On the Distributed Trigger Counting Problem for Dynamic Networks

Che-Cheng Chang[1], Jichiang Tsai[2], Tien-Yu Chang[3]

[1] Department of Information Engineering and Computer Science, Feng Chia University, Taiwan
[2] Department of Electrical Engineering & Graduate Institute of Communication Engineering, National Chung Hsing University, Taiwan
[3] Department of Electrical Engineering, National Chung Hsing University, Taiwan
checchang@fcu.edu.tw, jichiangt@nchu.edu.tw, eltonchang166@gmail.com

Abstract

The Distributed Trigger Counting (DTC) problem is a fundamental block for many distributed applications. Such a problem is to raise an alert while the whole system receives a pre-defined number of triggers. There have been several algorithms proposed to solve the DTC problem in the literature. However, these existing algorithms are all under the assumption that there is no event regarding process moving, leaving and joining in the network. In other words, they can be only applicable to static networks. The foregoing assumption is not practical for dynamic networks with continually changing topology. In this paper, we investigate the DTC problem for dynamic networks and introduce a distributed algorithm without any global assumption. Moreover, to reduce the message complexity of the above algorithm, we further propose a more message-efficient version, only with one additional requirement that all processes have learned ahead the upper bound on number of processes involved in the computation.

Keywords: Distributed trigger counting, Distributed algorithms, Dynamic networks

1 Introduction

The Distributed Trigger Counting (DTC) problem is a basic block for many distributed applications, such as monitoring [1-11], global snapshots [12-15], synchronizers [14-15] and so on. The underlying system will raise an alert upon receiving a certain number of triggers corresponding to natural events. Hence, this problem is especially important to monitoring applications of Wireless Sensor Networks (WSNs). For example, on the battlefield, sensors can be deployed at certain strategic points to detect and track enemies. Furthermore, in traffic management, when the number of vehicles on the road exceeds a pre-defined threshold, the system can raise an alarm to inform the supervisor. Likewise, in environment/habitat monitoring, sensors can prevent a conflagration by detecting if temperatures monitored by most sensors are higher than normal.

Several algorithms have been proposed in the literature to solve the DTC problem [14-22]. Most of those efforts mainly focused on how to design efficient methods based on specific network topologies [14-19]. First, in [14-15], the authors proposed a centralized algorithm and also demonstrated that its message complexity is near-optimal. Particularly, in their algorithm, a certain process acts as a coordinator, and all other processes communicate with the unique coordinator directly for trigger counting. Obviously, such a centralized algorithm can only work in a network topology with a sole central process and all other processes directly connecting to the central one. Furthermore, for the above algorithm, each process in the network has to know what kind of role it plays in advance.

On the other hand, some decentralized DTC algorithms have been proposed in [16-18]. More specifically, the authors in [16-17] proposed a decentralized model instead of the centralized one. In this model, all processes are first arranged into a pre-defined number of layers, and each process knows in advance the layer that it belongs to as well as executing a corresponding algorithm. After a process receives sufficient triggers, it informs some process residing in the upper layer. Moreover, upon receiving sufficient triggers, the unique process in the root layer starts a negotiation procedure with all participants to calculate the total number of triggers received by the entire system until now. In addition, based on the foregoing distributed model, the authors in [18] adjusted the threshold value for informing a process in the upper layer to design a more message-efficient algorithm. However, just like the work in [14-15], every process in the algorithm also has to know ahead not only what kind of network topology it resides in but also what

*Corresponding Author: Jichiang Tsai; E-mail: jichiangt@nchu.edu.tw
DOI: 10.53106/160792642021072204013

kind of role it plays in the system. With a similar concept, in [19], the authors designed an approximate algorithm for the DTC problem in a ring network and derived its message complexity. After that, in [20], still in a ring network, the authors further proposed an improved version to deal with the DTC problem in a deterministic way. In summary, these above efforts mainly focused on how to reduce the message complexity of a DTC algorithm in similar network topologies. Moreover, though these algorithms are claimed to be decentralized, they still need a particular process as the unique coordinator and especially a pre-defined underlying network topology. Due to the inherent decentralized and self-organized properties of dynamic networks, these aforementioned algorithms with specific global assumptions are not practical. In particular, all processes in WSNs tend to play the same role during the computation and do not know ahead what kind of topology the whole network constitutes.

Recently, some researches have explored how to eliminate the foregoing unreasonable global assumptions [21-22]. For example, a randomized DTC algorithm was introduced in [21], which can solve the DTC problem with a less restrictive assumption, especially unnecessary to direct any process in the system to play a certain role during the computation. Unfortunately, such an algorithm is approximate. Namely, it may fail to raise an alert upon receiving the exact number of triggers. Though the authors showed that there seems to be little probability of failure, it is better not to trust an approximate algorithm in critical situations. In contrast, the authors in [22] proposed two exact distributed algorithms to solve the DTC problem, free of any above global assumptions. More importantly, their algorithms will always raise an alert precisely.

Yet, all existing DTC algorithms introduced in the literature were devised for static networks, in which there is no process leaving or joining the considered system. Therefore, they cannot deal with some applications of dynamic networks, where the network topology will continually change. In this paper, based on the two algorithms for static networks proposed in [22], we design two distributed algorithms for solving the DTC problem in dynamic networks. Particularly, like their static versions, our two dynamic algorithms are both exact and thus can raise an alert upon receiving the exact number of triggers.

This paper is structured as follows. The system model and the specification of the DTC problem are described in Section 2. Furthermore, in Section 3 and Section 4, we introduce our first dynamic DTC algorithm and improve it in terms of message overhead, respectively. Then, a simulation study is presented in Section 5. At last, we conclude our work and present possible future work in Section 6.

2 Preliminaries

2.1 System Model

A finite set Π of processes $\{p_1, p_2, \ldots, p_n\}$ with $n > 1$ is considered in the subject. Unlike the model of traditional networks, the processes in Π are not necessarily aware of each other initially, even at the end of computation. This assumption reveals the nature of considered dynamic networks, where there is no central authority that initializes each process with certain context information. Moreover, processes in Π communicate by exchanging messages through reliable channels, that is, there is no message creation, corruption and duplication. Hence, if processes of two ends are correct, a message sent is eventually delivered to its destination exactly once. In particular, process p_i can send a message to another process p_j if p_j is within the transmission range of p_i. Here, we assume that the communication between any two processes is symmetric, i.e., if p_i can send a message to p_j, p_j can also send a message to p_i [23]. Below, we further describe two technical terms employed in the text:

Definition 1: In a dynamic network, a process p_j is called a neighbor of some other process p_i if and only if p_j is within the transmission range of p_i.

