Detección de nodos tramposos en procesos de consenso en redes

M. Rebollo*†
*Grupo de Tec. Informática – Inteligencia Artificial
Universitat Politècnica de València
Valencia, Spain
mrebollo@upv.es

R.M. Benito†, J.C. Losada†, J. Galeano†
†Grupo de Sistemas Complejos
Universidad Politécnica de Madrid
Madrid, Spain
{rosa.benito, juancarlos.losada, javier.galeano}@upm.es

***Resumen*—Los procesos de consenso en redes son procesos de difusión de información que permiten realizar cálculos de forma distribuida en una red, intercambiando información únicamente con los vecinos directos y sin tener conocimiento global de la estructura, tamaño, valores ni otras características de la red. Sin embargo, este proceso requiere que todos los nodos sigan el mismo algoritmo y cualquier cambio provoca una alteración en el valor de consenso alcanzado. Este trabajo propone un modelo para la detección de nodos que de forma deliberada o accidental no siguen el proceso de consenso. La red podrá, de forma autónoma y no supervisada, detectar estas desviaciones y corregirlas. Se muestra su aplicación a tres escenarios: un sistema de voto distribuido, la manipulación de imágenes por *adversarial examples* y el problema de los generales bizantinos, que es la base de los protocolos usados para la manipulación de cadens de bloques.**

***Index Terms*—complex networks, consenso, votaciones, blockchain, adversarial examples, tolerancia a fallos, resiliencia**

I. Introducción

Los procesos de consenso en redes permiten calcular de forma distribuida el valor de una función común. Cada nodo emplea únicamente sus datos y la información de sus vecinos directos para recalcular de forma iterativa el valor de la función y propagarlo a través de sus vecinos. Este proceso converge a un valor único final para la función que se está calculando. Los nodos no tienen ningún tipo de conocimiento sobre el tamaño, la topología de la red ni ninguna otra característica.

Sea $G = (V, E)$ un grafo no dirigido formado por un conjunto de vértices V y un conjunto de enlaces $E \subseteq V \times V$ donde $(i, j) \in E$ si existe un enlace entre los nodos i y j. Un vector $x = (x_1, \ldots, x_n)^T$ contiene los valores iniciales de las variables asociadas a cada nodo de la red. Olfati–Saber and Murray [1] proponen un algoritmo cuya aplicación iterativa converge al valor medio de x. La convergencia de este método está asegurada.

$$x_i(t+1) = x_i(t) + \varepsilon \sum_{j \in N_i} [x_j(t) - x_i(t)] \tag{1}$$

Este proceso converge al valor medio de los valores iniciales

$$\lim_{t \to \infty} x(t) = \frac{1}{n} \sum_i x_i(0) = \bar{x} \tag{2}$$

This work is supported by the PROMETEOII/2013/019, TIN2015-65515-C4-1-R and MTM2015-63914-P projects of the Spanish government

y en todas las iteraciones se conserva la suma de x.

$$\sum_i x_i(0) = \sum_i x_i(t) \ \forall t \leq 0 \tag{3}$$

Sin embargo, algunas funciones no pueden calcularse por este proceso. Particularmente, aquellas que requieren la consideración de valores repetidos (como la moda) o valores acumulados (por ejemplo, obtener la suma de los valores iniciales). Por otra parte, todos los nodos deben seguir exactamente el proceso de forma obligatoria para que el resultado sea correcto. Para alterar el resultado, basta con que uno de los nodos altere el valor que intercambia en una de las iteraciones para que el valor de consenso se desvíe del valor esperado. Existen mecanismos para detectar cuándo se produce una variación, pero requieren guardar el estado completo de la red en cada iteración y además es necesario mantener una visión global de la red y conocer su estructura, pues se basan en matrices de observación que se calculan a partir de la matriz de adyacencia de la red completa [2]. Otras alternativas requieren la existencia de un nodo que hace las funciones de coordinador [3]. También están relacionados otros trabajos enfocados a reducir el efecto del ruido en redes de sensores, pero sus soluciones están centradas en obtener la topología de red óptima para minimizar el efecto del ruido [4], [5]

En el presente trabajo se propone un método completamente descentralizado que no requiere información global de la red y que es independiente de la topología, si bien es cierto que existen determinadas restricciones que se deben cumplir para poder anular por completo las desviaciones producidas.

