Brief Announcement: A Simple and Efficient Distributed Trigger Counting Algorithm based on Local Thresholds

Consider a large-scale distributed system in which each computing device is observing triggers from an external source. Distributed Trigger Counting (DTC) algorithm is used to detect the state when the aggregated number of the observed triggers reaches a predefined value. In this paper, we propose a simple and efficient DTC algorithm: Cascading Thresholds (CT). We mathematically show that CT is an optimal DTC algorithm in terms of the total number of exchanged messages among the devices (*message complexity*). For the maximum number of received messages per node (*MaxRcv*), CT is sub-optimal. The average message complexity of CT is $O(N \log (W/N))$, and *MaxRcv* of it is $O(k \log (W/N) + N)$, where W is the number of triggers to be detected, N is the number of devices, and k is the degree of a node in the tree-like structure. Compared with previous optimal algorithm (TreeFill), CT is much more simple.

CCS Concepts: • **Theory of computation** → **Distributed algorithms**; • **Mathematics of computing** → **Probabilistic algorithms**.

Additional Key Words and Phrases: Distributed trigger counting, Distributed counting

ACM Reference Format:
. 2022. Brief Announcement: A Simple and Efficient Distributed Trigger Counting Algorithm based on Local Thresholds. 1, 1 (February 2022), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Consider a large-scale networked system in which the participating devices are counting triggers from an external source. The Distributed Trigger Counting (DTC) algorithm is to raise an alarm when the total number of triggers counted by all the devices reaches a predefined value. DTC algorithms can be used for many use cases. For example, there is an API service through which customers can subscribe to trending social network data like Twitter data API [7]. For this kind of service, multiple servers need to provide APIs to customers, and API call count through multiple servers should be aggregated to monitor call volume per customer and to charge a fee to the customer. By using a DTC algorithm, the service can raise an alarm when total API calls count reached a threshold without running distributed aggregation frequently. DTC algorithms can also be used for environment surveillance with sensor networks [6] and global snapshots [4].

We use the following notations throughout this paper: W denotes the total number of triggers to be detected, and N is the number of devices in the distributed system. We consider only the case where $W \gg N$, otherwise counting triggers becomes an easy problem [1][2]. To estimate the efficiency and scalability of our algorithm, we use *message complexity*, i.e., the total number of exchanged messages among the devices. To show how message load is evenly distributed among the devices, we use *MaxRcv*, i.e., the maximum number of received messages per device.

Author's address:

Manuscript submitted to ACM

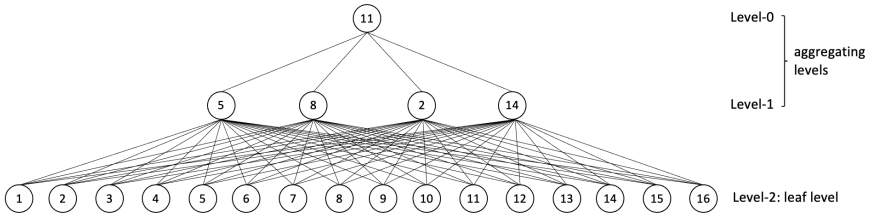

Fig. 1. An example of CT in Round 1 when $N = 16$, $k = 4$, and $h = 2$.

Garg et al. suggested the tree-based DTC algorithm and the centralized one in order to design distributed snapshot algorithms for large-scale distributed systems [4]. Their centralized algorithm has optimal message complexity, but *MaxRcv* is higher than optimal. Chakaravarthy et al. suggested the sub-optimal algorithm for both message complexity and *MaxRcv*. Their algorithm, CoinRand, uses a tree-like network topology [1]. Kim et al. suggested TreeFill, an optimal DTC algorithm [5]. However, it uses a more complicated distributed communication protocol than [1]. Emek et al. improved lower bounds on DTC algorithms and proposed the probabilistic DTC algorithm, which significantly improves the previous results [3]. Chang et al. also suggested the DTC algorithm that can work with any network topology [2].

