sequencing protocols, we use the following value for the flood period f.

$$f = hmax * tmax + 1$$

(We choose this value for f, instead of the minimum value $(hmax - 1) * tmax + 1$, to keep our proofs of the stabilization properties simple.)

Note that the above flood period is computed to guarantee that no two consecutive flood messages ever collide with each other. In a typical execution of the protocol, each sensor chooses its forwarding period at random in the range $1..tmax$, and so most sensors likely receive the flood messages within $(hmax - 1) * tmax/2$ time units, instead of $(hmax - 1) * tmax$ time units. Therefore, the half (or even less) of the flood period may be used without significantly degrading the stabilization property and performance of a flood sequencing protocol.

5 First Protocol: Sequencing Free

In this section, we discuss a first flood sequencing protocol where no sequence number is attached to each flood message, and so a sensor cannot distinguish between fresh and redundant flood messages, resulting that the sensor accepts every received message. This protocol is called the *sequencing free* protocol.

To initiate the flood of a new message, sensor 0 sends a data($hmax$) message, and then sets its timeout to expire after f time units to broadcast the next message. A formal specification of sensor 0 is given in Fig. 1. Note that sensor 0 does not receive any messages.

```
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
```

Figure 1. A specification of sensor 0 in the sequencing free protocol

Each sensor u that is not sensor 0 maintains a variable called new. The value of new is true only when u is in the forwarding period (i.e. u has a flood message that has been received earlier but has not been forwarded yet). When sensor u receives a data(h) message, u always accepts the message. Sensor u forwards the message as data($h - 1$), if $h > 1$ in the received message and $new = false$ in u. A formal specification of sensor u is given in Fig. 2. (Each sensor u also maintains a received data message that u will forward later, even though this is not explicitly specified in the specification.)

```
sensor $u$:1 .. $n-1$
const  hmax        : integer,    {max hop count}
       tmax        : integer     {max forwarding period}
var    $h$,$hlast$ : 1 .. hmax,  {rcvd,last hop count}
       $new$       : boolean     {true if u has msg to forward}
begin
    timeout-expires →  if $new$ →       $new$ := false;
                                        send data($hlast$)
                       [] ¬ $new$ →     skip
                       fi; timeout-after random(1,tmax)

    [] rcv data($h$) → {accept msg}
                       if h>1 ∧ ¬ $new$ →$new$ := true;
                                            $hlast$ := $h-1$
                       [] $h$≤ 1 ∨ $new$ →  skip
                       fi
end
```

Figure 2. A specification of sensor u in the sequencing free protocol

Note that in all the flood sequencing protocols presented in this paper, the value of timer.0 is at most f time units, and the value of timer.u is at most $tmax$. This is maintained by the executions of all the protocols.

A state S of the sequencing free protocol is *legitimate* iff either S is a state where the predicate (timer.0= 1) $\wedge$ (for all u, $u \neq 0$, new.u=$false$) holds or S is a state that is reachable from a state, where this predicate holds, by some execution of the protocol.

It follows from this definition that if the protocol is executed starting from a legitimate state, then every time sensor 0 initiates a new flood, previous flood messages (whether initiated by sensor 0 legitimately or other sensors illegitimately due to some fault) are no longer forwarded in the network.

The stabilization property of the sequencing free protocol can be stated by the following three theorems. Theorem 1A gives an upper bound on the convergence time of the protocol from an illegitimate state to legitimate states. Theorem 1B gives an upper bound on the number of fresh messages that can be discarded by each sensor during the convergence. Theorem 1C gives an upper bound on the number of redundant messages that can be accepted by each sensor during the convergence. (In general, the stabilization property of each of the other three protocols can be stated by three theorems: Theorem iA, Theorem iB, and Theorem iC, where i=2,3, and 4.) In proofs below, we use the notation <var>.u to denote the value of variable <var> in a sensor u.

Theorem 1A: *In the sequencing free protocol, starting from any illegitimate state, the protocol reaches a legitimate state within $2 * f$ time units, and continues to execute within legitimate states.*

Theorem 1B: *In the sequencing free protocol, starting from any illegitimate state, every sensor discards no fresh message (before the protocol converges to a legitimate state).*

Note that starting from any legitimate state, every sensor discards no fresh message, since the sensor accepts every received message.

