Applicability and Resilience of a Linear Encoding Scheme for Computing Consensus

Michel Toulouse and Bùi Quang Minh
Vietnamese-German University, Binh Duong New City, Vietnam

Keywords: Consensus Algorithms, Observability Theory, Attack Mitigation, Edge Computing.

Abstract: Consensus algorithms have a set of network nodes converge asymptotically to a same state which depends on some function of their initial states. At the core of these algorithms is a linear iterative scheme where each node updates its current state based on its previous state and the state of its neighbors in the network. In this paper we review a proposal from control theory which uses linear iterative schemes of asymptotic consensus and observability theory to compute consensus states in a finite number of iterations. This proposal has low communication requirements which makes it attractive to address consensus problems in a limited resource environment such as edge computing. However it assumes static networks contrary to wireless edge computing networks which are often dynamic and prone to attacks. The main purpose of this paper is to assess the network conditions and attack scenarios where this algorithm can still be considered useful in practice to address consensus problems in IoT applications. We introduce some new lower and exact bounds which further improve the communication performance of the algorithm. We also have some technical contributions on how to speed up mitigation of malicious activities and handling network instabilities. Numerical results confirm the communication performance of the algorithm and the existence of scenarios where the system can be considered cost effective resilient to errors injected intentionally or unintentionally.

1 INTRODUCTION

Consensus problems arise whenever a set of network nodes seek to agree on a same value that depends on parameters distributed across a network. They are coordination problems such as rendezvous (Cortes et al., 2006), agreement problems (Saber and Murray, 2003), leader-following problems (Peng and Yang, 2009), distributed fusion of sensor data (Xiao et al., 2005), distributed optimization problems (Tsitsiklis et al., 1986) and many others. Consensus problems are often hidden in larger applications. Recently, blockchain and distributed update consensus algorithms have been proposed to address security and privacy of IoT nodes data exchanges (Christidis and Devetsikiotis, 2016; Bahga and Madisetti, 2016). We also find consensus problems in service-oriented IoT applications (Li et al., 2014), IoT resource allocation (Colistra et al., 2014), and others (Pilloni et al., 2017; Carvin et al., 2014). More consensus problems are likely to arise as IoT expands its applications, some of which, such as smart cities, because of security and the sheer volume of data generated, may require close to the sources data fusion and analytic.

At the core of consensus algorithms is a linear iterative scheme where each node updates its current state based on its previous state and the state of its neighbors in the network. Assume the consensus function f is for each node to compute the average sum of the initial states, i.e. $f = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$, for n is the number of network nodes and $x_i(0)$ the initial state of node i. Let also assume the computer network is modeled as an undirected graph $G = (V,E)$ where $V = \{1,2,\ldots,n\}$ stands for the set of nodes, and where the edge set E represents network links. Let $\mathcal{N}_i$ denotes the set of nodes adjacent to node i in G. The average sum consensus problem is solved by a distributed average consensus algorithm where each node i computes iteratively the following local average sum:

$$x_i(k+1) = \frac{1}{|\mathcal{N}_i|+1}(x_i(k) + \sum_{j\in\mathcal{N}_i} x_j(k)), \qquad (1)$$

Local averages in (1) are initially poor approximations of the true average sum as they are based on a partial sum of the initial states. However, local averages get increasingly closer to the true average as further iterations carry information from nodes that are further away from node i. The local average rule

Toulouse, M. and Quang Minh, B.
Applicability and Resilience of a Linear Encoding Scheme for Computing Consensus.
DOI: 10.5220/0006806001730184
In *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018)*, pages 173-184
ISBN: 978-989-758-296-7

in Equation (1) can be expressed in a more general form:

$$x_i(k+1) = W_{ii}x_i(k) + \sum_{j \in \mathcal{N}_i} W_{ij}x_j(k), \qquad (2)$$

where W_{ij} holds the coefficient of a weighted linear combination of x_i and the values x_j, $j \in \mathcal{N}_i$ (note, the properties of W are summarized in the APPENDIX). Equation (2) is the core linear model found in all discrete linear consensus algorithms. The corresponding network wide update rule is as followed:

$$x(k+1) = Wx(k), k = 0, 1, \ldots \qquad (3)$$

where $x(k)$ represents the state of the nodes at iteration k ($x(k) = [x_1(k), x_2(k), \ldots, x_n(k)]^T$).

In this paper we consider a consensus algorithm that we call *exact consensus* because consensus functions are computed exactly in a finite number of iterations. This algorithm has first been proposed in (Sundaram and Hadjicostis, 2008). It is based on the same linear iterative scheme as in Equation (2) but interpreted differently. For asymptotic consensus, Equation (2) is a computational step in a distributed algorithm, it computes a refined approximation of the consensus state for node i. For exact consensus, Equation (2) is a linear encoding scheme.

This interpretation of the linear iterative scheme in Equation (2) as an encoding scheme is analogue to network coding protocols (Ahlswede et al., 2000) where, at the network layer, packets from different sources are recombined into a single packet to improve network's throughput. In Equation (2), states $x_j(k), j \in \mathcal{N}_i$, and state $x_i(k)$ are combined at iteration k to yield a new state $x_i(k+1)$. State $x_i(k+1)$ encodes the states of iteration k which is then forwarded to nodes $j \in \mathcal{N}_i$ at iteration $k+1$.

The second important contribution making exact consensus possible is the parallel drawn between linear iterative rules such Equation (2) and *observability theory* (Sundaram and Hadjicostis, 2013). Observability theory is not foreign to consensus algorithms. It has been a central theme in approaches proposed by researchers in the control theory community to address security issues of asymptotic consensus algorithms (Pasqualetti et al., 2009; Teixeira et al., 2010; Silvestre et al., 2013). Observer-based techniques in process control which detect faulty system components are adapted to detect and isolate data falsification attacks on asymptotic consensus iterations (Pasqualetti et al., 2007).

In the context of exact consensus, observability theory is applied to recover in a finite number of steps the only guaranteed uncorrupted state of the system, i.e. state $x(0)$ (in this paper we assume the initial states are genuine). The state $x(0)$ is inferred by a node i from observations of states $x_j(k), j \in \mathcal{N}_i$ that flow through node i at each iteration of Equation (2). Once the initial states have been recovered by a node i, the consensus function is computed locally.

This paper is structured as follow. Section 2 briefly describes the exact consensus algorithm. In Section 3 we introduce new lower and exact bounds on the number of communication steps for the algorithm, as well as a few technical approaches to compute these bounds. Section 4, briefly summarizes the algorithm in (Sundaram and Hadjicostis, 2011), which is an extension of the algorithm in Section 2 to mitigate the hacking of some nodes by malicious actors. Section 5 is an assessment of the performance and potential of both algorithms based on numerical tests and technical proposals to improve the resilience of exact consensus. Section 6 concludes.