Namely, a neighbor of p_i is a process with one hop away from p_i. As an example, in Figure 1, p_3 is the unique neighbor of p_1.

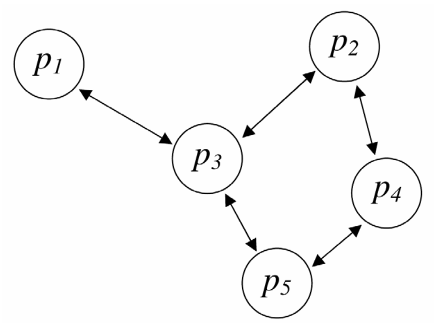


Figure 1. A network topology with five processes

Definition 2: In a dynamic network, a process p_j is called a participant of some other process p_i if and only if p_i can route a message to p_j via some intermediate process(es).

In other words, there exists at least one path between the process and any of its participants. For instance, in Figure 1, p_2, p_3, p_4 and p_5 are all participants of p_1. Finally, we denote the set of neighbors of p_i as $neighbors_i$ and the set of participants of p_i as $participants_i$ in this paper.

2.2 DTC Specification

In the Distributed Trigger Counting (DTC) problem, every process p_i receives triggers from some external sources. After the number of triggers received by the whole system reaches a pre-specified amount w, the system will raise an alert to inform the supervisor. More specifically, the DTC problem is solved under the following two conditions [16]:

1. The order of triggers received by a process is unknown in advance.

2. The number of triggers received by a process is unknown beforehand.

In addition, according to the requirement on the time of raising the alert, we can classify algorithms for the DTC problem into two categories [21]:

1. Exact: This kind of algorithm will raise an alert upon receiving exactly w triggers.

2. Approximate: This kind of algorithm will raise an alert when it receives about w triggers. This means that it may fail to raise an alert precisely.

The dynamic DTC problem that we intend to address in this research work is to solve the aforementioned DTC problem on the dynamic system model defined in the previous subsection.

3 Dynamic Intuitive Algorithm

First, we make an assumption that the time for a message to diffuse throughout a considered network is much less than that between the receipt of two continuous triggers from external sources. This assumption is reasonable in real situations since the time for propagating a message over a dynamic network is negligible in comparison to that between two continuous triggers related to a natural event happening. Also, it is necessary for exact DTC algorithms because when the considered system has received w-1 triggers, if two distinct processes can receive a trigger almost at the same time, it is possible that the whole system has received more than w triggers upon raising an alarm. The foregoing assumption is also adopted by all exact DTC algorithms introduced in [14-21], just without being mentioned explicitly. Moreover, solving the DTC problem requires that all processes in the network can communicate with each other to cooperatively count the triggers received by the whole system. If the network becomes disconnected due to process moving or leaving, processes in one of the two disconnected parts obviously have no way to communicate with those in the other part. Thus we need to make an assumption that regardless of how many processes leave the network and how a process moves within the network, the resultant topology of the network is still connected.

Next, if a process leaves before broadcasting out its information regarding the trigger count, such information will be lost and thus the resultant trigger number calculated by the entire system will be imprecise. Here, we want to remark that a process may leave the system permanently for being crashed, or may leave temporarily due to movement or low battery and will join the system again afterward. The former case is the trickiest part to tackle. One viable way to achieve this assumption is to attach a watchdog timer to every process to store the latest information about the trigger number periodically. Upon detecting its corresponding process being crashed, the watchdog timer will provide necessary information to the neighbors of the crashed process. As for the latter case, we suppose that as soon as a process detects that it will leave the system, e.g., being warned low battery by some supervisory voltage system or recognizing low signals from all other processes, the process will immediately broadcast out its latest information regarding trigger counting. Besides, the leaving process will store all variables used in the algorithm into its nonvolatile memory for its rejoining afterward.

On the other hand, it is also required to make a joining process able to learn the total number of triggers that the system has received so far. Specifically, a joining process has to ask other processes for the latest information regarding trigger counting no matter it is a recovering or newly-joining process. Here, a recovering process means that such a process had joined the system for cooperatively calculating the number of triggers but left due to some event, like low battery, and joins the cooperation again via some recovery procedure, like recharge. Such a kind of process will restore all variables stored in the nonvolatile memory prior to its leaving to facilitate the collaboration with other processes. As for a newly-joining process, we mean that it is the first time for this process to join the system for collecting triggers. This kind of process will first execute the **init** phase of the algorithm to do the initialization. Note that for the sake of brief presentation, in our following two algorithms for dynamic networks, we assume that all trigger-receiving, process-leaving and process-joining actions proceed in a totally sequential fashion. The assumption is reasonable because the times for the entire system processing these three kinds of actions are much shorter than that between two continuous events occurring.

Below, we introduce the core property of our first DTC algorithm for dynamic networks, called the dynamic intuitive algorithm. This property is directly derived form the result of [22]:

Theorem 1: In a connected component of a network topology, if the first time a process receives a message from a neighbor, it will broadcast the message to all its neighbors, such a message will finally reach all participants of the process from which the message originates.

Furthermore, as mentioned in [22], though the above idea can make information propagate to all processes in the whole connected network, it will cause a process to send a redundant message. To distinguish useful messages from redundant ones, a feasible way is to attach more information to make each message distinct, e.g., attaching the unique ID of the originating process and the number of triggers received by the process so far.