Theorem 1C: *In the sequencing free protocol, starting from any illegitimate state, every sensor accepts at most $2 * f$ redundant messages (before the protocol* *converges to a legitimate state).*

Note that even starting from any legitiamte state, the sensor cannot distinguish between fresh and redundant flood messages. The number of redundant copies of the same message accepted by a sensor u depends on the value of $hmax$ and the network topology. In worst case, u can accept a redundant copy of the same message at each time instant during the flood period of the message. Thus, starting from any legitimate state, every sensor accepts at most f redundant copies of the same message.

6 Second Protocol: Linear Sequencing

In this section, we discuss a second flood sequencing protocol where each flood message carries a unique sequence number that is linearly increased, and so a sensor accepts a flood message that has a sequence number larger than the last sequence number accepted by the sensor. This protocol is called the *linear sequencing* protocol.

```
sensor 0                                  {base station}
const  hmax   : integer,                  {max hop count}
       f      : integer                   {flood period}
var    $slast$  : integer                   {last seq number}
begin
    timeout-expires →  {generate new msg}
                       $slast := slast + 1$;
                       send data(hmax,$slast$);
                       timeout-after f
end
```

Figure 3. A specification of sensor 0 in the linear sequencing protocol

Each flood message in this protocol is of the form data(h,s), where field h is the remaining number of hops to be made by this message, and field s is the unique sequence number of this message.

Whenever sensor 0 broadcasts a new message, sensor 0 increases the sequence number of the last message by one, and attaches the increased sequence number to the message. A formal specification of sensor 0 is given in Fig. 3.

Each sensor u that is not sensor 0 keeps track of the last sequence number accepted by u in a variable called $slast$. When sensor u receives a data(h, s) message, sensor u accepts the message if $s > slast$, and forwards the message if $h > 1$. A formal specification of sensor u is given in Fig. 4.

A state S of the linear sequencing protocol is *legitimate* iff either S is a state where the predicate
(timer.0= 1) $\wedge$
(for all u, $u \neq 0$, new.u=$false$ $\wedge$ slast.$u \leq$ slast.0)
holds or S is a state that is reachable from a state, where this predicate holds, by some execution of the protocol.

It follows from this definition that if the protocol is executed starting from a legitimate state, then every time sensor 0 initiates a new flood, previous flood messages are no longer forwarded in the network, and the new flood message has a sequence number that is larger than every slast.u in the network, so that every u accepts the message.

```
sensor $u$:1 .. $n-1$
const  hmax        : integer,        {max hop count}
       tmax        : integer         {max forwarding period}
var    $h$,$hlast$    : 1 .. hmax,      {rcvd,last hop count}
       $s, slast$    : integer,        {rcvd,last seq number}
       $new$         : boolean         {true if u has msg to forward}
begin
  timeout-expires →  if $new$ →       $new$ := false;
                                    send data($hlast, slast$)
                     [] ¬ $new$ →     skip
                     fi; timeout-after random(1,tmax)

  [] rcv data($h, s$) →  if $s > slast$ →{accept msg} $slast := s$;
                          if h>1 →   $new$ := true;
                                     $hlast := h - 1$
                          [] $h \leq 1$ →  skip
                          fi
                       [] $s \leq slast$ →  {discard msg} skip
                       fi
end
```

Figure 4. A specification of sensor u in the linear sequencing protocol

Let k be the maximum value between 1 and k', where k' is the maximum difference $slast.u - slast.0$ for any sensor u in the network at an initial state. Note that the value of k is finite but it is unbounded.

Theorem 2A: *In the linear sequencing protocol, starting from any illegitimate state, the protocol reaches a legitimate state within $(k+1)*f$ time units, and continues to execute within legitimate states.*

Theorem 2B: *In the linear sequencing protocol, starting from any illegitimate state, every sensor discards at most $(k+1)*f$ fresh messages (before the protocol converges to a legitimate state).*

Theorem 2C: *In the linear sequencing protocol, starting from any illegitimate state, every sensor accepts at most $n-1$ redundant messages (before the protocol converges to a legitimate state).*

The linear sequencing protocol requires sensors to use unbounded sequence numbers. Thus, this protocol is very expensive to implement for sensor networks that have limited resources. However, once the protocol starts its execution from any legitimate state, every sensor accepts every fresh message and discards every redundant message under any degree of message loss.

