electronics

MDPI

Article

A Simple and Efficient Tree-Based Algorithm for the Distributed Trigger Counting Problem

Jaeheung Lee [1] and Yongsu Park [2,*]

[1] Department of Computer and Information Security, Daejeon University, Daejeon 34520, Korea; leejh@dju.kr
[2] Department of Computer Science, Hanyang University, Seoul 04763, Korea
* Correspondence: yongsu@hanyang.ac.kr; Tel.: +82-2-2220-2382

Abstract: The distributed trigger counting (DTC) problem is defined as raising an alarm and notifying a user when the total number of received triggers reaches a predefined value w in a distributed system of n nodes. DTC algorithms can be used for environmental surveillance with sensor networks and global snapshots. In this paper, we propose a simple and efficient algorithm for the DTC problem. The proposed algorithm is based on a tree structure of degree $\sqrt{n}$ and height 2. The proposed algorithm operates in three phases depending on the remaining number of triggers. We prove the correctness of the proposed algorithm: the probability of not notifying a user even though the total number of received triggers reaches w is 0. Experimental results show that the proposed algorithm has lower message complexity than the best previous algorithms: *CoinRand* and *TreeFill*. MaxRcv (the maximum number of received messages per node) of the proposed algorithm is also smaller than *CoinRand* and *TreeFill* when the number of nodes is not very large.

Keywords: distributed trigger counting; distributed system; distributed monitoring; global snapshot

Citation: Lee, J.; Park, Y. A Simple and Efficient Tree-Based Algorithm for the Distributed Trigger Counting Problem. *Electronics* **2022**, *11*, 1127. https://doi.org/10.3390/electronics11071127

Academic Editor: George Angelos Papadopoulos

Received: 24 January 2022
Accepted: 20 March 2022
Published: 2 April 2022

1. Introduction

Consider a large-scale networked system in which the participating nodes are counting external triggers. The distributed trigger counting (DTC) problem is defined as raising an alarm and notifying a user when the total number of received triggers reaches a predefined value w in a distributed system of n nodes [1–4]. We assume that no statistical data on the triggers (e.g., the sequence of nodes receiving the triggers and the number of triggers received by each node) are given to the system ahead of time. Only the case where the number of triggers is significantly greater than the number of nodes, i.e., $w \gg n$, is taken into account. Otherwise, the DTC problem can be solved with $O(n)$ messages [2,5].

DTC algorithms can be utilized for distributed monitoring [6–8] and global snapshots [9–12]. Monitoring is essential to manage distributed networks such as sensor networks [13]. Sensor networks monitor environmental or physical status such as traffic volume, wild animal behavior, troop movement, and atmospheric conditions. For example, in traffic management, an alarm can be raised when the number of vehicles on the road surpasses a certain threshold. When observing wild animal behavior, an alarm can be raised when the number of a specific species in a specific area surpasses a certain threshold. For a data network, you can also monitor the amount of traffic or the number of remote logins to detect DDoS (distributed denial-of-service) attacks. To declare that a global snapshot of a distributed system is valid, all messages in transit must be recorded. Garg et al. [14] proved that the DTC problem can be used to solve the problem of deciding whether all messages in transit have been received.

Message complexity and MaxRcv are major performance metrics for DTC algorithms [2]. Message complexity indicates the total number of messages sent and received by all nodes, and MaxRcv means the maximum number of received messages per node. While message complexity evaluates the performance of the overall algorithm, MaxRcv represents the overload of a specific node due to the algorithm.

In this paper, we propose a simple and efficient algorithm for the DTC problem. The proposed algorithm is based on a tree structure of degree $\sqrt{n}$ and height 2. The proposed algorithm operates in three phases depending on the remaining number of triggers. We prove the correctness of the proposed algorithm; namely, we prove that the probability of not notifying a user when the total number of triggers received from a distributed system reaches a predefined value w is 0. Experimental results show that the proposed algorithm has lower message complexity than *CoinRand* [2] and *TreeFill* [3]. MaxRcv of the proposed algorithm is also smaller than *CoinRand* and *TreeFill* when the number of nodes is not very large.

The rest of the paper is organized as follows. In Section 2, we summarize the related works on DTC algorithms. The proposed algorithm is described in Section 3 and the failure probability of the proposed algorithm is analyzed in Section 4. In Section 5, the experimental results are discussed. Finally, we conclude this paper in Section 6.

2. Related Works

For global snapshots, DTC algorithms can be utilized as a primitive operation [14]. By employing an efficient DTC algorithm, the message complexity for storing global snapshots can be significantly lowered when compared to existing global snapshot algorithms [7,13,15–17]. The message complexity of recording channel states in existing global snapshot algorithms is typically $O(n^2)$ [12]. By using an efficient DTC algorithm, the cost of recording channel states can be reduced to $O(n \log(w/n))$ [14].

Garg et al. proposed three DTC algorithms: one grid-based, one tree-based, and one centralized algorithm. They also proved that the lower bound of message complexity for generic DTC algorithms is $O(n \log(w/n))$ [14]. The centralized algorithm shows an optimal message complexity, but the MaxRcv of it is not bounded.