2 EXACT CONSENSUS

According to control theory, a system is observable to an external observer if its internal states can be inferred in finite time from observing a sequence of the system's outputs. In the context of the exact consensus algorithm, the internal system states are $x(k)$, and the external observers are each node in the network. Each node observes a sequence of states that flows directly through it, i.e. its own state and the states of its neighbors. Then each node infers $x(0)$, from the sequence of observations. Therefore we divide the presentation of the exact consensus algorithm into two sections. The *observation phase* deals with the recording of states that flow through a node. The *recovery phase* is related to the recovery of initial states from the stored encoded data.

2.1 Observation Phase

Let x be a state vector of n entries representing the current state of an n network nodes, and W a weight matrix. The following linear system models how information is recorded at node i:

$$\begin{aligned} x(k+1) &= Wx(k) \\ y_i(k) &= C_i x(k) \end{aligned} \qquad (4)$$

The relevant quantity in Equation (4) is $y_i(k) = C_i x(k)$ which specifies the values of $x(k)$ observed by node i. At each iteration k, $y_i(k)$ stores $(deg_i + 1)$ states, i.e. $x_i(k)$ and $x_j(k) \in \mathcal{N}_i$ ($deg_i = |\mathcal{N}_i|$). Notations $y_i(k-1)$ and $y_i(k)$ refer to two different sets of entries in y_i. The size of $y_i(0 : v-1)$ is $(deg_i + 1)v$, where v is the length of the sequence of observations performed by node i. C_i is a $(deg_i + 1) \times n$ matrix with a single 1 in

each row, indicating the value of vector $x(k)$ stored in $y_i(k)$ (entry $C_i[k,j] = 1$ if $j \in \mathcal{N}_i$, otherwise $C_i[k,j] = 0$).

As an example, consider the ring network in Figure 1 and the system described in Equation (4).

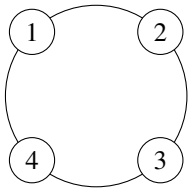

Figure 1: Ring Network.

Assume the weight matrix W is the same Metropolis-Hasting matrix (16) for computing asymptotic average consensus. For a ring of 4 nodes, the coefficients of the Metropolis-Hasting matrix are the following:

$$W = \begin{bmatrix} 0.33 & 0.33 & 0 & 0.33 \\ 0.33 & 0.33 & 0.33 & 0 \\ 0 & 0.33 & 0.33 & 0.33 \\ 0.33 & 0 & 0.33 & 0.33 \end{bmatrix}$$

We focus on node 1 as external observer. The corresponding C_1 matrix is as followed:

$$C_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

The entries of y_1 are obtained as follow. We know from linear algebra that $x(k) = W^k x(0)$ for W^k the product of matrix W with itself k times. Therefore, $y_1(0) = C_1 W^0 x(0) = [x_1, x_2, x_4]^T$, $y_1(1) = C_1 W x(0)$, $y_1(2) = C_1 W^2 x(0)$ and $y_1(v-1) = C_1 W^{v-1} x(0)$. After v observations, the column vector y_1 would have stored the $3v$ states that have passed through node 1.

The observation sequence of node i fills the column vector $y_i(0 : v-1)$. As the example above suggests, the observation sequence is a mapping $x(0) \rightarrow y_i(0 : v-1)$ that can be represented in matrix form called the *observability matrix*, denoted as $O_{i,v-1}$:

$$O_{i,v-1} = \begin{bmatrix} C_i \\ C_i W \\ C_i W^2 \\ \vdots \\ C_i W^{v-1} \end{bmatrix} \tag{5}$$

The observability matrix for node 1 in the ring network of Figure 1 and $v = 2$ is $O_{1,1}$ =

$$\begin{bmatrix} C_1 \\ C_1 W \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0.33 & 0.33 & 0 & 0.33 \\ 0.33 & 0.33 & 0.33 & 0 \\ 0.33 & 0 & 0.33 & 0.33 \end{bmatrix}$$

$O_{i,v-1}$ is a mathematical expression that computes the data recorded during the observation sequence at node i:

$$\begin{bmatrix} y_i(0) \\ y_i(1) \\ y_i(2) \\ \vdots \\ y_i(v_i-1) \end{bmatrix} = \begin{bmatrix} C_i \\ C_i W \\ C_i W^2 \\ \vdots \\ C_i W^{v_i-1} \end{bmatrix} x(0) \tag{6}$$

The observability matrix is needed to decode the data in *observation vector* $y_i(0 : v-1)$ in order to recover $x(0)$ during the recovery phase.

2.2 Recovery Phase

After the observation phase, the column vector $y_i(0 : v-1)$ of node i stores the initial state of all the other nodes in the network, though this information is stored in an encoded form. In order to compute a consensus function f, either f is mapped into an equivalent function f_i' that can be applied on the arguments provided by y_i or a decoding function is applied on y_i that recovers the initial state $x(0)$.

2.2.1 Mapping f into f_i'

The consensus f is mapped into an equivalent function f_i' expressed in matrix form Γ_i. First f is written in a matrix form Q. From linear algebra, if Q is in the row space of the matrix $O_{i,v-1}$, then linearly independent rows of $O_{i,v-1}$ can be treated as a base for Q. The matrix Q is then expressed in terms of this new base. The new expression of Q is found by solving the following system of linear equations:

$$O_{i,v-1}^T \Gamma_i^T = Q^T \tag{7}$$

where Γ_i computes $f(x(0))$ using the arguments available in the observation vector $y_i(0 : v-1)$:

$$\Gamma_i \begin{bmatrix} y_i(0) \\ y_i(1) \\ y_i(2) \\ \vdots \\ y_i(v-1) \end{bmatrix} = \Gamma_i O_{i,v-1} x(0) = Q x(0). \tag{8}$$

As an example consider the consensus function $f = \frac{1}{n}\sum_{i=1}^{4} x_i(0)$ and the four nodes ring network in Figure 1. In matrix form, f is the row vector $Q = [0.25, 0.25, 0.25, 0.25]$. Solving for Q and node 1, we have the system of linear equations $O_{1,2}^T \Gamma_1^T = Q^T$:

$$\begin{bmatrix} 1 & 0 & 0 & 0.33 & 0.33 & 0.33 \\ 0 & 1 & 0 & 0.33 & 0.33 & 0 \\ 0 & 0 & 0 & 0 & 0.33 & 0.33 \\ 0 & 0 & 1 & 0.33 & 0 & 0.33 \end{bmatrix} \Gamma_1^T = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}$$

This system has infinitely many solutions, one is $[-0.66,-0.33,-0.75,2,-0.25,1]$ which expresses Q in terms of a set of linearly independent rows of $O_{1,2}$ as a new base for Q.