7 Third Protocol: Circular Sequencing

In this section, we discuss a third flood sequencing protocol where each flood message carries a sequence number that is circularly increased within a limited range, and so a sensor accepts a flood message that has a sequence number "logically" larger than the last sequence number accepted by the sensor. This protocol is called the *circular sequencing* protocol.

Each flood message is augmented with a sequence number that has a value in the range 0 .. $smax$, where $smax > 1$. We assume that $smax$ is an even number (to keep our presentation simple).

Whenever sensor 0 broadcasts a new message, sensor 0 increases the sequence number of the last message by one circularly within the range 0 .. $smax$, i.e. $slast$:= ($slast + 1$) mod ($smax$+1), and attaches the increased sequence number to the message.

From the viewpoint of each sequence number s in the range 0 .. $smax$, the range can be divided into two subranges, where one subrange consists of the sequence numbers that are logically "smaller" than s, and the other subrange consists of the sequence numbers that are logically "larger" than s. Thus, sequence number s has $\frac{smax}{2}$ numbers logically smaller than it and $\frac{smax}{2}$ numbers logically larger than it. For example, if $smax = 8$, number 0 is logically smaller than 1, 2, 3, and 4, and is logically larger than 5, 6, 7, and 8.

When a sensor u receives a data(h, s) message, sensor u checks if s is logically larger than $slast$. Sensor u calls the function "Larger($s, slast$)" that returns true if s is logically larger than $slast$, and otherwise returns false. Sensor u accepts the message if Larger($s, slast$) returns true, and forwards it if $h > 1$. Otherwise, sensor u discards the message.

To prove the stabilization property of the circular sequencing protocol, we make an assumption of bounded message loss as follows:

- *Bounded message loss*: Starting from any state, if sensor 0 broadcasts $\frac{smax}{2}$ consecutive flood messages, then every sensor in the network receives at least one of those flood messages.

Two explanations concerning the above assumption are in order. First, the protocol may not be self-stabilizing without any bound on message loss. For example, consider a scenario where $smax$=8. Assume that sensor 0 sends a flood message with sequence number 0 and a sensor u accepts the message. If sensor u does not receive the next 4 (i.e. $\frac{smax}{2}$) consecutive messages with sequence numbers 1, 2, 3 and 4, and later receives a fresh message with sequence number 5, it discards the message since sequence number 5 is not logically larger than sequence number 0. Sensor u also discards the next flood messages with sequence numbers 6, 7, 8, and 0, if it receives them. In this scenario, if sensor u does not receive the flood messages with sequence numbers 1, 2, 3 and 4, it keeps discarding fresh flood messages. Thus, some assumption of bounded message loss is necessary for the stabilization property of the protocol.

Second, the above assumption becomes acceptable if the value of $smax$ is reasonably large enough for a given network setting. Selecting an appropriate value for $smax$ depends on the size of the network, the topology of the network, and a flood sequencing protocol used in the network. (In Section 9, we show how different values are selected for $smax$ depending on these factors.)

A state S of the circular sequencing protocol is *legitimate* iff either S is a state where the predicate

```
(timer.0=1) $\wedge$
(for all $u$, $u \neq 0$,
        (new.$u$=false) $\wedge$
        (slast.$u$ = slast.0 $\vee$
         slast.$u$ = (slast.0$-$1) mod ($smax$+1) $\vee$
         ...
         slast.$u$ = (slast.0$-\frac{smax}{2}$+1) mod ($smax$+1)
        )
) $\wedge$
(sensor 0 has already initiated at least $\frac{smax}{2}$+2 floods)
```

holds or S is a state that is reachable from a state, where this predicate holds, by some execution of the protocol.

It follows from this definition that if the protocol is executed starting from a legitimate state, then every time sensor 0 initiates a new flood, previous flood messages are no longer forwarded in the network, and the new flood message has a sequence number that is logically larger than every slast.u in the network, so that every u accepts the message.

