DISTRIBUTED QUICKEST DETECTION OF SIGNIFICANT EVENTS IN NETWORKS

Shaofeng Zou[†] *Venugopal V. Veeravalli*[⋆] *Jian Li*[‡] *Don Towsley*[‡] *Ananthram Swami*[∗]

[†]University at Buffalo, the State University of New York [⋆]University of Illinois at Urbana-Champaign
[‡]University of Massachusetts Amherst [∗]U.S. Army Research Laboratory
Emails: szou3@buffalo.edu, vvv@illinois.edu, jianli@cs.umass.edu,
towsley@cs.umass.edu, ananthram.swami.civ@mail.mil

ABSTRACT

The problem of quickest detection of significant events in networks is studied. A distributed setting is investigated, where there is no fusion center, and each node only communicates with its neighbors. After an event occurs in the network, a number of nodes are affected, which changes the statistics of their observations. The nodes may possibly perceive the event at different times. The goal is to design a distributed sequential detection rule that can detect when the event is "significant", i.e., the event has affected no less than η nodes, as quickly as possible, subject to false alarm constraints. A distributed algorithm is proposed, which is based on a novel combination of the alternating direction method of multipliers (ADMM) and average consensus approaches. Numerical results are provided to demonstrate the performance of the proposed algorithm.

***Index Terms*—** ADMM, average consensus, distributed algorithm, network event detection, quickest change detection

1. INTRODUCTION

We are interested in a network event detection problem, in which the network consists of a set of L interconnected nodes. We consider a distributed setting, where there is no fusion center in the network, and information exchanges are only among neighbors. Following the occurrence of an event in the network, one or multiple unknown nodes are affected, which changes the distribution of their observations. Moreover, the event can be dynamic, and affects more nodes over time. Our goal is to detect "significant" event quickly before it may potentially dominate the whole network. Specifically, the goal is to detect if more than η nodes are affected, as quickly as possible, subject to false alarm constraints. Applications of this model can be found in epidemic detection [1, 2], spreading viruses in computer networks [3], environmental monitoring [4], and intrusion detection in the Internet of Battlefield Things [5].

This problem has been studied previously in a centralized setting [6, 7], where there is a fusion center that collects information from all the nodes without delay. A Spartan CuSum (S-CuSum) algorithm was proposed. At each time k, the generalized likelihood ratio test (GLRT) between the two hypotheses of whether or not there are at least η sensors affected by the event is computed. An alarm is raised if the GLRT is above a threshold; otherwise, a new sample is taken at each node. It was shown that such a test is equivalent to the one that compares the sum of the smallest $L-\eta+1$ local CuSums [8] to the same threshold. The S-CuSum algorithm was shown to be first-order asymptotically optimal as the false alarm rate goes to zero. A Network CuSum (N-CuSum) algorithm was further developed, which exploits the fact that the event propagates along edges in the network, so that the affected nodes form a connected subgraph. The N-CuSum was also shown to be first-order asymptotically optimal, and it has better performance than S-CuSum, numerically.

However, the centralized algorithms are often difficult to implement due to limited communication bandwidth and long communication distance (hence delays), especially in large-scale networks. Moreover, for a sequential detection problem, to make a timely decision, the computational efficiency is of particular interest. In a centralized setting, the computation needs to be done at one fusion center, the complexity of which usually scales with the network size. Third, the event usually propagates locally in the network (along network edges), e.g., opinion propagation in social networks. Distributed algorithms address all the three concerns: (i) distributed algorithms do not need a fusion center; (ii) distributed algorithms distribute computational burden to all nodes in the network so that the computation can be done in parallel; and (iii) information exchanges are only among neighbors, which naturally incorporates the local nature of the event.