Chakaravarthy et al. proposed an almost optimal DTC algorithm called *LayeredRand* [1]. The message complexity of *LayeredRand* is $O(n \log n \log w)$ with high probability, and its MaxRcv is $O(\log n \log w)$. Chakaravarthy et al. also proposed two DTC algorithms, *CoinRand* and *RingRand*, which can be regarded as improvements of *LayeredRand* [2]. The message complexity of *CoinRand* is $O(n(\log w + \log n))$ with high probability, and its MaxRcv is $O(\log w + \log n)$. It is based on a network topology similar to a binary tree. By using a randomized approach in the message aggregation process, *CoinRand* outperforms *LayeredRand*. The message complexity of *RingRand* is $O(n \log n \log w)$, and its MaxRcv is $O(\log n \log w)$ with high probability.

Kim et al. proposed *TreeFill*, an optimal DTC algorithm [3]. The message complexity and MaxRcv of *TreeFill* are $O(n \log(w/n))$ and $O(\log(w/n))$, respectively. *TreeFill* is also based on a tree-like network topology.

Emek and Korman proposed two DTC algorithms called *CompTreeRand* and *CompTreeDet* with more general assumptions about node-to-node communication [18]. The algorithms are built on a tree network with nodes which are only able to communicate with their immediate neighbors. The message complexity of *CompTreeRand* is $O(n \log w (\log \log n)^2)$; however, MaxRcv of *CompTreeRand* is not investigated. The message complexity and MaxRcv of *CompTreeDet* are, respectively, $O(n(\log w \log n)^2)$ and $O((\log w \log n)^2)$.

For global snapshots, Kshemkalyani suggested a *hypercube-based* algorithm [12]. The message complexity of the *hypercube-based* algorithm is $O(n \log n)$, which is lower than the optimal message complexity for DTC problems of $O(n \log(w/n))$. The *hypercube-based* algorithm, on the other hand, has a message size of $O(n)$, whereas DTC algorithms have a message size of $O(1)$.

Tsai used the general grid interconnection network, which is an extension of the *hypercube-based* network to prove the lower bound of message complexity for global snapshot algorithms [19].

Chang et al. proposed a DTC algorithm without any assumption about the network topology [5]. The algorithm they propose primarily focuses on sensor networks whose network topology is unknown ahead of time. In the worst case scenario, their algorithm

solves the DTC problem using $x(n\lceil \log \frac{w-n}{n^2-n} / \log \frac{n}{n-1} \rceil + n^2 - 1)$ messages, where x is double the number of edges in the network.

Recently, Kim et al. proposed *DDR-coin*, an efficient probabilistic DTC algorithm [4]. It is a Monte Carlo algorithm in the sense that the system may fail to raise the alarm when it receives w triggers. Compared with the previous work (*CoinRand*, *RingRand* and *TreeFill*), it shows a smaller message complexity and MaxRcv. However, for a small n, *DDR-coin* shows larger a message complexity and MaxRcv. Furthermore, it has a limitation in that a mathematical analysis of message complexity and MacRcv is incomplete.

Table 1 compares major performance metrics for DTC algorithms.

Table 1. Comparison of major performance metrics for DTC algorithms.

Algorithm	Message Complexity	MaxRcv	Exact or Probabilistic
Centralized [14]	$O(n \log(w/n))$	-	Exact
Tree-based [14]	$O(n \log n \log(w/n))$	$O(\log n \log(w/n))$	Exact
LayeredRand [1]	$O(n \log n \log w)$	$O(\log n \log w)$	Exact
CompTreeRand [18]	$O(n \log w (\log \log n)^2)$	-	Probabilistic
CompTreeDet [18]	$O(n(\log w \log n)^2)$	$O((\log w \log n)^2)$	Exact
CoinRand [2]	$O(n(\log w + \log n))$	$O(\log w + \log n)$	Exact
RingRand [2]	$O(n \log n \log w)$	$O(\log n \log w)$	Probabilistic
TreeFill [3]	$O(n \log(w/n))$	$O(\log(w/n))$	Exact
DDR-coin [4]	$O(n \log_n(w/n))$	$O(\log_n(w/n))$	Probabilistic

3. Proposed Algorithm

In this section, we present a simple and efficient tree-based algorithm for the distributed trigger counting problem. It is an exact algorithm in that it has no false positives and no false negatives. To make the problem easier to understand, we assume all nodes are fully linked and that there are no message losses, no node failures, and no external attackers. Events are triggered by arbitrary distribution on the nodes in the system. We want to detect when w or more triggers occur in the system and raise an alarm.

The proposed algorithm works as follows. For ease of explanation, we assume the number of nodes $n = k^2$ for some positive integer k. The n nodes are arranged in three layers: layer 0, 1, and 2. Layer 0 consists of a single node, and layer 1 consists of k nodes. As in *CoinRand*, all the n nodes are arranged in layer 2. Among the n nodes, one node is arranged in layer 0 and other k nodes are arranged in layer 1. The $(k+1)$ nodes which play dual roles in the proposed algorithm are selected in a round-robin fashion. We assume that all nodes in the system know the layering information.