Para mostrar su utilidad, se aplica a tres casos: un mecanismo de voto distribuido, la difusión de imágenes a través de una red y el problema de los generales bizantinos, que es la base de las cadenas de bloques. En todos ellos nos centraremos en cómo detectar y corregir las posibles manipulaciones sobre el proceso de consenso para alterar el resultado de la votación, modificar la imagen para que no se clasifique correctamente (usando *adversarial example*) o evitar la difusión de mensajes falseados en el caso de los generales.

II. Detección de desviaciones en el proceso de consenso

El proceso de consenso definido en (1) asume que todos los participantes ejecutan el mismo algoritmo. Si un nodo desea

alterar el resultado basta con enviar a los vecinos cualquier otro valor. Por ejemplo, si un nodo i envía un valor fijo $x_i(t) = x_i(0)\ \forall t$, la red completa convergerá a dicho valor $x_i(0)$. En general, podemos modelar el proceso de consenso con desviaciones como

$$x_i(t+1) = x_i(t) + \varepsilon \sum_{j \in N_i} [x_j(t) - x_i(t)] + u_i(t) \quad (4)$$

donde $u_i(t)$ es el error introducido por el nodo i en la iteración t. La pregunta que se plantea es si los vecinos de i pueden detectar cuándo $u_i(t) \neq 0$.

II-A. *Detección de fallos en el protocolo*

Despejando (1) podemos obtener la siguiente expresión:

$$dv_i(t+1) = x_i(t+1) + (\varepsilon d_i - 1)x_i(t) - \varepsilon \sum_{j \in N_i} x_j(t) \quad (5)$$

siendo d_i el grado del nodo i. Si los nodos de la red siguen el algoritmo, cada iteración debe cumplir

$$dv_i(t) = 0\ \forall t \geq 0 \quad (6)$$

y cuando $dv_i(t) > 0$ el nodo puede cuestionar la validez de los datos recibidos. La primera parte de la expresión depende solo de los valores de i y el último término es la suma de los valores recibidos en la iteración anterior, así que basta con guardar esta suma para controlar el proceso. La única restricción del método es que el primer intercambio debe ser el real, es decir, el valor inicial no contendrá errores.

De esta manera se detecta que el resultado obtenido en el proceso de consenso no es correcto. La siguiente pregunta que se plantea es ¿puede corregirse esta desviación?

II-B. *Corrección de las desviaciones*

Una primera aproximación es compensar la desviación cuando se detecta en los nodos de la red. Vamos a asumir, sin perder generalidad y para facilitar la explicación, que la desviación se produce una única vez y en uno de los nodos i. Todos sus vecinos $j \in N_i$ lo detectan según (5), con $dv_j(t) > 0$. En la siguiente iteración, estos nodos descuentan la desviación de su valor calculado, haciendo

$$z_j(t+1) = x_j(t+1) - dv_j(t+1) \quad (7)$$

siendo $z_j(t+1)$ el valor a intercambiar en la siguiente iteración. Si todos los nodos se comportan de esta forma, la suma se conserva y además se puede demostrar que $u(t) = \sum_i dv_i(t+1)$. El principal problema de esta solución es que el nodo que introduce la desviación debe colaborar para compensar el resultado, lo que no es probable.

De (5) se puede deducir cómo se reparte el exceso entre los vecinos. El nodo tramposo i se guarda $(1 - \varepsilon d_i)u_i(t)$ para él mismo y reparte $\varepsilon u_i(t)$ para cada uno de sus vecinos, luego

$$dv_j(t+1) = \varepsilon \sum_{k \in N_j} u_k(t), \qquad \frac{dv_j(t+1)}{\varepsilon} = \sum_{k \in N_j} u_k(t) \quad (8)$$

Dividiendo la desviación obtenida en el nodo por el factor ε, cualquier nodo conoce cuál es el error total que se ha introducido en el consenso. Sin embargo, dado que los valores de $\sum_{k \in N_j} u_k(t)$ están agregados, lo que no es posible es determinar qué nodo ha introducido el error.

Esta solución permite a los vecinos del nodo con la desviación saber que algo está mal en el proceso de consenso y corregirlo. Para que esta corrección llegue a todos los nodos se debe propagar también por la red, por lo que se efectúa un proceso de consenso paralelo

Es importante destacar que este método no impide la manipulación ni lo corrige cuando lo detecta. El proceso converge al valor manipulado. Sin embargo, cuando finaliza el proceso, los nodos disponen de la información necesaria para saber si se ha producido alguna desviación de los datos y cómo corregirla.