In this paper, we construct a distributed implementation of S-CuSum in [7]. We note that distributed quickest change detection has been studied in [9, 10], which assume that all the nodes in the network are affected after the event occurs,

Research reported in this paper was sponsored in part by the Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196 (IoBT CRA). The work of S. Zou and V. V. Veeravalli was also supported in part by the National Science Foundation (NSF) under grant CCF 16-18658, and by the Air Force Office of Scientific Research (AFOSR) under grant FA9550-16-1-0077.

and the goal is to quickly detect the event ($\eta = 1$). Here we consider a more general setting, where an event first affects a subset of nodes, and then propagates to affect additional nodes. We are interested in detecting the event after it has affected at least η nodes.

The proposed algorithm is a three-step procedure. In the first step, each node distributedly learns which nodes have the smallest $L - \eta + 1$ local CuSums using the alternating direction method of multipliers (ADMM) approach [11, 12]. In the second step, each node distributedly computes the sum of the smallest $L - \eta + 1$ local CuSums using the result from the first step and an average consensus algorithm [13]. In the third step, an alarm is raised if the statistic at any node exceeds a threshold. This three-step procedure is repeated every time a new sample is observed until an alarm is triggered.

For the proposed distributed algorithm, the computational complexity at each node is linear in the number of its neighbors, and communication is only between neighbors. We study the performance of the proposed distributed algorithm numerically. We also compare its performance to the centralized S-CuSum algorithm.

2. PROBLEM MODEL

Consider a set of L nodes interconnected according to an unweighted, undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Each node observes samples of some process sequentially in a synchronized fashion. There is no fusion center in the network, and each node i is only allowed to exchange information with its neighbors, denoted by $\mathcal{N}(i)$. If an event occurs in the network and affects node i at some unknown time v_i (change-point), then the distribution of the samples at node i changes from g_0 to g_1 [1]. Specifically, if we assume that the samples are independent and denote the observation made by node i at time k by $X_i[k]$, then

$$X_i[k] \sim \begin{cases} g_0, & \text{if } k < v_i, \\ g_1, & \text{if } k \geq v_i. \end{cases} \tag{1}$$

We are interested in sequentially detecting a "significant" event, i.e., when the event has affected no less than η nodes in the network. Specifically, if an alarm is triggered at a time when less than η nodes are affected, it is considered as a false alarm.

We denote by $\boldsymbol{v} = \{v_1, \ldots, v_L\}$ the set of all change-points, and assume that this set is unknown. Without loss of generality, we further assume that $v_1 \leq v_2 \leq \cdots \leq v_L$, and that the ordering is unknown in advance. Therefore, v_η is the first time at which there are no less than η affected nodes, and the problem is then to detect the change at v_η as quickly as possible subject to false alarm constraints.

[1] In this paper, we focus on the case where all the sensors have identical pre-/post-change distributions. The results can be easily generalized to heterogeneous sensors with different pre-/post-change distributions.

We let $d_i = v_{i+1} - v_i$ denote the time it takes the event to propagate from i affected nodes to $i + 1$ affected nodes, and let $\boldsymbol{d} = \{d_1, \ldots, d_{L-1}\}$. We use $\mathbb{P}_{\boldsymbol{v}}$ to denote the probability measure of the samples with change-points being $\boldsymbol{v}$, and let $\mathbb{E}_{\boldsymbol{v}}$ denote the corresponding expectation.

For any stopping rule τ, we define the worst-case average run length (WARL) to false alarm:

$$\text{WARL}(\tau) = \inf_{\boldsymbol{v}: v_\eta = \infty} \mathbb{E}_{\boldsymbol{v}}[\tau].$$

For any fixed $\{d_\eta, d_{\eta+1}, \ldots, d_{L-1}\}$, we further define the worst-case average detection delay (WADD) under Pollak's criterion [14]:

$$J_{\text{P}}[\tau] = \sup_{\boldsymbol{v}: v_1 \leq \cdots \leq v_\eta < \infty} \mathbb{E}_{\boldsymbol{v}}[\tau - v_\eta | \tau \geq v_\eta].$$

Our goal is to construct a distributed algorithm that achieves

$$\inf_{\tau} \quad J_{\text{P}}[\tau], \quad \text{s.t.} \quad \text{WARL}(\tau) \geq \gamma. \tag{2}$$

3. THE S-CUSUM ALGORITHM

In this section, we briefly review the S-CuSum algorithm for the centralized setting [7].

