A Trigger Counting Mechanism for Ring Topology

Sushanta Karmakar[1] **Subhrendu Chattopadhyay** [1]

[1] Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati, India, 781039
Email: sushantak@iitg.ernet.in, subhrendu@iitg.ernet.in

.

Abstract

Consider a distributed system with n processors, which receive triggers from the outside world. The Distributed Trigger Counting (DTC) problem is to raise an alarm if the number of triggers over the system reaches w, which is an user specified input. DTC is used as a primitive operation in many applications, such as distributed monitoring, global snapshot etc. In this paper, we propose an algorithm for the DTC problem in a ring topology with a message complexity of $O(n^2 \log(w/n))$ and each node in the system receives $O(n \log(w/n))$ number of messages. We also discuss about the possible tuning of the algorithm which results better complexities.

Keywords: Distributed algorithm, distributed monitoring, distributed trigger counting, ring topology

1 Introduction

Distributed trigger counting (DTC) is an important problem in distributed systems. Consider a distributed system with n processes. Each process receives some triggers (signals) from an external source. The DTC problem is to detect the state when the number of triggers received by the system reaches w which is a user defined input to the system. Note here that w may be much larger than n. The sequence of processors receiving the w triggers is not known apriori to the distributed system. Our goal is to propose an algorithm for the DTC problem under a known topological setting. More specifically, in this paper we propose an algorithm for the DTC problem in a ring network.

The DTC problem arises in many applications in distributed systems. Some major application areas can be distributed monitoring, global snapshot etc. Monitoring is an important aspect in wireless networks as well as in wired networks. A wireless sensor network is typically used to monitor physical or environmental conditions such as border surveillance, forest fire detection, traffic management in highways etc. In traffic management, one may be interested in detecting whether the number of vehicles on a highway exceeds a certain threshold. Similar applications may be found in border surveillance and forest fire detection as well. In distributed system a global snapshot state is said to be valid provided all the in-transit messages are recorded. It was shown by Garg et al. (2006) that the problem of detecting whether all the in-transit messages have been recorded can be reduced to the DTC problem. Awerbuch (1985) proposed the concept of synchronizers which is a tool to transform a synchronous distributed algorithm to another that runs on an asynchronous distributed system. Here a distributed system is required to generate a signal (or pulse) when all the messages generated after the previous signal have been delivered. This problem can also be formulated as a variant of the DTC problem. The performance of an algorithm for the DTC problem is often measured by the following two metrics.

- *Message complexity:* It is the total number of messages sent by all the nodes of the systems.
- *MaxRcvLoad:* It is the maximum number of messages received by any node in the system.

The DTC problem was studied by Garg et al. (2006) for a general distributed system. In their work, they proposed two algorithms for the DTC problem: a centralized algorithm and a tree-based algorithm. The centralized algorithm has a message complexity of $O(n \log w)$. Its MaxRcvLoad has a complexity of $O(n \log w)$. This is a tight bound for the centralized algorithm. The tree based algorithm has a message complexity of $O(n \log n \log w)$. Again the MaxRcvLoad of the tree based algorithm has a complexity of $O(n \log n \log w)$. In the work the authors also showed that any deterministic algorithm for the DTC problem must have a message complexity of $\Omega(n \log(w/n))$. Huang et al. (2007) proposed a novel solution to the problem of efficient detection of an aggregate predicate over cumulative triggers over a time-varying window in a distributed monitoring system. They provided a queueing theory based analysis of the problem which provides the user the power to trade-off between desired accuracy and communication overhead. However, their approach is based on a single coordinator. Hence the algorithm is not fully distributed. In another work (Chakaravarthy, Choudhury, Garg & Sabharwal 2011) a randomized distributed algorithm was proposed for trigger counting in a tree based topology. The algorithm has a message complexity of $O(n \log n \log w)$ and MaxRcvLoad complexity of $O(\log n \log w)$ with high probability. Later in another work (Chakaravarthy, Choudhury & Sabharwal 2011) the authors proposed an improved algorithm for the DTC problem having message complexity of $O(n \log w)$ and MaxRcvLoad of $O(\log w)$. However, this work has the limitation that the algorithm is a randomized algorithm and it does not provide any bound on the messages sent per node.