III. Casos de aplicación

A continuación se van a presentar tres casos de aplicación de la detección y corrección de fallos a problemas para los que disponer de sistemas robustos y seguros es de vital importancia: (i) un sistema de votación electrónica, (ii) la detección de imágenes manipuladas mediante *adversarial examples* y (iii) el problema de los generales bizantinos, base de las cadenas de bloques.

III-A. *Votación distribuida*

Supongamos que se desea obtener el resultado de un votación que se realiza de forma distribuida en una red. Supongamos que se dispone de m opciones y cada nodo solo puede votar a una de ellas. El proceso de consenso consistirá en intercambiar con los vecinos un vector $x_i = (x_i^1, \ldots, x_i^m)$, donde $x_i^k = 1$ si k es la opción seleccionada por i and $x_i^j = 0\ \forall j \neq k$. Es decir, el vector contiene un 1 en la posición de la opción votada y ceros en todas las demás. Cuando se realiza el proceso de consenso, el resultado converge al valor medio de la votación. Para obtener el resultado general, basta con multiplicar el resultado por el número de nodos, que puede obtenerse de forma sencilla con un proceso de consenso en paralelo inicializado de una forma particular.

El proceso de consenso converge a la media de los valores iniciales. Si $\sum_i y_i(0) = 1$, el proceso de consenso convergerá para algún instante de tiempo t a $y_i(t) = \frac{1}{n}$ por lo que el tamaño de la red $n = \frac{1}{y_i(t)}$. Ambos procesos de consenso son independientes y pueden ejecutarse en paralelo.

Extendemos el vector de votos con una columna adicional

$$(x_i|y_i) = (x_i^1, \ldots, x_i^m \mid y_i) \quad (9)$$

e, inicialmente, $y_i = 0\ \forall i$. Sin perder generalidad, podemos introducir un nodo adicional en la red cuyos valores iniciales sean

$$(x_0|y_0) = (\underbrace{0, \ldots, 0}_{m} \mid 1) \quad (10)$$

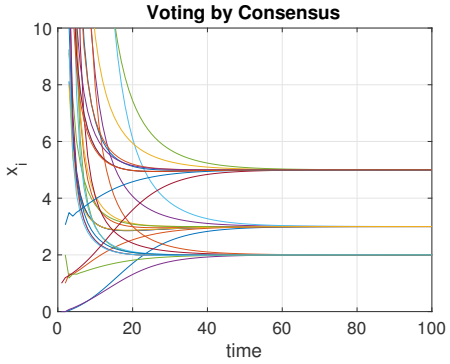

Figura 1. Evolución del proceso de consenso para calcular el resultado de una votación de forma distribuida. red aleatoria con $n = 10$ nodos y $m = 3$ opciones. Los votos son $x(0) = \{3, 3, 1, 3, 2, 1, 1, 2, 3, 3\}$, lo que da un resultado de $\{3, 2, 5\}$.

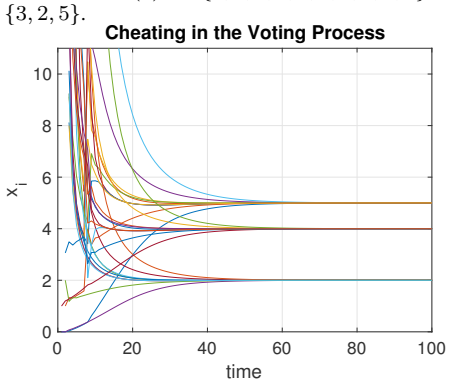

Figura 2. El nodo $i = 7$ añade 2 cotos a la primera opción y eliminar un voto de la tercera en la iteración $t = 6$. El resultado alterado converge a $\{5, 2, 4\}$.