The quickest detection problem in Section 2 is reformulated as a dynamic hypothesis testing problem, i.e., to distinguish the following two hypotheses at each time k:

$$\begin{aligned} \mathcal{H}_0[k] &: \sum_{i=1}^{L} \mathbb{1}_{\{v_i \leq k\}} < \eta, \\ \mathcal{H}_1[k] &: \sum_{i=1}^{L} \mathbb{1}_{\{v_i \leq k\}} \geq \eta. \end{aligned} \tag{3}$$

The null hypothesis corresponds to the scenario in which there are less than η affected nodes at time k, the alternative hypothesis corresponds to the scenario in which there are η or more affected nodes at time k, and both the null and alternative hypotheses are composite.

The generalized log-likelihood ratio for this composite hypothesis testing problem is computed as follows:

$$W[k] = \log \left(\frac{\max\limits_{\boldsymbol{v}: \sum_{i=1}^{L} \mathbb{1}_{\{v_i \leq k\}} \geq \eta} \mathbb{P}_{\boldsymbol{v}}(\boldsymbol{X}[1,k])}{\max\limits_{\boldsymbol{v}: \sum_{i=1}^{L} \mathbb{1}_{\{v_i \leq k\}} < \eta} \mathbb{P}_{\boldsymbol{v}}(\boldsymbol{X}[1,k])} \right), \tag{4}$$

where $\boldsymbol{X}[k] = \{X_1[k], \ldots, X_L[k]\}$, and $\boldsymbol{X}[k_1, k_2] = \{\boldsymbol{X}[k_1], \ldots, \boldsymbol{X}[k_2]\}$. The corresponding stopping time is then given by comparing $W[k]$ against a pre-determined positive threshold: $\tilde{\tau}(b) = \inf\{k \geq 1 : W[k] > b\}$, where b is selected to meet the false alarm constraint.

It was shown in [7] that the above test is equivalent to

$$\hat{\tau}(b) = \inf \left\{ k \geq 1 : \min_{\substack{S \subseteq \{1,\ldots,L\} \\ |S| = L - \eta + 1}} \sum_{i \in S} W_i[k] \geq b \right\}, \tag{5}$$

where $W_i[k] = \max_{1\le v_i\le k} \sum_{j=v_i}^{k} \log \frac{g_1(X_i[k])}{g_0(X_i[k])}$, is the local CuSum statistic at node i and time k [8]. We refer to (5) as the S-CuSum algorithm. Here, it is clear that S-CuSum first determines the smallest $L-\eta+1$ local CuSums, computes their sum, and then compares the sum to a threshold b. The S-CuSum algorithm was shown to be first-order asymptotically optimal [7, Theorem 4].

4. DISTRIBUTED S-CUSUM

In this section, we propose a distributed algorithm for the problem in Section 2. The distributed algorithm is a three-step procedure. The first step is to distributedly learn which nodes have the smallest $L-\eta+1$ local CuSums using distributed optimization [11, 12], the second step is to compute the average of the smallest $L-\eta+1$ local CuSums using average consensus [13], and the third step is to decide whether to stop by comparing the local statistic to a threshold b. Such a procedure is repeated every time a new sample is observed, until an alarm is triggered.