Table 1: Summary of DTC algorithms

Algorithm	*Messages*	*MaxRcvLoad*	*Nature*
Tree-based (Garg et al. 2006)	$O(n \log n \log w)$	$O(n \log n \log w)$	randomized
Centralized (Garg et al. 2006)	$O(n \log w)$	$O(n \log w)$	centralized
(Chakaravarthy, Choudhury, Garg & Sabharwal 2011)	$O(n \log n \log w)$	$O(\log n \log w)$	randomized
(Chakaravarthy, Choudhury & Sabharwal 2011)	$O(n \log w)$	$O(\log w)$	randomized
(Emek & Korman 2010)	$O(n(\log n \log w)^2)$	$O((\log n \log w)^2)$	deterministic
Unidirectional Ring (this paper)	$O(n^2 \log \frac{w}{n})$	$O(n \log \frac{w}{n})$	deterministic

Also this algorithm assumes a clique of n nodes as the topology. In fact they concluded that designing a deterministic algorithm with message complexity of $O(n \log w)$ and *MaxRcvLoad* of $O(\log w)$ for any arbitrary network is an open DTC problem.

In this paper, we propose a deterministic distributed algorithm for the DTC problem in a unidirectional ring network. A ring of nodes is an interesting topology and is used in solving many distributed computing problems such as leader election, mutual exclusion, distributed snapshot etc. Also if a solution of a problem in distributed computing can be obtained for a ring topology then the same can be applied to have a solution for the same problem under arbitrary topology by means of embedding a virtual ring topology over the network (e.g. Eulerian graph). Therefore for many reasons a ring topology is important and in this paper we propose a solution to the DTC problem assuming a ring topology. We also provide an outline of how the DTC problem can be solved in a general network using our proposed solution to the problem in a ring network.

The main challenge of the DTC problem is that outside triggers may be detected by any node and one would like to minimize the communication overhead for determining when a total of w external triggers have been counted. It is important to see that there is a tradeoff between minimizing the communication overhead and having a timely detection of the w triggers. The simplest way to minimize the number of messages is to let every node detect triggers and when any one has detected w triggers, the global alarm signal is sent. However, in this simple scheme there is obviously a big risk that the system will be seriously delayed in sending the global alarm signal which has to be sent as soon as w or more triggers are received by all the nodes. The obvious way to avoid such delays is to send a message as soon as a trigger is detected. However this will generate a lot of messages (potentially $O(w)$). Hence there should be some compromise between these two extremes. In an earlier work (Garg et al. 2006) it was shown that any deterministic algorithm for the DTC problem must have a message complexity lower bound of $\Omega(n \log(w/n))$. However this result was analytical and no algorithmic instance has been found so far satisfying this lower bound.

Let there be n nodes in a unidirectional ring. The nodes are denoted as v_1, v_2, v_3, ... , v_n. Any node v_i can send a message to v_{i+1} and receive a message from v_{i-1}. Specifically, v_n sends a message to v_1. We assume an asynchronous model of computation and communication (via messages). We assume that the channels are reliable and FIFO. There is no node or link failure. Also messages are not corrupted or spuriously introduced. It is assumed that each node has a unique identifier. Out of the n nodes, there is one node v_n designated as the *master*. All other nodes (v_i such that $i \neq n$) act as slaves. The algorithm is stated in the form of guarded statements. A guarded statement is of the form $g \rightarrow a$ where g is the guard, and a is the action. The action a is executed if and only if g is true. The program for any node i contains a sequence of statements $\{S_1, S_2, \ldots, S_n\}$ where each S_j is of the form $g_j \rightarrow a_j$. The j-th statement of the program at node i is denoted by $S_j(i)$. The guard corresponding to $S_j(i)$ is denoted by $g_j(i)$ and the action corresponding to $S_j(i)$ is denoted by $a_j(i)$. Also it is assumed that the guarded actions are atomic. The main result of our algorithm is as follows. The distributed trigger counting algorithm over a ring of n processors has a message complexity of $O(n^2 \log \frac{w}{n})$ and *MaxRcvLoad* of $O(n \log \frac{w}{n})$ where w is the number of triggers to be counted.

The rest of this paper is organized as follows. Section 2 contains the related work on the DTC problem. The proposed algorithm for the DTC problem in a unidirectional ring is presented in Section 3. Section 4 contains the analysis for proving the correctness and message complexity of the proposed algorithm. Section 5 discusses the tuning of the algorithm with respect to some parameters. The trade-off between message complexity and delay in raising the alarm is discussed in Section 6. Solving the DTC problem under arbitrary network is discussed in Section 7. Finally we conclude in Section 8.