Figure 1 shows the hierarchical structure of the proposed algorithm when $k = 4$. In the first round, node 0 is assigned to layer 0 and nodes 1, 2, 3, and 4 are assigned to layer 1 as in Figure 1. In the second round, node 5 is assigned to layer 0 and nodes 6, 7, 8, and 9 are assigned to layer 1.

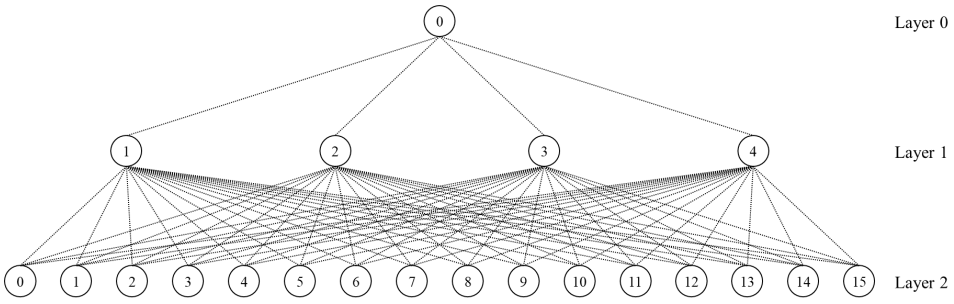


Figure 1. The hierarchical structure of the proposed algorithm when $k = 4$.

The algorithm works in three phases depending on the number of triggers that has not yet been detected, $\hat{w}$ ($\leq w$). The first phase is when $\hat{w} \geq 2n$, the second phase is when $2k \leq \hat{w} < 2n$, and the third phase is when $\hat{w} < 2k$.

3.1. The First Phase ($\hat{w} \geq 2n$)

In the first phase, the proposed algorithm works on a round basis. At the start of each round, the system must know how many triggers have not yet been detected. The system counts it by keeping a counter for each node that stores the number of triggers the node has received in each round. A variable $\hat{w}$ is used to store the initial value for each round. We set $\hat{w} = w$ in the first round because all the w triggers have not yet been detected.

Hereafter, we describe the behavior of specific rounds. Each node calculates a leaf node threshold value $\tau_{leaf} = \lfloor \hat{w}/2n \rfloor$. Each node also keeps a counter variable $C(x)$ to indicate the number of triggers received by the node x in each round. Each time the node x receives a trigger from external sources, it increments $C(x)$ by 1. When the counter variable $C(x)$ reaches the leaf node threshold τ_{leaf}, the node x decreases $C(x)$ by τ_{leaf} and chooses a node y uniformly assigned to layer 1 at random and sends a coin message to y. Each node assigned to layer 1 maintains another counter variable $D(y)$ to indicate the number of coin messages received by the node y in each round. Upon receiving a coin message, the node y increments $D(y)$ by 1. When the counter variable $D(y)$ reaches the internal node threshold $\tau_{internal} = \lfloor k/2 \rfloor$, the node y decreases $D(y)$ by $\tau_{internal}$ and sends a coin message to the root node assigned to layer 0. Finally, when the number of coin messages that the root node receives reaches k, the root node computes the total number of triggers received by all the nodes in this round and updates $\hat{w}$. It is possible via a simple broadcast and upcast procedure in a pre-determined binary tree as in *CoinRand*. The aggregation notification is broadcast to all the nodes in a recursive top–down fashion. Similarly, aggregation values are computed from the leaf nodes to the root node in a recursive bottom–up fashion. After that, the root node updates $\hat{w}$ and broadcasts it to all the nodes, again recursively. Upon receiving the updated $\hat{w}$, all the nodes update τ_{leaf} values for the next round. If the newly computed $\hat{w}$ is less than $2n$, the algorithm enters the second phase. Algorithm 1 shows the first phase of the proposed algorithm.

3.2. The Second Phase ($2k \leq \hat{w} < 2n$)

The proposed algorithm also works on a round basis in the second phase. A variable $\hat{w}$ is also used to store the initial value for each round. Whenever every node receives a trigger from external sources, it chooses a node y uniformly assigned to layer 1 at random and sends a coin message to y (i.e., $\tau_{leaf} = 1$). Recall that each node assigned to layer 1 maintains a counter variable $D(y)$ to indicate the number of coin messages received by the node y in the current round. Upon receiving a coin message, the node y increments $D(y)$ by 1. When the counter variable $D(y)$ reaches another internal node threshold $\tau_{internal'} = \lfloor \hat{w}/2k \rfloor$, the node y decreases $D(y)$ by $\tau_{internal'}$ and sends a coin message to the root node assigned to layer 0. Finally, when the number of coin messages that the root node receives reaches k, the root node computes the total number of triggers received by all the nodes in this round and updates $\hat{w}$. If the newly computed $\hat{w}$ is less than $2k$, the algorithm enters the third phase. Algorithm 2 shows the second phase of the proposed algorithm.