Theorem 3A: *In the circular sequencing protocol, starting from any illegitimate state, the protocol reaches a legitimate state within $(smax + 2) * f$ time units, and continues to execute within legitimate states.*

Theorem 3B: *In the circular sequencing protocol, starting from any illegitimate state, every sensor discards at most $(smax + 2) * f$ fresh messages (before the protocol converges to a legitimate state).*

Theorem 3C: *In the circular sequencing protocol, starting from any illegitimate state, every sensor accepts at most $f + 1$ redundant messages (before the protocol converges to a legitimate state).*

Note that starting from any legitimate state, every sensor accepts every fresh message and discards every redundant message under the assumption of bounded message loss.

8 Fourth Protocol: Differentiated Sequencing

In this section, we discuss the last flood sequencing protocol where the sequence numbers of flood messages are in a limited range, similar to the circular sequencing protocol. However, in this protocol, a sensor accepts a flood message if the sequence number of the message is different from the last sequence number accepted by the sensor. This protocol is called the *differentiated sequencing* protocol.

Each flood message is augmented with a sequence number that has a value in the range 0 .. $smax$, where $smax > 0$. We assume that $smax$ is an even number (to keep our presentation simple).

Sensor 0 in this protocol is identical to the one in the circular sequencing protocol. However, when a sensor u receives a data(h, s) message, sensor u accepts the message if s is different from $slast$, i.e. $s \neq slast$, and forwards the message if $h > 1$. Otherwise, sensor u discards the message.

Similar to the circular sequencing protocol, if a sensor does not receive a large number of consecutive flood messages, the differentiated sequencing protocol may not be self-stabilizing. Thus, the proofs of the stabilization property of this protocol are based on the assumption of bounded message loss described in Section 7.

A state S of the differentiated sequencing protocol is *legitimate* iff either S is a state where the predicate

```
(timer.0=1) $\wedge$
(for all $u$, $u \neq 0$,
        (new.$u$=false) $\wedge$
        (slast.$u$ = slast.0 $\vee$
         slast.$u$ = (slast.0$-$1) mod ($smax$+1) $\vee$
         ...
         slast.$u$ = (slast.0$-\frac{smax}{2}$+1) mod ($smax$+1)
        )
)
```

holds or S is a state that is reachable from a state, where this predicate holds, by some execution of the protocol.

It follows from this definition that if the protocol is executed starting from a legitimate state, then every time sensor 0 initiates a new flood, previous flood messages are no longer forwarded in the network, and the new flood message has a sequence number that is different from every slast.u in the network, so that every u accepts the message.

Theorem 4A: *In the differentiated sequencing protocol, starting from any illegitimate state, the protocol reaches a legitimate state within $(\frac{smax}{2} + 2) * f$ time units, and continues to execute within legitimate* *states.*

Theorem 4B: *In the differentiated sequencing protocol, starting from any illegitimate state, every sensor discards at most* $(\frac{smax}{2} + 2) * f$ *fresh messages (before the protocol converges to a legitimate state).*

Theorem 4C: *In the differentiated sequencing protocol, starting from any illegitimate state, every sensor accepts at most* $f + 1$ *redundant messages (before the protocol converges to a legitimate state).*

Note that starting from any legitimate state, every sensor accepts every fresh message and discards every redundant message under the assumption of bounded message loss.

We compare the stabilization properties of the four flood sequencing protocols in Table 1. We also compare the properties of the flood sequencing protocols after convergence (or starting from a legitimate state) in Table 2. We call these properties the *stable properties* of the protocols. In Tables 1 and 2, “free”, “lin”, “cir”, and “dif” represent the sequencing free, linear sequencing, circular sequencing, and differentiated sequencing protocols, respectively. We conclude that the differentiated sequencing protocol has better stabilization property than those of the other three protocols.

9 Simulation Results

We have developed a simulator that can simulate the execution of the four flood sequencing protocols. In this simulator, a network is an $N * N$ grid where N is the number of sensors in each side of the grid, and the distance between a sensor (i, j) and each of $(i + 1, j)$, $(i, j + 1)$, $(i - 1, j)$, and $(i, j - 1)$, if it exists, where $0 \leq i, j < N$, is 1.

For the purpose of simulation, sensor 0 is (0,0) which is located at the left-bottom conner in a grid, and the following two types of topologies that have different network density were used.