Step 1: Distributed optimization. If we denote by $\boldsymbol{W}[k] = [W_1[k], W_2[k], \ldots, W_L[k]]^\top$, and $\boldsymbol{\alpha}^{(k)} \in \mathbb{R}^L$, then the S-CuSum statistic in (5) is equivalent to:

$$\begin{aligned} \min_{\boldsymbol{\alpha^{(k)}}} \ & (\boldsymbol{\alpha}^{(k)})^\top \boldsymbol{W}[k], \\ \text{s.t.} \ & \boldsymbol{\alpha}^{(k)}(i) \in \{0,1\}, \forall 1 \le i \le L, \\ & \mathbf{1}^\top \boldsymbol{\alpha}^{(k)} = L-\eta+1, \end{aligned} \tag{6}$$

where $\boldsymbol{\alpha}^{(k)}(j)$ is the j-th element of $\boldsymbol{\alpha}^{(k)}$. We define a set

$$C := \{\boldsymbol{\alpha} \in \mathbb{R}^L : \boldsymbol{\alpha}(j) \in [0,1], \mathbf{1}^\top \boldsymbol{\alpha} = L-\eta+1\},$$

and further define the following function

$$\theta_C(\boldsymbol{\alpha}) = \begin{cases} 0, & \text{if } \boldsymbol{\alpha} \in C, \\ \infty, & \text{otherwise.} \end{cases} \tag{7}$$

In the distributed setting, let $\boldsymbol{\alpha}_i^{(k)} \in \mathbb{R}^L$ denote the local copy of the optimization variable at node i and time k, then following steps similar to those in [11], we can show that (6) is equivalent to

$$\begin{aligned} \min_{\boldsymbol{\alpha_1^{(k)}}, \boldsymbol{\alpha_2^{(k)}}, \ldots, \boldsymbol{\alpha_L^{(k)}}, \boldsymbol{Z}_{i,j}} \ & \sum_{i=1}^{L} \left(\boldsymbol{\alpha}_i^{(k)}(i) W_i[k] + \theta_C(\boldsymbol{\alpha}_i^{(k)}) \right), \\ \text{s.t.} \ & \boldsymbol{\alpha}_i^{(k)} = \boldsymbol{\alpha}_j^{(k)} = \boldsymbol{Z}_{i,j}, \forall (i,j) \in E, \end{aligned} \tag{8}$$

where $\boldsymbol{Z}_{i,j}$ is an auxiliary variable imposing the consensus constraint on neighboring nodes i and j. Since it is assumed that G is connected, the last condition in the constraint means that $\boldsymbol{\alpha}_i^{(k)} = \boldsymbol{\alpha}_j^{(k)}, \forall 1 \le i, j \le L$.

Let $f_i^{(k)}(\boldsymbol{\alpha}_i^{(k)}) = \boldsymbol{\alpha}_i^{(k)}(i) W_i[k] + \theta_C(\boldsymbol{\alpha}_i^{(k)})$. Then, (8) can be rewritten as

$$\begin{aligned} \min_{\boldsymbol{\alpha_1^{(k)}}, \boldsymbol{\alpha_2^{(k)}}, \ldots, \boldsymbol{\alpha_L^{(k)}}, \boldsymbol{Z}_{i,j}} \ & \sum_{i=1}^{L} f_i^{(k)}(\boldsymbol{\alpha}_i^{(k)}), \\ \text{s.t.} \ & \boldsymbol{\alpha}_i^{(k)} = \boldsymbol{\alpha}_j^{(k)} = \boldsymbol{Z}_{i,j}, \forall (i,j) \in E. \end{aligned} \tag{9}$$

We note that different from [11], $f_i^{(k)}$ depends on the local CuSum statistic $W_i[k]$, which changes with k. A straightforward approach is to update $W_i[k]$ at each time $k, \forall 1 \le i \le L$, and then apply the ADMM algorithm in Algorithm 1 with arbitrary initializations of $\boldsymbol{\lambda}_i^{(0)}$ and $\boldsymbol{\delta}_i^{(0)}$, $1 \le i \le L$ until it converges, by choosing a large enough t_0. We note that $\nabla f_i(\boldsymbol{\alpha})$ is the gradient of f_i at $\boldsymbol{\alpha}$ if f_i is differentiable, or is a subgradient if $f_i^{(k)}$ is non-differentiable. The minimizers and multipliers for ADMM are then given by $\boldsymbol{\alpha}_i^{(k)} = \boldsymbol{\lambda}_i^{(t_0)}$, and $\boldsymbol{\beta}_i^{(k)} = \boldsymbol{\delta}_i^{(t_0)}$, for $1 \le i \le L$. However, this approach is inefficient in that it has to repeatedly apply Algorithm 1 with independent initializations at each time k with a large t_0. This will require many rounds of computations and local information exchanges.