Algorithm 1: The first phase of the proposed algorithm

When ith round begins:
  if $i = 1$ **then**
    $\hat{w} \leftarrow w$
  end if
  if $\hat{w} < 2n$ **then**
    Go to the second phase
  end if
  $\tau_{leaf} \leftarrow \lfloor \hat{w}/2n \rfloor$
  $\tau_{internal} \leftarrow \lfloor k/2 \rfloor$
  $C(x) \leftarrow 0$ for all node x
  $D(y) \leftarrow 0$ for all node y

When the node x receives a trigger:
  $C(x) \leftarrow C(x) + 1$
  if $C(x) = \tau_{leaf}$ **then**
    $C(x) \leftarrow C(x) - \tau_{leaf}$
    Choose a node y assigned to layer 1 uniformly at random
    Send a coin message to y
  end if

When the node y assigned to layer 1 receives a coin message:
  $D(y) \leftarrow D(y) + 1$
  if $D(y) = \tau_{internal}$ **then**
    $D(y) \leftarrow D(y) - \tau_{internal}$
    Send a coin message to the root node assigned to layer 0
  end if

When the root node z receives a coin message:
  $D(z) \leftarrow D(z) + 1$
  if $D(z) = k$ **then**
    Compute the total number of triggers received by all the nodes in this round and updates $\hat{w}$ (via a simple broadcast and upcast procedure)
    Go to the next round ($i \leftarrow i + 1$)
  end if

Algorithm 2: The second phase of the proposed algorithm

When ith round begins:
  if $\hat{w} < 2k$ **then**
    Go to the third phase
  end if
  $\tau_{internal'} \leftarrow \lfloor \hat{w}/2k \rfloor$
  $D(y) \leftarrow 0$ for all node y

When the node x receives a trigger:
  Choose a node y assigned to layer 1 uniformly at random
  Send a coin message to y

When the node y assigned to layer 1 receives a coin message:
  $D(y) \leftarrow D(y) + 1$
  if $D(y) = \tau_{internal'}$ **then**
    $D(y) \leftarrow D(y) - \tau_{internal'}$
    Send a coin message to the root node assigned to layer 0
  end if

When the root node z receives a coin message:
  $D(z) \leftarrow D(z) + 1$
  if $D(z) = k$ **then**
    Compute the total number of triggers received by all the nodes in this round and updates $\hat{w}$ (via a simple broadcast and upcast procedure)
    Go to the next round ($i \leftarrow i + 1$)
  end if

3.3. The Third Phase ($\hat{w} < 2k$)

In the third phase, all the nodes in the system send the received trigger information directly to the root node, and the root node updates $\hat{w}$ accordingly. When the newly updated $\hat{w}$ becomes zero, the root node raises an alarm and finishes the algorithm. Algorithm 3 shows the third phase of the proposed algorithm.

Algorithm 3: The third phase of the proposed algorithm

When the node x receives a trigger:
  Send a coin message to the root node assigned to layer 0

When the root node z receives a coin message:
  $\hat{w} \leftarrow \hat{w} - 1$
  if $\hat{w} = 0$ **then**
    Raise an alarm and finish the algorithm
  end if

4. Analysis

Theorems 1 and 2 show that in the first phase of the proposed algorithm, the probability of not notifying a user even though the total number of received triggers is greater than or equal to w is 0.

Theorem 1. *In the first phase of the proposed algorithm (where $\hat{w} \geq 2n$), each node sends a coin message to a node uniformly chosen at random among the k nodes assigned to layer 1 whenever it receives $\tau_{leaf} = \lfloor \hat{w}/2n \rfloor$ triggers from external sources. In this case, regardless of the trigger distribution in each round, at least n coin messages are always sent to the nodes assigned to layer 1 before a total of $\hat{w}$ triggers occur.*

Proof. Let us calculate the maximum number of triggers that can occur when n coin messages are sent to the nodes assigned to layer 1. The maximum number of triggers that each node can receive without generating a coin message is $\tau_{leaf} - 1 = \lfloor \hat{w}/2n \rfloor - 1$. Therefore, the maximum number of triggers that can occur when n coin messages are sent is less than $\hat{w}$ according to Equation (1):

$$\lfloor \hat{w}/2n \rfloor \times n + (\lfloor \hat{w}/2n \rfloor - 1) \times n < \hat{w} \tag{1}$$

□

Theorem 2. *In the first phase of the proposed algorithm (where $\hat{w} \geq 2n$), each k node assigned to layer 1 sends a coin message to the root node assigned to layer 0 whenever it receives $\tau_{internal} = \lfloor k/2 \rfloor$ coin messages. In this case, regardless of the trigger distribution in each round, at least k coin messages are always sent to the root node assigned to layer 0 before a total of $\hat{w}$ triggers occur.*

Proof. According to the Theorem 1, k nodes assigned to layer 1 always receive at least n coin messages before a total of $\hat{w}$ triggers occur. Let us calculate the maximum number of coin messages received by the nodes assigned to layer 1 when k coin messages are sent to the root node. The maximum number of coin messages that each node assigned to layer 1 can receive without generating a coin message is $\lfloor k/2 \rfloor - 1$. Therefore, when k coin messages are sent to the root node, the maximum number of coin messages sent to the nodes assigned to layer 1 is less than n according to Equation (2):

$$\lfloor k/2 \rfloor \times k + (\lfloor k/2 \rfloor - 1) \times k < k^2 = n \tag{2}$$

□

Theorem 3 shows that in the second phase of the proposed algorithm, the probability of not notifying a user even though the total number of received triggers is greater than or equal to w is 0.