- A topology for a sparse network:
  The edge probability between two sensors is labeled with a high probability 0.95 if their distance is at most 1, and with a low probability 0.5 if their distance is larger than 1 and less than 2. Otherwise, there is no edge between the two sensors. In this topology, each sensor (i,j) that is not on or near the boundary of the grid generally has 8 neighbors.

- A topology for a dense network:
  The edge probability between two sensors is labeled with probability 0.95 if their distance is at most 1.5, and with probability 0.5 if their distance is larger than 1.5 and less than 3. Otherwise, there is no edge between the two sensors. In this topology, each sensor (i,j) that is not on or near the boundary of the grid generally has 24 neighbors.

(The used probabilities, 0.95 and 0.5, were chosen based on some experiments on sensors. We refer the reader to [3] for details.)

The performance of a flood sequencing protocol can be measured by the following two metrics:

i. *Reach:* The percentage of sensors that receive a message sent by sensor 0.

ii. *Communication:* The total number of messages forwarded by all sensors in the network.

We ran simulations of the four flood sequencing protocols, and measured the above two metrics in 10*10 and 20*20 grids for both sparse and dense network topologies. (In the figures and tables below,“free”, “lin”, “cir”, and “dif” represent the sequencing free, linear sequencing, circular sequencing, and differentiated sequencing protocols, respectively.) In our simulations, we do not consider other techniques that can improve the performance of a flood protocol based on extra information such as probability, location, and neighbor information.

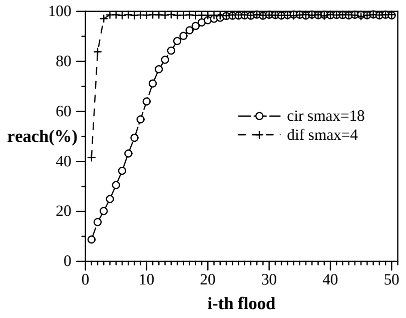
First, we studied the performance of the sequencing free protocol and the linear sequencing protocol starting from a legitimate state. The result of each simulation in this study represents the average value over the simulations of 100,000 floods. Staring from a legitimate state, the linear sequencing protocol never discards fresh messages and never accepts redundant messages under any degree of message loss, and so we consider its performance as the ideal one for flood sequencing protocols that attach a sequence number to a flood message. Note that when the value of $smax$ is reasonably large for a given network setting, the performance of the circular sequencing and differentiated

Table 1. Stabilization Properties of the Flood Sequencing Protocols

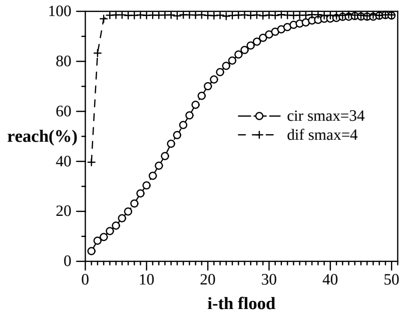
	Convergence time (time units)	Max # of fresh msgs discarded by u until convergence	Max # of redundant msg accepted by u until convergence	Stabilization property
free	$2 * f$	0	$2 * f$	good
lin	unbounded	unbounded	$n - 1$	bad
cir	$(smax + 2) * f$	$(smax + 2) * f$	$f + 1$	good
dif	$(\frac{smax}{2} + 2) * f$	$(\frac{smax}{2} + 2) * f$	$f + 1$	good

Table 2. Stable Properties of the Flood Sequencing Protocols

	Max # of fresh msgs discarded by u after convergence	Max # of redundant copies of the same msg accepted by u after convergence	Stable property
free	0	f	bad
lin	0	0	good
cir	0	0	good
dif	0	0	good



(a) A 10*10 network



(b) A 20*20 network

Figure 5. Reach of the circular and differentiated sequencing protocols starting from an illegitimate state in a sparse network

sequencing protocols becomes same as that of the linear sequencing protocol.

Tables 3 and 4 show the reach and communication of the sequencing free protocol and the linear sequencing protocol in a sparse network and in a dense network, respectively. Also the value of $hmax$ used in each network setting is specified in these tables. In these simulations, $tmax = 6$ was used for a sparse network, and $tmax = 7$ was used for a dense network. From the above results, one can observe that the sequencing free protocol requires the sensors to send much more messages than those that the linear sequencing protocol does. Specially in a sparse 20*20 network where a large value needs to be selected for $hmax$ (i.e. $hmax = 27$), the communication of the sequencing free protocol is around 7.39 times that of the linear sequencing protocol.