In this paper, we present a simple and efficient DTC algorithm: Cascading Thresholds (CT). CT is inspired by [1] (that has sub-optimal message complexity) and using similar network topology with it. Our contributions are as follows. We prove that CT has no false negatives: When W events have occurred, CT does not fail to raise an alarm. Also, CT is optimal in terms of average message complexity and sub-optimal in terms of average *MaxRcv*. CT uses much simpler distributed communication protocol than the optimal DTC algorithm, TreeFill [5]. This is because (unlike CT) each node in TreeFill should maintain state information of other nodes. Experimental results show that compared with previous work, CT has similar performance to TreeFill [5] in terms of message complexity and *MaxRcv*, which we omit due to lack of space (please refer to [paper with appendix]).

2 PROPOSED DTC ALGORITHM: CASCADING THRESHOLDS (CT)

Let W be the number of triggers to be detected and the total number of devices be $N = k^h$ where k is an even integer and $k \geq 4$. Devices are organized in a multi-leveled k-ary tree-like hierarchy, as follows. A total of N devices correspond to N leaf nodes in Level-h. At level l ($0 \leq l < h$), there are k^l nodes. Therefore, among N devices $k^0 + k^1 + \cdots + k^{h-1} = \frac{k^h - 1}{k-1}$ devices have dual-roles and are also associated with nodes in the Level-$0 \sim (h-1)$. E.g., Figure 1 shows an example of hierarchical structure when $k = 4, N = 16$, and $h = 2$. We call Level-h as a leaf level and all the other levels as aggregating levels.

The proposed algorithm operates on a round basis. Let $\hat{w}$ be the number of triggers that have not yet been detected at the start of each round. The initial value is $\hat{w} = W$. The detailed description of the algorithm is as follows.

(1) **(Detect-message generation routine)** Recall that all N nodes are at leaf level (level-h). In round i, each node has a local threshold $\tau_{leaf} = \lfloor \hat{w}/(2N) \rfloor$. Also, each node x ($1 \leq x \leq N$) has a variable $C(x)$ to count the number of observed triggers in x at this round. When a node x detects a trigger, x increases $C(x)$ by 1. If x has observed τ_{leaf} triggers, $C(x)$ is cleared to 0, and x chooses a node y assigned to level-$(h-1)$ uniformly at random and sends a *Detect*-message to y.

(2) **(Detect-message propagation routine)** At aggregating level l ($1 \leq l < h$), there are k^l nodes. Each aggregating node y at Level-l maintains another counter variable $D(y)$ to indicate the number of *Detect*-messages received by

the node y in the current round. Upon receiving a *Detect*-message, the node y increments $D(y)$ by 1. When the counter variable $D(y)$ reaches the threshold $\tau_{aggregating} = \lfloor k/2 \rfloor$, the node y decreases $D(y)$ by $\tau_{aggregating}$ and sends a *Detect*-message to the randomly-selected node at the upper level ($= l - 1$). If we repeat this procedure, the root node eventually receives at least k *Detect*-messages, by Theorems 1 and 2, which are described in Section 3. If the root receives k messages, it goes to End-of-round procedure.

(3) **(End-of-round procedure)** the root node computes the total number of received triggers by all nodes and updates $\hat{w}$ by using simple broadcast and upcast procedure in the spanning tree, which is explaned in the previous work [1, 5]: The aggregation notification is broadcast to all the nodes in a recursive top-down manner. Similarly, aggregation values are computed from the leaf nodes to the root node in a recursive bottom-up manner. Thus, the root node updates $\hat{w}$ and broadcasts it to all the nodes, again in a recursive fashion. Upon receiving the updated $\hat{w}$, all the nodes clear $C(x)$ and $D(x)$, and update τ_{leaf} for the next round.
If the number of not yet detected triggers exceeds $2N$, a new round starts by going back to the Detect-message generation routine. Otherwise, it goes to the Final rounds procedure.

(4) **(Final rounds procedure)** Every trigger makes a *Detect*-message. Instead of sending *Detect*-messages to level $h - 1$, those are sent to the root. If the root counts all remaining triggers, it raises an alarm.