Theorem 3. *In the second phase of the proposed algorithm (where $2k \leq \hat{w} < 2n$), each node sends a coin message to a node uniformly chosen at random among the k nodes assigned to layer 1 whenever it receives a trigger from external sources. In addition, each k node assigned to layer 1 sends a coin message to the root node assigned to layer 0 whenever it receives $\tau_{internal'} = \lfloor \hat{w}/2k \rfloor$ coin messages. In this case, regardless of the trigger distribution in each round, at least k coin messages are always sent to the root node assigned to layer 0 before a total of $\hat{w}$ triggers occur.*

Proof. Since each node sends a coin message to a node assigned to layer 1 whenever it receives a trigger, when a total of $\hat{w}$ triggers occur, the k nodes assigned to layer 1 also receive $\hat{w}$ coin messages. Let us calculate the maximum number of coin messages received by the nodes assigned to layer 1 when k coin messages are sent to the root node. The maximum number of coin messages that each node assigned to layer 1 can receive without generating a coin message is $\lfloor \hat{w}/2k \rfloor - 1$. Therefore, when k coin messages are sent to the root node, the maximum number of coin messages sent to the nodes assigned to layer 1 is less than $\hat{w}$ according to Equation (3):

$$\lfloor \hat{w}/2k \rfloor \times k + (\lfloor \hat{w}/2k \rfloor - 1) \times k < \hat{w} \tag{3}$$

□

From Theorems 1–3, we can obtain the following result.

Theorem 4. *In the proposed algorithm, the probability of not notifying a user even though the total number of received triggers is greater than or equal to w is 0.*

5. Experimental Results

In this section, we compare the simulation results of the proposed algorithm with those of previous works. Among the previous exact DTC algorithms, *CoinRand* [2] and *TreeFill* [3], which show the best performance in terms of message complexity and MaxRcv are chosen for comparison. The source code for the simulation is available online [20]. The simulation code is written in Python. It is assumed that triggers are received uniformly at random among all the nodes in the system. Table 2 shows the parameters for the simulation. For each number of triggers and nodes, the simulation is repeated 10 times and the average is used for comparison.

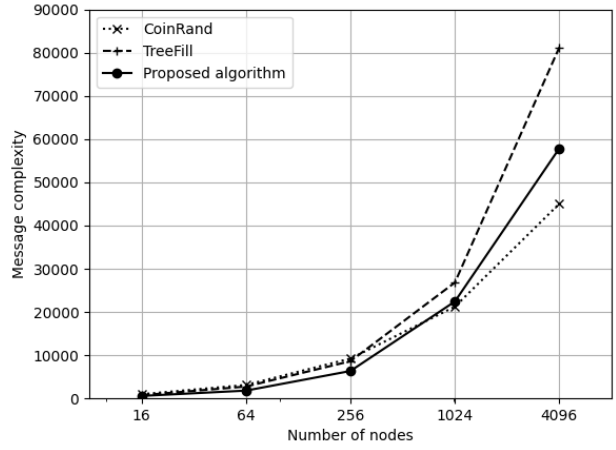
Table 2. The parameters for the simulation.

Parameter	Symbol	Values
The number of nodes	n	$16(2^4)$, $64(2^6)$, $256(2^8)$, $1024(2^{10})$, $4096(2^{12})$
The number of triggers to be detected	w	$10{,}000(10^4)$, $100{,}000(10^5)$, $1{,}000{,}000(10^6)$

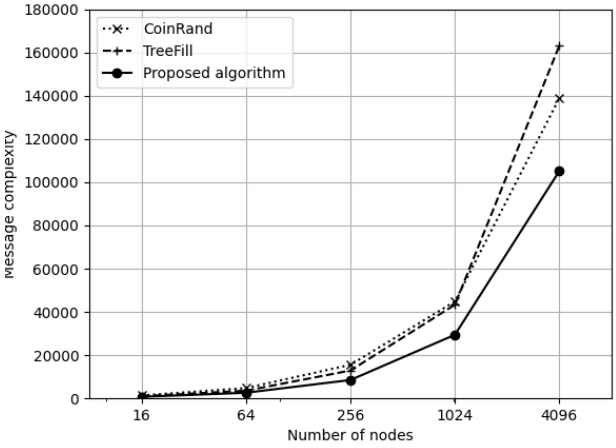
Figure 2 shows the message complexity of *CoinRand*, *TreeFill*, and the proposed algorithm. As shown in the figure, the proposed algorithm has more efficient message complexity than *CoinRand* and *TreeFill*, except when w = 10,000 and $n = 1024, 4096$. Since the DTC problem can be easily solved when the number of triggers is not significantly larger than the number of nodes, the parameters w = 10,000 and $n = 1024, 4096$ need not be taken seriously. When the number of triggers is $1{,}000{,}000(10^6)$ and the number of nodes is $16(2^4)$, $64(2^6)$, $256(2^8)$, $1024(2^{10})$, and $4096(2^{12})$, the message complexity of *CoinRand* is 2.04 times, 2.16 times, 1.98 times, 1.85 times, and 1.60 times larger than that of the proposed algorithm. Furthermore, the message complexity of *TreeFill* is 1.43 times, 1.67 times, 1.56 times, 1.54 times, and 1.48 times larger than that of the proposed algorithm.

