[IIC3272] Proyecto II Criptomonedas 2020-1: Consensus from Trust

Camilo Berríos - Felipe Gómez - Saúl Langarica - Pablo Rojas

Julio de 2020

1. Introducción: El problema de los generales bizantinos

Un sistema de computación confiable debe continuar su buen funcionamiento a pesar de la falla de alguno de sus componentes al propagar información, aquí entra el concepto de *Consensus from Trust* (en español Consenso desde la confianza), es un método para propagar mensajes en una red nodos conectados ante la presencia de nodos maliciosos.

Esto puede ser ilustrado a través de la metáfora de los generales bizantinos. Donde un comandante debe coordinar a sus subordinados, los tenientes, a través de mensajes. Sin embargo, no necesariamente puede enviar mensajes directamente, y los tenientes tienen que mandar los mensajes entre ellos. A su vez hay tenientes traidores que tratarán de confundir a los otros. El objetivo del problema es asegurarse de que todos los tenientes leales reciban el mensaje enviado por el comandante. Donde un teniente al recibir ordenes contradictorias de distintos nodos, puede decidir quien es el traidor al comparar el mayor numero de ordenes entre las distintas ordenes, por ejemplo, si recibe dos ordenes de atacar y una de retirada, decidirá que quien le envío la orden de retirada es el traidor y la orden de atacar es la genuina.

El método *Consensus from Trust* no utiliza mining como prueba para propagar el mensaje. En vez se basa recibir un mismo mensaje de varios nodos diferentes y corroborar su veracidad si todos son idénticos. Para esto es necesario que la mayoría de los nodos sean genuinos.

En el presente informe se detalla una simulación que se realizó de una red de nodos usando el método de *Consensus from Trust* en la propagación de mensaje. Luego se pasará a exponer y analizar los resultados, y se explicará porque no es usado en sistemas como bitcoin.

2. Explicación del código implementado

Esta sección hará una explicación general del código fuente, el archivo **net.py** y el archivo **simulations.py**, que implementa a **net.py** generando las simulaciones y resultados, así como de las librerías utilizadas y del modo de ejecución del archivo.

2.1. Librerías utilizadas

- **Networkx**: librería diseñada para la creación y manipulación de redes complejas
- **Itertools**: librería *built-in* para iteradores eficientes. Utilizada para calcular permutaciones de elementos a partir de una lista
- **Random**: librería *built-in* para obtener pseudo-aleatoriedad.
- **Numpy**: librería para cálculos científicos y matemáticos. La estructura básica de esta librería, numpy array, fue utilizada en varias partes del código
- **Matplotlib**: librería ideal para graficar en 2D. Utilizada para generar los gráficos resumen de las simulaciones realizadas.
- **Copy**: librería *built-in* que permite copiar objetos por valor fácilmente (a diferencia de lo que hace Python normalmente, que es copiar por referencia).
- **Datetime**: librería *built-in* para manipular fechas y horas. Se utilizó para obtener el tiempo de ejecución de las sucesivas simulaciones.

- **Argparse**: librería *built-in* para controlar los argumentos entregados por consola

2.2. Modo de ejecución

Para ejecutar el archivo por consola, se debe ingresar el comando: **python net.py n**, donde **n** es la cantidad de simulaciones a ejecutar. Así, **python net.py 100** ejecutará 100 simulaciones y entregará los gráficos con los resultados. De esta forma se utilizan los parámetros iniciales

n: 20; p: 0.4; ppp: 0.1; k: 50; pp: 0.2

Si no es entregado ningún número, se asumirá **n=15**. En un procesador Intel Core I7-8750H de octava generación, esto demoró aproximadamente 1 minuto en terminar. El tiempo de ejecución varía linealmente según cambia **n**.

Luego la ejecución de **simulations.py** no requiere parámetros, este ejecutará 6000 simulaciones, por lo que tomará un tiempo considerable.

2.3. Explicación del código

El consenso se modelará a través de un grafo dirigido, donde cada nodo subscrito a otro se modela con una arista de A a B, lo que significa que B confía en A y, por tanto, A enviará mensajes a B cuando le lleguen. En el código, el grafo y sus métodos se modelan en la clase **Network**. Los atributos de esta clase son los siguientes:

- **n**: el número de nodos de la red. Se utilizará este número al crear el grafo.
- **p**: representa la probabilidad de que el nodo A esté conectado directamente con el nodo B. Se utilizará este número al crear el grafo.
- **pp**: representa la probabilidad de cada nodo de efectivamente recibir el mensaje cuando se le es enviado. Se utilizará en el método **broadcast_transactions**.
- **ppp**: probabilidad de que un nodo sea malicioso. Se utilizará este valor en el método **malicious_nodes**.
- **k**: número de pasos utilizados en cada simulación. Se utiliza en el método **simulations** e indirectamente en la función **simulate_N**.
- **messages_per_round**: número de mensajes utilizados en cada ronda
- **nTx**: Total de transacciones de la red. Se calcula como: **k * messages_per_round**
- **Tx_id**: ID de cada transacción (incrementa en 1 cada vez que se agrega una transacción a la red)
- **G**: contiene el grafo dirigido que sale como output del método **GenNetwork**, que se explicará a continuación

El método init, además de guardar los atributos de cada instancia de la clase Network, ejecuta **GenNetwork** y guarda su resultado en el atributo **G**. Luego de crear la red, se utiliza el método **malicious_nodes()** para señalar cuáles nodos serán maliciosos y cuáles no.