Algorithm 1 ADMM [11]

Input & Initialization:
$\left\{\boldsymbol{\lambda}_i^{(0)}, \boldsymbol{\delta}_i^{(0)}, f_i\right\}_{i=1}^{L}, t_0$
c : algorithm parameter
$t \leftarrow 0$
Method:
while $t < t_0$ **do**
  for $i = 1$ to L **do**
$\boldsymbol{\lambda}_i^{(t+1)} \leftarrow (\nabla f_i + 2c|\mathcal{N}_i|I)^{-1}\left(c|\mathcal{N}_i|\boldsymbol{\lambda}_i^{(t)} + c\sum_{j\in\mathcal{N}_i} \boldsymbol{\lambda}_j^{(t)} - \boldsymbol{\delta}_i^{(t)}\right)$
  end for
  for $i = 1$ to L **do**
$\boldsymbol{\delta}_i^{(t+1)} \leftarrow \boldsymbol{\delta}_i^{(t)} + c\left(|\mathcal{N}_i|\boldsymbol{\lambda}_i^{(t+1)} - \sum_{j\in\mathcal{N}_i} \boldsymbol{\lambda}_j^{(t+1)}\right)$
  end for
  $t \leftarrow t+1$
end while
Return $\left\{\boldsymbol{\lambda}_i^{(t_0)}, \boldsymbol{\delta}_i^{(t_0)}\right\}_{i=1}^{L}$

To construct an efficient algorithm, we use the output of the ADMM algorithm at time k as the initialization of the ADMM algorithm at time $k+1$, i.e., $\boldsymbol{\lambda}_i^{(0)} \leftarrow \boldsymbol{\alpha}_i^{(k)}$ and $\boldsymbol{\delta}_i^{(0)} \leftarrow \boldsymbol{\beta}_i^{(k)}$ (see Algorithm 3). The idea is that although $f_i^{(k)}$ changes with time, the minimizers $\boldsymbol{\alpha}_i^{(k)}$, $1 \le i \le L$ do not change very fast with k, for those entries corresponding to the affected nodes. Therefore, a smaller t_0 should be able to guarantee accurate recovery of local $\boldsymbol{\alpha}_i^{(k)}$, for $1 \le i \le L$.

Step 2: Average consensus. Suppose $\boldsymbol{M} \in \mathbb{R}^{L\times L}$ is a matrix satisfying the following conditions.

- $\boldsymbol{M} \cdot \mathbf{1} = \mathbf{1}$.
- $\boldsymbol{M}^\top = \boldsymbol{M}$.
- $\boldsymbol{M}_{ij} > 0$ if node i and node j are one-hop neighbors, and $\boldsymbol{M}_{ij} = 0$, otherwise.

Algorithm 2 AvgCon

Input & Initialization:

$\left\{d_i^{(k)}, \boldsymbol{\alpha}_i^{(k+1)}, \boldsymbol{\alpha}_i^{(k)}, W_i[k+1], W_i[k]\right\}_{i=1}^{L}, t_1$

$t \leftarrow 0$

Method:

For $1 \leq i \leq L$,

$$d_i^{(k+1)} \leftarrow \sum_{j \in \mathcal{N}_i} \boldsymbol{M}_{i,j} \Big(d_j^{(k)} + \boldsymbol{\alpha}_j^{(k+1)}(j) W_j[k+1] - \boldsymbol{\alpha}_j^{(k)}(j) W_j[k] \Big)$$

while $t \leq t_1$ **do**

$d_i^{(k+1)} \leftarrow \sum_{j \in \mathcal{N}_i} \boldsymbol{M}_{i,j} d_i^{(k+1)}$, for $1 \leq i \leq L$

$t \leftarrow t + 1$

end while

Return $\left\{d_i^{(k+1)}\right\}_{i=1}^{L}$

- The second largest eigenvalue modulus is less than 1.

