Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-4852 - Tópicos Especiales en Computación: Tecnologías Blockchain
Septiembre-Diciembre 2021

Proyecto 1 (20 %)

En este proyecto Ud. deberá desarrollar una aplicación P2P que simule el comportamiento de una red Blockchain simplificada. Este prototipo usará definiciones de bloque y transacciones como las usadas por la red Bitcoin[1].

Objetivo: Al finalizar el proyecto, el estudiante entenderá claramente la interacción entre los diferentes conceptos y estrategias que definen un blockchain.

En las siguientes secciones se definirán los requerimientos básicos para este prototipo.

1. Actividad 1: Generadores

En esta sección Ud. desarrollará dos generadores que permitan crear los escenarios para la correcta ejecución de su prototipo.

1.1. Generador de Identidades

Debe desarrollar una utilidad que permita crear un conjunto de identidades ficticias. Cada una de ellas tendrá asociada su nombre, correo, una clave privada, una pública y una dirección asociada a esta última. Cree una salida con saldo inicial de 10.000.000 satoshis para cada cuenta. Estas identidades incluyen los nodos del blockchain.

Sintaxis: `genIdenti -i` $< NumIdentidades > -n < NumNodos >$

En caso de los nodos, las identidades deben tener nombres de la forma: $nodoI | I \in \mathbb{N}$.

Debe ofrecer una función que retorne la clave pública de la identidad asociada a un nombre. Esta función podrá ser usada por los otros programas en su proyecto. Se asume que los nombres son únicos.

[1] Cualquier modificación de estas definiciones deben ser aprobados por su profesor.

1.2. Generador de Transacciones

Este generador tiene como función interactuar con los nodos de la red para solicitar el procesamiento de transacciones. Tendrá la función de usuario y billetera al mismo tiempo. Por lo tanto, podrá usar las claves privadas de las diferentes identidades en los casos que haga falta. Adicionalmente, llevará control de las `UTXO` de cada usuario.

Para cada transacción, las entradas tendrán un único remitente y será seleccionado en forma aleatoria. Asimismo, el destinatario de cada salida será seleccionado en forma aleatoria. Cuide los valores asociados a las salidas. En la mayoría de los casos no deberían sumar más que las entradas ya que no se podría probar su aplicación. Una vez que una transacción sea generada, ésta debe ser enviada a un nodo para solicitar su aceptación en la red.

`Sintaxis: genTransac -f <archivo config> -n <archivo nodo> -d <dir>`

Donde

- $< archivo \quad config >$ será el nombre de un archivo que cumplirá la especificación siguiente:

  - `frecuencia`: Número de transacciones por minuto
  - `NumEntradasMin`: Número de entradas Mínimo
  - `NumEntradasMax`: Número de entradas Máximo
  - `NumSalidasMin`: Número de salidas Mínimo
  - `NumSalidasMax`: Número de salidas Máximo

  Los números anteriores deben ser seleccionados en forma uniforme entre los valores mínimos y máximos correspondientes, inclusive en ambos casos.

- $< archivo \quad nodo >$ será el nombre de un archivo que tendrá los nombres y puertos de los nodos a los que se le puede enviar la transacción. Recuerde que en este caso, los nombres no corresponden con el nombre internet (ip) sino con el identificador en la red P2P. Se debe seleccionar aleatoriamente sólo uno nodo.

- $< dir >$: Directorio dónde el nodo colocará el archivo de nombre $< nombre \quad nodo >$.log

2. Actividad 3: Desarrollar un prototipo de Blockchain

Su prototipo deberá implementar acciones asociadas con el siguiente protocolo.

2.1. Protocolo

Se definen los siguientes mensajes:

2.1.1. Ingresar Transacción

Este mensaje será usado por el generador de transacciones para intentar ingresar una transacción al sistema.

Mensaje: Transacción Nueva

nombre_nodo_remitente
transacción

Mensaje: Transacción Nueva Ack

nombre_nodo_remitente
Estado

donde estado es `Si` o `No` indicando si la transacción fue aceptada (validada) por el nodo receptor o no.

Tanto el generador como los nodos adoptarán la semántica de `Pay-to-Script-Hash (P2SH)` para la simulación de los scripts `scriptSig` y `scriptPubKey`.

2.1.2. Presentación

Este mensaje le permite a un nodo presentarse con otro. El envío de otro mensaje entre dos nodos está condicionado al éxito de la verificación mutua realizada usando este mensaje.

Mensaje: Presentación

nombre_nodo_remitente

Mensaje: PresentacionAck

nombre_nodo_remitente

Si ambos nodos logran identificar a su contra parte, queda establecida la confianza mutua.

2.1.3. Propagar Transacción

Este mensaje será usado por un nodo para propagar una transacción validada a los nodos conocidos. Una transacción previamente validada y propagada no deberá propagarse.

Mensaje: Transacción

nombre_nodo_remitente transacción

Mensaje: Transacción Ack

nombre_nodo_remitente Estado

donde estado es `Si` o `No` indicando si la transacción fue aceptada (validada) por el nodo. Una transacción se considerará válida si:

- la evaluación de los *scripts* P2SH dan cierto.
- Tiene tanto entradas como salidas.
- Suma de las entradas debe ser mayor a suma de las salidas.
- Cada entrada debe corresponder con una salida no gastada.

2.1.4. Propagar bloque candidato

Este mensaje será usado tanto por un nodo que logre minar un bloque así como por un nodo que reciba un nodo propuesto y necesite propagarlo después de confirmarlo. Si un nodo recibe un bloque que ya confirmó y propagó no deberá hacerlo nuevamente.

Mensaje: Bloque

nombre_nodo_remitente bloque

Mensaje: Bloque Ack

nombre_nodo_remitente Estado

donde estado es `Si` o `No` indicando si el bloque fue aceptado por el nodo.