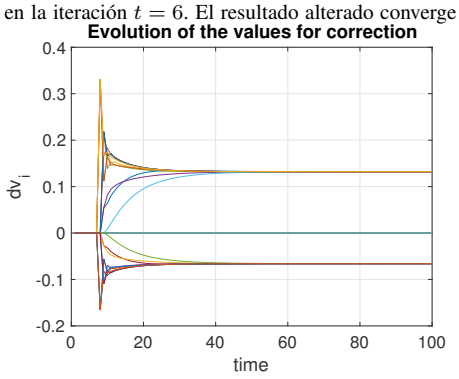

Figura 3. El proceso de consenso sobre las desviaciones detectadas converge a $dv(t) = (0{,}1320, 0, -0{,}0660)$. Aplicando (8) adaptada a votaciones, con $\varepsilon = 0{,}1650$ y $w = 0{,}4$ se obtiene la corrección deseada $\frac{dv/\varepsilon}{w} = (2, 0, -1)$

Este nodo no afecta al resultado de la votación puesto que no vota por ninguna opción. Cuando el proceso de consenso converge, basta con que cada nodo divida los valores finales x_i por el valor obtenido en la última columna y_i para determinar el resultado general de la votación tal y como muestra (11)

$$\frac{x_i(t)}{y_i(t)} = \frac{(\bar{x^1}, \ldots, \bar{x^m})}{1/n} = (n\bar{x^1}, \ldots, n\bar{x^m}) = (\sum_j x_j^1, \ldots, \sum_j x_j^m) \tag{11}$$

La Figura 1 muestra un ejemplo de la aplicación de este proceso sobre una red aleatoria de diez nodos. Los nodos eligen entre 3 opciones y la elección de cada uno es $x(0) = \{3, 3, 1, 3, 2, 1, 1, 2, 3, 3\}$. Por lo tanto, $(x_1|y_1) = (0, 0, 1|0)$, $(x_2|y_2) = (0, 0, 1|0)$, $(x_3|y_3) = (1, 0, 0|0)$, y así sucesivamente. Las líneas muestran la evolución del cociente $\frac{x_i^k(t)}{y_i(t)}$. Puede observarse que la red converge al resultado de la votación $\{3, 2, 5\}$.

Supongamos que el nodo $i = 7$ desea añadir dos votos a la primera opción y eliminar un voto de la tercera: $u_7 = (2, 0, -1)$. El resultado del consenso convergerá a $\{5, 2, 4\}$ y la opción ganadora sería la primera, tal y como deseaba el nodo 7 (ver Figura 2). Aplicando (5), la red puede detectar esta manipulación. La Figura 3 muestra el proceso de consenso sobre los valores de $dv_i(t)$. Para corregir las desviación en el proceso de voto, el consenso de las desviaciones también se extiende con un valor w_i adicional que en este caso se emplea para determinar cuántos nodos han detectado la desviación. El resultado obtenido al aplicar (8) debe dividirse por este valor w_i. En el caso de la Figura 3, $\varepsilon = 0{,}1650$ y la manipulación es detectada por cuatro nodos, lo que arroja un valor de $w = 0{,}4$, con lo que la desviación detectada a partir de los valores de $dv(t)$ obtenidos será $(2, 0, -1)$ en todos los nodos.

De esta forma, al terminar el proceso de consenso, aunque el resultado esté manipulado, todos los nodos de la red tienen la información que les permite corregirlo.

III-B. Adversarial examples

Un segundo caso de aplicación es la transmisión de imágenes a través de una red. Asumiremos que hay un nodo que tiene la imagen (por ejemplo, una imagen capturada en una cámara, como la matrícula de un coche) y quiere propagarla a todos los demás. Sin perder generalidad, será $i = 1$.

La representación de la imagen será a través de un vector que contiene el mapa de bits, con tantos elementos como resolución tiene la imagen y con el valor de RGB de cada pixel. El proceso de consenso se debe modificar de forma semejante al apartado anterior. A dicho vector, se le añade una columna y_i que valdrá cero para todos los nodos excepto para el que contenga la imagen, para el que $y_1 = 0$. Si la imagen contiene p píxeles, $(x_1|y_1) = (x_1^1, \ldots, x_1^p \mid 1)$ y para el resto de nodos $(x_i|y_i) = (0, \ldots, 0 \mid 0)$. Cuando el proceso de consenso converge, basta con dividir el valor obtenido para cada pixel entre $y_i(t)$ para recuperar la imagen original.

Sin embargo, un nodo malicioso puede tratar de modificar la imagen para que el sistema no la reconozca o la identifica erróneamente. Una de las técnicas empleadas se conoce como *adversarial example* [6] y consiste en añadir a la imagen una ligera distorsion de manera que los algoritmos de *deep learning* fallen y clasifiquen la imagen de forma incorrecta (ver Figura 4).