After the first step, each node i has its local $\boldsymbol{\alpha}_i^{(k)}$, i.e., each node has an estimate of which nodes have the smallest $L - \eta + 1$ local CuSums. But every node does not know the CuSum statistics of other nodes. To compute the average of the smallest $L - \eta + 1$ local CuSums, we then apply the average consensus (AvgCon) algorithm, which is described in Algorithm 2. We note that as t_1 increases, $d_i^{(k+1)}$ converges to $\sum_j \boldsymbol{\alpha}_j^{(k+1)}(j) W_j[k+1]/L$, for $1 \leq i \leq L$.

Step 3: Stopping rule. The algorithm stops and raises an alarm if any of the $d_i^{(k)}$ goes above a pre-specified threshold b, i.e., the stopping time is $\tau_d = \inf\left\{k : \max_{1 \leq i \leq L} d_i^{(k)} \geq b\right\}$.

The distributed S-CuSum algorithm is summarized in Algorithm 3.

5. NUMERICAL RESULTS

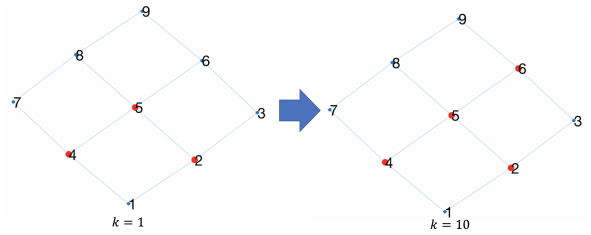

Fig. 1. A growing event in a lattice network.

In this section, we present numerical results showing the comparison between the proposed distributed S-CuSum (Algorithm 3) and the centralized S-CuSum [7] algorithms.

We consider a 3×3 lattice network with 9 nodes (see Fig.1). We set $\eta = 2$, $g_0 = \mathcal{N}(0,1)$ and $g_1 = \mathcal{N}(1,1)$. To simulate the average detection delay (ADD), we assume that nodes 2, 4, 5 are affected at time 1, node 6 is then affected at time 10, and no other nodes are affected later. To simulate the average run length (ARL) to false alarm, we assume that nodes 4 and 5 are affected at time 1, and no other nodes are affected later. For the distributed S-CuSum, we set $t_0 = 1$ in the ADMM, and $t_1 = 3$ in the AvgCon. We set $c = 0.5$ for the ADMM algorithm. We plot ADD versus ARL in Fig. 2 by varying the threshold.

We observe from Fig. 2 that the distributed S-CuSum has a slight loss in performance compared to the centralized S-CuSum algorithm, which can be mitigated by increasing t_0 and t_1. Furthermore, it should be noted that with a larger network, t_0 and t_1 may need be increased to achieve good performance.

Algorithm 3 Distributed S-CuSum

Input & Initialization:

$\left\{\boldsymbol{\alpha}_i^{(0)}, \boldsymbol{\beta}_i^{(0)}\right\}_{i=1}^{L}, d_i^{(0)} \leftarrow 0, 1 \leq i \leq L$

c, t_0, t_1

$k \leftarrow 0, \tau_d \leftarrow 0$

Method:

while $\tau_d = 0$ **do**

Observe $X_i[k+1]$, for $1 \leq i \leq L$

$W_i[k+1] = (W_i[k])^+ + \log \frac{g_1(X_i[k+1])}{g_0(X_i[k+1])}$, for $1 \leq i \leq L$