Assume $x(0) = [1,2,3,4]$, therefore $f = \frac{1}{n}\sum_{i=1}^{4} x_i(0) = 2.5$. We consider again the case of node 1. During the observation phase, after the first observation, the content of the column vector $y_1(0)$ is

$$y_1(0) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$$

At the second observation, the values stored in $y_1(1) = C_1 W x(0)$

$$\begin{bmatrix} 0.33 & 0.33 & 0 & 0.33 \\ 0.33 & 0.33 & 0.33 & 0 \\ 0.33 & 0 & 0.33 & 0.33 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 2.31 \\ 2.0 \\ 2.66 \end{bmatrix}$$

After two observations, the column vector $y_1^T = [1,2,4,2.31,2,2.66]$. According to Equation (8) the consensus value can be computed as $\frac{1}{n}\sum_{i=1}^{4} x_i(0) = \Gamma_i y_i$. Indeed, the product of $\Gamma_1 y_1 = 2.46 \approx 2.5$, the difference is accounted for by rounding errors.

2.2.2 Recovering the Initial State $x(0)$ from y_i

The procedure to recover $x(0)$ simply consists to solve Equation (8) for the identity matrix I_n, assuming I_n is the row space of $O_{i,v-1}$. If this the case, then I_n is expressed in terms of a linear combination of rows in $O_{i,v-1}$, from which $\Gamma_i y_i$ returns $x(0)$. Consider the four nodes ring network in Figure 1, to compute the average sum function, first the identity matrix I_4 is expressed in a base provided by the $O_{1,2}$ matrix, by solving the system of linear equations $O_{1,2}^T \Gamma_1^T = I_4$:

$$\begin{bmatrix} 1 & 0 & 0 & 0.33 & 0.33 & 0.33 \\ 0 & 1 & 0 & 0.33 & 0.33 & 0 \\ 0 & 0 & 0 & 0 & 0.33 & 0.33 \\ 0 & 0 & 1 & 0.33 & 0 & 0.33 \end{bmatrix} \Gamma_1^T = I_4$$

The solution of this system of equations is as follows:

$$\Gamma_1^T = \begin{bmatrix} 0.917 & -0.028 & -0.836 & -0.082 \\ -0.083 & 0.953 & -0.336 & -0.033 \\ -0.083 & -0.008 & -0.336 & 0.869 \\ 0.250 & 0.083 & -0.497 & 0.249 \\ 0.000 & 0.058 & 1.515 & -0.149 \\ 0.000 & -0.058 & 1.515 & 0.149 \end{bmatrix}$$

Assuming the initial values $x_i(0)$ of nodes 1 to 4 are the same as above, then $y_1^T = [1,2,4,2.31,2,2.66]$, the same as for the average sum example. One can verify that $\Gamma_1 y_1 = [0.9965,1.9994,3.0598,4.0015]$, rounding errors accounting for the differences. Node 1 has then learned the initial state of the system, therefore node 1 can computed directly the average sum using the $f = \sum_{i=1}^{n} x_i(0)$ or almost any other function of $x(0)$.

The difference between mapping f to f' and recovering $x(0)$ from y_i is mainly in the number of observations needed to compute each approach. As the number of columns in I_n is the same as the number of columns in $O_{i,v-1}$, for I_n to be in the row space of $O_{i,v-1}$, the matrix $O_{i,v-1}$ has to be full column-rank. This usually requires more observations compared to solving for a matrix representation of the function f. On the other hand, computing the consensus function f directly from the arguments in y_i requires a mapping $f \rightarrow f'$ for each consensus function, which could be tedious. It is more simple to recover $x(0)$, from which any consensus function can be computed.

The observability matrix $O_{i,v-1}$ as well as Γ_i are pre-computed for each node prior to the execution of the observation phase. The constitution of the observability matrix $O_{i,v-1}$ depends on the weight matrix W, the matrix C_i and the value of v which are known prior to the observation phase. Γ_i depends on the function Q and $O_{i,v-1}$ which are also known prior to the observation phase. Only Γ_i needs to be stored by node i, the observability matrix is not needed during or after the observation phase.

In this brief description of the exact consensus algorithm several issues have been omitted on relations existing between the value of v, the coefficients of the weight matrix W, the rank of the observability matrix and the existence of a base in the row space of $O_{i,v-1}$ for a given consensus function f. Discussions and results about these issues can be found in (Sundaram, 2009). One open issue is finding better bounds on v. We address this issue in the next section.

3 BOUNDS ON THE NUMBER OF OBSERVATIONS

The length v of the sequence of observations is a communication cost that exact consensus incurs prior to compute the consensus function. It is important to reduce the number of observations in order to improve the efficiency of the algorithm. In this section we briefly recall known results in the literature on upper bounds for v. Then, we address open issues regarding the more difficult task of defining lower and exact bounds on the size of $O_{i,v-1}$.

In this section, we only consider the general case of $Q = I$, where we focus on the recovery of the initial

state. We also assume that W is a random weight matrix. In order to recover the initial state of every node successfully, it is necessary and sufficient to have a full column-rank observability matrix $O_{i,v-1}$ for all i. It is proven that random weight matrices can generate a full column-rank $O_{i,v-1}$ for all i with probability equal to 1 (Sundaram, 2009).

3.1 Upper Bounds

The number of nodes n in the network is a trivial upper bound on the number of observations. According to the local update rule in Equation (1), at iteration $k = 0$, the initial state $x_i(0)$ of any node i is pushed to all its neighbors. For any node $j \in \mathcal{N}_i$, at iteration $k = 1$, the value $x_i(0)$ is pushed to all the neighbors of j, therefore after 2 iterations $x_i(0)$ has reached all the nodes at distance 2 for node i. In at most n iterations, $x_i(0)$, the initial value of node i, thought in an encoded form, will have been pushed to all the nodes in the network. In practice a better upper bound can be computed by noticing that a node i receives the initial values of all its neighbors in one iteration. Therefore, after the first iteration, each node i has received $deg_i + 1$ initial values (including its own initial value). This yields the following upper bound on v (Sundaram and Hadjicostis, 2008):

$$v \leq n - deg_i + 1 \tag{9}$$

3.2 Lower Bounds

A trivial lower bound on the dimension of $O_{i,v-1}$ is the diameter of the graph (Sundaram and Hadjicostis, 2008). The intuition is that if the initial value of some node j does not reach node i, it cannot recover the state of node j by any mean. We call this lower bound the *data availability* requirement. Indeed, if this requirement does not hold, this means there exists two nodes i and j where the observations of node i do not depend on $x_j(0)$. This translates into an observability matrix $O_{i,v-1}$ with all zero entries at the j-th column and therefore $O_{i,v-1}$ is not full column-rank.