Now we begin to elaborate our dynamic intuitive DTC algorithm, which is presented in Figure 2, and all definitions of messages and variables for this algorithm are listed in Table 1. Foremost, the part for counting triggers is similar to its counterpart for static networks proposed in [22]. Particularly, each process p_i takes the number of triggers that have to be received by the whole system, i.e., w, as the input (line 1). Then in the **init** phase, p_i designates w_i as the number of triggers received by it so far and w_i' as the number of triggers that all processes except p_i have received so far (lines 2-3). Since there is no trigger received by the system in the very beginning, both w_i and w_i' are initialized to zero (lines 4-5). After completing the **init** phase, upon receipt of a trigger from an external source, w_i is increased by one to denote that p_i has just received a trigger (line 6). Besides, p_i broadcasts *increase*$_i(w_i)$ to inform its neighbors that it has received one more trigger (line 7). Then, p_i examines whether the sum of w_i and w_i' equals w; if so, p_i is the process that receives the last trigger such that it needs to raise an alarm to inform the supervisor (lines 8-9). While receiving *increase*$_j(w_j)$ for $j \neq i$, if it is the first time for p_i to receive such a message, p_i stores the current number of triggers received by p_j in dynamically-allocated variable *latest_received*$_i^j$ (line 13). Here, if such a variable does not exist, p_i first allocates memory space for it (lines 11-12). Moreover, p_i adds w_i' by one and broadcasts *increase*$_j(w_j)$ out (lines 14-15). Note that the reason of exploiting dynamically-allocated variables is that the algorithm is assumed to not learn the accurate number of processes in the system. Thus doing so makes every process merely allocate necessary memory space to record numbers of triggers received by other processes until now.

```
The dynamic intuitive DTC algorithm for process p_i

input:
(01) w: the number of triggers that have to be received;
init:
(02) w_i: the number of triggers that p_i has received so far;
(03) w_i': the number of triggers that all processes except p_i have
received so far;
(04) w_i ← 0;
(05) w_i' ← 0;
upon receipt of a trigger from an external source:
(06) w_i ++;
(07) broadcast increase_i( w_i );
(08) if w_i + w_i' = w
(09)     raise an alarm;
upon receipt of increase_j( w_j ) from some other process:
(10) if it is the first time to receive increase_j( w_j ) ∧ j ≠ i
(11)     if latest_received_i^j does not exist
(12)         allocate latest_received_i^j;
(13)     latest_received_i^j ← w_j;
(14)     w_i' ++;
(15)     broadcast increase_j( w_j );
upon leaving:
(16) broadcast leave_i( );
upon receipt of leave_j( );
(17) if it is the first time to receive leave_j( )
(18)     if leaving_process_i^j does not exist
(19)         allocate leaving_process_i^j;
(20)     leaving_process_i^j ← true;
(21)     broadcast leave_j( );
upon joining:
(22) broadcast join_i( );
upon receipt of join_j( ):
(23) if it is the first time to receive join_j( ) ∧ j ≠ i
(24)     if leaving_process_i^j does not exist
(25)         allocate leaving_process_i^j;
(26)     leaving_process_i^j ← false;
(27)     broadcast join_j( );
         /* Provide the current information of the system for p_j. */
(28)     broadcast inf_for_join_i( w_i, j );
(29)     for any x do
(30)         if leaving_process_i^x = true
(31)             broadcast inf_for_join_i( latest_received_i^x, j );
upon receipt of inf_for_join_j( w_k, ℓ ):
(32) if it is the first time to receive inf_for_join_j( w_k, ℓ )
       /* Broadcast this message. */
(33)     if j ≠ i ∧ k ≠ i ∧ ℓ ≠ i
(34)         broadcast inf_for_join_j( w_k, ℓ );
       /* Make the joining process p_i able to learn the information
       regarding p_k. */
(35)     if ℓ = i
       /* p_k is strange to p_i. */
(36)         if latest_received_i^k does not exist
(37)             w_i' ← w_i' + w_k;
(38)             allocate latest_received_i^k;
(39)             latest_received_i^k ← w_k;
       /* p_i has known p_k but it needs to update information. */
(40)         if latest_received_i^k exists ∧ w_k > latest_received_i^k
(41)             w_i' ← w_i' + w_k - latest_received_i^k;
(42)             latest_received_i^k ← w_k;
       /* According to the source of the message, we can learn if
       p_k is a leaving process. */
(43)         if leaving_process_i^k does not exist
(44)             allocate leaving_process_i^k;
(45)         if j ≠ k
(46)             leaving_process_i^k ← true;
(47)         else
(48)             leaving_process_i^k ← false;
```

Figure 2. The dynamic intuitive DTC algorithm for receiving *w* triggers

Table 1. The definitions of all messages and variables in the dynamic intuitive DTC algorithm

Variable	Definition
w	The number of triggers the entire system has to receive
w_i	The number of triggers p_i has received so face
w_i'	The number of triggers all processes except p_i have received so far
$increase_i(w_i)$	To indicate if p_i has received the w_i^{th} trigger
$latest\_receoved_i^j$	The information p_i has known regarding the number of triggers received by p_j so far
$leave_i()$	To indicate if p_i has left now
$leaving\_process_i^j$	The information p_i has known regarding if p_i has left now
$join_i()$	To indicate if p_i *has joined now*
$in\_for\_join_i(w_i, j)$	To provide the information for p_j regarding the number of triggers p_i has received so far

As for the part for dealing with process leaving, since a process immediately broadcasts out a message to inform other processes upon receiving a trigger (line 7), a leaving process p_i does not need to provide any information regarding trigger counting when it leaves. Instead, p_i just lets its neighbors know its leaving (line 16). Correspondingly, upon learning some process p_j leaving the system for the first time, p_i sets dynamically-allocated Boolean variable $leaving\_process_i^j$ to *true* to denote that p_j has left the system and broadcasts this information out (lines 17-21).

In contrast, a joining process p_i has to ask all processes in the system for the latest number of triggers received by the entire system by broadcasting message $join_i()$ out (line 22). Correspondingly, upon receiving $join_j()$ for the first time, p_i sets dynamically-allocated Boolean variable $leaving\_process_i^j$ to *false* to denote that p_j has joined the system and broadcasts out $join_j()$ as well as message $inf\_for\_join_i(w_i, j)$ (lines 23-28). Furthermore, to make p_j able to learn the total number of triggers that the whole system has received so far, p_i also needs to provide p_j with its known numbers of triggers received by all leaving processes (lines 29-31). More importantly, upon receiving message $inf\_for\_join_j(w_k, l)$, where w_k represents the latest number of triggers received by some process p_k known by process p_j, an intermediate process p_i, i.e., $i \neq j$, $i \neq k$ and $i \neq l$, forwards the message to other processes if it is the first time to receive this message (lines 32-34). If the destination of such a message is p_i itself, namely, $i=l$, p_i updates its knowledge about the system according to the information carried on the message (lines 35-48). In particular, according to information carried on the message, p_i can determine whether p_k is a leaving process. Namely, if $j \neq k$, this means that p_k is a leaving process, and p_j offers p_i required information for it. Otherwise, p_k still remains in the system. After the above procedure, a joining process can obtain the numbers of triggers received by all processes involved in the computation, especially those having left.