2.2. Requerimientos

- Toda actividad realizada por un nodo o cliente debe ser registrada en un archivo log indicando los participantes, la actividad, el `timestamp` (aa/mm/dd hh:mm:ss) así como los datos asociados a la operación que se registra (hash del bloque o transacción entre otros). Al diseñar este formato, tenga presente que esta información deberá ser procesada por el visualizador.

  La idea es que se pueda estudiar toda la vida del blockchain después de su ejecución.

- Todos los mensajes enviados deben estar firmados por el remitente y encriptados para los destinatarios.

- Se debe usar `PoW` como mecanismo de consenso.

2.3. Ejecución del nodo

Un nodo de la red debe ser invocado usando la siguiente sintaxis.

```
Sintaxis:
   nodo -n <nombre nodo> -d <dir> -f <archivo red> -c <archivo config>
```

donde:

- $< nombre \quad nodo >$: el nombre que usará el nodo para identificarse.
- $< dir >$: Directorio dónde el nodo colocará el archivo de nombre $< nombre \quad nodo >$.log
- $< archivo \quad red >$: es el nombre del archivo que tendrá la información de la red virtual.

  Especificación del archivo:

```
n
nodo1 puerto1
nodo2 puert2
...
nodon puerton
m
nodoi1 nodoj1
```

```
nodoi2 nodoj2
...
nodoim nodojm
```

La primera línea indicará el número n de nodos en la red. Luego aparecerán n líneas, cada una con el nombre de nodo y el puerto que usará ese nodo. Luego estará el número de conexiones m. Por último, habrá m pares de nodos.

- $< archivo \quad config >$: Archivo que tendrá un conjunto de variables. Una por línea.

```
TamanioMaxBloque: 512              # en bytes
TiempoPromedioCreacionbloque: 1    # en minutos
DificultadInicial: 1000
```

3. Actividad 4: Exploradores

Implemente dos utilitarios para mostrar información de los bloques y de las transacciones.

- Explorador de bloques: Mostrará un resumen de la información en un bloque.

```
Sintaxis:
   exploradorBloque {-a <altura del bloque> | -h <hash del bloque>}
```

No se podrá usar las dos opciones al mismo tiempo.

- Explorador de transacciones: Mostrará un resumen de la información en una transacción.

```
Sintaxis:
   exploradorTransac -h <hash de la transacción>
```

4. Actividad 5: Visualizador de Logs

Este utilitario tomará un directorio con archivos logs de los nodos, los combinará y mostrará la evolución del blockchain. En otras palabras, se mostrarán los mensajes ordenándolos por timestamp. La interfaz tiene las siguientes presentaciones:

- Paralela: Una interfaz gráfica bastante sencilla en base a una cuadrícula donde cada celda corresponda con un Textbox no editable. La cuadrícula será de un tamaño predeterminado. Cada casilla estará asociada un nodo de la red.

  La asignación de un nodo a una casilla se realizará en forma automática. Sin embargo, la aplicación permitirá al usuario cambiar esa asignación, con un `Combobox`, por ejemplo. Pueden quedar nodos sin un Textbox cuando haya menos celdas que nodos en la red.

  Es importante que esta salida vaya apareciendo a una velocidad que pueda ser graduada e incluso pausada. Imagine su uso como el de un player de vídeo.

- Secuencial: Debe presentar todos los mensajes en forma clara de manera de seguir las conversaciones entre los nodos. Se puede implementar como el modo anterior con un solo Textbox grande.

Adicionalmente, la interfaz debe ofrecer la posibilidad filtrar:

- por rango de timestamp.
- por tipo de operación: Todas, operación sobre transacciones u operación sobre bloques.
- todas las acciones, o la acción asociada a un mensaje del protocolo.

Sintaxis `visualizador -d` $< dirlogfile > \{-mt \quad | -mg\}$

donde

- $< dirlogfile >$ es el directorio donde estarán los logs a analizar
- $-mt$ y $-mg$ se refieren a los modos indicados. No se pueden usar juntas.

Nota: Para la implementación de este visualizador está permitido reusar código o librerías publicadas por terceros. En este caso, deberá dejar claro el origen de estos fuentes y las modificaciones realizadas.

5. Informe

Elabore un informe con su diseño, las consideraciones tomadas y estado del proyecto.

6. Actividad extra

Diseñe e implemente un mecanismo de ajuste de dificultad para mantener el tiempo promedio de generación de bloque.

7. Sugerencias

- Haga una lista de requerimientos y funcionalidades
- Clasifique y ordene los requerimientos. Por ejemplo, diseñe considerando el visualizador pero deje su implementación para el final.
- Diseñe su aplicación en base a las capas estudiadas en clase. Discútalo con su profesor.
- Divida el trabajo de programación con su compañero. Note que el diseño debe ser en conjunto. La programación de cada fase no necesita ser así.
- Mantenga contacto constante con su compañero. Vayan integrando y probando los avances individuales.
- Informe oportunamente a su profesor sobre problemas encontrados.
- Use `stubs` para funciones que aún no hayan sido realizadas. Cuando estén listas, las agregan y prueban.
- Para la verificación de `P2SH` no debe implementar una máquina de pila. Sin embargo, encapsule esta verificación en un método separado. Preste atención a los parámetros que debe recibir este método.
- Preste atención a cómo enviará la información a través del medio. Recuerde que no puede mandar una referencia por un socket. Debe mandar una representación que pueda ser reconstruida en el receptor. Considere implementar un `__str__` para los tipos transacción y bloque. Y constructores que reciban un string con esos formatos.
- Para funciones muy especificas, puede usar librerías públicas. Consulte con su profesor.
- Pruebe su aplicación usando varias topologías de red P2P.