Figure 3 shows MaxRcv of *CoinRand*, *TreeFill*, and the proposed algorithm. As shown in the figure, the proposed algorithm has more efficient MaxRcv than *CoinRand* and *TreeFill* when the number of nodes is 16, 64, and 256. When the number of nodes is 4096, *CoinRand* is more efficient than the proposed algorithm. This is because, as the number of nodes increases, the degree of the root node and the nodes assigned to layer 1 increases as well (by $\sqrt{n}$), which increases the MaxRcv of the proposed algorithm. When the number of triggers is $1{,}000{,}000(10^6)$ and the number of nodes is $16(2^4)$, $64(2^6)$, $256(2^8)$, $1024(2^{10})$, and $4096(2^{12})$, MaxRcv of *CoinRand* is 1.97 times, 1.88 times, 1.68 times, 1.21 times, and 0.71 times larger than that of the proposed algorithm. Furthermore, MaxRcv of *TreeFill* is 1.44 times, 1.77 times, 2.17 times, 2.62 times, and 2.69 times larger than that of the proposed algorithm.

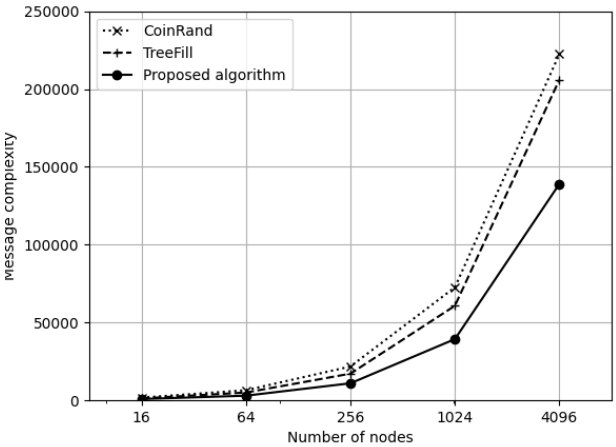
Figure 4 shows the number of rounds of *CoinRand*, *TreeFill*, and the proposed algorithm. The number of rounds of the proposed algorithm is significantly smaller than that of *CoinRand*, regardless of the number of nodes and triggers. When the number of triggers is $1{,}000{,}000(10^6)$ and the number of nodes is $16(2^4)$, $64(2^6)$, $256(2^8)$, $1024(2^{10})$, and $4096(2^{12})$, the number of rounds of *CoinRand* is 1.83 times, 1.93 times, 1.83 times, 1.77 times, and 1.74 times larger than that of the proposed algorithm. However, the number of rounds of the proposed algorithm is significantly larger than that of *TreeFill*, regardless of the number of nodes and triggers. When the number of triggers is $1{,}000{,}000(10^6)$ and the number of nodes is $16(2^4)$, $64(2^6)$, $256(2^8)$, $1024(2^{10})$, and $4096(2^{12})$, the number of rounds of *TreeFill* is 0.73 times, 0.74 times, 0.63 times, 0.59 times, and 0.54 times larger than that of the proposed algorithm.