Here, we illustrate the above algorithm with the instance shown in Figure 1. After p_1 receives the first trigger from an external source, it broadcasts $increase_1(1)$ to its neighbors. Upon receipt of $increase_1(1)$, the unique neighbor of p_1, that is, p_3, knows that p_1 has just received a trigger and records this event. Because it is the first time to receive such a message, p_3 broadcasts $increase_1(1)$ to its neighbors. Later, p_1, p_2, and p_5 receive $increase_1(1)$. According to the information carried on the message, p_1 learns that this message broadcast by p_3 is a redundant message. Hence, p_1 discards it. On the other hand, p_2 and p_5 realize that it is the first time for them to receive such a message. They record $increase_1(1)$ and then broadcast it to their neighbors. Finally, message $increase_1(1)$ will travel throughout the entire network. This means that all processes in the network will eventually learn that p_1 has just received a trigger from an external source. Moreover, since we assume that all trigger-receiving, process-leaving and process-joining actions progress in a totally sequential fashion, all processes will know every previous trigger-receiving event before the next trigger is received by the system.

Next, we formally demonstrate that the algorithm presented in Figure 2 is an exact DTC algorithm for a dynamic network.

Theorem 2: The algorithm presented in Figure 2 is sufficient for solving the DTC problem in a dynamic network and also an exact algorithm.

Proof. Based on the result of [22], we know that the algorithm presented in Figure 2 is sufficient for solving the DTC problem if no dynamic event happens. Hence, we only need to discuss the events of leaving process and joining process here. First, because a process immediately broadcasts out a message to inform other processes upon receiving a trigger (line 7), information about trigger counting of a leaving process will not be lost. Furthermore, a joining process can obtain the numbers of triggers received by all processes involved in the computation from some other process (lines 35-48) such that it can cooperate with other processes remaining in the system to count triggers as if the joining process had participated in the computation from the very beginning.

More importantly, seeing that we assume that all trigger-receiving, process-leaving and process-joining actions progress in a totally sequential fashion, the network topology can be regarded static at any duration of dealing with a trigger received from an external source. Therefore, we have that all processes will know every previous trigger-receiving event before the next trigger is received by the system. Namely, when the system has received w-1 triggers, all processes in the system can know this result prior to receiving the w^{th} trigger. This means that when some process in the system receives the w^{th} trigger from an external source, it will precisely raise an alarm to inform the supervisor (lines 8 and 9), and all other processes will learn such an event before they receive one more trigger. From the above discussions, we can conclude that the algorithm presented in Figure 2 is sufficient for solving the DTC problem in a dynamic network and also exact.

4 Dynamic Improved Algorithm

Although the algorithm presented in Figure 2 is sufficient for solving the DTC problem in the exact way, it will have the whole system yield much message overhead during the computation. This is obviously a critical issue for energy-limited dynamic networks, especially WSNs. The method adopted by the second DTC algorithm for static networks introduced in [22] to reduce the message overhead is to lessen the frequency of broadcasting actions performed by a process upon receiving a trigger from external sources. More specifically, the above algorithm directs a process p_i not to broadcast information about trigger counting so frequently that p_i only broadcasts a message to inform its neighbors after receiving a certain amount of triggers, i.e., f. Trivially, doing so can significantly lower the message complexity of the entire system. Though a larger value of f can reduce more message overhead of the whole system, it causes the problem that the system may not raise an alarm exactly upon receiving the desired number of triggers. Therefore, to lower the message complexity as well as avoiding an imprecise alarm, the second DTC algorithm in [22], which is round-based, makes the value of f vary round by round, with the requirement that the number of processes in the system is known beforehand so as to appropriately adjust the value of f during the computation. In addition, to calculate the number of remaining triggers for a coming round, all n processes in the system need to broadcast a message by the end of the current round to inform other processes of how many triggers they have received until now. Hence, the benefit of the threshold for lowering the message complexity may not be achieved when its value is less than the number of processes in the system. Thus as the value of f is less than n, the foregoing algorithm acts like the first one introduced in [22].

Here, we exploit the concept of the second DTC algorithm in [22] to improve our dynamic intuitive algorithm. The resultant dynamic improved algorithm contains two parts, which are presented in Figure 3 and Figure 4, respectively, and all definitions of messages and variables for this algorithm are listed in Table 2. The former part is responsible for counting triggers and is also similar to its static counterpart in [22]. Below, we only explain the techniques that are not employed in the foregoing dynamic intuitive one. First, each process p_i takes the initial number of processes in the system, n, and the upper bound on number of processes during the computation, n_u, as the input (lines 2-3). The upper bound n_u can be any integer not less than n, and it means that during the whole course of collecting w triggers, the number of processes involved in the computation at any moment will never exceed this value. Such a bound is used for properly adjusting the threshold value in every non-final round. Moreover, in the **init** phase, p_i defines some variables with certain purposes. In particular, $round_i$ is the number of the current round in p_i, which is initialized to 1 (lines 6 and 10), and f_i is the threshold value to end the current round, which is defined as $\lfloor(w{-}w_i{-}w_i')/n_u\rfloor$ (lines 7 and 11). Because the initial number of processes in the system is assumed to be known, each process first allocates memory space for storing information about initial participants. More specifically, Boolean variable $leaving\_process_i^x$ is utilized for denoting whether p_x has left the system or not, and $infor_i^x$ is used for checking if p_i has received information about trigger counting from p_x in the current round (lines 15-18). Seeing that there is no corresponding event happening in the beginning of the algorithm, the above dynamically-allocated variables are all initialized to zero or *false*.

Next, upon receipt of a trigger from an external source, if p_i is not in the final round, namely, $round_i \neq 0$, f_i is decreased by one (lines 20-21). Then, if f_i equals zero, p_i broadcasts $finish\_this\_round_i(w_i, round_i)$ to its neighbors to inform them that it has received enough triggers in the current round and also records the number of triggers that it has received so far, i.e., w_i, in $latest\_received_i^i$ (lines 22-24). Otherwise, if p_i is in the final round, it will work as the previous intuitive one (lines 36-39). As for upon receiving $finish\_this\_round_j(w_j, round_j)$ for the first time, p_i broadcasts the message out, updates the trigger number of p_j, and sets $infor_i^j$ to $round_j$ for denoting that it has already received the information from p_j in the current round (lines 40-43). Furthermore, if p_i has not provided its trigger count in the current round, it broadcasts $finish\_this\_round_i(round_i, w_i)$ out and updates corresponding variables as well (lines 44-47). Since leaving processes cannot support their trigger numbers, p_i utilizes its knowledge regarding them for computing the new threshold for the coming round (lines 26-28 and 48-50).

More importantly, to deal with the extreme scenario that there is merely one process remaining in the system, the process terminating the current round will verify this scenario via checking if the value of every existing $infor_i^x$ equals $round_i$ (lines 26-28). Finally, after receiving enough information, that is, receiving $finish\_this\_round_j(w_j,\ round_j)$ originating from any remaining process p_j in the current round, p_i calculates the threshold value for the next round (lines 29-30 and 51-52). If the new value of f_i is not less than n_u, p_i enters a new non-final round and increases $round_i$ by one accordingly (lines 31-32 and 53-54). Otherwise, p_i calculates the total number of triggers received by all processes but itself so far and sets $round_i$ to 0 to enter the final round (lines 33-35 and 55-57).