2 Related Work

Many of the earlier works primarily consider the DTC problem in a centralized setting and solve it using randomized algorithm. A randomized algorithm was proposed by Emek & Korman (2010) for the DTC problem with message complexity of $O(n(\log \log n)^2 \log w)$ and average message complexity of $O((\log \log n)^2 \log w)$. In this paper it is assumed that the input is constructed by an adaptive adversary whose decisions may depend on previous coin tosses of the randomized protocol but not on future ones. Here a separator decomposition of a tree of n nodes is constructed over which the events are aggregated. By changing the internal parameters of the randomized protocol they have obtained a deterministic protocol with message complexity of $O(n(\log n \log w)^2)$ and MaxRcvLoad of $O((\log n \log w)^2)$.

A fundamental class of problems called "thresholded counts" was introduced by Keralapura et al. (2006) where the goal is to return the aggregate frequency count of an event, that is continuously monitored by distributed nodes with a user-specified accuracy, whenever the actual count exceeds a given threshold value. They studied the cases under static as well as dynamic threshold values. Cormode et al. (2011) defined a function monitoring problem as a 4-tuple $(\kappa, f, \tau, \epsilon)$ where κ denote the number of nodes, f denote the function that is being monitored by the nodes, τ denote the threshold such that if $f \geq \tau$ then the system generates some alarm 1 and the alarm is 0 if $f \leq (1-\epsilon)\tau$. The authors give lower and upper bounds of communication cost for the $(\kappa, f, \tau, \epsilon)$

MasterProcess(*TriggerCount* w) // for master node v_n

Initialization: *TargetTriggerCount* $w' = w$; $C = 0$; $sflag = true$; $fsorFlag = false$; $eorFlag = false$; ψ; τ

(S_1) $sflag \rightarrow \psi = w'/2$; $\tau = w'/2n$
send $< start - of - round, \tau >$ to v_1
$fsorFlag = true$; $sflag = false$

(S_2) upon receiving $< Trigger > \wedge fsorFlag \rightarrow C = C + 1$

(S_3) upon receiving $< Coin, factor > \wedge fsorFlag \rightarrow C = C + factor \times \tau$

(S_4) $fsorFlag \wedge (C \geq \psi) \rightarrow efactor = 0$
Send $< end - of - round, efactor >$ to v_1

(S_5) upon receiving $< end - of - round, efac >$ from $v_{n-1} \rightarrow$
$w' = w' - (C + efac \times \tau)$
if $(w' > 0)$ then $sflag = true$
else $sflag = false$
$fsorFlag = false$
send $< TargetReached >$ to v_1

Figure 1: Pseudocode of master node for DTC algorithm for a ring

problem with different f. The algorithms proposed by the authors are randomized. One example of f can be the aggregate function. Here each input trigger i is associated with a value α_i. The goal is to raise an alarm when the aggregate of these values crosses a threshold.

A decentralized and randomized algorithm called LayeredRand algorithm was proposed by Chakaravarthy, Choudhury, Garg & Sabharwal (2011). In this work, the nodes are organized in a tree topology and they communicate only with the nodes in the adjacent layers. The algorithm proceeds in multiple rounds and in each round it achieves a trigger count which is approximately half of the required count. In another work, Chakaravarthy, Choudhury & Sabharwal (2011) proposed an approximate algorithm with the assumption $w \leq 2^n$, and it exhibits a message complexity of $O(n \log n \log w)$ and MaxRcvLoad of $O(\log n \log w)$. Similar works were done in (Korman & Kutten 2007, Emek & Korman 2011). No prior work on the DTC problem considered a ring as the underlying topology.

3 Algorithm for DTC in a Unidirectional Ring

In this section, we describe an algorithm for the DTC problem in a unidirectional ring topology. Its message complexity is $O(n^2 \log(w/n))$ and *MaxRcvLoad* is $O(n \log(w/n))$. We assume that the triggers which come from the outside world are distributed to the n nodes of the ring randomly. The distribution of the triggers among n nodes is arbitrary and not known apriori. The algorithm consists of a number of rounds. The total number of triggers to be counted by the system is denoted by w. The system has n number of nodes connected in a logical unidirectional ring configuration. The nodes in the ring are denoted as v_i $(1 \leq i \leq n)$. There is one node v_n which is designated as the *master* node. All the other nodes in the ring are called *slaves*. Each *slave* node has the following state variables: C is a counter indicating the number of received triggers that have not yet contributed to the distributed trigger counting and initialized to 0; $fsorFlag$ is a boolean flag which indicates the start of the DTC algorithm, and it is initialized to $false$; $cflag$ is a boolean flag which indicates whether a node has received a *Coin* message, and it is initialized to $false$; *coinCount* is an integer that denotes the number of *Coin* messages sent by the node; $eorFlag$ is a boolean flag that indicates the end of the current round and it is set to $true$ on receipt of a *end-of-round* message. For the *master*, there are some other state variables in addition to the aforesaid variables. They are: w' is the remaining number of triggers yet to be counted by the system and is initialized to w; $sflag$ is the start of round flag for each round and it is initialized to $true$. The *master* node also maintains two variables, ψ and τ, which are known as *global threshold* and *local threshold*. Each slave node gets the value of τ for the current round from its previous node in the ring. To reduce the number of messages, each slave node sends a *Coin* message only when its trigger count crosses the local threshold, τ. Similarly the *master* node sends a *end-of-round* message only when the total trigger count at the master crosses its global threshold, ψ.