En este caso, el nodo $i = 5$ aplica una perturbación a la imagen en la iteración $t = 20$, tal y como muestra la Figura 4. A la perturbación se le aplica un factor de 0.007, que es suficiente para hacer que los clasificadores que no contemplan *adversarial example* fallen, aunque para el ojo humano parece que la imagen no ha sido alterada. La

Figura 4. Adversarial example. Si a la imagen de la derecha (panda) se le añade la perturbación (centro), da como resultado una imagen que es erróneamente clasificada como un gibón (derecha)

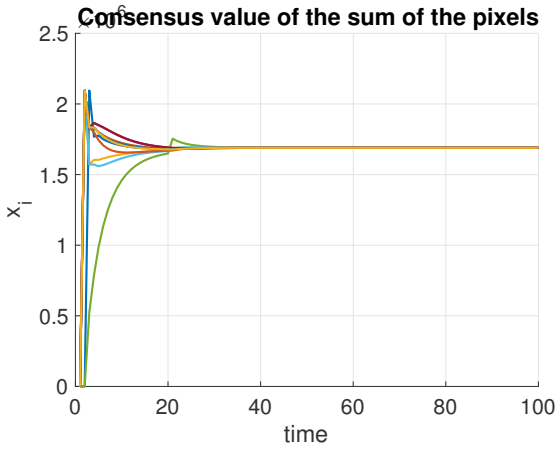

Figura 5. Alteración del consenso en la iteración 20 para incluir la imagen modificada para *adversarial example*. El valor de x_i es la suma del mapa de bits asociado a la imagen

Figura 6. *Adversarial example*. Imagen corregida después del proceso de consenso. Se clasifica como panda con una precisión del 0.9986 (googleNet)

Figura 5 muestra el proceso de consenso, donde el valor es la suma del mapa de bits asociado a la imagen. Se aprecia la perturbación introducida. Cuando converge, todos los nodos tienen la imagen alterada.

Pero de la misma forma que en el caso de la votación, la alteración se detecta inmediatamente por los vecinos utilizando (8) y esta desviación respecto a la imagen original se puede corregir cuando el proceso ha finalizado.

En esta ocasión, vamos a relajar la corrección, de manera que no deseamos recuperar la imagen original, si no simplemente tratar de paliar el efecto de la perturbación para que la imagen vuelva a ser clasificada correctamente. El resultado que muestra la Figura 6 se ha obtenido haciendo un consenso sobre las desviaciones dv_i obtenidas. De esta forma, la parte de la perturbación que se queda el nodo que modifica la imagen $\varepsilon(1 - d_i)u_i(t)$ no se puede recuperar.

Pero como se ponde de manifiesto den lo resultados de la Tabla I, aunque no se recupere la imagen original, la corrección es lo suficientemente buena como para que la imagen vuelva a ser correctamente clasificada.

Los métodos actuales suelen incorporar bibliotecas que incorporan el *adversarial example*, por lo que frecuentemente el único efecto que tiene es disminuir la precisión del clasificador en la imagen seleccionada, pero sigue clasificándose de forma correcta. La imagen corregida por consenso muestra un pixelado mayor, perceptible a simple vista (Figura 6) Sin embargo, la precisión que obtiene por el clasificador es equivalente a la imagen original. La implementación de los distintos algoritmos corresponden a las versiones de Matlab para la *Neural Network Toolbox*. En todos los casos excepto VGG-16, las imágenes son clasificadas como panda gigante, la imagen modificada obtiene una menor precisión y la imagen corregida, sin ser una corrección total, logra resultados comparables a la inmagen original. En el caso cd VGG-16, se observa el comportamiento detectado en el trabajo de Goodfellow [6] y la imagen manipulada se clasifica como un gibón. La corrección vuelve a clasificarse correctamente, y esta vez con una precisión notablemente superior.

III-C. Problema de los generales bizantinos

Un tercer caso de aplicación de particular interés: el problema de los generales bizantinos. La primera solución la plantean Lamport et al. [7] y propone una solución basada en prueba de trabajo *(proof of work –POW–)* como las que incorporan en la actualidad los algoritmos para cadenas de bloques y algunas criptomonedas, como bitcóin.

Cuadro I
PRECISIÓN DE LA CLASIFICACIÓN DE IMÁGENES POR DISTINTOS ALGORITMOS. LA TABLA MUESTRA LA PRECISIÓN DE LA IMAGEN ORIGINAL, LA IMAGEN MODIFICADA POR *adversarial example*, Y LA IMAGEN CORREGIDA DURANTE EL PROCESO DE CONSENSO.