However, the above lower bound is not tight in many cases. Consider a torus network such as the network of 25 nodes in Figure 2. The diameter of this graph is 4. As torus is a regular graph, the dimensions of the matrix C_i are the same for all nodes, it is 5×25 for the graph in Figure 2. If the value of v is set to 4, the column vector y_i for any node i is $5v = 20$, the dimension of $O_{i,v-1}$ is 20×25, this observability matrix has only 20 rows. Thus $O_{i,3}$ cannot be full column-rank as the number of rows is less then the number of columns.

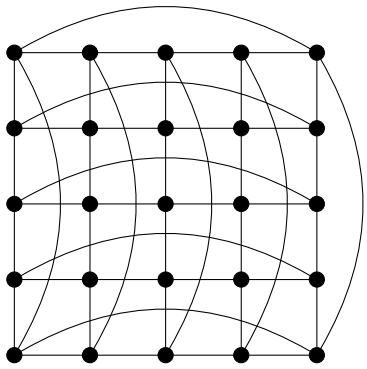

Figure 2: Torus network with 25 nodes.

This brings us to introduce a second condition on the minimum length the observation sequence must satisfy, we call this second condition the *intake capacity* requirement. In one observation, a node cannot gather more states than the number of links adjacent to it in the network. The lower bound according to this requirement could be formulated as $\lceil \frac{n}{\min_{i \in V}(\deg_i)+1} \rceil$, which implies that the observability matrix $O_{i,v-1}$ for any node would have at least n rows. A better bound can be obtained by eliminating the rows which are obviously linearly dependent on other rows. Notice that, by definition, the state $x_i(k)$ is a linear combination of its state $x_i(k-1)$ and the states of its neighbors $x_j(k-1), j \in \mathcal{N}_i$ of the previous iteration $k-1$. Therefore, at node i, $x_i(k)$ can be eliminated (except for the initial one $x_i(0)$), thus there are only at most $\deg_i$ independent states that are observed at each iteration. Put everything together, the lower bound corresponding to this requirement is:

$$v \geq \frac{n-1}{\min_{i \in V}(\deg_i)} \tag{10}$$

However, this requirement does not cover the data availability requirement. Consider the graph in Figure 3. In this case, the minimum degree is 2 and thus, the lower bound according to the intake capacity requirement is 3, but the diameter is 4. These two requirements generate two different lower bounds which both need to be satisfied. Neither of these two bounds is tight however for the graph in Figure 3 which requires empirically a sequence of 5 observations.

Tight lower bounds must also consider local properties and the topology of the graph. Intuitively, *bottlenecks* are one of those properties. Bottlenecks for a particular node are those nodes in the graph which limit the intake capacity. For example, in Figure 3, with respect to node 0, node 2 is a bottleneck. Though node 0 has two neighbors, this node can only extract one independent state observation of the part of the

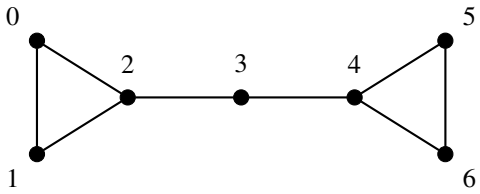

Figure 3: Example graph.

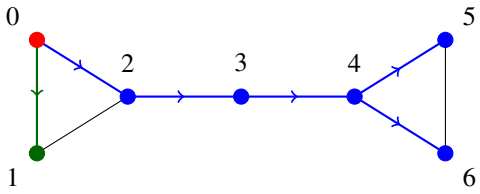

Figure 4: The spanning tree of the graph in Fig. 3.

graph beyond node 2 (i.e. nodes 3 to 6). To examine this analytically, consider $x_1(k), k \geq 1$, which is the linear combination of $x_j(k-1), j \in \{0,1,2\}$. All those three states derived from observations of node 0 at iteration $k-1$, therefore $x_1(k)$ always depends on existing observations of node 0 from iteration 1 and can be eliminated. This implies that observing the state of a same node twice would contribute nothing into recovering $x(0)$.

We derive a method to determine an exact lower bound on v for any graph, which is correct empirically, but with no proof. This method was also mentioned in (Sundaram, 2009) as a conjecture. For a specific node i, draw a spanning tree rooted at i which has the largest branch whose size is minimum. The number of observations requires at that particular node is equals to the size of that largest branch. For the graph in Figure 3, and node 0, the number of iterations (which is 5) could be explained by looking at Figure 4 (node 0 is colored in red, the blue branch is the largest one and contains 5 edges, the green branch is the other branch of the tree).

3.3 Exact Bound for Symmetric Graphs

A symmetric graph is a graph where any pair of adjacent nodes could be mapped to another pair of adjacent nodes (Babai, 1995). Formally, in a symmetric graph, for any two pairs of adjacent nodes (u_1, v_1) and (u_2, v_2), there exists an automorphism f such that $f(u_1) = u_2$ and $f(v_1) = v_2$. Such a graph is both vertex-transitive (implying regularity) and edge-transitive, and the connectivity of any node is exactly d where d is the degree of every node in the graph.

The lower bound defines by the intake capacity requirement is tight for symmetric graphs, which is $\lceil \frac{n-1}{d} \rceil$. While we still don't have a formal proof, the intuition could be explained using the result in (Sundaram, 2009), which states that the minimum size of the largest branch is the upper-bound on the number of observations with probability 1 (where the lower-bound is kept as a conjecture). In a symmetric graph, there exists a spanning tree whose branches have equal size or only have a difference of 1 (tree with balanced branches). Because a symmetric graph is a vertex-transitive graph (where any node has the same role), we only need to analyze a single node.

To draw such a spanning tree, consider any node as a reference node v_0. The other nodes are organized into layers based on the distance between them and the reference node. Assign the nodes at distance 1 different labels (branch label). For any node at distance $dist+1$, its label will be the union of the labels of its adjacent nodes at distance $dist$. This structure is shown in figure 5 and has some noticeable properties. First, because the connectivity of any node is exactly d, any layer (except for the last one) must have at least d nodes. Second, at any layer, the label counts of each branch are equal. This is because we can map any pair (v_0, v_1) to (v_0, v_2) where v_1, v_2 are any pair of nodes at distance 1, thus the labels of v_1 could be map bijectively to the labels of v_2. Based on these two properties, we can manually draw a tree with balanced branches, however we don't have yet an explicit procedure to draw this tree, though we confirmed this observation with many well-known symmetric graphs, as shown in figures 5, 6, and 7.

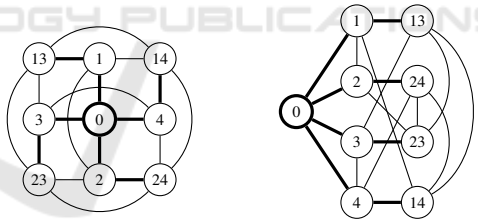

Figure 5: The distance-based representation of torus 3×3.