Next, we studied the stabilization properties of the circular sequencing and differentiated sequencing protocols, and their performance while stabilizing. We simulated the sequences of floods starting from 1000 different illegitimate states, and computed the average

Table 3. Sequencing free and linear sequencing protocols in a sparse network

	A 10*10 network			A 20*20 network		
	hmax	Reach	Comm.	hmax	Reach	Comm.
free	13	99%	351.3	27	99.2%	2885.7
lin	15	98.5%	97.8	28	98.5%	390.3

Table 4. Sequencing free and linear sequencing protocols in a dense network

	A 10*10 network			A 20*20 network		
	hmax	Reach	Comm.	hmax	Reach	Comm.
free	7	99.8%	200.5	13	99%	1262
lin	7	98.5%	87.5	14	98.8%	376.4

reach for each i-th flood. We attempted to select an appropriate value of $smax$ for each network setting such that the assumption of bounded message loss becomes acceptable, while the convergence time of each protocol is minimized.

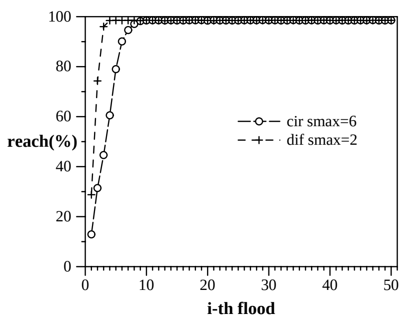
Figures 5 and 6 show the reach of the circular sequencing and differentiated sequencing protocols starting from an illegitimate state in a sparse network and in a dense network, respectively. From these results, one can observe the followings for the circular sequencing protocol. For a sparse network, a large value needs to be selected for $smax$ such as $smax = 18$ in a 10*10 network and $smax = 34$ in a 20*20 network. Also the size of a network affects selecting a value for $smax$. On the contrary, for a dense network, a relatively small value can be selected for $smax$, regardless of a network size (whether 10*10 or 20*20). During the convergence time, each sensor has a higher probability to receive a (fresh) flood message from one of its neighbors in a dense network than that in a sparse network, since the number of its neighbors in a dense network is larger than that in a sparse network. Thus, this protocol converges faster to a legitimate state in a dense network than in a sparse network.

In the differentiated sequencing protocol, a small value can be selected for $smax$, regardless of a network size (whether 10*10 or 20*20) in both sparse and dense networks. During the convergence time, each sensor has a higher probability to accept a received fresh message in the differentiated sequencing protocol (where it accepts a message if the message has a different sequence number than its last sequence number) than that in the circular sequencing protocol (where it accepts a message if the message has a logically larger sequence number than its last sequence number). Thus, this protocol generally reaches a legitimate state faster than the circular sequencing protocol does.

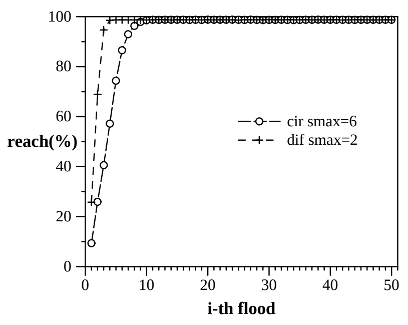
In summary, starting from a legitimate state, the performance of any flood sequencing protocol that attaches a sequence number to a flood message is better than that of the sequencing free protocol in terms of communication. Starting from an illegitimate state, the differentiated sequencing protocol converges to a legitimate state quickly in all simulated network settings. Thus, we conclude that the differentiated sequencing protocol has better stabilization property, and provide better performance compared to those of the other three protocols.

10 Concluding Remarks

In this paper, we discussed a family of the four flood sequencing protocols that use sequence numbers to distinguish between fresh flood messages and redundant flood messages. The members of our family are the sequencing free protocol, the linear sequencing protocol, the circular sequencing protocol, and the differentiated sequencing protocol. We analyzed the stabilization and stable properties of these four protocols, and also studied their performance, using simulation, over various settings of sensor networks. We concluded that the differentiated sequencing protocol has better overall performance in terms of communi-



(a) A 10*10 network



(b) A 20*20 network

Figure 6. Reach of the circular and differentiated sequencing protocols starting from an illegitimate state in a dense network

cation and stabilization and stable properties compared to those of the other three protocols.