	Google Net	Res–Net 50	Alex Net	VGG-16
original	0.9873	0.9844	0.7479	0.5715 (panda)
alterada	0.9313	0.9429	0.6887	0.4439 (gibón)
corregida	0.9986	0.9863	0.9220	0.8302 (panda)

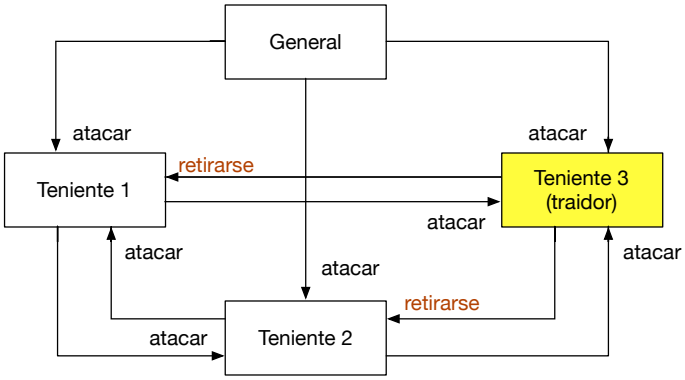

Figura 7. Problema de los generales bizantinos con 4 generales y con un mensaje simple de atacar o retirarse. Los generales solo atacarán si están seguros de que van a hacerlo todos.

El problema de los generales bizantinos consiste en lo siguiente. Supongamos que un conjunto de generales están sitiando una ciudad y tienen que coordinarse para atacar. Si no lo hacen todos al mismo tiempo, fracasarán. Dentro del grupo hay algunos generales traidores que tratarán de prevenir el acuerdo. Para poder atacar, uno de los generales mandará un mensaje al resto (los llamaremos tenientes para diferenciarlos) con la hora a la que van a atacar o indicando si se van a retirar. Cada teniente leal transmitirá el mensaje tal y como lo ha recibido, Pero los traidores alterarán el mensaje indicando una hora distinta. Por ejemplo, si el ataque es a las 9:00, un general traidor mandará un mensaje para que el ataque sea a las 8:00 Cuando lleguen las 8:00, los generales que hayan sido engañados atacarán y, al no ser suficientes, perderán. Luego, los generales restantes atacarán a las 9:00 y de nuevo perderán por no ser suficientes.

¿Cómo puede el algoritmo de consenso resolver este problema? En primer lugar, veamos cómo se pueden sincronizar mediante consenso. Supongamos que no hay traidores. La inicialización será la siguiente: el general (nodo $i = 1$) la hora de ataque como valor inicial $x_1(0) = 8$ y la columna adicional con $y_1 = 1$, con un vector extendido $(x_1|y_1) = (8 \mid 1)$. El resto de nodos se inicializan a cero: $(x_i|y_i) = (0 \mid 0), \forall i \neq 1$. Una vez alcanzada la convergencia, todos los generales sabrán la hora de ataque, que será $\frac{x_i(t)}{y_i(t)} = 8$.

Si hay algún traidor, como muestra la Figura 8, la red converge al valor modificado. En este caso, el nodo $i = 15$ a incrementado en una la hora de ataque y la red converge a 9.

En este caso necesitamos obtener la hora precisa, por lo que es necesario hacer una corrección completa siguiendo (8). De esta forma, los generales determinan que la hora recibida de ataque tiene una desviación de 1 hora adicional sobre la hora que contenía el mensaje original, luego la hora de ataque serán las 8:00 (Figura 9).