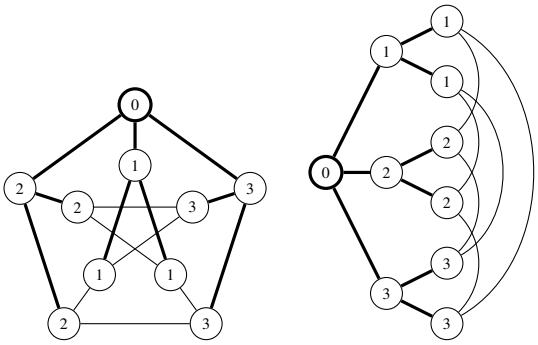

Figure 6: The distance-based representation of Petersen graph.

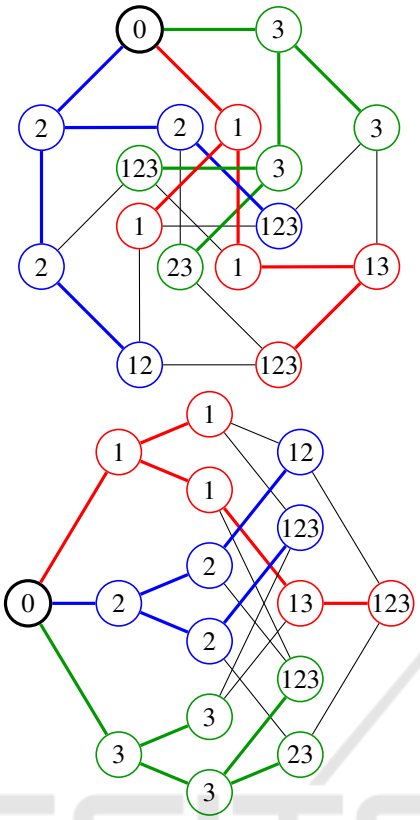

Figure 7: The distance-based representation of Möbius-Kantor graph.

4 RESILIENCE OF EXACT CONSENSUS

Implementations of the algorithm describe in Section 2 are susceptible to be hacked by malicious actors taking control of some nodes from which the current state of the nodes will be updated incorrectly. This form of attacks has been addressed in (Sundaram and Hadjicostis, 2011) where a solution approach is proposed. In this section we briefly describe the proposed attack model, the procedure detecting the existence of compromised nodes, the recovery procedure of $x(0)$ in the presence of malicious actors and finally the identification and removal of compromised nodes.

4.1 Modeling Malicious Activities

Attacks are modeled as an additive error $u_i(k)$ to the linear iterative Equation (2) at any given iteration k of node i :

$$x_i(k+1) = W_{ii}x_i(k) + \sum_{j\in\mathcal{N}_i} W_{ij}x_j(k) + u_i(k), \quad (11)$$

In matrix form, referring to the possibility that several nodes may be compromised, this is modeled as

$$x(k+1) = Wx(k) + B_{\mathcal{F}}u_{\mathcal{F}}(k) \quad (12)$$

where $\mathcal{F} = \{i_1, i_2, \ldots, i_f\}$ is the set of the compromised nodes, $B_{\mathcal{F}} = \begin{bmatrix} e_{i_1} & e_{i_2} & \cdots & e_{i_f} \end{bmatrix}$ (where e_l denotes a unit vector with entry l storing 1) and $u_{\mathcal{F}}(k) = \begin{bmatrix} u_{i_1}(k) & u_{i_2}(k) & \cdots & u_{i_f}(k) \end{bmatrix}^T$. From this, the observation vector of a node i at iteration k is written as a linear combination of the initial states and the injected additive errors:

$$y_i(0:k) \triangleq \begin{bmatrix} y_i(0) \\ y_i(1) \\ \vdots \\ y_i(k) \end{bmatrix} = O_{i,k}x(0) + \mathcal{M}_{i,k}^{\mathcal{F}} \begin{bmatrix} u_{\mathcal{F}}(0) \\ u_{\mathcal{F}}(1) \\ \vdots \\ u_{\mathcal{F}}(k-1) \end{bmatrix} \quad (13)$$

in which $\mathcal{M}_{i,k}^{\mathcal{F}}$ is the *fault matrix* corresponding to the subset $\mathcal{F}$ of compromised nodes, defined as:

$$\mathcal{M}_{i,k}^{\mathcal{F}} \triangleq \begin{bmatrix} 0 & 0 & \cdots & 0 \\ C_iB_{\mathcal{F}} & 0 & \cdots & 0 \\ C_iWB_{\mathcal{F}} & C_iB_{\mathcal{F}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ C_iW^{k-1}B_{\mathcal{F}} & C_iW^{k-2}B_{\mathcal{F}} & \cdots & C_iB_{\mathcal{F}} \end{bmatrix}$$

4.2 Detection

At the end of the observation phase, the algorithm run tests to detect malicious activities. Let κ be the vertex-connectivity of the graph. Assume that at some iteration L, for every subset $\mathcal{S}$ with size less then $\kappa/2$, $O_{i,L}$ and $\mathcal{M}_{i,L}^{\mathcal{S}}$ satisfy this condition:

$$\rho\left(\begin{bmatrix} O_{i,L} & \mathcal{M}_{i,L}^{\mathcal{S}} \end{bmatrix}\right) = n + \rho\left(\mathcal{M}_{i,L}^{\mathcal{S}}\right) \quad (14)$$

which means that every column in $O_{i,L}$ is linearly independent with the columns in $\mathcal{M}_{i,L}^{\mathcal{S}}$ and $O_{i,L}$ is full column-rank (ρ denotes the rank of a matrix). If the coefficients of the weight matrix W are generated randomly, this condition will be satisfied with probability 1. From Equation (13), if there is any compromised node in the system, the following test will fail:

$$\rho\left(\begin{bmatrix} O_{i,L} & y_i(0:L) \end{bmatrix}\right) = \rho\left(O_{i,L}\right) = n \quad (15)$$

i.e. if there is no abnormal event happening in the system, the observation vector $y_i(0:L)$ is a linear combination of columns in $O_{i,L}$. Note that this is only true when L is large enough such that $O_{i,L}$ has already achieved full column-rank. If the test is passed, the recovery algorithm in section 2 could be used. Otherwise, a different recovery process must be applied to recover $x(0)$.

The test in Equation (14) is executed prior to the observation phase. If f, the number of potentially compromised nodes, is pre-determined (for reasons described in Section 5) than the test in Equation (14) only needs to be applied to subsets $\mathcal{S}$ of size f. Once an L has been found for which Equation (14) is satisfied for all subsets $\mathcal{S}$, then the length of the observation phase is known, it is $L+1$. We found empirically (see Section 5) that Equation (14) is successful even when condition $\mathcal{S}$ with size less then $\kappa/2$ is not satisfied.

4.3 Recovery

If the presence of malicious activities has been detected, legitimate nodes could recover the initial states from their observation vector. The following procedure describes this recovery process.