For any node v_i, v_{i-1} is called the predecessor of v_i and v_{i+1} is the successor of v_i. Note that the successor of v_n is v_1. Similarly the predecessor of v_1 is v_n. The trigger counting algorithm proceeds in multiple rounds. At the start of each round, the system should know the number of triggers which are yet to be counted to raise an alarm. This can be known by subtracting the sum of all the triggers received in the previous round from w'. Each round of the algorithm is started by the *master* node v_n by sending the *start-of-round* message to v_1. Node v_n calculates ψ and τ, and sends τ to v_1 along with the *start-of-round* message. It updates its state variables as follows: $fsorFlag(v_n) = true$ and $sflag(v_n) = false$. When a *slave* v_i gets (*start-of-round*, τ') message it also calculates its local threshold as $\tau = \tau'$. It sets its local state variables as follows: $fsorFlag(v_i) = true$, $eorFlag(v_i) = false$. Since it has not yet sent any *Coin* message, v_i sets $coinCount(v_i) = 0$. It also sends a (*start-of-round*, τ) message to v_{i+1}.

With $fsorFlag(v_i) = true$ $(1 \leq i \leq n)$, any node v_i increments its local counter $C(v_i)$ on receiving a *Trigger* message. So each node v_i independently

SlaveProcess() // code for node v_i where $i \neq n$

Initialization: $C = 0$; $fsorFlag = false$; $cflag = false$; τ; $coinCount = 0$; $factor$; $eorFlag = false$

(S_6) upon receiving $< start - of - round, \tau' > \rightarrow$ $\tau = \tau'$
Send $< start - of - round, \tau >$ to v_{i+1}
$coinCount = 0$
$fsorFlag = true$; $eorFlag = false$

(S_7) upon receiving $< Trigger > \wedge\ fsorFlag \rightarrow C = C + 1$

(S_8) upon receiving $< Coin, fac > \wedge\ fsorFlag \wedge \neg cflag \rightarrow C = C + fac \times \tau$
$cflag = true$

(S_9) $fsorFlag \wedge (C \geq \tau \vee cflag) \wedge \neg eorFlag \wedge coinCount \leq n \rightarrow$ $factor = C/\tau$
Send $< Coin, factor >$ to v_{i+1}
$C = C - factor \times \tau$
$coinCount = coinCount + 1$
$cflag = false$

(S_{10}) upon receiving $< end - of - round, efactor > \rightarrow$
$eorFlag = true$
$factor = C/\tau$
$C = C - factor \times \tau$
$efactor = efactor + factor$
Send $< end - of - round, efactor >$ to v_{i+1}

(S_{11}) upon receiving $< TargetReached > \rightarrow$ $fsorFlag = false$
send $< TargetReached >$ to v_{i+1}

Figure 2: Pseudocode of slave node for DTC algorithm for a ring

counts the number of triggers it receives. Whenever v_i finds that $C(v_i) \geq \tau$ or $cflag(v_i) = true$ then it sends a *Coin* message to v_{i+1} $(i \neq n)$. Note here that $cflag(v_i) = true$ only if v_i has received a *Coin* message from its previous node in the ring. In this case v_i computes the $factor = C/\tau$ and sends the $factor$ along with the *Coin* message to its successor node v_{i+1} in the ring. So a *Coin* message essentially transfers $\tau \times factor$ amount of triggers from v_i to the successor v_{i+1}. Node v_i also reduces the value of its counter C appropriately and remembers the send of a *Coin* message by incrementing the variable *coinCount*. Then it sets $cflag(v_i) = false$. It is clear from S_9 in the pseudocode of Figure 2 that a slave can send at most n number of *Coin* messages in a round of the algorithm.