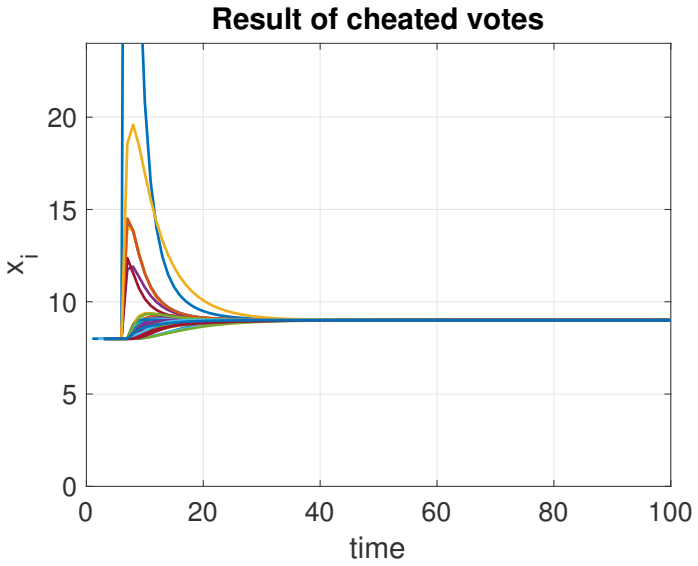

Figura 8. Proceso de consenso para determinar la hora de ataque en una red aleatoria de 50 nodos. El nodo $i = 15$ es un traidor y cambia la hora de ataque de las 8:00 a las 9:00.

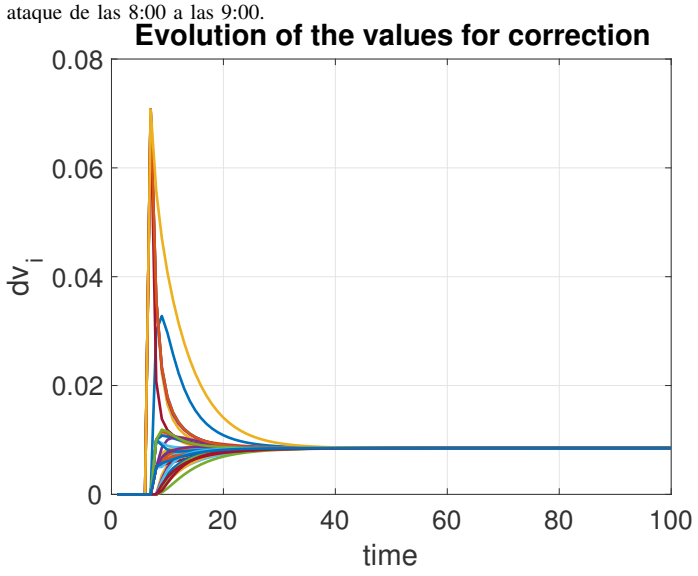

Figura 9. Evolución del factor de corrección $dv_i(t)$. El valor de convergencia es de 0.0085 y aplicando (8) se obtiene que la diferencia con la hora de ataque tiene un exceso de 1 hora

III-D. Validación en redes sintéticas

Finalmente, se han realizado un conjunto de experimentos para validar cómo se degrada la solución a medida que aumentan los nodos que no siguen el algoritmo de consenso. Las pruebas se han realizado sobre redes aleatorias de 100 nodos. Se ha variado el número de nodos que no siguen el proceso desde 5 hasta 50 (represen ta el 50 % de la red). Para cada caso, se han generado 20 redes diferentes y en cada una de ellas se han ejecutado 50 experimentos variando qué nodos

hacen trampas, con un total de 1000 ejecuciones para cada caso. En la Figura 10 se muestra el valor medio (la línea central) y la desviación típica (el área sombreada). Puede verse que siempre hay una desviación mínima, debido a la ubicación de los nodos tramposos.

Para que la solución de Lamport para problema de los generales bizantinos o la corrección planteada por Sundaram funcionen, se requiere que

1. en la red haya al menos $3m + 1$ nodos si hay m nodos tramposos
2. al menos la mitad de los vecinos de cada nodo deben comportarse correctamente

Puesto que para estas pruebas se han generado redes aleatorias sin ningún control sobre el tipo de nodos y su conectividad, y que los ejemplos llegan a una distribución de nodos de confianza y tramposos de 50–50, es de esperar que haya casos en los que la red no sea capaz de compensar la desviación. Resulta interesante ver que el proceso de consenso sigue funcionando aunque se supere el límite de un tercio de nodos tramposos, si bien es algo que debe estudiarse en más profundidad y queda para trabajos futuros.