Algorithm 1: Recovery algorithm.

1. Let f be equal to the (assumed) number of malicious nodes.
2. For each combination $\mathcal{S}$ of f nodes among n:
   (a) If this test is passed:
   $$\rho\left(\begin{bmatrix} O_{i,L} & \mathcal{M}_{i,L}^{\mathcal{S}} & y_i(0:L) \end{bmatrix}\right) = \rho\left(\begin{bmatrix} O_{i,L} & \mathcal{M}_{i,L}^{\mathcal{S}} \end{bmatrix}\right)$$
   then remember $\mathcal{S}$ and go to step 3.
   (b) If there is no combination left (all $\binom{n}{f}$ tests have failed), increase f and repeat step 2.
   (c) Otherwise (there are still some combinations), continue to the next combination.
3. Find $\mathcal{N}_{i,L}^{\mathcal{S}}$ whose rows form the basis for the left null space of $\mathcal{M}_{i,L}^{\mathcal{S}}$.
4. Calculate $\mathcal{P}_{i,L}^{\mathcal{S}} = \left(\mathcal{N}_{i,L}^{\mathcal{S}} O_{i,L}\right)^{+} \mathcal{N}_{i,L}^{\mathcal{S}}$ where A^{+} denotes the pseudo-inverse of matrix A.
5. Recover the initial state $x(0) = \mathcal{P}_{i,L}^{\mathcal{S}} y_i(0:L)$.

4.3.1 Malicious Nodes Identification

If the initial state can be recovered using $x(0) = \mathcal{P}_{i,L}^{\mathcal{S}} y_i(0:L)$, then the modified state at iteration 1 could be calculated in the same way:

$$x(1) = \mathcal{P}_{i,L}^{\mathcal{S}} y_i(1:L+1)$$

Compare it with the expected state at iteration 1 $\bar{x}(1) = Wx(0)$, the malicious nodes at iteration 1 could be identified. Conduct the consensus protocol and apply the same strategy until iteration $L+l$, if a node j is malicious at iteration l, then

$$x_j(l) \neq \bar{x}_j(l)$$

where $x(l) = \mathcal{P}_{i,L}^{\mathcal{S}} y_i(l:L+l)$ and $\bar{x}(l) = Wx(l-1)$. Note that the next expected state is calculated based on the modified state, because that is the state non-malicious nodes use in reality.

5 ASSESSMENT OF EXACT CONSENSUS

In this section we assess the exact consensus algorithms in sections 2 and 4 for their communication and computation cost, as well as resilience to attacks and network changes. This assessment derived from observations on the design of the algorithm as well as numerical results. Numerical results have been obtained from the insertion of each of the two exact consensus algorithms in a simulator for consensus based network intrusion detection (Toulouse et al., 2015). For these simulations, we have run tests with 5 different regular network topologies: rings with 9 and 25 nodes, 2-D torus with 9 and 25 nodes and the Petersen graph (10 nodes 15 edges).

5.1 No Malicious Activities

In this section we assess the performance of the algorithm describes in Section 2. Column "Steps" in Table 1 lists the number of observations performed to recover $x(0)$. Column "Time" displays the cost to recover $x(0) = \Gamma_i y_i$, expressed in millisecond on a dual-core 2.00 GHz CPU. The time results reported are averaged over 100000 runs of exact consensus. Clearly, the computation time is negligible, which makes the time complexity of this algorithm depends solely on the number of observations. Columns 10^{-2} and 10^{-10} report the convergence speed (number of iterations) of an asymptotic consensus algorithm which was substituted to exact algorithm in the simulations. The values in column 10^{-2} are the number of iterations of the asymptotic algorithm to obtain a solution with a precision no less than 10^{-2} from the true solution (similarly for the column 10^{-10}). We see that that

Table 1: Convergence speed: Exact vs Asymptotic.

Topology	Steps	Time	10^{-2}	10^{-10}
Ring 9	4	0.131	20	129
Ring 25	12	0.219	136	1007
Torus 9	2	0.008	5	26
Torus 25	6	0.128	12	69
Petersen	3	0.014	7	33

the number of iterations for exact consensus is substantially lower compared to asymptotic consensus. It also worth noticing that the values in the column "Steps" follow the intake capacity lower bound tightly for the five network topologies.

The performances of this algorithm are quite good. We now analyze the impact on the consensus function evaluation of errors that are introduced during the observation phase either intentionally (hackers) or unintentionally (system instability). Applications often have some tolerance to error in the consensus function evaluation, so maybe, as for asymptotic algorithms, this algorithm can be used to compute approximations of the true consensus value.

Table 2 records the errors in the recovered initial states $x(0)$ when every node injects random errors within some range. The header of each column indicates the value of the error injected. If a is the header of a column, then the injected error is drawn randomly and uniformly in the range $[-10^a, 10^a]$. The entries of the table contain the error in $x(0)$ represented in $\log_{10}$ form, so if the value in an entry of Table 2 is b, then the raw error is 10^b. The raw error is the L^2 norm between the true $x(0)$ and the recovered one. Results in Table 2 are based the same number of observations as in Table 1 for exact consensus.

The error in Table 2 grows linearly in the injected data range with coefficient 1. Fixing the linear coefficient (slope) at 1, let y be the injection range and x be the error (in log form), we have the model $y = x + \beta$ where β is the base error. The base error is the intrinsic error of the system, which depends on the topology and represents the rounding error occurring during the calculation. The base error is estimated by $\beta = \bar{y} - \bar{x}$ where $\bar{y}$ and $\bar{x}$ is the mean of y and x. The base error values are listed in Table 3.

In table 3, a base error value of 4.01 means the system will lose over 4 decimal places of precision, no matter which floating-point format is being used. If the single-precision format (32-bit) is used with around 7 decimal places of precision, the recovered $x(0)$ will only have 3 meaningful decimal places left. We observe from Table 3 that network topology with fewer degree performs worse than the one with more degree (e.g. Ring 9 $<$ Petersen (10) $<$ Torus 9, where $a < b$ means a is worse than b). We also observe that the more nodes a network has, the more base error it will suffer. Overall, the results in Table 3 pretty much preclude the application of the exact consensus algorithm in Section 2 in an environment characterized by system instability and malicious activities.

5.2 Coping with Resilience Issues

In this section we first analyze numerically the strategy described in Section 4 for coping with malicious activities in the network. Then we discuss the applicability of this algorithm to handle attacks and network instability.