The main part of the dynamic improved DTC algorithm for process p_i

```
input:
(01) w: the number of triggers that have to be received;
(02) n: the initial number of processes in the system;
(03) n_u: the upper bound on number of processes in the system;
init:
(04) w_i: the number of triggers that p_i has received so far;
(05) w_i': the number of triggers that all processes except p_i have
received so far;
(06) round_i: the current round number of p_i;
(07) f_i: the number of triggers to end a round;
(08) w_i ← 0;
(09) w_i' ← 0;
     /* The procedure starts from the first round. While round_i is
     increased by 1, the procedure enters the next round. On the
     other hand, while it goes to 0, the procedure enters the final
     round. */
(10) round_i ← 1;
(11) f_i ← ⌊ (w - w_i - w_i') / n_u ⌋;
     /* Allocate the memory space for storing information of
     these initial processes. */
(12) for 1 ≤ x ≤ n do
(13)     allocate latest_received_i^x;
(14)     latest_received_i^x ← 0;
(15)     allocate leaving_process_i^x;
(16)     leaving_process_i^x ← false;
(17)     allocate infor_i^x;
(18)     infor_i^x ← 0;
upon receipt of a trigger from an external source:
(19) w_i ++;
(20) if round_i ≠ 0
(21)     f_i --;
(22)     if f_i = 0
(23)         broadcast finish_this_round_i(w_i, round_i);
(24)         latest_received_i^i ← w_i;
(25)         infor_i^i ← round_i;
(26)         for any x do
(27)             if leaving_process_i^x = true
(28)                 infor_i^x ← round_i;
         /* Adjust the threshold value in every non-final round
         dynamically to reduce the message overhead. */
(29)         if all infor_i^x equal round_i
(30)             f_i = ⌊ (w – the sum of all latest_received_i^x) / n_u) ⌋;
(31)             if f_i ≥ n_u
(32)                 round_i ++;
(33)             else
(34)                 w_i' = the sum of all latest_received_i^x except
                            latest_received_i^i;
(35)                 round_i ← 0;
(36) else
(37)     broadcast increase_i(w_i);
(38)     if w_i + w_i' = w
(39)         raise an alarm;
upon receipt of finish_this_round_j(w_j, round_j):
(40) if it is the first time to receive finish_this_round_j(w_j, round_j)
     ∧ j ≠ i
(41)     broadcast finish_this_round_j(w_j, round_j);
(42)     latest_received_i^j ← w_j;
(43)     infor_i^j ← round_j;
(44)     if p_i has not provided its trigger count in the current round
(45)         broadcast finish_this_round_i(w_i, round_i);
(46)         latest_received_i^i ← w_i;
(47)         infor_i^i ← round_i;
(48)         for any x do
(49)             if leaving_process_i^x = true
(50)                 infor_i^x ← round_i;
(51)     if all infor_i^x equal round_i
(52)         f_i = ⌊ (w – the sum of all latest_received_i^x) / n_u) ⌋;
(53)         if f_i ≥ n_u
(54)             round_i ++;
(55)         else
(56)             w_i' = the sum of all latest_received_i^x except
                        latest_received_i^i;
(57)             round_i ← 0;
upon receipt of increase_j(w_j):
(58) if it is the first time to receive increase_j(w_j) ∧ j ≠ i
(59)     latest_received_i^j ← w_j;
(60)     w_i' ++;
(61)     broadcast increase_j(w_j);
```

Figure 3. The main part of the dynamic improved DTC algorithm for process p_i

```
The auxiliary part of the dynamic improved DTC algorithm for process p_i

upon leaving:
(62) broadcast leave_i( w_i );
upon receipt of leave_j( w_j );
(63) if it is the first time to receive leave_j( w_j )
(64)     leaving_process_i^j ← true;
(65)     latest_received_i^j ← w_j;
(66)     broadcast leave_j( w_j );
upon joining:
(67) broadcast join_i( );
upon receipt of join_j( ):
(68) if it is the first time to receive join_j( ) ∧ j ≠ i
(69)     if leaving_process_i^j does not exist
(70)         allocate latest_received_i^j;
(71)         allocate leaving_process_i^j;
(72)         allocate infor_i^j;
(73)     latest_received_i^j ← 0;
(74)     leaving_process_i^j ← false;
(75)     broadcast join_j( );
         /* Provide the current information of the system for p_j. */
(76)     broadcast inf_for_join_i( w_i, round_i, f_i, j );
(77)     for any x do
(78)         if leaving_process_i^x = true
(79)             broadcast inf_for_join_i( latest_received_i^x, round_i, f_i,
                     j );

upon receipt of inf_for_join_j( w_k, round_j, f_j, ℓ ):
(80) if it is the first time to receive inf_for_join_j( w_k, round_j, f_j, ℓ )
         /* Broadcast this message. */
(81)     if j ≠ i ∧ k ≠ i ∧ ℓ ≠ i
(82)         broadcast inf_for_join_j( w_k, t_k, round_j, f_j, t_ℓ );
         /* Make the joining process p_i able to learn the information
         regarding p_k. */
(83)     if ℓ = i
         /* If p_k is strange to p_i, p_i needs to allocate new memory
         space, else it just updates the information. */
(84)         if latest_received_i^k does not exist
(85)             allocate latest_received_i^k;
(86)             allocate leaving_process_i^k;
(87)             allocate infor_i^k;
(88)         latest_received_i^k ← w_k;
         /* According to the source of the message, we can learn if p_k
         is a leaving process. */
(89)         if j ≠ k
(90)             leaving_process_i^k ← true;
(91)         else
(92)             leaving_process_i^k ← false;
(93)         round_i ← round_j;
(94)         f_i ← f_j;
```

Figure 4. The auxiliary part of the dynamic improved DTC algorithm for process p_i

Table 2. The definitions of all messages and variables in the dynamic improved DTC algorithm