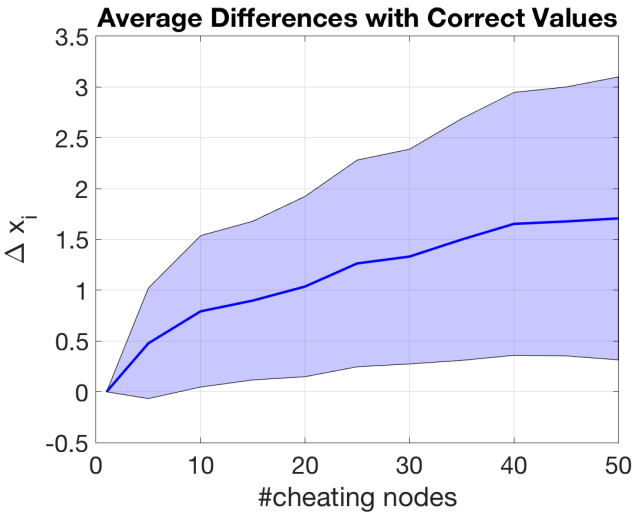

Figura 10. Desviaciones sin corregir en una red de 100 nodos variando el número de nodos tramposos. La línea central es el valor medio y el área sombreada la desviación típica.

IV. Conclusiones

Los procesos de consenso en redes nos permiten realizar cálculos distribuidos sobre el valor de cierta función sobre los valores iniciales de la red, como la media, máximos, mínimos o valores agregados, utilizando tan solo información local y sin conocer la topología ni ninguna otra característica de la red. Sin embargo, para que el proceso funcione, todos los nodos están obligados a seguir el mismo algoritmo. Si tan solo uno de ellos realiza un cambio en uno de los pasos, el valor al que converge la red se ve alterado. Para evitarlo, en el presente trabajo ha desarrollado un mecanismo que permite (i) detectar cuándo se produce una desviación en el proceso de consenso y (ii) corregir la desviación sobre el valor final obtenido.

Se ha mostrado su utilidad en tres problemas de interés: un sistema de votación distribuido, la alteración de imágenes usando *adversarial example* y la resolución del problema de los generales bizantinos. En todos ellos, se ha realizado una adaptación del proceso de consenso para poder hacer cálculos sobre valores agregados, aprovechando la asignación concreta de los valores iniciales y realizando un consenso doble en paralelo sobre variables independientes. En los tres casos, se ha aplicado con éxito el algoritmo de consenso con detección de desviaciones. En el caso de las votaciones, se evita que un nodo intente añadir o eliminar votos a alguna de las opciones. Para la difusión de imágenes, se pueden anular los efectos en los errores de clasificación de imágenes a las que se les ha introducido una perturbación mediante *adversarial example*. Finalmente, en el problema de los generales bizantinos se corrigen los cambios que los traidores hacen en los mensajes y se recupera la hora original. De esta manera, se consigue un método robusto que permite seguir aplicando el consenso en situaciones de incertidumbre, en las que se pueden producir fallos de forma accidental o deliberada y los valores que se transmiten no son los correctos.

Una limitación del método expuesto es que los nodos necesitan tener un número mínimo de vecinos de confianza que propague la información siguiendo el algoritmo. La cantidad de nodos tramposos no debe superar un tercio del total de nodos de la red. Los resultados obtenidos experimentalmente indican que utilizando procesos de consenso se pueden reducir estos límites, pero es necesario un estudio teórico en profundidad sobre el comportamiento del algoritmo presentado. Por otra parte, no se contempla la posibilidad de que los nodos tramposos coordinen sus ataques. Se asume que cada nodo intenta afectar el resultado de forma individual frente a toda la red. Un planteamiento que resuelva estas situaciones se contempla como trabajo futuro.

Referencias

[1] R. Olfati-Saber and R. M. Murray, *Consensus problems in networks of agents with switching topology and time-delays* IEEE TAC. **49**(9), 1520–1533 (2004)

[2] S. Sundaram, S., and C.N. Hadjicostis, *Distributed function calculation via linear iterative strategies in the presence of malicious agents*. IEEE TAC, **56**(7), 1495–1508 (2011)

[3] M. Rafailescu, Fault Tolerant Leader Election In Distributed Systems. IJCSIT, **9**(1), 13–20 (2017)

[4] Ma, C., Li, T. and Zhang, J., Consensus control for leader-following multi-agent systems with measurement noises. J Syst Sci Complex **23**(1), 35–49 (2010)

[5] Wang, J.Z., Mareels, I. and Tan, Y. Robustness of Distributed Multi-Agent Consensus, IFAC Proceedings Volumes **41**(2), 1510–1515 (2016)

[6] Goodfellow, Ian, Shlens, Jonathon and Szegedy, Christian. *Explaining and Harnessing Adversarial Examples*. arXiv 1412.6572. (2014)

[7] Lamport, L. and Shostak, R. and Pease, M. *The Byzantine Generals Problem*. ACM TPLS, **4**(3), 382–401 (1982)