(**a**) w = 10,000

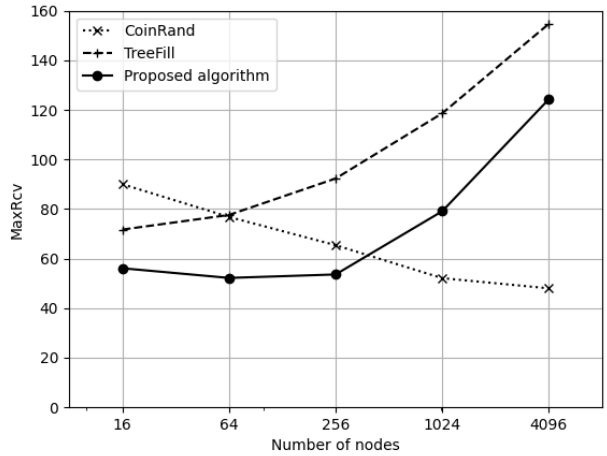


(**b**) w = 100,000

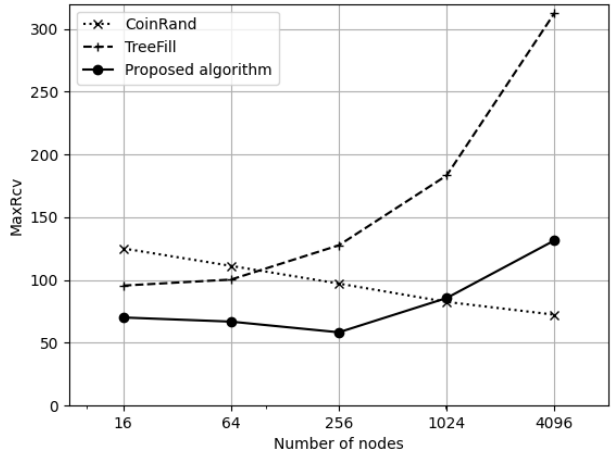


(**c**) w = 1,000,000

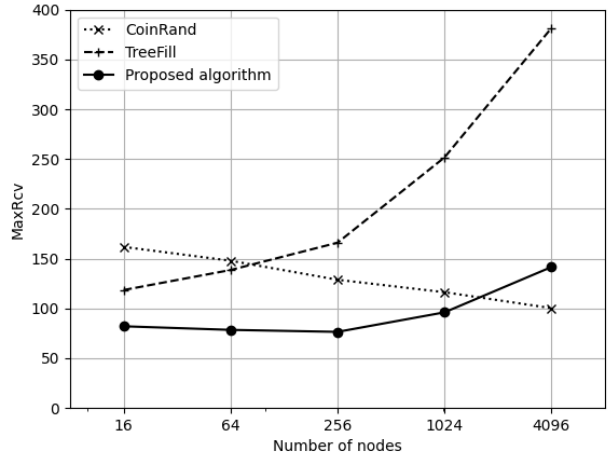
Figure 2. Comparison of message complexity between *CoinRand*, *TreeFill*, and the proposed algorithm when the number of nodes is 16, 64, 256, 1024, and 4096.



(**a**) w = 10,000

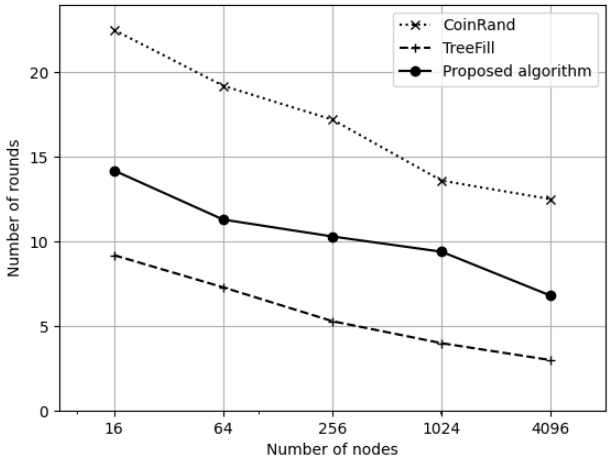


(**b**) w = 100,000

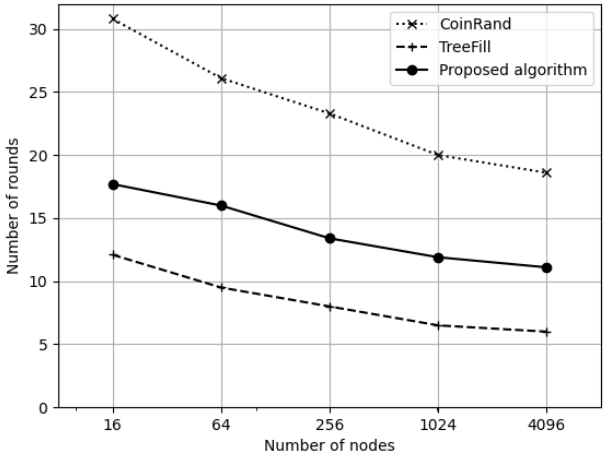


(**c**) w = 1,000,000

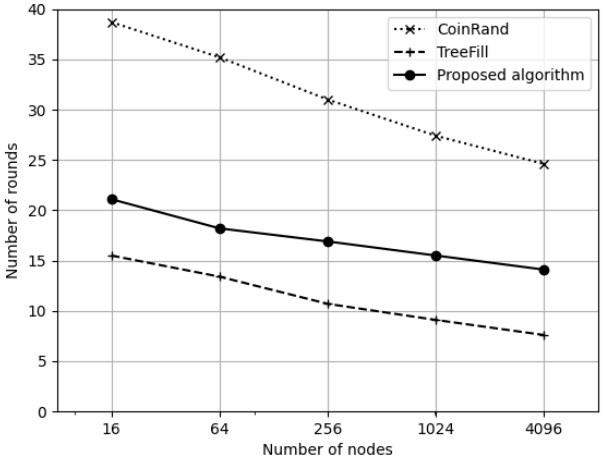
Figure 3. Comparison of MaxRcv between *CoinRand*, *TreeFill*, and the proposed algorithm when the number of nodes is 16, 64, 256, 1024, and 4096.



(**a**) w = 10,000



(**b**) w = 100,000



(**c**) w = 1,000,000

Figure 4. Comparison of the number of rounds between *CoinRand*, *TreeFill*, and the proposed algorithm when the number of nodes is 16, 64, 256, 1024, and 4096.