3 ANALYSIS

First, we show CT has no false negatives in Section 3.1 and then analyze average message complexity and average *MaxRcv* in Section 3.2.

3.1 Mathematical analysis on CT

We first show 2 properties of *CT* by Theorems 1 and 2.

Theorem 1. *Regardless of the trigger distribution in each round, at the leaf level, always at least N* Detect*-messages are sent to the nodes assigned to level-$(h-1)$ before a total of $\hat{w}$ triggers occur.*

Proof. Let us calculate the maximum number of triggers that can occur when N *Detect*-messages are sent to the nodes assigned to level-$(h-1)$. The maximum number of triggers that each node can receive without generating a *Detect*-message is $\tau_{leaf} - 1 = \lfloor \hat{w}/(2N) \rfloor - 1$. Therefore, the maximum number of triggers that can occur when N *Detect*-messages are sent is less than $\hat{w}$ because: $\lfloor \hat{w}/(2N) \rfloor \cdot N + (\lfloor \hat{w}/(2N) \rfloor - 1) \cdot N < \hat{w}$. □

Recall that when an aggregating node receives $\lfloor k/2 \rfloor$ *Detect*-messages, it creates a new *Detect*-message and sends it to a randomly chosen node at the upper level. If we repeat this procedure, the root node eventually receives at least k *Detect*-messages, by Theorem 2.

Theorem 2. *Recall that there are k^i nodes in aggregating level i $(0 < i < h)$. In level-i, always at least k^i* Detect*-messages are forwarded to the upper level before a total of k^{i+1}* Detect*-messages have received from the lower level.*

Proof. Let us calculate the maximum number of received *Detect*-messages from the lower level when k^i *Detect*-messages are sent to the nodes assigned to the upper level. The maximum number of *Detect*-messages that each node can receive without generating a *Detect*-message is $\lfloor k/2 \rfloor - 1$. Therefore, the maximum number of *Detect*-messages received at Level-i when k^i *Detect*-messages are forwarded to the upper level is less than k^{i+1} because: $\lfloor k/2 \rfloor \cdot k^i + (\lfloor k/2 \rfloor - 1)k^i < k^{i+1}$. □

By Theorem 1, regardless of the trigger distribution in each round, at the leaf level, always at least $k^h = N$ *Detect*-messages are sent to the nodes assigned to Level-$(h-1)$ before a total of $\hat{w}$ triggers occur. By Theorem 2, the root at Level-0 eventually receives k *Detect* messages before a total of $\hat{w}$ triggers occur. This means, CT always goes to the End-of-round procedure and the final rounds procedure. This implies that CT has no false positives.

3.2 Analysis on message complexity and *MaxRcv*

Average number of rounds: Consider the round i. Recall that when the root at level 0 receives k *Detect*-messages, it goes to the next round. At Level-1, there are k nodes. To send a *Detect*-messages from Level-1 to the root, the node at level 1 should receive at least $(1/2)k$ messages from Level-2. When it goes to the next round, the expected value of the variable $D(y)$ is $(1/2)k(1/2)$. This implies that to send a message from Level-1 to the root, each node at level 1 has received $(1/2)k + (1/2)k(1/2) = 3k/4$ messages from level-2 in average. Since there are k nodes at Level-1, they have received $3k^2/4$ *Detect*-messages from Level-2 in average.

Suppose that in Level-2, nodes have sent $3k^2/4$ *Detect*-messages. Since there are k^2 nodes in Level-2, in average each node have sent (3/4) *Detect*-message. To send a *Detect*-message, the node should have received at least $(1/2)k$ messages from Level-3. Hence, To send $(3k^2/4)$ messages, k^2 nodes have received $(3/4)k^2(k/2)$ messages from lower-level. When it goes to the next round, in average each node's $D(y)$ is $(1/2)k(1/2)$. Hence, in average k^2 nodes have received $(3/4)k^2(k/2) + (k/2)(1/2)k^2 = (5/8)k^3$ messages from Level-3.