Variable	Definition
w	The number of triggers the entire system has to receive
n	The initial number of processes in the system
n_u	The upper bound on number of processes in the system
w_i	The number of triggers p_i has received so far
w_i'	The number of triggers all processes except p_i have received so far
$round_i$	The current round number of p_i
f_i	The threshold value for p_i to end the current round
$latest\_receoved_i^j$	The information p_i has known regarding the number of triggers received by p_j so far
$leaving\_process_i^j$	The information p_i has known regarding if p_i has left now
$infor_i^j$	To indicate if p_i has received information about trigger counting from p_j in the current round
$Finish\_this\_round_i$	To inform participants that p_i has received enough triggers in the current round
$increase_i(w_i)$	To indicate if p_i has received the w_i^{th} trigger
$leave_i(w_i)$	To indicate if p_i has left and received w_i trigger now
$join_i()$	To indicate if p_i has joined now
$in\_for\_join_i(w_i, round_i, f_i, j)$	To provide the information for p_j regarding the number of triggers p_i has received so far, along with the current round number and the current threshold value of p_i

On the other hand, the part shown in Figure 4 is for dealing with process leaving and joining. In particular, now that no process will broadcast out its information about trigger counting until the end of a non-final round, a process p_i needs to send out its number of triggers received so far to let its neighbors learn its latest trigger count upon its leaving (line 62). While knowing the leaving action of some process p_j for the first time, p_i updates its knowledge about p_j as well as broadcasting this information out (line 63-66). As for the joining procedure, upon learning the first time that some process has joined the system, beside updating its knowledge regarding this joining process, p_i informs the process of all known information about the system (lines 68-79). Furthermore, upon receiving *inf_for_join*$_j$(w_k, *round*$_j$, f_j, l) from some other process, if p_i is exactly the destination of such a message, it updates its knowledge about the system accordingly (lines 80-94).

Here, we also use Figure 1 as an example to illustrate the improved algorithm. First, after p_1 receives enough f_1 triggers in the first round, it broadcasts *finish_this_round*$_1$(f_1, *round*$_1$) to inform its sole neighbor p_3 that it will end the first round. Upon receipt of this message, p_3 knows that p_1 has terminated the first round and thus records the information carried on the message. Because p_3 receives such a message for the first time, p_3 broadcasts *finish_this_round*$_1$(f_1, *round*$_1$) and *finish_this_round*$_3$(w_3, *round*$_3$) to its neighbors. Subsequently, all p_1, p_2, and p_5 receive these two messages. After receiving such two messages, p_1 discards the first message, which is redundant to it, and records the information carried on *finish_this_round*$_3$(w_3, *round*$_3$). Also, p_1 broadcasts *finish_this_round*$_3$(w_3, *round*$_3$) to its neighbors. Likewise, p_3 discards the above redundant message from p_1. On the other hand, both p_2 and p_5 know that it is the first time for them to receive the two messages separately originating from p_1 and p_3. Hence, p_2 and p_5 broadcast to their neighbors *finish_this_round*$_2$(w_2, *round*$_2$) and *finish_this_round*$_5$(w_5, *round*$_5$), respectively, along with the two messages from p_1 and p_3. At last, after receiving the information about the trigger number from all other processes, every process p_i in the system, $\forall i \in [1, 5]$, can learn how many triggers the whole system has received in the first round and then calculate the new threshold value for the second round.

Theorem 3: The algorithms presented in Figure 3 and Figure 4 are sufficient for solving the DTC problem in a dynamic network and also an exact algorithm.

Proof. Based on the result of [22], we know that the algorithm presented in Figure 3 and Figure 4 are sufficient for solving the DTC problem if no dynamic event happens. Hence, we also only need to discuss the events of leaving process and joining process here. First, because a leaving process will broadcast out its latest number of received triggers to its neighbors as it leaves (line 62), the information about trigger counting of the leaving process will not be lost. In addition, a joining process will obtain the total numbers of triggers received by the whole system so far from some other process (lines 83-94) such that it can count triggers cooperatively with other processes remaining in the system.

Moreover, for showing that the algorithm is also exact, we need to prove that the system will still have some remaining triggers by the end of a round except the final one. Thus it will continuously enter a new round until the final one is reached. Since the procedure of the final round in the algorithm is the same as that of the algorithm in Figure 2, which is proven exact by Theorem 2, we only need to demonstrate that in any non-final round, the number of triggers received by the whole system will never exceed the number of remaining triggers that the system has not yet received prior to this round.

Let the number of remaining triggers having not been received by the system before entering the k^{th} round, which is not the final one, is w^k, where $k \geq 1$. The value of w^k as well as that of the corresponding threshold f^k for the k^{th} round is calculated by the end of the $(k\text{-}1)^{th}$ round for $k \geq 2$ (lines 51 and 52), while the value of w^1 for the first round is obtained from the input (line 1) and that of f^1 for the same round is computed in the initialization part (line 11). Note that in a non-final round, the system will receive the maximum number of triggers when triggers are evenly delivered to all processes. In particular, in this case, only one process will receive sufficient triggers to end the current round, and all other n-1 processes will receive one less trigger than this process. Hence, the maximum number of triggers that can be received in the k^{th} round is $f^k + (n\text{-}1)(f^k\text{-}1) = nf^k\text{-}n+1 = n\lfloor w^k/n_u \rfloor\text{-}(n\text{-}1)$. Because of $\lfloor w^k/n_u \rfloor \leq \lfloor w^k/n \rfloor \leq w^k/n$ and $n > 1$, we have $n\lfloor w^k/n_u \rfloor\text{-}(n\text{-}1) \leq n\lfloor w^k/n \rfloor\text{-}(n\text{-}1) \leq n(w^k/n)\text{-}(n\text{-}1) \leq w^k\text{-}(n\text{-}1) < \mathrm{w}^k$.

5 Simulation Results

Now, we start to perform a simulation study to compare our two algorithms more comprehensively in dynamic and randomized situations. Particularly, we use the PARSEC language [24] to conduct the simulation. Since the topologies of dynamic networks are varied, we execute the DTC application on systems with different settings of node numbers, edge numbers and process leaving/joining frequencies. The parameter settings of our simulation experiments are listed below:

1. The environment:
   - □ The considered environment is a square with sides 1000m long.
2. The processes:
   - □ For each experiment, there are 20, 30 or 40 processes uniform-randomly scattered in the considered environment, where the upper bound on number of processes is set to the value of the number of initial processes plus 10.
   - □ Both process leaving and joining times are determined by a Poisson distribution with a mean interval λ equal to the duration of receiving 1000, 2000 or 3000 triggers.
3. The transmission range:
   - □ The transmission range of every process is set to 100m, 200m or 300m in the simulation.
4. The triggers:
   - □ There are 10000 triggers that the system has to

receive. Moreover, the destination of a trigger is uniformly random.

For each setting, we generate 1000 different connected network topologies and separately execute the dynamic intuitive and improved algorithms on these systems.