If a node v_i $(i \neq 1)$ receives a *Coin* message from its predecessor and $fsorFlag(v_i) = true$ and $cflag(v_i) = false$ then v_i increments it local trigger count $C(v_i)$ by $\tau \times fac$, where fac is the multiplicative factor received along with the *Coin* message. Also v_i sets $cflag(v_i) = true$. Therefore *Coin* messages move from one node to another and eventually reaches the *master* node v_n. When the *master* receives a *Coin* message, v_n increments $C(v_n)$ by $\tau \times fac$, where fac is the the multiplicative factor that v_n received from its predecessor. When the *master* finds that $C(v_n) \geq \psi$ then v_n initiates the end of the round by sending a *end-of-round* message to v_1. During the propagation of the *end-of-round* message from one node to another, nodes continue to receive triggers from the outside world. These triggers are counted in a similar way.

If a slave v_i receives (*end-of-round*, $efactor$) message then it first set $eorFlag(v_i) = true$ such that no further *Coin* message is sent in this round. It computes $factor = C/\tau$ and adds this with the received multiplicative factor $efactor$, and sends this modified $efactor$ to v_{i+1} along with a *end-of-round* message. In this way the (*end-of-round*, $efactor$) message moves from one slave to another and eventually reaches the *master*. When the *master* receives (*end-of-round*, $efac$) message from v_{n-1} it computes the remaining number of triggers yet to be counted in future rounds. This is given by $w' = w' - (C(v_n) + efac \times \tau)$. If the remaining number of triggers yet to be counted is more than zero then v_n sets it variable $sflag(v_n) = true$ so that it can start another round (by S_1 of Figure 1).

4 Correctness and Analysis

Lemma 1. *A slave node sends at most n number of* Coin *message in each round.*

Proof. By S_9, a node v_i sends a *Coin* message to v_{i+1} if the G_9 is true. Again G_9 is true if $coinCount \leq n$. By S_6, $coinCount = 0$ at the start of each round. Also by S_9, if v_i sends a *Coin* message to v_{i+1} then it increments its *coinCount*. Hence in each round, v_i can send at most n number of *Coin* messages. □

Lemma 2. *If* master *node* v_n *gets the (*end-of-round, $efac$*) message in a round and E is the number of triggers contributed by all nodes except* v_n *during the end-of-round phase then* $E \geq 0$.

Proof. By S_4, the master node v_n sends a (*end-of-round*, $efactor$) message to v_1 with $efactor = 0$. Any node v_i, on receiving a (*end-of-round*, $efactor$) message, computes its own contribution as $factor = C/\tau$. It then adds this $factor$ with the $efactor$ and forwards the (*end-of-round*, $efactor$) message to v_{i+1} with the modified $efactor$ value. Therefore eventually v_n will receive the (*end-of-round*, $efac$) message from v_{n-1}. Note here that the contribution of triggers

by v_n itself in this phase is done through the variable $C(v_n)$. Hence $E = efac \times \tau$. Since $efac \geq 0$ and $\tau \neq 0$ therefore $E \geq 0$. □

Lemma 3. *In each round, at least $\psi = w'/2$ number of triggers are counted by the system where w' is the remaining number of triggers yet to be counted at the begining of the round.*

Proof. By S_7 (or S_2) and S_8 (or S_3)) it is clear that $C(v_i)$ is updated when v_i receives a *Trigger* or a *Coin* message. Also by S_4, the master node v_n sends a (*end-of-round*, $efactor$) message to v_1 only if $fsorFlag(v_n) = true$ and $C(v_n) \geq \psi$. Again by Lemma 2, the number of triggers contributed by all nodes except v_n during the propagation of *end-of-round* message (from the time it is sent by v_n to the time it is received by v_n) is $E \geq 0$. Hence total number of triggers counted in a round is given by $C(v_n) + E$. Since $C(v_n) \geq \psi$, therefore the lemma is proved.

□

Theorem 1. *The distributed trigger counting takes $O(\log \frac{w}{n})$ rounds.*

Proof. Here we will analyze how the value of w is decreasing from one round to the next round. By Lemma 2, it depends on $E \geq 0$. Initially $w' = w$. By S_5, the remaining number of triggers to be counted in future rounds is given by $w' = w-(C(v_n)+E)$. Upper bound of the number of rounds can be obtained when $E = 0$. Here w' decreases as w, $\frac{1}{2}w$, $(\frac{1}{2})^2w$, $(\frac{1}{2})^3w$, ... , $(\frac{1}{2})^kw$. The number of rounds continue till $w' \geq 0$.

Let after $(r + 1)$ rounds $\tau < 1$ for the first time. So, in the r-th round $\tau \geq 1$. Therefore $\frac{(1/2)^r w}{n} \geq 1$. Therefore $r = O(\log \frac{w}{n})$. At $(r + 1)$-th round, $\tau \leq 1$. Hence $w'/2n \leq 1$. Hence $w' \leq 2n$. Since $\tau \leq 1$, if a node gets 1 trigger then its local threshold is crossed and a *Coin* message is sent. Therefore to count $w' \leq 2n$ number of triggers, at most $2n$ *Coin* messages will be required. In worst case this will require $2n$ rounds. Hence the total number of rounds is $O(2n + \log \frac{w}{n}) = O(\log \frac{w}{n})$ since $w >> n$.