For Level-3, to send $(5k^3/8)$ messages, in average k^3 nodes have received $(5/8)(k^3)(k/2) + (k/2)(1/2)k^3 = (9/16)k^4$ messages from Level-4.

If we generalize, for Level-j, in average k^j nodes have received $\frac{2^j+1}{2^{j+1}}k^{j+1}$ messages from Level-$j+1$. Hence, for Level-h where h is the height, $(1/2 + 1/(2^h))k^h \approx (1/2)N$ *Detect*-messages have been generated in average. This implies that $(\hat{w}/2N)(1/2)N = \hat{w}/4$ triggers have occurred and the remaining trigger is $\hat{w} - \hat{w}/4 = 3\hat{w}/4$. In the final round, $\hat{w}_f = W(\frac{3}{4})^f < 2N$. Hence, the number of rounds, $f = O\log(W/N)$.

Message complexity for each round: (From previous paragraphs,) for each round, the number of generated *Detect*-messages is $k + \frac{3}{4}k^2 + \frac{5}{8}k^3 + \ldots + (1/2 + \frac{1}{2}^h)k^h \leq k + k^2 + k^3 + \ldots + k^h = \frac{k^h-1}{k-1} = \frac{N-1}{k-1}$. Hence, The number of *Detect*-messages for each round is $O(N)$.

In the end-of-round, the root requests the number of triggers that have occurred so far to all nodes. If we assume that CT uses the spanning tree, number of messages for propagating the request is $O(N)$ and gets the sum through the spanning tree again (=$O(N)$), and then this is shared by all nodes through the spanning tree: $O(N)$. Therefore, the number of messages occurring in each round is $O(N)$.

Message complexity: The average number of rounds is $\log(W/N)$, and the average number of messages per round is $O(N)$. In the final round, the number of remaining messages is less than $2N$. Hence, the message complexity is $O(N\log(W/N))$.

***MaxRcv* analysis:** Recall that in each round, an average number of Detect-messages is less than $\frac{N-1}{k-1}$. This means that in average, each node has sent/received far less than 1 message. *MaxRcv* is for the root, which receives k messages for each round. In the end-of-round, if aggregation is performed through the $k-ary$ spanning tree, the root node sends $2k$ messages and receives k messages, and the internal node receives $k+1$ messages and sends $2k$ messages. Each leaf node receives 2 messages and sends 1 message. Therefore, in each round, *MaxRcv* is for the root and is $k + k + 2$. The average number of rounds is $O(\log(W/N))$ and in the final round the root receives less than $2N$ messages, so *MaxRcv* = $O((2k+2)(\log(W/N)) + 2N) = O(k(\log(W/N)) + N)$.

REFERENCES

[1] V. T. Chakaravarthy, A. R. Choudhury, and Y. Sabharwal. 2011. Improved Algorithms for the Distributed Trigger Counting Problem. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. 515–523.

[2] Che-Cheng Chang and Jichiang Tsai. 2016. Distributed trigger counting algorithms for arbitrary network topology. *Wireless Communications and Mobile Computing* 16, 16 (2016), 2463–2476.

[3] Y. Emek and A. Korman. 2010. Efficient threshold detection in a distributed environment: extended abstract. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing* (Zurich, Switzerland) *(PODC '10)*. ACM, New York, NY, USA, 183–191.

[4] Rahul Garg, Vijay K Garg, and Yogish Sabharwal. 2010. Efficient algorithms for global snapshots in large distributed systems. *Parallel and Distributed Systems, IEEE Transactions on* 21, 5 (2010), 620–630.

[5] Seokhyun Kim, Jaeheung Lee, Yongsu Park, and Yookun Cho. 2013. An Optimal Distributed Trigger Counting Algorithm for Large-scale Networked Systems. *SIMULATION: Transactions of The Society for Modeling and Simulation International* (May 2013).

[6] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke. 2017. A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements. *IEEE Access* 5 (2017), 1872–1899.