According to the algorithms, we can know the corresponding message costs of dealing with different dynamic events, e.g., the function, upon receipt of *leaving*$_i$(), is for dealing with the event of a leaving process. Furthermore, since the sizes of messages induced by our DTC algorithms are different, we further consider the total sizes of all messages produced by the algorithms during the entire procedure to make the simulation comparisons more convincible (Table 3). Particularly, we suppose that the size of a trivial message carrying no variable, e.g., *join*$_i$(), is one; while that of a message with y variables is y+1. For example, the size of message *increase*$_i$(w_i) is 2.

Table 3. The comparison of weighted overhead of each message in our algorithms and [22]

Message \ Algorithm	intuitive		Improved	
	Static	Dynamic	Static	Dynamic
increase(*w*)	1	2	1	2
leave()	N/A	1	N/A	1
join()	N/A	1	N/A	1
inf_for_join(*w*, *j*)	N/A	3	N/A	N/A
finish_this_round(*w*, *round*)	N/A	N/A	1	3
leave(*w*)	N/A	N/A	N/A	2
in_for_join(*w*, *round*, *f*, *j*)	N/A	N/A	N/A	5

In addition, distinct settings in the simulation experiments are used to verify if our algorithms can meet various conditions, e.g., the number of processes for the scalability of the system, the frequency of dynamic events for the environment and application, and the transmission range for the limitation of the hardware and environment.

The simulation results are summarized in Table 4, Table 5 and Table 6, where we present the average total message size, standard deviation and the ratio of the message size induced by the improved algorithm to that by the intuitive one for each transmission range setting, respectively. Note that if the topology is disconnected in any moment of simulation, we will discard it. More specifically, from the simulation results of all settings with the same number of processes, we can see that the smaller the value of λ is, the higher message overhead the two algorithms produce during the computation. The underlying reason is that with a smaller value of λ, both process leaving and joining events happen more frequently, and thus more messages are produced to manipulate these events. On the other hand, because the two DTC algorithms direct all existing processes to provide a joining process with the up-to-date information of the system and the latest trigger count of a leaving process also needs to be broadcast to the whole system, a system with more initial processes and a process with longer transmission range, resulting in a topology with higher connectivity, will induce more message overhead. As expected, the simulation results of all settings with the same value of λ exactly conform to this observation.

Table 4. The simulation results for transmission range 100m

Number of processes	λ = 1000					λ = 2000					λ = 3000				
	Intuitive		Improved		Ratio	Intuitive		Improved		Ratio	Intuitive		Improved		Ratio
	Mean	Standard deviation	Mean	Standard deviation		Mean	Standard deviation	Mean	Standard deviation		Mean	Standard deviation	Mean	Standard deviation	
20	647962.15	250604.14	306439.73	121151.76	0.4729	493361.39	199896.13	123241.40	59662.30	0.2498	466676.12	185878.97	78322.88	38509.28	0.1678
30	1590083.49	427111.64	1011462.74	293860.32	0.6361	1181777.22	333577.85	434980.71	156996.32	0.3681	1089985.78	303721.39	290150.18	104332.63	0.2662
40	3126706.25	621555.10	2301553.00	523216.74	0.7361	2240689.77	493855.18	992489.62	305735.77	0.4429	2020795.14	425618.16	625696.66	191474.14	0.3096

Table 5. The simulation results for transmission range 200m

Number of processes	λ = 1000					λ = 2000					λ = 3000				
	Intuitive		Improved		Ratio	Intuitive		Improved		Ratio	Intuitive		Improved		Ratio
	Mean	Standard deviation	Mean	Standard deviation		Mean	Standard deviation	Mean	Standard deviation		Mean	Standard deviation	Mean	Standard deviation	
20	2368984.90	663377.88	1119260.99	326714.13	0.4725	1800914.40	470774.58	450594.88	171582.81	0.2502	170467.14	421133.93	286666.23	105086.62	0.1682
30	5855927.10	1106696.15	3708280.15	780525.96	0.6333	437136136	843081.68	1602337.56	458696.52	0.3666	4031393.59	735015.93	1076190.98	309473.40	0.2670
40	11503274.83	1642552.75	8466750.26	1449546.42	0.7360	8256762.63	1283929.10	3661319.72	1004132.60	0.4434	7453087.23	1057056.55	2307606.61	625945.39	0.3096

Table 6. The simulation results for transmission range 300m

Number of processes	$\lambda = 1000$					$\lambda = 2000$					$\lambda = 3000$				
	Intuitive		Improved		Ratio	Intuitive		Improved		Ratio	Intuitive		Improved		Ratio
	Mean	Standard deviation	Mean	Standard deviation		Mean	Standard deviation	Mean	Standard deviation		Mean	Standard deviation	Mean	Standard deviation	
20	4845149.65	1243885.89	2290969.20	610138.64	0.4728	3692487.40	833191.68	92643.20	339337.26	0.2501	3495892.59	727390.27	588279.34	204955.35	0.1683
30	11963263.96	2095894.75	7571915.44	1495571.19	0.6329	8940271.80	1528954.78	3284950.10	931001.82	0.3674	8247105.54	1343799.30	2206267.81	620489.22	0.2675
40	23514562.76	3094378.28	17307564.48	2776668.36	0.7360	16885369.26	2405820.31	7486031.19	1998480.81	0.4433	15234234.72	1947740.97	4719675.66	1250411.66	0.3098

Next, to show the efficiency of the dynamic improved algorithm for reducing message overhead, the ratio between message sizes induced by our two dynamic algorithms is presented as well. According to that, we can observe that regardless of which setting is considered, the dynamic improved algorithm is much more efficient in message overhead than the dynamic intuitive one. However, we can also learn that the dynamic improved algorithm pays more message overhead to address topology variations of networks than the intuitive one, especially when the number of initial processes is large.

6 Conclusions

In this paper, we have proposed two exact algorithms for solving the DTC problem in dynamic networks, where a process will move, leave or join. Therefore, our algorithms are much more applicable to the real world, especially monitoring applications of WSNs. In addition, we have comprehensively compared our two DTC algorithms and found that the technique adopted by the latter does take effect in reducing the message overhead. For the future work, it is interesting to propose novel dynamic DTC algorithms with other techniques to achieve better performances.

Acknowledgments

This research was supported by Ministry of Science and Technology, Taiwan, R.O.C. under grant 109-2221-E-035-067-MY3, 109-2622-H-035-001-, and 108-2221-E-035-063-.

References

[1] C.-Y. Chong, S. P. Kumar, Sensor Networks: Evolution, Opportunities, and Challenges, *Proceedings of the IEEE*, Vol. 91, No. 8, pp. 1247-1256, August, 2003.