□

Theorem 2 (Partial Correctness). *If the DTC algorithm has terminated then at least w triggers have been counted.*

Proof. Let us assume that the DTC algorithm has terminated. However at least w triggers have not yet been counted. Therefore by S_5, $w' > 0$. So v_n sets $sflag(v_n) = true$. Therefore G_1 becomes enabled. So eventually G_6 will be enabled. However this contradicts the assumption that the algorithm has terminated. Therefore $w' \leq 0$. Hence at least w triggers will be counted.

□

Theorem 3 (Termination). *The DTC algorithm for a ring network eventually terminates.*

Proof. By Lemma 1, each slave node sends at most n number of *Coin* messages in a round. After this, no slave will send a *Coin* message in the current round even if it crosses its local threshold or receives *Coin* message from its predecessor. Also if $C(v_n) \geq \psi$ then v_n sends a *end-of-round* message to v_1. Each slave node v_i, on receiving a *end-of-round* message, forward it to its successor. Eventually when v_n gets back a *end-of-round* message, it does not send any further *end-of-round* message in the current round. Here v_n checks is $w' > 0$. By Lemma 3, eventually $w' \leq 0$ will hold. Therefore v_n will set $sflag(v_n) = false$ and $fsorFlag(v_n) = false$. Hence G_1, G_2, G_3, G_4 are all $false$. Also by S_{11}, $fsorFlag(v_i) = false$ for any slave node v_i. Hence G_7, G_8, G_9 are $false$. Since G_1 is $false$, therefore v_n does not send *start-of-round* message. Here G_6 is $false$. Similarly G_{10} and G_{11} are also $false$. Therefore all the guards are $false$. Therefore the algorithm eventually terminates.

□

Theorem 4. *The message complexity of the DTC algorithm is $O(n^2 \log \frac{w}{n})$.*

Proof. In a round, each slave node can send n number of *Coin* messages. Also each of them is forwarded by the other slaves towards the master node. Hence in each round $O(n^2)$ number of messages are sent overall by all the nodes. By Theorem 1, the algorithm takes $O(\log \frac{w}{n})$ rounds. Hence the overall message complexity is $O(n^2 \log \frac{w}{n})$.

□

Theorem 5. *The MaxRcvLoad of the DTC algorithm in a ring is $O(n \log(w/n))$.*

Proof. In a round each node v_i receives $O(n)$ number of messages. Since the algorithm has overall $O(\log \frac{w}{n})$ rounds, the MaxRcvLoad per node is $O(n \log(w/n))$.

□

5 Tuning the Algorithm

The algorithm can be tuned by appropriately setting the parameters ψ and τ. In this case we have chosen $\tau = \psi/n$. If we choose $\tau = \psi$ then we may achieve a better message complexity and MaxRcvLoad. Let us assume that $\psi = w/k$ and $\tau = \psi/l$. Here we can claim that,

$$\begin{aligned} \textit{No of rounds} = \mathcal{R} &= O\left(\log_{\frac{k}{k-1}}(w/n)\right) \\ &= O\left(\log_{\frac{w}{w-\psi}}(w/n)\right) \end{aligned}$$

It can be observed that the round complexity is primarily dependent on ψ. This is due to the fact that one round completes only when all the nodes together counts a global threshold, ψ, number of triggers. The local threshold, τ, does not affect the number of rounds. The variation of the number of rounds with respect to ψ is shown in Figure 3. We consider three different values of w. It is observed that as ψ increases, the number of rounds required decreases.

The overall message complexity of the algorithm can be given as,

$$\mathcal{M} = O\left(\frac{n\psi}{\tau}\left(\log_{\frac{w}{w-\psi}}(w/n)\right)\right)$$

It is obvious from the above equation that $\mathcal{M}$ is directly proportional to $l = \psi/\tau$. In the proposed algorithm in this paper, $l = n$. If we assume $l = 1$ (i.e. $\psi = \tau$) then we can achieve a better overall message complexity of $O(n \log_{\frac{w}{w-\psi}}(w/n))$. Figure 4 depicts the variation of message complexity with respect to $l = \psi/\tau$. It is clear that as l increases the messages complexity increases. This is due to the fact that l can increse only when ψ is large and τ is small. In this case, each node crosses its local threshold (due