Table 4 reports the performance of the algorithm in Section 4 for one and two compromised nodes. The column "Compromised" indicates the number of compromised nodes in the network. The column "Subsets" indicates the number $\binom{n}{f}$ of subsets of nodes the algorithm may have to examine before finding the subset of compromised nodes (step 2 of the recovery algorithm, Section 4.3). The column "Steps" displays the number of observations performed to recover $x(0)$ (this number is pre-computed as defined in Section 4.3). The last three columns record the empirical time to execute the recovery algorithm, measured in millisecond. The best case happens when the first subset always passes the test in step 2a of the recovery algorithm and the same for the worst case with the last subset. For the average time, the malicious nodes are selected randomly at every run of the exact consensus. Time is averaged over 1000 consensus runs.

In Table 4, "rank test is degenerate" means the system cannot calculate the rank of the matrix reliably anymore. The system uses SVD to calculate the rank. Rank is degenerated when the precision of the floating-point number (we used double floating-point number) is not enough for the system to decide whether a singular value is extremely small or it is zero. For the cases of ring 9 and ring 25 topologies, if there are 2 compromised nodes, this will separate the network, it is then mathematically impossible to recover $x(0)$ at any node. Results in Table 4 show computation time greatly increases for scenarios where there are 2 compromised nodes. The minimum number of observations to recover $x(0)$ also increases rapidly. For Torus 9, it goes from 2 when no attack to 3 and 5 respectively for one and two compromised nodes, similarly for Torus 25 with 6-8-12.

Note that we have run the recovery algorithm with 2 attackers for the Petersen graph which has a vertex connectivity $\kappa = 3$ and the 2D-Torus networks (9 and 25 nodes) with $\kappa = 4$. In both cases $f = 2 \geq \kappa/2$. However, in both cases we have been able to find an L such that the test in Equation (14) is satisfied, allowing the recovery algorithm to compute $x(0)$ for these last 3 network topologies.

5.2.1 Computational Time-complexity

The most time-consuming part of exact consensus in Section 4 is step 2 of the recovery procedure. The

Table 2: Robustness of exact consensus against error injection.

Topology	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
Ring 9	-10.6	-9.0	-7.4	-7.8	-5.2	-5.3	-4.1	-2.3	-1.9	-1.7	-0.6	1.8	2.1	3.1
Ring 25	-2.1	-3.2	-0.9	-1.0	-0.2	0.5	2.2	2.9	3.4	5.2	5.3	7.2	8.3	9.5
Torus 9	-11.8	-10.8	-10.0	-9.3	-8.4	-6.6	-5.7	-5.0	-4.4	-3.0	-0.2	-0.7	0.4	0.8
Torus 25	-10.5	-9.6	-8.5	-7.7	-5.9	-5.7	-4.7	-2.7	-1.5	-1.6	-0.8	0.8	1.3	3.2
Petersen	-10.4	-10.6	-9.8	-8.9	-7.0	-6.0	-5.6	-4.2	-3.4	-1.1	-2.0	-1.0	0.3	2.1

Table 3: Base error for each topology in exact consensus.

Topology	Base error β
Ring 9	4.01
Ring 25	10.15
Torus 9	2.17
Torus 25	3.64
Petersen	2.68

number of subsets that may need to be tested is $\binom{n}{f}$, therefore the algorithm is exponential in f. In practice, base on the results in Table 4, this algorithm can only detect 2 malicious nodes or less. However, the time-complexity of the recovery algorithm grow in polynomial time in terms of the number of nodes n, but the order is high. The number of subsets for a specified n is $O(n^f)$, but matrix multiplication and SVD also have their own time-complexity (which is $O(n^3)$), which is in $O(n^{f+3})$ in total. If the combination $\mathcal{S}$ is a superset of the set of malicious nodes $\mathcal{F}$, $\mathcal{S}$ will still satisfy the test in Step 2a of the recovery algorithm. Therefore, if there is high probability that 2 malicious nodes will appear, we can set the starting of f to $f = 2$, and then use an alternating combination enumeration such as $\{0,1\},\{2,3\},\{4,5\},\ldots$ to cover all the subsets with 1 node in a first few trials. In this way, the computational time trying for the case $f = 1$ could be saved without any significant trade-off in performance. The same strategy could be applied for $f = 3$, however, it is harder to design an enumeration strategy which covers all the subsets of 2 nodes without repeating some subsets.

5.3 Dynamic Networks

The recovery of the initial states at a given node depends on information encoding rather than states diffusion. If changes occur in the network topology, sequences of information encoding will take place at run time that are not modeled in the pre-existing observability matrix. Despite some proposals for exact consensus in dynamic networks (Ahmad et al., 2012), none can actually work reasonably in a network where node/link failures and removal may occur any where without informing the whole network. The proposal in (Ahmad et al., 2012) is limited to cyclic network topology changes. It assumes the existence of a pre-computed observability matrix for specific sequence of network topologies, changes in the network have take place according the sequence embedded in the observability matrix otherwise recovery of initial states will fail.

Our proposal to handle dynamic networks is to consider changes in the network configuration as a set of malicious nodes and apply the expensive recovery process of the algorithm in Section 4. Practically, as the detection algorithm can only recover the initial state with 2 malicious nodes, it's equivalent to 2 node failures or 1 link failure. The observation phase of exact consensus is relatively short, with an upper bound of n iterations. In contrast, the convergence speed of asymptotic consensus algorithms is slow, needing several iterations to compute a consensus solution. Thus, despite that exact consensus cannot tolerate many network changes, its speed makes modifications in the network configuration less likely to happen. And even if the system cannot recover in an initial observation phase, it can initiate a second observation phase to retransmit the data, whose total number of iterations will still be less than the time for a single asymptotic consensus phase. Therefore, exact consensus is still a useful solution in dynamic networks, which is applicable when the cost of transmission is high and the network is sufficiently stable.

6 CONCLUSION

The present paper describes an exact consensus algorithm which has proven low communication cost. Our numerical results further indicate that it should definitely be preferred to asymptotic consensus algorithms when the physical network is stable, reliable and secure. Further, as initial states are completely recovered by each node, different nodes can compute different functions of the initial states, they are not limited to compute the same consensus function. This extends the range of IoT applications of this algorithm beyond those related to consensus problems.

Exact consensus has a mathematically sound mitigation strategy to recover initial states from nodes compromised by malicious actors. Unfortunately it is known that this strategy does not scale well. Our tests

Table 4: Exact consensus speed test result.

Compromised	Topology	Subsets	Steps	Best	Average	Worst
1	Ring 9	9	8	0.429	0.851	1.398
	Ring 25	25	Rank test is degenerate			
	Torus 9	9	3	0.288	0.599	0.825
	Torus 25	25	8	1.124	5.608	9.991
	Petersen	10	5	0.361	0.797	1.289
2	Ring 9	Theoretically impossible				
	Ring 25	Theoretically impossible				
	Torus 9	36	5	0.458	2.714	5.186
	Torus 25	300	12	2.345	133.514	272.231
	Petersen	45	8	0.659	5.909	10.695

and analysis have shown that this strategy is only reasonably applicable for attack scenarios involving 1 or 2 compromised nodes or for network configurations that are impacted by no more than 2 nodes failures or disappearances or by one link failure. However these scenarios are not too constraining. The algorithm has a procedure to identify and phase out compromised nodes once consensus has been computed, so that malicious actors will not accumulate in the system. The very short communication phase means that proportionally very few configuration changes can occur during a communication phase.