[2] D. Steere, A. Baptista, D. McNamee, C. Pu, J. Walpole, Research Challenges in Environmental Observation and Forecasting Systems, *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, Boston, Massachusetts, USA, 2000, pp. 292-299.

[3] D. Estrin, R. Govindan, J. Heidemann, S. Kumar, Next Century Challenges: Scalable Coordination in Sensor Networks, *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Seattle, Washington, USA, 1999, pp. 263-270.

[4] C.-Y. Chong, F. Zhao, S. Mori, S. Kumar, Distributed Tracking in Wireless Ad Hoc Sensor Networks, *Proceedings of the 6th International Conference of Information Fusion*, Queensland, Australia, 2003, pp. 431-438.

[5] F. Zhao, J. Shin, J. Reich, Information-Driven Dynamic Sensor Collaboration, *IEEE Signal Processing Magazine*, Vol. 19, No. 2, pp. 61-72, March, 2002.

[6] M. L. Massie, B. N. Chun, D. E. Culler, The Ganglia Distributed Monitoring System: Design, Implementation, and Experience, *Parallel Computing*, Vol. 30, No. 7, pp. 817-840, July, 2004.

[7] W. Zhang, G. Cao, DCTC: Dynamic Convoy Tree-based Collaboration for Target Tracking in Sensor Networks, *IEEE Transactions on Wireless Communications*, Vol. 3, No. 5, pp. 1689-1701, September, 2004.

[8] K. S. Park, V. S. Pai, CoMon: A Mostly-scalable Monitoring System for PlanetLab, *ACM SIGOPS Operating Systems Review*, Vol. 40, No. 1, pp. 65-74, January, 2006.

[9] B. Li, B. Qi, J. Yang, Y. Sun, W.-X. Cui, H.-G. Yan, OEEABed-Online Distributed Energy Efficiency Analysis Testbed and Novel Monitoring Approach under Wireless Sensor Network, *Journal of Internet Technology*, Vol. 14, No. 3, pp. 467-475, May, 2013.

[10] F. Lamberti, A. Sanna, A Java Web-based Multichannel Architecture for Distributed System Monitoring, *Journal of Internet Technology*, Vol. 3, No. 4, pp. 235-244, October, 2002.

[11] C. Liu, G. Cao, Distributed Monitoring and Aggregation in Wireless Sensor Networks, *2010 IEEE Proceedings INFOCOM*, San Diego, CA, USA, 2010, pp. 1-9.

[12] S. Ji, J. He, A. S. Uluagac, R. Beyah, Y. Li, Cell-based Snapshot and Continuous Data Collection in Wireless Sensor Networks, *ACM Transactions on Sensor Networks*, Vol. 9, No. 4, pp. 1-29, July, 2013.

[13] T. H. Lai, T. H. Yang, On Distributed Snapshots, *Information Processing Letters*, Vol. 25, No. 3, pp. 153-158, May, 1987.

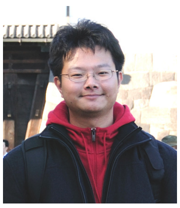
[14] R. Garg, V. K. Garg, Y. Sabharwal, Scalable Algorithms for Global Snapshots in Distributed Systems, *Proceedings of the 20th Annual International Conference on Supercomputing*, Queensland, Australia, 2006, pp. 269-277.

[15] R. Garg, V. K. Garg, Y. Sabharwal, Efficient Algorithms for Global Snapshots in Large Distributed Systems, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 21, No. 5, pp. 620-630, May, 2010.

[16] V. T. Chakaravarthy, A. R. Choudhury, V. K. Garg, Y. Sabharwal, Brief Announcement: A Decentralized Algorithm for Distributed Trigger Counting, *Proceedings of the 24th*

International Conference on Distributed Computing, Cambridge, MA, USA, 2010, pp. 398-400.
[17] V. T. Chakaravarthy, A. R. Choudhury, V. K. Garg, Y. Sabharwal, An Efficient Decentralized Algorithm for the Distributed Trigger Counting Problem, *Proceedings of the 12th International Conference on Distributed Computing and Networking*, Bangalore, India, 2011, pp. 53-64.
[18] S. Kim, J. Lee, Y. Park, Y. Cho, An Optimal Distributed Trigger Counting Algorithm for Large-scale Networked Systems, *Simulation*, Vol. 89, No. 7, pp. 846-859, July, 2013.
[19] S. Karmakar, S. Chattopadhyay, A Trigger Counting Mechanism for Ring Topology, *Proceedings of the 37th Australasian Computer Science Conference*, Auckland, New Zealand, 2014, pp. 81-87.
[20] S. Karmakar, A. C. Reddy, An Improved Algorithm for Distributed Trigger Counting in Ring, *The Computer Journal*, Vol. 57, No. 7, pp. 980-986, July, 2014.
[21] V. T. Chakaravarthy, A. R. Choudhury, Y. Sabharwal, Improved Algorithms for the Distributed Trigger Counting Problem, *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, Anchorage, AK, USA, 2011, pp. 515-523.
[22] C.-C. Chang, J. Tsai, Distributed Trigger Counting Algorithms for Arbitrary Network Topology, *Wireless Communications and Mobile Computing*, Vol. 16, No. 16, pp. 2463-2476, November, 2016.
[23] F. Buckley, F. Harary, *Distance in Graphs*, Addison-Wesley, 1990.
[24] X. Zeng, R. Bagrodia, M. Gerla, GloMoSim: A Library for Parallel Simulation of Large-scale Wireless Networks, *Proceedings of the 12th Workshop on Parallel and Distributed Simulations*, Alberta, Canada, 1998, pp. 154-161.

Biographies



Che-Cheng Chang received his PhD degree in department of electrical engineering from National Chung Hsing University, Taiwan in 2015. Then he joined the department of information engineering and computer science, Feng Chia University, Taiwan, as an assistant professor in 2018. His current research interests include autonomous vehicles, machine learning, and distributed computing.

Jichiang Tsai joined the Department of Electrical Engineering, National Chung Hsing University, Taiwan, as an assistant professor in 2002 and then was promoted to associate professor in 2005. From 2017, he has been a full professor of the same department. His current research interests include unmanned vehicles, distributed computing and machine learning.

Tien-Yu Chang received his Ph. D. degree in the Department of Electrical Engineering, National Chung Hsing University, Taiwan in 2019. He has been a software design engineer in the National Chung-Shan Institute of Science and Technology, Taiwan from 2003. His research interests include distributed systems and wireless sensor networks.