A continuación, se describen los principales métodos de la clase **Network**:

- **GenNetwork**: genera **n** nodos, cada uno conectado a otro con probabilidad **p**. Es requisito que el grafo sea fuertemente conexo, es decir, que cada nodo sea alcanzable desde todos los otros nodos, donde .ªlcanzableïndica que hay un camino que une a dos nodos. En otras palabras, se pide que un mensaje de un nodo de la red pueda ser transmitido a todos los otros (asumiendo nodos honestos). Si después de generar el grafo no se cumple esta condición, se generará otro grafo y se evaluará la condición. Por tanto, un grafo con **n** grande y **p** muy pequeño podría tardar un gran número de intentos en generar un grafo fuertemente conexo.
- **malicious_nodes**: agrega el comportamiento malicioso a la red recién creada. Cada nodo será marcado como malicioso con probabilidad **ppp**. Se asigna aleatoriamente uno de los tres posibles comportamientos maliciosos, que serán descritos en el método **malicious_node_behaviors**.
- **generate_transactions**: genera una lista de transacciones, cada una de las cuales se representa como un diccionario con los siguientes pares key-value:

  - **type**: el tipo de la transacción. Para este método siempre se ocupará 'Transaction'
  - **value**: el valor de la transacción. Es un float elegido aleatoriamente entre los parámetros de entrada **tx_min_value** y **tx_max_value**
  - **uniqueID**: el identificador de cada transacción.

- **broadcast_transactions**: por cada transacción inicial, se define para cada nodo si recibe el mensage inicial con probabilidad **pp**, si este lo recibe, enviará cada transacción a sus vecinos (nodos que confían en él). aplicando la funcion recursiva **send_message_to_neighbors**

- **send_message_to_neighbors**: este método modela el envío de una transacción desde un único nodo a todos sus vecinos, guarda la transacción en la lista del nodo y se aplica esta función con la misma transacción a cada vecino.

- **malicious_node_behaviors**: este metodo es llamado en la función**send_message_to_neighbors** cuando un nodo es malicioso, presenta uno de los siguientes comportamiento (cada uno con igual probabilidad):

  - Actuará como un "nodo muertoz nunca enviará mensajes a sus vecinos
  - Solo emitirá mensajes al inicio de la ronda
  - Alternará entre ambos comportamientos en rondas diferentes

- **consensus**: dos nodos están en consenso si su lista de transacciones recibidas es exactamente igual. Este método calcula el número de nodos que está en consenso con cada otro nodo del grafo. El resultado es una lista con este número para cada nodo

- **simulation**: Ejecuta una simulación de **k** rondas. En cada una, se utiliza **generate_transactions**, luego **broadcast_transactions** y finalmente **consensus**, descritos anteriormente. A partir del resultado de la función consensus se calcula la cantidad de nodos en consenso, el número de consensos dentro de la red y la cantidad de transacciones en el pool de transacciones de los nodos en el consenso mayor como una medida de cuántas transacciones son transmitidas exitosamente a través de la red.

Las siguientes funciones se utilizarán para fines estadísticos:

- **simulate_N**: se crea una red con los parámetros de entrada, se ejecuta **N** veces el método **simulation** y se calcula media y desviación estándar para el máximo de consensos y el número de consensos de cada ronda de cada simulación. Esta función es la que se llama al ejecutar **net.py**

- **vary_parameters**: Esta función acepta como parámetro N, initial_params y vary_dimension, llama a **simulate_N** con el parámetro N repetidamente variando el parámetro especificado en vary_dimension y un numero de veces dependiendo de la misma variable. Por ejemplo: N = 75, vary_dimension={'k':[10, 110, 10]}. se varia el parámetro k desde 10 hasta 110 con un paso de 10 (python excluye el 110), es decir se llama 10 veces a 75 simulaciones, 750 simulaciones en total. Se retorna un diccionario con
  key = {dimension_variante}:{valor_iteracion}
  value = diccionario con promedio y desviación estándar de max_consensus y n_consensus

- **plot_simulation**: Simplemente genera la imagen de los gráficos con la data generada por el método **simulation_N**.
  el eje X son las k iteraciones, se muestra la media , el máximo y mínimo de las N simulaciones en cada iteracion, tanto del Max consensus (máxima cantidad de nodos en consenso) como del n_consensus (la cantidad de clusters en consensos diferentes).

- **plot_vary_parameters**: Se generan los gráficos con la data generada por el método **vary_parameters**.
  el eje X es la variable que está siendo cambiada, se muestra la media , el maximo y minimo de las N simulaciones en cada variacion del parametro, tanto del Max consensus como del n_consensus.