An open question is looming in the context of distributed edge computing. Beyond the above limited scenarios of dynamic networks, can the linear encoding scheme of exact consensus be a substitute to the less communication efficient but more robust values diffusion gossip and flooding algorithms? Likely some answers to this question can be found in research on time-varying multihop networks in network coding and applications of observability theory to complex networks. This is the direction of our future research on exact consensus algorithms.

ACKNOWLEDGMENTS

Funding for this project comes from the Professorship Start-Up Support Grant VGU-PSSG-02 of the Vietnamese-German University. The authors thank this institution for supporting this research.

REFERENCES

Ahlswede, R., Cai, N., Li, S. Y., and Yeung, R. W. (2000). Network information flow. *IEEE Trans. Inf. Theor.*, 46(4):1204–1216.

Ahmad, M. A., Azuma, S., and Sugie, T. (2012). Distributed function calculation in switching topology networks. *SICE Journal of Control, Measurement, and System Integration*, 5(6):342–348.

Babai, L. (1995). Handbook of combinatorics (vol. 2). chapter Automorphism Groups, Isomorphism, Reconstruction, pages 1447–1540. MIT Press, Cambridge, MA, USA.

Bahga, A. and Madisetti, V. K. (2016). Blockchain platform for industrial internet of things. *Journal of Software Engineering and Applications*, 9(10):533–546.

Carvin, D., Owezarski, P., and Berthou, P. (2014). A generalized distributed consensus algorithm for monitoring and decision making in the iot. In *2014 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, pages 1–6.

Christidis, K. and Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303.

Colistra, G., Pilloni, V., and Atzori, L. (2014). The problem of task allocation in the internet of things and the consensus-based approach. *Computer Networks*, 73:98 – 111.

Cortes, J., Martinez, S., and Bullo, F. (2006). Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions. *IEEE Transactions on Automatic Control*, 51(8):1289–1298.

Li, S., Oikonomou, G., Tryfonas, T., Chen, T. M., and Xu, L. D. (2014). A distributed consensus algorithm for decision making in service-oriented internet of things. *IEEE Transactions on Industrial Informatics*, 10(2):1461–1468.

Pasqualetti, F., Bicchi, A., and Bullo, F. (2007). Distributed intrusion detection for secure consensus computations. In *Decision and Control, 2007 46th IEEE Conference on*, pages 5594–5599.

Pasqualetti, F., Bicchi, A., and Bullo, F. (2009). On the security of linear consensus networks. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 4894–4901.

Peng, K. and Yang, Y. (2009). Leader-following consensus problem with a varying-velocity leader and time-varying delays. *Physica A: Statistical Mechanics and its Applications*, 388(2):193 – 208.

Pilloni, V., Atzori, L., and Mallus, M. (2017). Dynamic involvement of real world objects in the iot: A

consensus-based cooperation approach. *Sensors 2017*, 17(3).

Saber, R. and Murray, R. (2003). Consensus protocols for networks of dynamic agents. In *American Control Conference, 2003. Proceedings of the 2003*, volume 2, pages 951–956.

Silvestre, D., Rosa, P. A. N., Cunha, R., Hespanha, J. P., and Silvestre, C. (2013). Gossip average consensus in a byzantine environment using stochastic set-valued observers. In *CDC*, pages 4373–4378. IEEE.

Sundaram, S. (2009). *Linear Iterative Strategies for Information Dissemination and Processing in Distributed Systems*. University of Illinois at Urbana-Champaign.

Sundaram, S. and Hadjicostis, C. N. (2008). Distributed function calculation and consensus using linear iterative strategies. *IEEE Journal on Selected Areas in Communications*, 26(4):650–660.

Sundaram, S. and Hadjicostis, C. N. (2011). Distributed function calculation via linear iterative strategies in the presence of malicious agents. *IEEE Transactions on Automatic Control*, 56(7):1495–1508.

Sundaram, S. and Hadjicostis, C. N. (2013). Structural controllability and observability of linear systems over finite fields with applications to multi-agent systems. *IEEE Trans. Automat. Contr.*, 58(1):60–73.

Teixeira, A., Sandberg, H., and Johansson, K. H. (2010). Networked control systems under cyber attacks with applications to power networks. In *Proceedings of the 2010 American Control Conference*, pages 3690–3696.

Toulouse, M., Minh, B. Q., and Curtis, P. (2015). A consensus based network intrusion detection system. In *IT Convergence and Security (ICITCS), 2015 5th International Conference on*, pages 1–6. IEEE.

Tsitsiklis, J., Bertsekas, D., and Athans, M. (1986). Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *Automatic Control, IEEE Transactions on*, 31(9):803–812.

Xiao, L., Boyd, S., and Kim, S.-J. (2007). Distributed average consensus with least-mean-square deviation. *J. Parallel Distrib. Comput.*, 67(1):33–46.

Xiao, L., Boyd, S., and Lall, S. (2005). A scheme for robust distributed sensor fusion based on average consensus. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN '05, Piscataway, NJ, USA. IEEE Press.

APPENDIX

The weight matrix W in systems like Equations (2,3) is modeled on the adjacency structure of the graph G, i.e. $W_{ij} = 0$ if $(i, j) \notin E$, otherwise W_{ij} represents a weight on edge $(i, j) \in E$. The graph G has self-edge, i.e. $W_{ii} \neq 0$. The system described in Equation (3) converges asymptotically to a steady state where $x_i(k) \approx x_j(k), \forall i, j \in \{1, 2, \ldots, n\}$ provided it satisfies some conditions. The most general one is that the graph G must be connected (strongly connected in oriented graphs). Other conditions depend on the consensus problem solved, and are associated to the weight matrix. For the specific case of the average sum problem, the system in Equation (3) converges towards the true average sum if W is row stochastic (Xiao et al., 2007), i.e. $\sum_{j=1}^{n} W_{ij} = 1$, the sum of the weights of each row equal 1. The following weight matrix (known as the *Metropolis-Hasting* matrix) satisfies this second convergence condition for the average sum problem:

$$W_{ij} = \begin{cases} \frac{1}{1+\max(deg_i, deg_j)} & \text{if } i \neq j \text{ and } j \in \mathcal{N}_i \\ 1 - \sum_{k \in \mathcal{N}_i} W_{ik} & \text{if } i = j \\ 0 & \text{if } i \neq j \text{ and } j \notin \mathcal{N}_i \end{cases} \quad (16)$$

where deg_i denotes the degree of node $i \in G$.