6. Conclusions

In this paper, we proposed a simple and efficient algorithm for the DTC problem. The proposed algorithm is based on a tree structure of degree $\sqrt{n}$ and height 2. This algorithm operates in three phases depending on the remaining number of triggers. We proved that

the probability of not notifying a user when the total number of triggers received from a distributed system reaches a predefined value w is 0. Experimental results show that the proposed algorithm has lower message complexity than *CoinRand* and *TreeFill*. The MaxRcv of the proposed algorithm is also smaller than *CoinRand* and *TreeFill* when the number of nodes is not very large.

As future work, we plan to evaluate the performance of the proposed algorithm through various additional analyses and experiments. We also plan to improve the proposed algorithm by increasing the height of the tree and reducing the node degree. By reducing the node degree, we expect to mitigate a rapid increase in MaxRcv as the number of nodes increases.

Author Contributions: Conceptualization, J.L.; methodology, J.L.; validation, J.L. and Y.P.; investigation, J.L.; writing—original draft preparation, J.L.; writing—review and editing, J.L. and Y.P.; funding acquisition, Y.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT), grant number 2020R1F1A1048443. This research was also supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2017R1C1B5076925).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chakaravarthy, V.T.; Choudhury, A.R.; Sabharwal, Y.; Garg, V. An Efficient Decentralized Algorithm for the Distributed Trigger Counting Problem. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6522, pp. 53–64.
2. Chakaravarthy, V.T.; Choudhury, A.R.; Sabharwal, Y. Improved algorithms for the distributed trigger counting problem. In Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium, Anchorage, AK, USA, 16–20 May 2011; pp. 515–523.
3. Kim, S.; Lee, J.; Park, Y.; Cho, Y. An optimal distributed trigger counting algorithm for large-scale networked systems. *Simulation* **2013**, *89*, 846–859. [CrossRef]
4. Kim, S.; Park, Y. DDR-coin: An Efficient Probabilistic Distributed Trigger Counting Algorithm. *Sensors* **2020**, *20*, 6446. [CrossRef]
5. Chang, C.C.; Tsai, J. Distributed trigger counting algorithms for arbitrary network topology. *Wirel. Commun. Mob. Comput.* **2016**, *16*, 2463–2476. [CrossRef]
6. Hsin, C.; Liu, M. A distributed monitoring mechanism for wireless sensor networks. In Proceedings of the 1st ACM Workshop on Wireless Security (WiSE '02), Atlanta, GA, USA, 28 September 2002; Association for Computing Machinery: New York, NY, USA, 2002; pp. 57–66. [CrossRef]
7. Changlei, L.; Guohong, C. Distributed monitoring and aggregation in wireless sensor networks. In Proceedings of the INFOCOM, 2010 Proceedings IEEE, San Diego, CA, USA, 14–19 March 2010; pp. 1–9.
8. Kshemkalyani, A.D.; Raynal, M.; Singhal, M. An introduction to snapshot algorithms in distributed computing. *Distrib. Syst. Eng.* **1995**, *2*, 224–233. [CrossRef]
9. Chandy, K.M.; Lamport, L. Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.* **1985**, *3*, 63–75. [CrossRef]
10. Lai, T.H.; Yang, T.H. On distributed snapshots. *Inf. Process. Lett.* **1987**, *25*, 153–158. [CrossRef]
11. Mattern, F. Efficient algorithms for distributed snapshots and global virtual time approximation. *J. Parallel Distrib. Comput.* **1993**, *18*, 423–434. [CrossRef]
12. Kshemkalyani, A.D. Fast and message-efficient global snapshot algorithms for large-scale distributed systems. *IEEE Trans. Parallel Distrib. Syst.* **2010**, *21*, 1281–1289. [CrossRef]
13. Chitnis, L.; Dobra, A.; Ranka, S. Aggregation methods for large-scale sensor networks. *ACM Trans. Sens. Netw.* **2008**, *4*, 1–36. [CrossRef]
14. Garg, R.; Garg, V.K.; Sabharwal, Y. Efficient algorithms for global snapshots in large distributed systems. *IEEE Trans. Parallel Distrib. Syst.* **2010**, *21*, 620–630. [CrossRef]
15. Massie, M.L.; Chun, B.N.; Culler, D.E. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Comput.* **2004**, *30*, 817–840. [CrossRef]
16. Park, K.; Pai, V.S. CoMon: A mostly-scalable monitoring system for PlanetLab. *ACM SIGOPS Oper. Syst. Rev.* **2006**, *40*, 65–74. [CrossRef]
17. Wensheng, Z.; Guohong, C. DCTC: Dynamic convoy tree-based collaboration for target tracking in sensor networks. *IEEE Trans. Wirel. Commun.* **2004**, *3*, 1689–1701. [CrossRef]

18. Emek, Y.; Korman, A. Efficient threshold detection in a distributed environment: Extended abstract. In Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC '10, Zurich, Switzerland, 25–28 July 2010; ACM: New York, NY, USA, 2010; pp. 183–191.
19. Tsai, J. Flexible symmetrical global-snapshot algorithms for large-scale distributed systems. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *24*, 493–505. [CrossRef]
20. Distributed Trigger Counting. 2021. Available online: https://github.com/leejh257/DTC-2021 (accessed on 5 January 2022).