$\left\{\boldsymbol{\alpha}_i^{(k+1)}, \boldsymbol{\beta}_i^{(k+1)}\right\}_{i=1}^{L} \leftarrow \text{ADMM}\left(\left\{\boldsymbol{\alpha}_i^{(k)}, \boldsymbol{\beta}_i^{(k)}, f_i^{(k)}\right\}_{i=1}^{L}, t_0, c\right)$

$\left\{d_i^{(k+1)}\right\}_{i=1}^{L} \leftarrow \text{AvgCon}\left(\left\{d_i^{(k)}, \boldsymbol{\alpha}_i^{(k+1)}, \boldsymbol{\alpha}_i^{(k)}, t_1\right\}_{i=1}^{L}, t_1\right)$

if $\max_{1 \leq i \leq L} d_i^{(k+1)} \geq b$ **then**

$\tau_d = k + 1$

end if

$k \leftarrow k + 1$

end while

Return τ_d

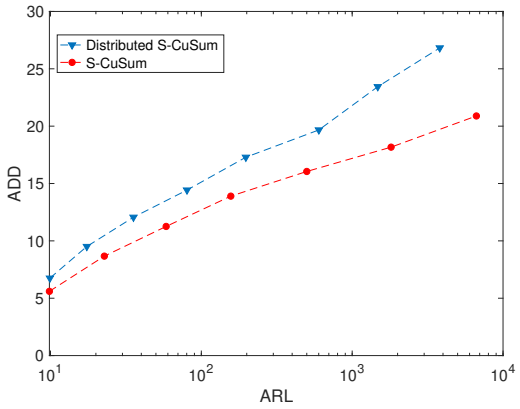

Fig. 2. Comparison between the distributed S-CuSum and the centralized S-CuSum.

6. REFERENCES

[1] N. A. Christakis and J. H. Fowler, "Social network sensors for early detection of contagious outbreaks," *PLOS ONE*, vol. 5, no. 9, pp. e12948, Sept. 2010.

[2] F. Pervaiz, M. Pervaiz, N. A. Rehman, and U. Saif, "Flubreaks: Early epidemic detection from Google flu trends," *Journal of Medical Internet Research*, vol. 14, no. 5, Apr. 2012.

[3] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer, "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," *Information Security Technical Report*, vol. 14, no. 1, pp. 16–29, Feb. 2009.

[4] B. Manly and D. Mackenzie, "CUSUM environmental monitoring in time and space," *Environmental and Ecological Statistics*, vol. 10, no. 2, pp. 231–247, 2003.

[5] S. Russell and T. Abdelzaher, "The Internet of battlefield things: The next generation of command, control, communications and intelligence decision-making," in *Proc. Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 737–742.

[6] S. Zou and V. V. Veeravalli, "Quickest detection of dynamic events in sensor networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Alberta, Canada, April 2018.

[7] S. Zou, V. V. Veeravalli, J. Li, and D. Towsley, "Quickest detection of dynamic events in networks," *Submitted to IEEE Transactions on Information Theory*, 2018.

[8] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, pp. 100–115, June 1954.

[9] Q. Liu and Y. Xie, "Distributed change detection based on average consensus," *arXiv preprint arXiv: 1710.10378*, 2017.

[10] S. S. Stankovic, N. Ilic, M. S. Stankovic, and K. H. Johansson, "Distributed change detection based on a consensus algorithm," *IEEE Trans. Signal Proc.*, vol. 59, no. 12, pp. 5686–5697, Dec. 2011.

[11] W. Shi, Q. Ling, K. Yuan, G. Wu, and W. Yin, "On the linear convergence of the ADMM in decentralized consensus optimization.," *IEEE Trans. Signal Proc.*, vol. 62, no. 7, pp. 1750–1761, 2014.

[12] Q. Li, B. Kailkhura, R. Goldhahn, P. Ray, and P. K. Varshney, "Robust decentralized learning using ADMM with unreliable agents," *ArXiv: 1710.05241*, 2017.

[13] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[14] M. Pollak, "Optimal detection of a change in distribution," *The Annals of Statistics*, pp. 206–227, Mar. 1985.