Finalmente se explica el funcionamiento del archivo **simulations.py**. Este básicamente llama al método **vary_parameters** para cada parámetro y los resultados son almacenados en disco. Para cada variación se hacen 75 simulaciones. el parámetro nodos n y el parámetro k, se realizan 10 variaciones, 750 cada uno. Mientras que para los parametros p, pp, ppp se realizan 20 variaciones, es decir 1500 simulaciones cada uno.
En total se ejecutan 6000 simulaciones. Los detalles de estas se mencionan a continuación.

3. Simulaciones

A continuación se detallan los resultados obtenidos al variar los distintos parámetros de la simulación utilizando el código detallado anteriormente.

En cada caso se reporta la cantidad máxima de nodos en consenso para un n determinado, la cantidad de consensos dentro de la red y por último, el porcentaje del total de transacciones transmitidas a la red que están en el pool de transacciones de todos los nodos del consenso más grande.

3.1. Parámetros de simulación iniciales

Como primer experimento se hizo una simulación con los parámetros iniciales recomendados en el enunciado, es decir:

Cuadro 1: Parámetros iniciales

Parámetro	Valor
n	20
k	10
% p	0.4
% pp	0.2
% ppp	0.1

Como se explico anteriormente, se realizaron $N = 75$ simulaciones con los mismos parámetros y luego se promediaron los resultados para obtener un resultado más robusto dado el comportamiento probabilístico de la red.

- **Máxima cantidad de nodos en consenso**

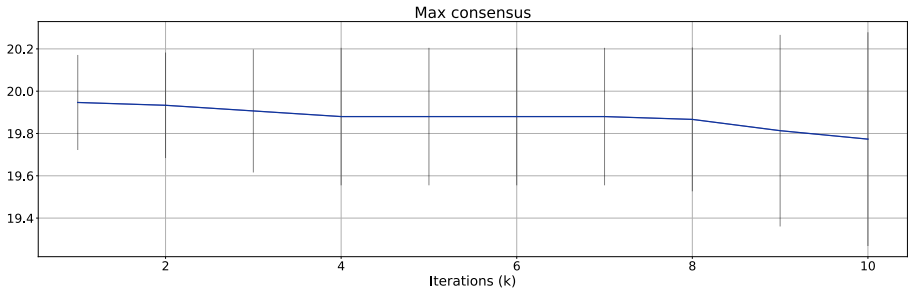

Figura 1: Máxima cantidad de nodos en consenso utilizando los parámetros iniciales

- **Número de consensos en la red**

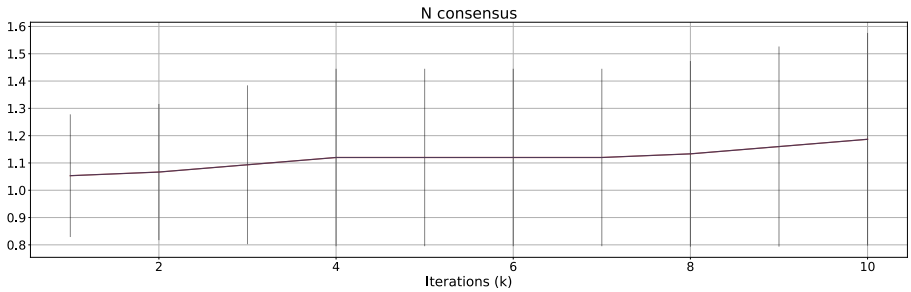

Figura 2: Número de consensos en la red utilizando los parámetros iniciales

- **Porcentaje de las transacciones en el pool de los nodos del consenso mayor**

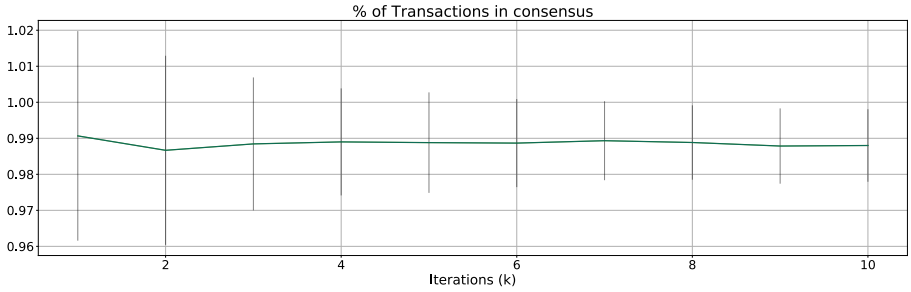

Figura 3: Porcentaje de las transacciones en el pool del consenso mayor con los parámetros iniciales

A partir de los gráficos anteriores se puede ver claramente que con los parámetros iniciales de simulación, virtualmente todos los nodos de la red llegan a un único consenso que casi contiene todas las transacciones transmitidas a la red (se pierde solo un 1 % de las transacciones). Por lo que en este caso, los nodos maliciosos no afectan de gran manera.