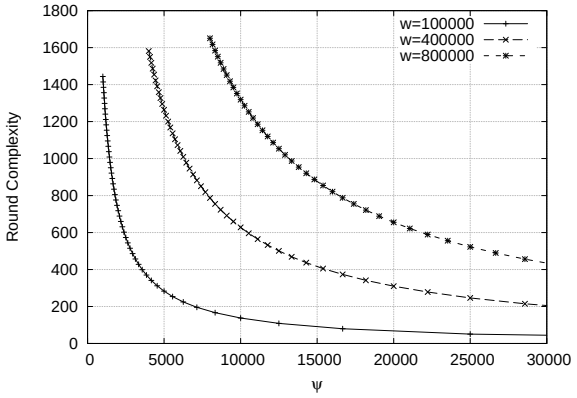

Figure 3: The variation of number of rounds with respect to ψ

to the receive of triggers) many more times and thus more number of *Coin* messages are sent in a round. This increases the overall message complexity of the system.

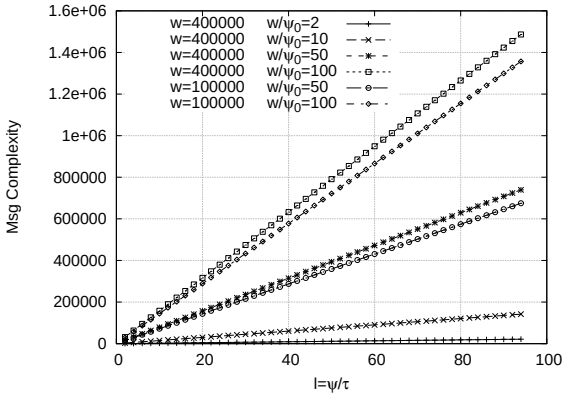

Figure 4: The variation of message complexity with respect to l

Similarly in Figure 5, we see the variation of message complexity with respect to τ. Here we keep w and ψ fixed and observe the variation of message complexity with respect to τ. It is found that as τ increases, message complexity falls sharply initially. However as the value of τ approaches ψ, the rate of fall in message complexity is small and not as sharp as earlier. This is an indication that we may not improve the message complexity by arbitrarily increasing the τ.

6 Message Complexity and Delay Trade-off

There is a trade-off between the message complexity of distributed trigger counting and the time of raising an alarm when at least w triggers have been counted. The simplest way to minimize the number of messages is to let every node detect triggers and when any one has detected w triggers, the global alarm signal is sent. However, in this simple scheme there is obviously a big risk that the system will be seriously delayed in sending the global alarm signal which has to be sent as soon as w or more triggers are received by all the nodes. The obvious way to avoid such delays is to send a message as soon as a trigger is detected.

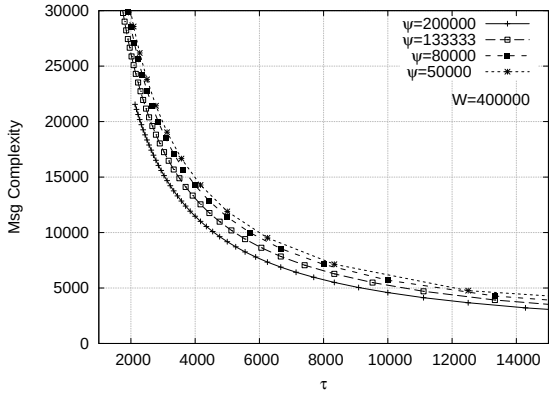

Figure 5: The variation of message complexity with respect to τ

However this will generate a lot of messages (potentially $O(w)$). Hence there should be some compromise between these two extremes.

Let t_0 be the time when w-th trigger enters the system (any node). Let t_1 be the time at which v_n sends the *TargetReached* message to v_1. We define the delay of generating the alarm as $\mathcal{T} = |t_1 - t_0|$. In the following we give a measure of $\mathcal{T}$.

Let e_1 be the $(w-1)$-th trigger and e_2 be the w-th trigger. Now there can be two cases:

Case 1: e_1 and e_2 enter the system in same round. Every trigger that enters the system is counted by the *master* node through *Coin* message or excess triggers during *end-of-round*. The maximum delay between the times e_1 and e_2 are recorded at the *master* node is the delay between the receive of two successive *Coin* messages by the *master* v_n. Essentially this is equal to the time required by τ triggers to arrive at any arbitrary node (where trigger count is zero). Let this time be denoted by Δ. In this case $\mathcal{T} = \Delta$. Here we assume that processing delay and communication delay is negligible.