[7] Inc Twitter. 2020. *Twitter Developer Site*. https://developer.twitter.com/en.html

APPENDIX

3.3 Simulation results

We wrote simulation code using NetLogo 6.2.1 to compare the message complexity and MaxRcv of CT with other DTC algorithms (TreeFill and CoinRand). The simulation code can be found at https://github.com/SeokhyunKim/dtc-algos. To run our simulation, the newest version of NetLogo needs to be installed. (https://ccl.northwestern.edu/netlogo/download.shtml) Figure 2 shows the simulation screenshot of CT algorithm.

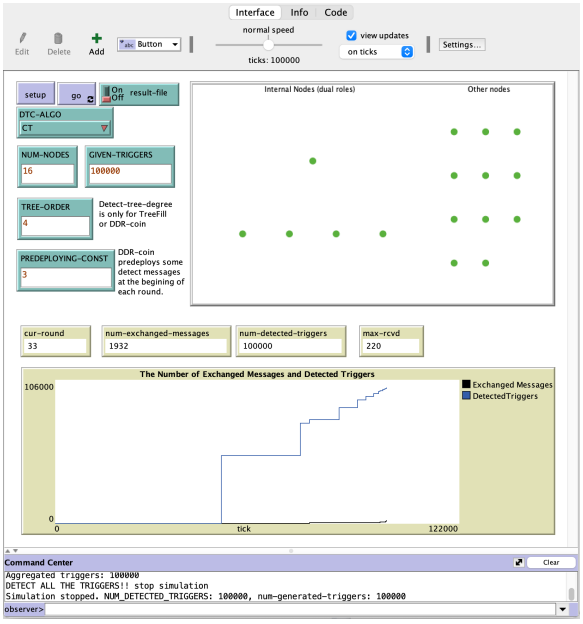

Fig. 2. Simulation of CT algorithm using NetLogo when $N = 16$, $k = 4$, and $h = 2$.

We used the following parameters: the number of triggers to be detected (W) was 100, 000. The number of nodes (N) was 4^i, where $2 \leq i \leq 5$. We set $k = 4$ for CT. We repeated 10 times to get the average value of message complexity and *MaxRcv*.

Message complexity: Figure 3 shows message complexity of CT, TreeFill, and CoinRand. As shown in this figure, among them TreeFill has the smallest number of messages. While CT has slightly larger message complexity than TreeFill, it has much smaller than CoinRand. E.g., when $N = 1024$, message complexity of CT is only about 1.19 times larger than that of TreeFill while that of CoinRand is 2.87 times bigger. As the number of nodes increases, the difference in message complexity also increases. Especially, CoinRand shows the fastest increase.