It was mentioned that the flood period is computed to guarantee that no two consecutive flood messages ever collide with each other. Thus, in practice, the half (or even less) of the flood period may be used without significantly degrading the stabilization property and performance of a flood sequencing protocol. Moreover, each of the flood sequencing protocols can be extended to support multiple floods within one flood period.

A spanning tree can be used to distinguish fresh flood messages and redundant flood messages [1, 6]. Flood protocols using a spanning tree require extra overheads to build and maintain the spanning tree. When sensors are mobile, the spanning tree needs to keep changed. Thus, these protocols may not be suitable for some sensor networks.

References

[1] B. R. Bellur and R. G. Ogier. A Reliable, Efficient Topology Broadcast Protocol for Dynamic Networks. In *INFOCOM*, pages 178–186, 1999.

[2] D. Ganesan, B. Krishnamurthy, A. Woo, D. Culler, D. Estrin, and S. Wicker. An Empirical Study of Epidemic Algorithms in Large Scale Multihop Wireless Networks. *IRP-TR-02-003*, 2002.

[3] M. Gouda and Y. Choi. A State-based Model of Sensor Protocols. In *Proceedings of 9th International Conference on Principles of Distributed Systems (OPODIS 2005)*, December 2005.

[4] M. Heissenbttel, T. Braun, M. Waelchli, and T. Bernoulli. Optimized Stateless Broadcasting in Wireless Multi-hop Networks. In *IEEE INFOCOM*, 2006.

[5] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*, volume 353, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[6] T. Korkmaz and M. Krunz. Hybrid flooding and tree-based broadcasting for reliable and efficient link-state dissemination. In *Proceedings of the IEEE GLOBECOM 2002 Conference - High-Speed Networks Symposium*, Taiwan, November 2002.

[7] J. Li and P. Mohapatra. A Novel Mechanism for Flooding Based Route Discovery in Ad Hoc Networks. In *Proceedings of the IEEE Global Telecommunications Conference*, 2003.

[8] 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.

[9] S. Ni, Y. Tseng, Y. Chen, and J. Sheu. The Broadcast Storm 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.

[10] Y. Sasson, D. Cavin, and A. Schiper. Probabilistic broadcast for flooding in wireless mobile ad hoc networks. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2003)*, 2003.

[11] I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination based broadcasting algorithms in wireless networks. *IEEE Transac-*

tions on Parallel and Distributed Systems, 13(1):14–25, January 2002.

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

[13] 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*.

Appendix

The proof of Theorem 0:

When sensor 0 broadcasts a data($hmax$) message at time t, an out-neighbor u of sensor 0 can receive the message at t and choose the maximum possible value $tmax$ for the forwarding period. At time $t + tmax$, u forwards the message as data($hmax - 1$). Similarly, an out-neighbor u' of sensor u can receive the message at $t + tmax$ and choose $tmax$ for the forwarding period. This forwarding process continues until this message makes $hmax$ hops. Therefore, some sensor u can receive the last data(1) message at time $t + (hmax - 1) * tmax$ in the worst case. Thus, the flood period needs to be at least $(hmax-1)*tmax+1$ time units to guarantee that no forwarded messages from two consecutive floods collide with one anther. □

The proof of Theorem 1A:

(Sketch) Starting from any state, any flood initiated by some sensor u due to wrong initial values of new and $hlast$ in u will be terminated within f time units. This is because sensor u will timeout within $tmax$ time units, and the maximum lifetime of a flood message is $(hmax - 1) * tmax$ time units. After all wrongly initiated floods are terminated, $new.u$ for every sensor u always becomes false when timer.u=1. The value of $timer.0$ becomes 1 again within $2 * f$ time units. Thus, the protocol reaches a legitimate state within $2 * f$ time units, and continuously stays in legitimate states. □

The proof of Theorem 1C:

A sensor u can receive at most one message at each time instant. Thus, in the worst case, u can accept a redundant message at each time instant during the convergence time, and so the maximum number of redundant messages accepted by u until convergence is $2*f$. □