3.2. Variación de la cantidad n de nodos

En este caso se varió la cantidad de nodos de la red desde 10 hasta 100 manteniendo todos los demás parámetros constantes.

Para realizar este experimento, se realizaron $N = 75$ simulaciones para cada valor de n, se promediaron los resultados y se reportan los mismos indicadores que en el experimento anterior pero solo para $k = 10$ (última iteración de cada simulación). Se obtienen así los siguientes resultados:

- **Máxima cantidad de nodos en consenso**

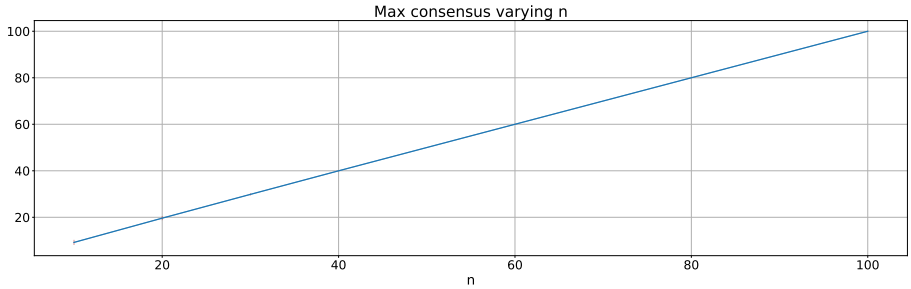

Figura 4: Máxima cantidad de nodos en consenso al variar n

- **Número de consensos en la red**

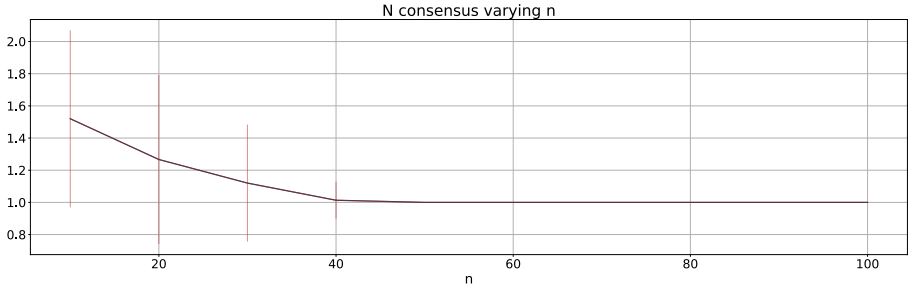

Figura 5: Número de consensos en la red al variar n

- **Porcentaje de las transacciones en el pool de los nodos del consenso mayor**

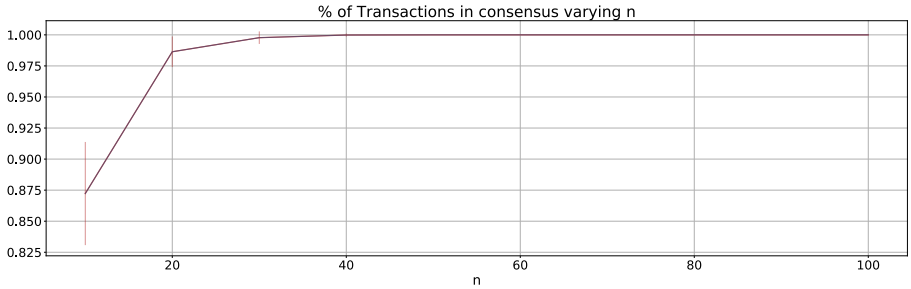

Figura 6: Porcentaje de las transacciones en el pool del consenso mayor al variar n

Como se puede ver en los gráficos, especialmente en las figuras 5 y 6, a medida que aumentan la cantidad de nodos en la red el consenso en la red se hace cada vez más robusto y el efecto de los nodos maliciosos es cada vez menor.

3.3. Variación de la cantidad k de iteraciones

En este caso se varió la cantidad de iteraciones de cada simulación desde $k = 10$ hasta $k = 100$ manteniendo todos los demás parámetros constantes con los valores de la tabla 1.

Al igual que en el caso anterior, para realizar este experimento se realizaron $N = 75$ simulaciones para cada valor de k, se promediaron los resultados y se reportan los indicadores pero solo para el valor máximo de k (última iteración de cada simulación). Se obtienen así los siguientes resultados:

- **Máxima cantidad de nodos en consenso**

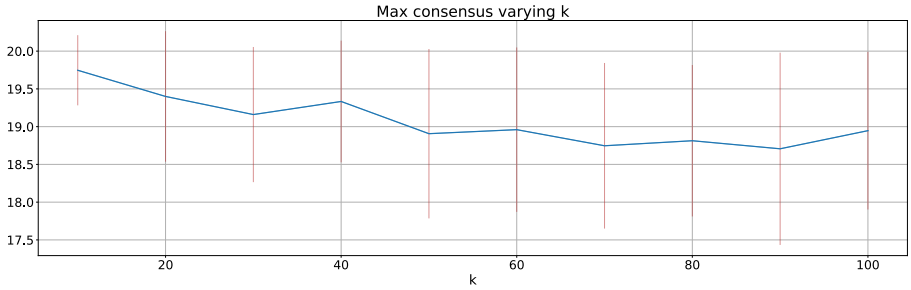

Figura 7: Máxima cantidad de nodos en consenso al variar k

- **Número de consensos en la red**

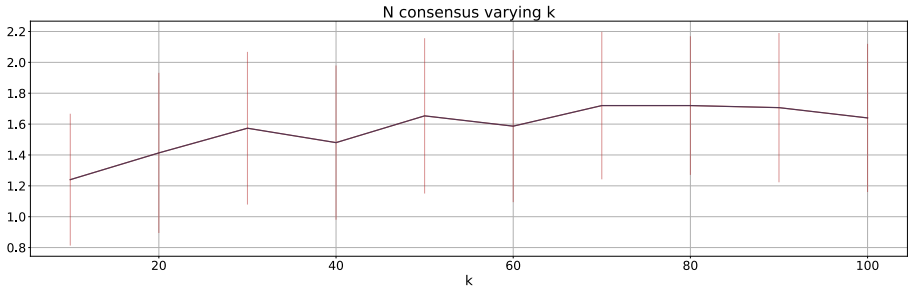

Figura 8: Número de consensos en la red al variar k

- **Porcentaje de las transacciones en el pool de los nodos del consenso mayor**

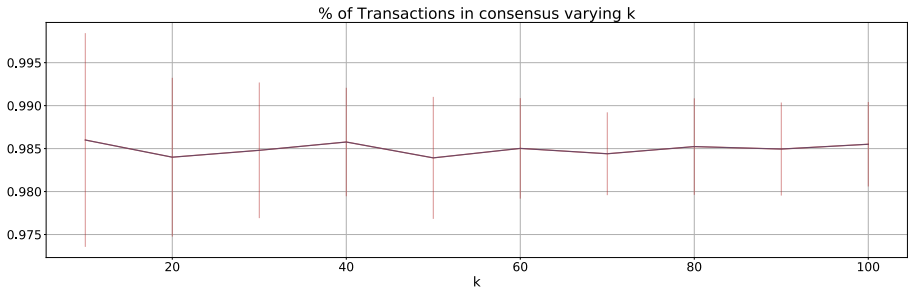

Figura 9: Porcentaje de las transacciones en el pool del consenso mayor al variar k

Se puede ver que en la medida que se incrementa k, disminuye levemente la cantidad de nodos que entran en consenso hasta llegar a $k = 70$ donde se llega a un equilibrio del efecto de los nodos maliciosos en la red. Esto hace que en promedio un nodo quede aislado y que se pierda el 2 % de las transacciones aproximadamente en el consenso mayor.

3.4. Variación de tasa p de conectividad

En este caso se varió la tasa de conectividad desde $p = 0{,}1$ hasta $p = 1$ manteniendo todos los demás parámetros constantes con los valores de la tabla 1.

Se realizó el mismo procedimiento que en los casos anteriores y se obtuvieron los siguientes resultados:

- **Máxima cantidad de nodos en consenso**

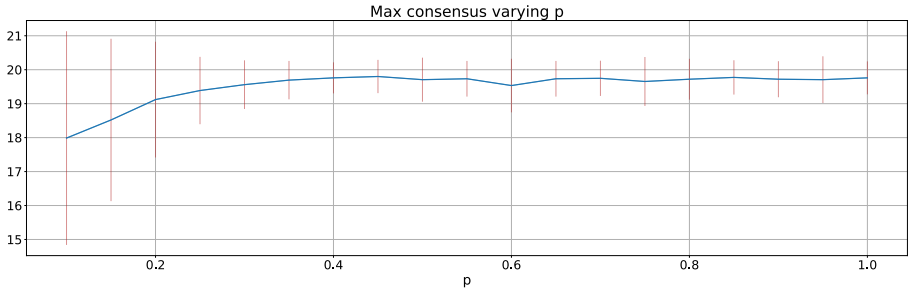

Figura 10: Máxima cantidad de nodos en consenso al variar p

- **Número de consensos en la red**

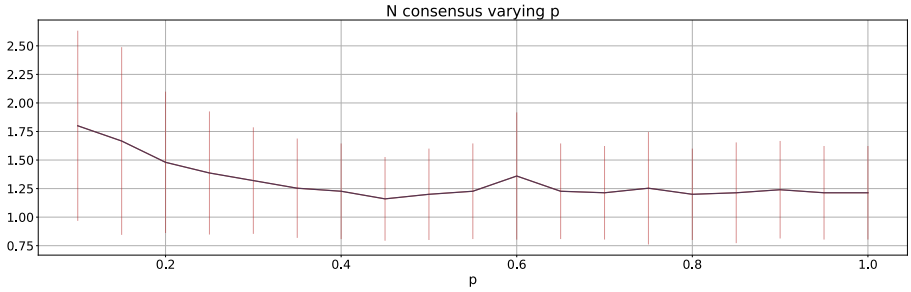

Figura 11: Número de consensos en la red al variar p

- **Porcentaje de las transacciones en el pool de los nodos del consenso mayor**

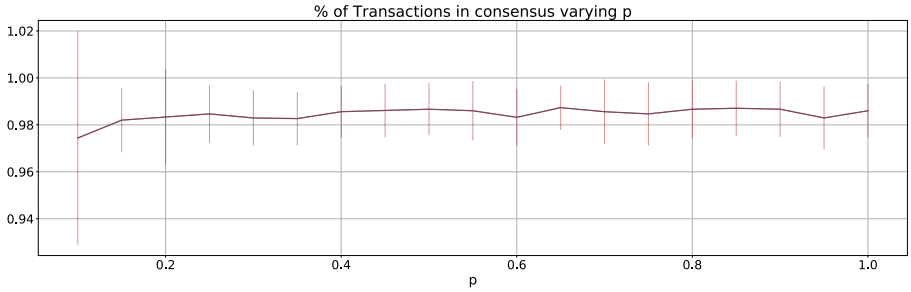

Figura 12: Porcentaje de las transacciones en el pool del consenso mayor al variar p

Se puede ver que cuando la tasa de conectividad es baja, como era de esperase, menos nodos entran en consenso y el número de consensos es mayor. Sin embargo, en la medida en que la tasa de conectividad se incrementa, el número de nodos en consensos se incrementa hasta llegar a un equilibrio a partir de $p = 0{,}5$ aproximadamente.

Como se puede ver en las figuras anteriores, para un p alto, es más probable que un nodo en particular reciba una transacción a pesar de que pueda tener un vecino malicioso, ya que puede tener uno o más vecinos no maliciosos.

3.5. Variación de la probabilidad *pp* de recibir un mensaje directamente

En este caso se varió la probabilidad de recibir un mensaje directamente desde $pp = 0{,}05$ hasta $pp = 1$ manteniendo todos los demás parámetros constantes con los valores de la tabla 1.

Se realizó el mismo procedimiento que en los casos anteriores y se obtuvieron los siguientes resultados:

- **Máxima cantidad de nodos en consenso**

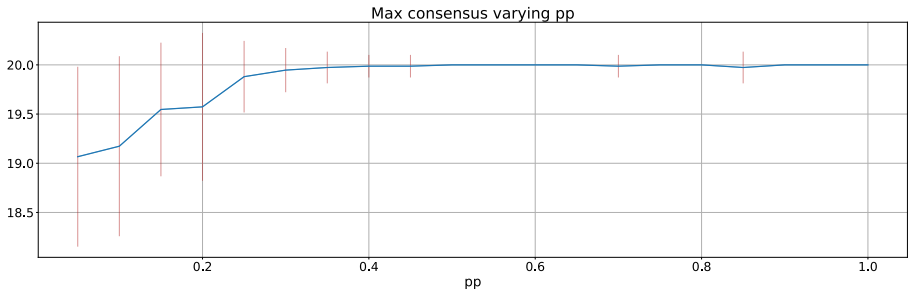

Figura 13: Máxima cantidad de nodos en consenso al variar *pp*

- **Número de consensos en la red**

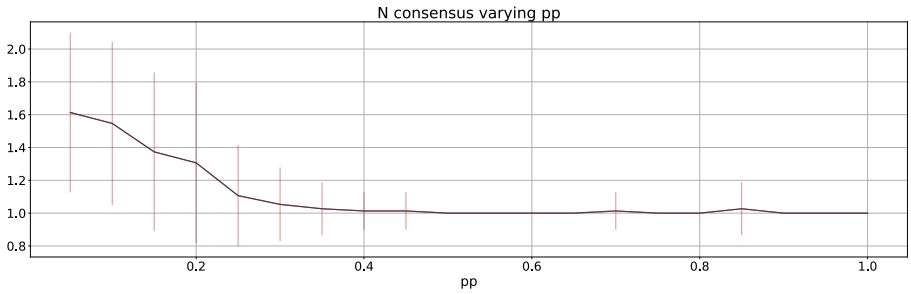

Figura 14: Número de consensos en la red al variar *pp*

- **Porcentaje de las transacciones en el pool de los nodos del consenso mayor**

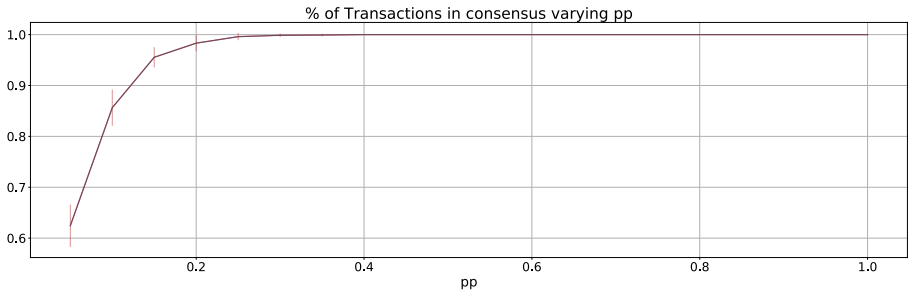

Figura 15: Porcentaje de las transacciones en el pool del consenso mayor al variar *pp*

Como era de esperarse, en la medida que aumenta la probabilidad *pp* de recibir una transacción, la cantidad de nodos que entran en consenso aumenta, hasta llegar a que todos los nodos entren en consenso a partir de $pp = 0{,}5$ aproximadamente.