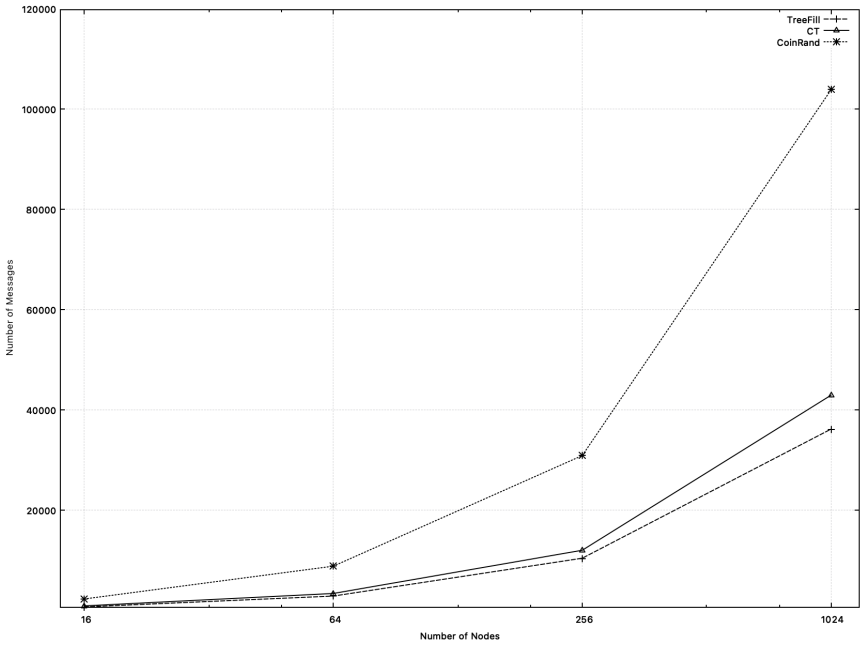

Fig. 3. Comparison of the numbers of messages of CT, TreeFill, and CoinRand when the number of nodes are 4^i where $2 \leq i \leq 5$. The number of triggers (W) is 100, 000.

***MaxRcv*:** Figure 4 shows the comparison of *MaxRcv* of CT, TreeFill, and CoinRand. As shown in this graph, TreeFill and CT shows similar *MaxRcv*. *MaxRcv* of CT is lower than that of CoinRand when N is small (about less than 20). However, *MaxRcv* of CT goes bigger as N becomes larger, which can be explained by mathematical analysis of *MaxRcv* in Section 3, *MaxRcv*=$O(k(\log(W/N)) + N)$. This is due to the final rounds procedure of CT, where all the messages go directly to the root and the *MaxRcv* of the root becomes $O(N)$. If we change the final rounds procedure as follows, *MaxRcv* of CT reduces signinifanctly and it is almost the same as that of CoinRand or TreeFill.

(Enhanced Final rounds procedure) Every trigger makes a *Detect*-message. Instead of sending *Detect*-messages to level $h - 1$, those are sent to a level $l' = \max_l (w_i \geq 2k^l)$. The root node can start next round similarly after receiving k *Detect*-messages. If it counts all remaining triggers, it raises an alarm.

Mathematical analysis of this enhanced version is complicated so we leave as future work.

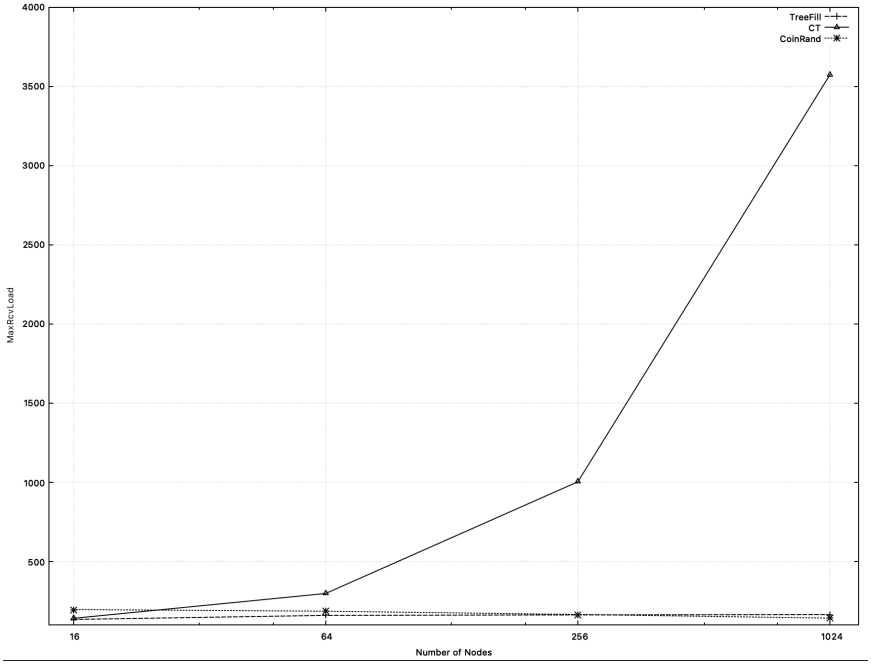

Fig. 4. Comparison of $MaxRcv$ of CT, TreeFill, and CoinRand, when the number of nodes is 4^i where $2 \leq i \leq 5$. The number of triggers (W) is $100,000$.