The proof of Theorem 2A:

(Sketch) There are two cases to consider. In the first case, initially $slast.0$ is larger than or equal to each $slast.u$ in the network, so that $k' < 1$ and $k = 1$. In the second case, initially $slast.0$ is less than some

$slast.u$ in the network, so that $k' \geq 1$ and $k \geq 1$. In the first case of $k' < 1$ and $k = 1$, the proof is similar to that of Theorem 1A. The protocol reaches a legitimate state within $2 * f$ time units. Consider the second case of $k' \geq 1$ and $k \geq 1$. Let s be the value of $slast.0$ at the initial state. After f time units, $new.u$ for every sensor u always becomes false when timer.0=1. When the value of timer.0 becomes 1 again, say in time unit ($t - 1$,t), within $2 * f$ time units, $slast.0$ is at least $s + 1$. Sensor 0 broadcasts a message with sequence number $s + 2$ at t. If $k' = 1$, then $s + 2$ is larger than each $slast.u$ in the network. Thus, in this case, the protocol reaches a legitimate state within $2 * f$ time units. If $k' > 1$, then sensor 0 broadcasts a message with sequence number $s+3$ at $t+f$ and so on. Finally at $t + (k - 2) * f$, sensor 0 broadcasts a message with sequence number $s + k$ and the flood of this message will be terminated by $t + (k - 1) * f$. Thus, the protocol reaches a legitimate state within $(k + 1) * f$ time units, and continuously stays in legitimate states. □

The proof of Theorem 2C:

Assume that the protocol starts from a state where $new.u$ for every sensor u is true. In this case, every sensor u can initiate a flood of the previous accepted message. Thus, sensor u can accept at most $n - 1$ redundant messages from every other sensor in the network. □

The proof of Theorem 3A:

(Sketch) After f time units, $new.u$ for every sensor u always becomes false when timer.0=1. The value of timer.0 becomes 1 again, say in $(t - 1, t)$ time unit, within $2 * f$ time units, and then sensor 0 broadcasts a flood message at t. By the assumption of bounded message loss, every sensor u is guaranteed to receive at least one (fresh) flood message by $t + \frac{smax}{2} * f$. When u receives a message, u computes whether the message is fresh or redundant based on the values of the received sequence number and $slast.u$. Because of message loss and/or wrong initial value of $slast.u$, u may compute that the received message is redundant. Assume that in $(t + \frac{smax}{2} * f - 1, t + \frac{smax}{2} * f)$, the value of $slast.0$ is s and the value of $slast.u$ for some sensor u is equal to $(s + \frac{smax}{2})$ mod ($smax$+1). At $t + (smax - 1) * f$, sensor 0 broadcasts a message with sequence number $(s+\frac{smax}{2})$ mod ($smax$+1), and this flood message will be terminated by $t + smax * f$. Therefore, the protocol reaches a legitimate state within $(smax + 2) * f$ time units, and continuously stays in legitimate states. □

The proof of Theorem 3C:

(Sketch) During the first f time units, any flood initiated by some sensor u due to wrong initial values of new and $hlast$ in u can exist in the network, and sensor u can accept at most f redundant messages. After f time units, sensor u can accept one redundant message initiated by sensor 0, and then u will not accept any redundant message any more. Thus, the maximum number of redundant messages accepted by sensor u is $f+1$. □

The proof of Theorem 4A:

(Sketch) After f time units, $new.u$ for every sensor u always becomes false when timer.0=1. The value of timer.0 becomes 1 again, say in time unit ($t - 1$,t), within $2 * f$ time units. Assume that sensor 0 broadcasts a new message with sequence number s at t. Then, sensor 0 broadcasts a new message with sequence number $(s + \frac{smax}{2} - 1)$ mod ($smax$+1) at $t + (\frac{smax}{2} - 1) * f$. In time unit ($t + \frac{smax}{2} * f - 1$,$t + \frac{smax}{2} * f$), every $slast.u$ has one of the values in s .. $(s+\frac{smax}{2}-1)$ mod ($smax$+1), since u receives at least one of those sequence numbers by the assumption of bounded message loss. Thus, the protocol reaches a legitimate state within $(\frac{smax}{2} + 2) * f$ time units, and continuously stays in legitimate states. □