Case 2: e_1 and e_2 enter the system in two different successive rounds. In this case, $\mathcal{T} = \Delta + \Delta_e$ where Δ_e is the amount of time required to collect all the excess triggers during the *end-of-round* phase.

If τ is large then delay will be large. However message complexity will be smaller. The reverse will happen if τ is small.

7 DTC in Arbitrary Network

To solve the DTC problem over an arbitrary network, we need to embed a logical unidirectional ring over the graph. For a certain class of graphs (Eulerian graphs), this can be done by forming an Eulerian circuit over the graph. Even though a Hamiltonian circuit provides the perfect ring embedding, the graph may not have an Hamiltonian circuit. Also finding a Hamiltonian circuit is NP-complete. Therefore our approach relies on Eulerian circuit construction. Makki (1999) proposed a distributed algorithm for construction of an Eulerian circuit. The algorithm has a message complexity of $(1 + r)(|\mathcal{E}| + n)$ where $0 \leq r < 1$. Here $\mathcal{E}$ denotes the set of edges of the graph. Our approach is to pre-process the input graph, provided it is Eulerian, to find an Eulerian circuit using the

above algorithm. Next on the virtual ring thus obtained, we apply the proposed trigger counting algorithm. Hence the overall message complexity of the DTC problem in an arbitrary network, which is Eulerian, is $O(|\mathcal{E}| + n^2 \log \frac{w}{n}) = O(n^2 \log \frac{w}{n})$. Also MaxRcvLoad for DTC in this case remains $O(n \log \frac{w}{n})$ since average message complexity of Eulerian circuit construction is $O(n)$.

8 Conclusion

We have presented a distributed algorithm for the DTC problem in a ring network with message complexity of $O(n^2 \log(w/n))$ and MaxRcvLoad of $O(n \log(w/n))$. One of the important issues is that the DTC algorithm may generate an alarm after counting w' number of triggers where $w' > w$. One of the future works may be to minimize the difference between w' and w. Proposing an algorithm for the DTC problem in an arbitrary network with optimal complexity can be a challenging work.

References

Awerbuch, B. (1985), 'Complexity of network synchronization', *Journal of ACM* **32**(4), 804–823.

Chakaravarthy, V. T., Choudhury, A. R., Garg, V. K. & Sabharwal, Y. (2011), An efficient decentralized algorithm for the distributed trigger counting problem, *in* 'Proceedings of the 12th International Conference on Distributed Computing and Networking', Springer-Verlag, Berlin, Bangalore, India, pp. 53–64.

Chakaravarthy, V. T., Choudhury, A. R. & Sabharwal, Y. (2011), Improved algorithms for the distributed trigger counting problem, *in* 'Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium', IEEE Computer Society, Washington, DC, USA, Anchorage, Alaska, USA, pp. 515–523.

Cormode, G., Muthukrishnan, S. & Yi, K. (2011), 'Algorithms for distributed functional monitoring', *ACM Trans. Algorithms* **7**(2), 1–20.

Emek, Y. & Korman, A. (2010), Efficient threshold detection in a distributed environment, *in* 'Proceedings of the 29th ACM Symposium on Principles of Distributed Computing (PODC)', ACM, New York, USA, Zurich, Switzerland, pp. 183–191.

Emek, Y. & Korman, A. (2011), 'New bounds for the controller problem', *Distributed Computing* **24**(3-4), 177–186.

Garg, R., Garg, V. K. & Sabharwal, Y. (2006), Scalable algorithms for global snapshots in distributed systems, *in* 'Proceedings of the 20th Annual International Conference on Supercomputing (ICS)', ACM, New York, USA, Cairns, Queensland, Australia, pp. 269–277.

Huang, L., Garofalakis, M., Joseph, A. D. & Taft, N. (2007), Communication-efficient tracking of distributed cumulative triggers, *in* 'Proceedings of the 27th International Conference on Distributed Computing Systems', IEEE Computer Society, Washington, DC, USA, Toronto, Canada, pp. 54–63.

Keralapura, R., Cormode, G. & Ramamirtham, J. (2006), Communication-efficient distributed monitoring of thresholded counts, *in* 'Proceedings of the ACM SIGMOD International Conference on Management of Data', ACM, New York, USA, Chicago, IL, USA, pp. 289–300.

Korman, A. & Kutten, S. (2007), Controller and estimator for dynamic networks, *in* 'Proceedings of the 26th ACM Symposium on Principles of Distributed Computing (PODC)', ACM, New York, USA, Portland, Oregon, USA, pp. 175–184.

Makki, S. A. M. (1999), 'Eulerian tour construction in a distributed environment', *Computer Communications* **22**(7), 621 – 628.