3.6. Variación de porcentaje *ppp* de nodos maliciosos en la red

En este caso se varió el porcentaje de nodos maliciosos desde $ppp = 0{,}05$ hasta $ppp = 1$ manteniendo todos los demás parámetros constantes con los valores de la tabla 1.

Se realizó el mismo procedimiento que en los casos anteriores y se obtuvieron los siguientes resultados:

- **Máxima cantidad de nodos en consenso**

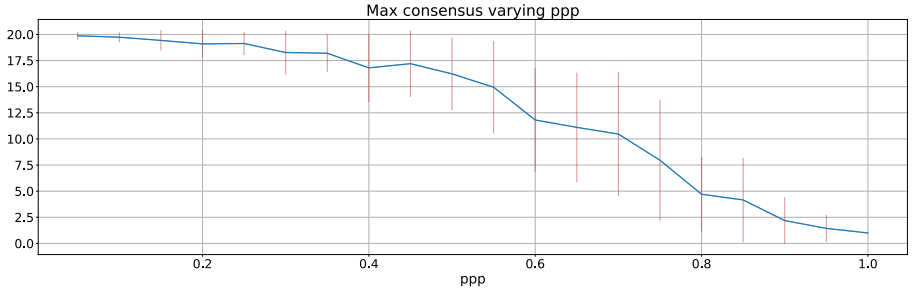

Figura 16: Máxima cantidad de nodos en consenso al variar *ppp*

- **Número de consensos en la red**

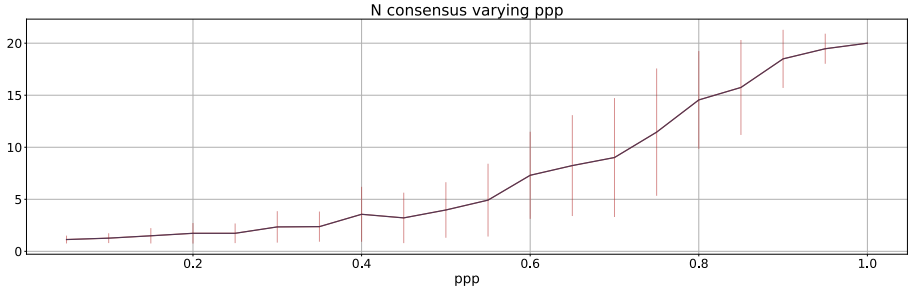

Figura 17: Número de consensos en la red al variar *ppp*

- **Porcentaje de las transacciones en el pool de los nodos del consenso mayor**

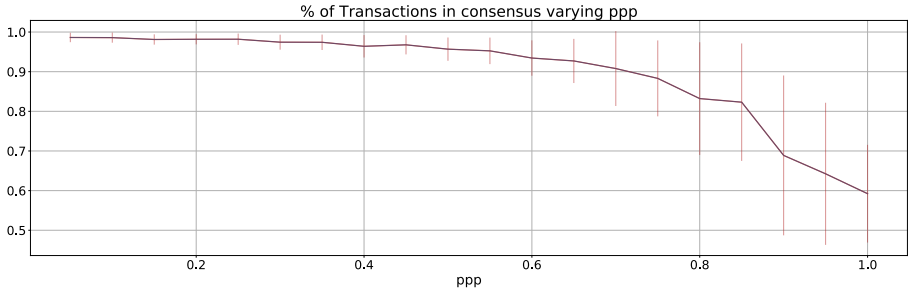

Figura 18: Porcentaje de las transacciones en el pool del consenso mayor al variar *ppp*

A partir de las figura 16 se puede ver como el algoritmo es más o menos robusto a los nodos maliciosos hasta $ppp = 0{,}3$ o un 30 % de nodos maliciosos, sin embargo, para porcentajes mayores de nodos maliciosos la cantidad de nodos en consenso cae drásticamente.

En la figura 17 se puede ver cómo en la medida que crece la cantidad de nodos maliciosos, se va fragmentando la red hasta que ningún nodo entra en consenso. Finalmente en la figura 18 se puede ver cómo cae drásticamente la cantidad de transacciones preservadas a medida que aumenta *ppp*.

3.7. Variación del comportamiento de los nodos maliciosos

En este experimento se varió el comportamiento de los nodos maliciosos y se observó su efecto sobre los indicadores. Tal como se menciona en el enunciado, los nodos maliciosos pueden tener tres comportamientos distintos.

1. **Behaviour 1**: Actuar como un nodo muerto y nunca enviar los mensajes a sus seguidores
2. **Behaviour 2**: Solo puede enviar los mensajes al inicio de cada ronda pero nunca enviar los mensajes recibidos por parte de otros nodos
3. **Behaviour 3**: Alternar entre ambos comportamientos.

En todos los experimentos anteriores, al crear un nodo malicioso se escogía uno de estos comportamientos al azar, por lo que cada nodos malicioso podía comportarse de alguna de estas tres maneras.

En este experimento se mantuvo constante un tipo de comportamiento por toda la simulación y se observó el efecto en los indicadores.

- **Máxima cantidad de nodos en consenso**

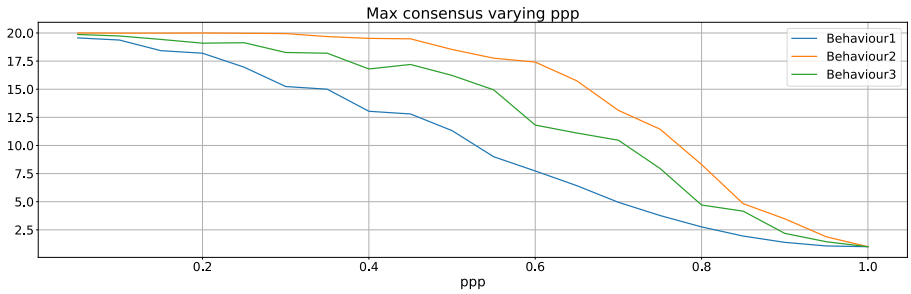

Figura 19: Máxima cantidad de nodos en consenso al variar *ppp* para distintos comportamientos

- **Número de consensos en la red**

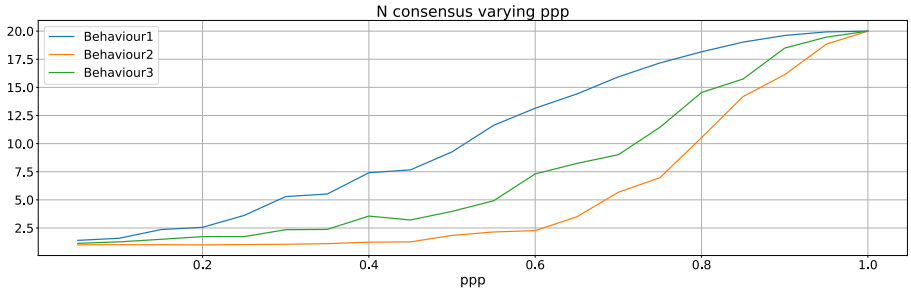

Figura 20: Número de consensos en la red al variar *ppp* para distintos comportamientos

- **Porcentaje de las transacciones en el pool de los nodos del consenso mayor**

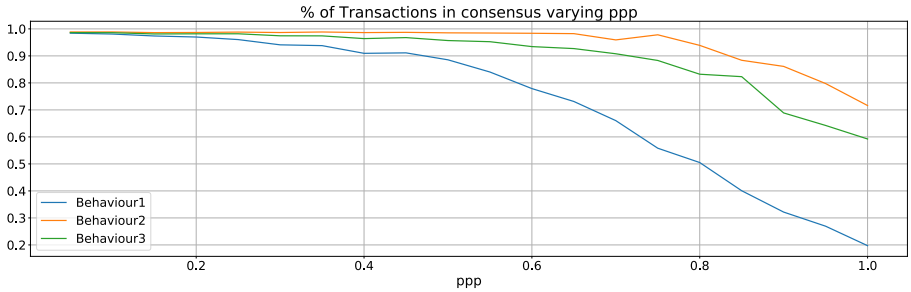

Figura 21: Porcentaje de las transacciones en el pool del consenso mayor al variar *ppp* para distintos comportamientos

A partir de los gráficos anteriores puede verse como el comportamiento 1 (actuar como nodo muerto) es el más perjudicial para la red y es donde se pierden más transacciones, como habría de esperarse. También es interesante notar que el comportamiento 3 tiene un efecto intermedio de el del comportamiento 1 y el comportamiento 2.

4. Conclusiones

A partir de lo realizado en este trabajo, se llegó a las siguientes conclusiones respecto al algoritmo de *consensus for trust*.

- Es necesario una red grande para mejorar las chances de un consenso unánime.
- En una red muy bien conectada, solo los nodos maliciosos no estarían en consenso.
- Sin embargo, no se pueden garantizar las condiciones ideales para un consenso unilateral y través de este método, habiendo una probabilidad a tener en cuenta de que se generen clusters de consenso, lo cual para Bitcoin es perjudicial. En nuestro ejemplo de los generales bizantinos, es como que un grupo de tenientes se ponga en acuerdo para atacar y otro grupo a retirarse. En Bitcoin, es como si en un mismo bloque se hicieran transacciones y en otra versión del mismo bloque no se hicieran esas transacciones.
- Es necesario un mínimo de centralización para colocar en acuerdo a los participantes de la red.

Referencias

[1] LAMPORT, Leslie; SHOSTAK, Robert; PEASE, Marshall. The Byzantine generals problem. En Concurrency: the Works of Leslie Lamport. 2019. p. 203-226. Disponible en: https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf