Solución Examen – 7 de febrero de 2024
(ref: solredes202402.odt)

Instrucciones

- **Indique su nombre completo y número de cédula en cada hoja.**
- **Numere todas las hojas e indique la cantidad total de hojas que entrega en la primera.**
- **Escriba las hojas de un solo lado y utilice una caligrafía claramente legible.**
- **Comience cada pregunta teórica y cada ejercicio en una hoja nueva.**
- **Solo se responderán dudas de letra. No se responderán dudas de ningún tipo los últimos 30 minutos del examen.**
- **El examen es individual y sin material. Apague su teléfono celular mientras esté en el salón del examen.**
- **Es obligatorio responder correctamente al menos 15 puntos en las preguntas teóricas y 20 de los problemas prácticos. Los puntos ganados en el curso se suman a los puntos de teórico.**
- **El puntaje mínimo de aprobación es de 60 puntos.**
- **Para todos los ejercicios, si es necesario, puede suponer que dispone de los tipos de datos básicos (p.ej. lista, cola, archivo, string, etc.) y sus funciones asociadas (ej: tail(lista), crear(archivo), concatenar(string, string).**
- **Justifique todas sus respuestas.**
- **Duración: 3 horas. Culminadas las 3 horas el alumno no podrá modificar las hojas a entregar de ninguna forma.**

Preguntas Teóricas

Pregunta 1 (10 puntos)

a) Mencione y describa los 3 principales componentes del servicio de correo electrónico vistos en el curso.
b) Realice una breve comparación entre los protocolos POP3 e IMAP.

Solución Pregunta 1

a) Los 3 componentes principales vistos en el curso son: agentes de usuario, servidores de correo y el protocolo simple de transferencia de correo (SMTP, Simple Mail Transfer Protocol).
Los agentes de usuario permiten a los usuarios leer, responder, reenviar, guardar y componer mensajes.
Los servidores de correo forman el núcleo de la infraestructura del correo electrónico. Cada destinatario tiene un buzón de correo ubicado en uno de los servidores de correo. Un mensaje típico inicia su viaje en el agente de usuario del emisor, viaja hasta el servidor de correo del emisor y luego hasta el servidor de correo del destinatario, donde es depositado en el buzón del mismo.
SMTP es el principal protocolo de la capa de aplicación para el correo electrónico por Internet. Utiliza el servicio de transferencia de datos fiable de TCP para transferir el correo desde el servidor de correo del emisor al servidor de correo del destinatario. Al igual que la mayoría de los protocolos de la capa de aplicación, SMTP tiene dos lados: el lado del cliente, que se ejecuta en el servidor de correo del emisor, y el lado del servidor, que se ejecuta en el servidor de correo del destinatario. Tanto el lado del cliente como el del servidor de SMTP se ejecutan en todos los servidores de correo. Cuando un servidor de correo envía mensajes de correo a otros servidores de correo, actúa como un cliente SMTP. Cuando un servidor de correo recibe correo de otros servidores, actúa como un servidor SMTP.

b) Ambos son protocolos de acceso al correo electrónico. POP3 es mucho más simple y cuenta con menos funcionalidades que IMAP (surgido posteriormente). Las implementaciones de IMAP resultan más complejas que POP3.
IMAP ofrece comandos para crear carpetas y mover los mensajes a ellas, permitiendo que el servidor de correo mantenga la configuración (estado) para cada usuario sin importar desde qué computadora (cliente) se está accediendo.
También, a diferencia de POP3, IMAP ofrece comandos que permiten al agente obtener determinadas partes de los mensajes, lo que permite una mejor gestión del ancho de banda de la conexión.

Pregunta 2 (10 puntos)

a) Discuta ventajas y desventajas del uso de TCP y UDP para transmisiones de streaming. Se busca que considere el efecto de la variación de retardo y de la pérdida de paquetes en el stream.
b) La visión clásica vista en el curso indica que UDP es el servicio de transporte más adecuado para streaming, mientras que TCP para transferencias de archivos. Considerando este punto, ¿cómo es posible que los servicios comerciales de streaming como Youtube, Netflix o Amazon Prime operen sobre HTTP sobre TCP? Use su análisis de la parte a) en esta respuesta.

Solución Pregunta 2

a) Si bien TCP ofrece un servicio orientado a conexión con garantías en la entrega de datos, de forma clásica éste no se considera el más adecuado para streaming. Se considera a UDP como un protocolo más adecuado, pues, el video acepta leves pérdidas de paquetes de forma imperceptible para el ojo humano o a lo sumo, pixelado de algunas regiones de la imagen. El tipo de datos que se transmiten acepta pérdidas y desde el punto de vista de la percepción, se asimila mejor una disminución transitoria de la calidad que la detención del stream mientras se re-transmite un dato perdido: es preferible ver un tiro penal pixelado que tener la secuencia perfecta 5 minutos más tarde (luego que los vecinos gritaron el gol). La variación de retardo o jitter, se mitiga colocando un buffer a la recepción que suavice los cambios en los tiempos de llegada y ofrezcan un servicio estable. La variación de retardo en TCP puede generar retransmisiones, mientras que en UDP pueden ser suavizadas.

b) Internet en general ha mejorado notablemente sus velocidades de servicio y su calidad. Tanto servicios de cobre como de Fibra Óptica logran tasas de error muy bajas, por lo que, son muy escasas las retransmisiones, así como, los niveles de ancho de banda ofrecen retardos de transmisión o cola muy bajos y estables, con un jitter bajo. Esto permite que la transmisión de video por TCP de excelentes resultados. Por otra parte, el uso generalizado de NAT hace muy difícil el uso de protocolos basados en UDP de forma generalizada. El uso de TCP también ofrece una solución para mitigar los problemas asociados al NAT.

Pregunta 3 (10 puntos)

En el contexto de la capa de red:

a) Describa plano de datos y plano de control. Cada descripción debe incluir su función, como la realiza y si es local o global.
b) Mencione dos ejemplos de protocolos por cada uno de los planos mencionados.
c) IPv4 permite fragmentación, ¿cuándo se utiliza este mecanismo?. Especifique cuales nodos pueden fragmentar y reensamblar.

Solución Pregunta 3

a) Respuesta extraída textualmente de las diapositivas del curso:

Plano de datos
- función **local** a cada enrutador
- determina cómo el datagrama que llega a un puerto de entrada se reenvía a un puerto de salida del enrutador
- implementa la **función de reenvío** o **forwarding**

Plano de control
- funcionalidad **global** en toda la red
- determina cómo se encamina un datagrama entre enrutadores a lo largo de la ruta desde el host de origen al host de destino, utilizando los algoritmos de enrutamiento.

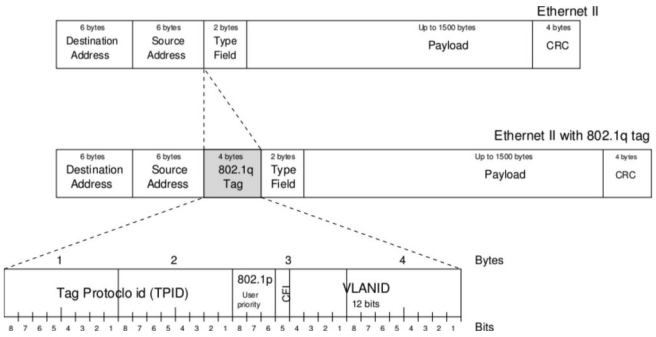
b) En el plano de control tenemos los algoritmos de enrutamiento, por ejemplo RIP y OSPF. El plano de datos tranfiere los datos de los usuarios, y por ende son protocolos de capa de aplicación los que llevan esos datos. Dos ejemplos posibles son HTTP y SMTP.
c) IPv4 prevee este mecanismo para cuando un paquete IP debe atravesar un enlace con un MTU menor que el tamaño del paquete. IPv4 permite entonces fragmentar el paquete en cualquier nodo de la red, para luego ser reensamblado por el host destino.

Pregunta 4 (10 puntos)

a) Describa diferencias en el formato de los cabezales de los protocolos 802.1 y 802.1q.
b) Presente un ejemplo de topologías equivalentes con y sin 802.1q, que demuestre alguna ventaja de usar este protocolo.

Solución Pregunta 4

a) Se le agrega una etiqueta de VLAN de 4 bytes, que destina 12 bits a VLANs. Esta desencadena un cambio en el tamaño de los frames.



b) Ejemplos:

- Múltiples LANs → el uso de VLANs permite unificar multiples switches en uno, agregando flexibilidad para conectar equipos a diferentes LANs.
- Múltiples LAN en varias localizaciones → permite reemplazar múltiples enlaces con un trunk, y permite migrar equipos entre localizaciones permaneciendo lógicamente en la misma LAN.
- “Router on a stick” → permite ocupar un solo puerto mediante un enlace trunck entre un router y un switch que soporta VLANs.

Problemas Prácticos

Problema 1 (30 puntos)

Suponga que deseamos disponer de un protocolo de tipo “Selective Repeat” o “Repetición Selectiva” (SR) que envíe mensajes de dos en dos. Es decir, el emisor enviará un par de mensajes y enviará el siguiente par de mensajes solo cuando sepa que los dos mensajes del primer dúo se han recibido correctamente. Suponga que el canal puede perder mensajes pero no corromperlos ni tampoco reordenarlos.

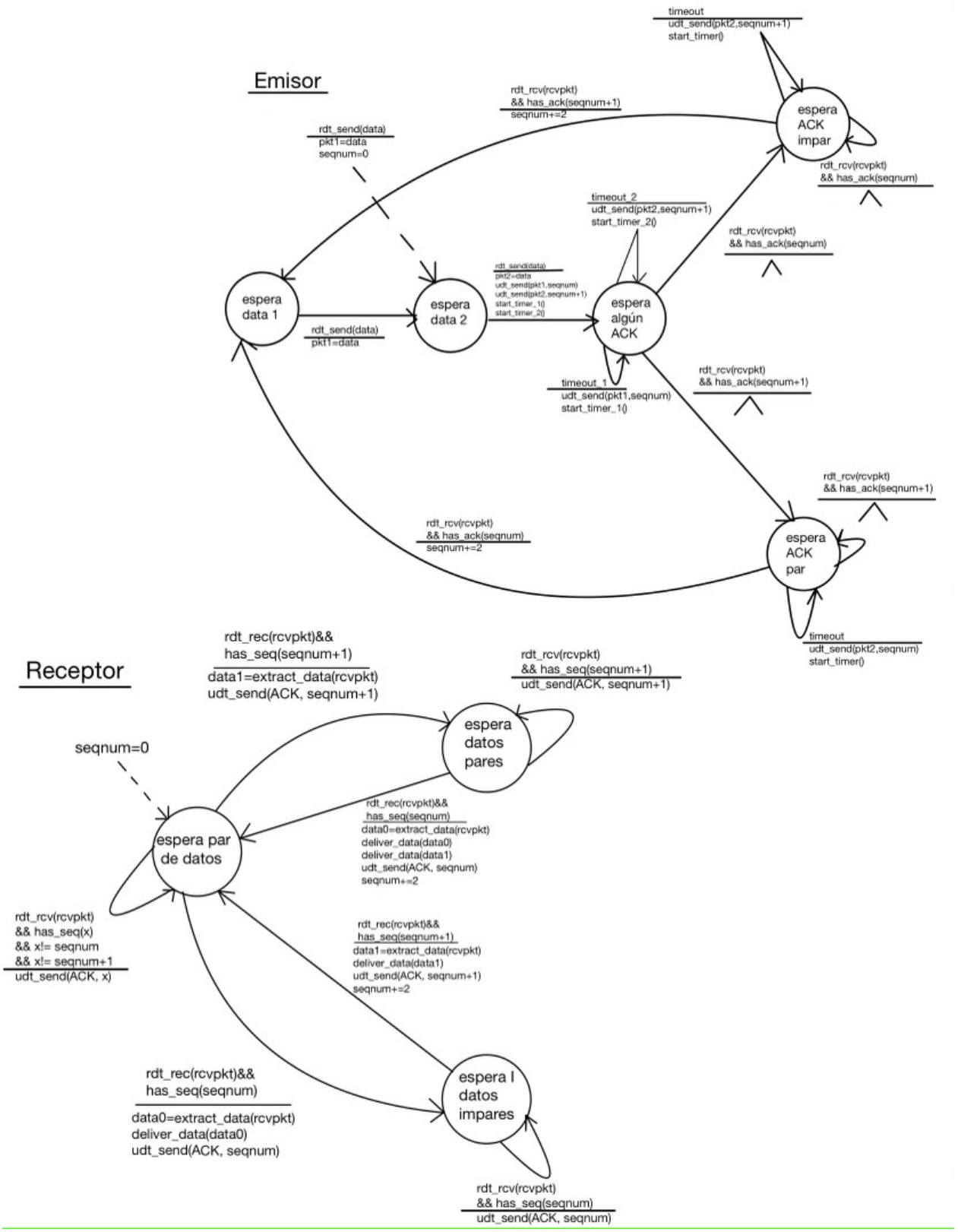
Se pide:

a) Diseñe un protocolo de control de errores para un servicio de transferencia de mensajes fiable y unidireccional. Proporcione una descripción de las máquinas de estados finitos del emisor y del receptor. Describa el formato de los paquetes intercambiados por el emisor y el receptor. Si utiliza alguna llamada a procedimiento distinta de las empleadas en el curso, defina claramente las acciones que realizan.

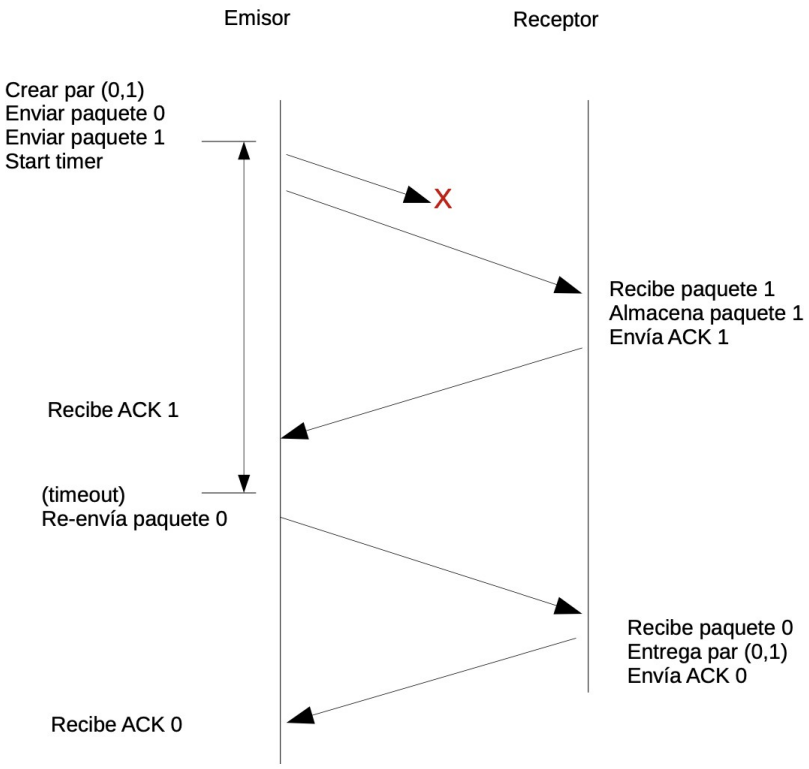
b) Proporcione un ejemplo (una gráfica temporal del emisor y del receptor como las vistas en el curso) que muestre cómo este protocolo se recupera de la pérdida de un paquete.

Solución

a)


Emisor
timeout
udt_send(pkt2,seqnum+1)
start_timer()
rdt_rcv(rcvpkt)
&& has_ack(seqnum+1)
seqnum+=2
espera ACK impar
rdt_rcv(rcvpkt)
&& has_ack(seqnum)
rdt_send(data)
pkt1=data
seqnum=0
timeout_2
udt_send(pkt2,seqnum+1)
start_timer_2()
rdt_rcv(rcvpkt)
&& has_ack(seqnum)
espera data 1
espera data 2
rdt_send(data)
pkt2=data
udt_send(pkt1,seqnum)
udt_send(pkt2,seqnum+1)
start_timer_1()
start_timer_2()
espera algún ACK
rdt_send(data)
pkt1=data
timeout_1
udt_send(pkt1,seqnum)
start_timer_1()
rdt_rcv(rcvpkt)
&& has_ack(seqnum+1)
rdt_rcv(rcvpkt)
&& has_ack(seqnum+1)
rdt_rcv(rcvpkt)
&& has_ack(seqnum)
seqnum+=2
espera ACK par
timeout
udt_send(pkt2,seqnum)
start_timer()
Receptor
rdt_rec(rcvpkt)&&
has_seq(seqnum+1)
data1=extract_data(rcvpkt)
udt_send(ACK, seqnum+1)
rdt_rcv(rcvpkt)
&& has_seq(seqnum+1)
udt_send(ACK, seqnum+1)
espera datos pares
seqnum=0
espera par de datos
rdt_rec(rcvpkt)&&
has_seq(seqnum)
data0=extract_data(rcvpkt)
deliver_data(data0)
deliver_data(data1)
udt_send(ACK, seqnum)
seqnum+=2
rdt_rcv(rcvpkt)
&& has_seq(x)
&& x!= seqnum
&& x!= seqnum+1
udt_send(ACK, x)
rdt_rec(rcvpkt)&&
has_seq(seqnum+1)
data1=extract_data(rcvpkt)
deliver_data(data1)
udt_send(ACK, seqnum+1)
seqnum+=2
rdt_rec(rcvpkt)&&
has_seq(seqnum)
data0=extract_data(rcvpkt)
deliver_data(data0)
udt_send(ACK, seqnum)
espera I datos impares
rdt_rcv(rcvpkt)
&& has_seq(seqnum)
udt_send(ACK, seqnum)

b)

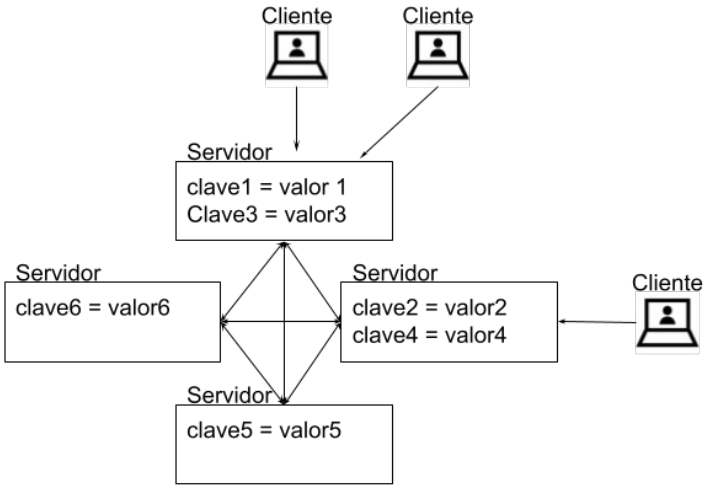

Emisor
Receptor
Crear par (0,1)
Enviar paquete 0
Enviar paquete 1
Start timer
X
Recibe paquete 1
Almacena paquete 1
Envía ACK 1
Recibe ACK 1
(timeout)
Re-envía paquete 0
Recibe paquete 0
Entrega par (0,1)
Envía ACK 0
Recibe ACK 0

Problema 2 (30 puntos)

Se propone desarrollar una base de datos de *clave-valor*, donde clave y valor son cadenas alfanuméricas sin espacios en blanco. El sistema está formado por las aplicaciones servidor y cliente. El servidor es el responsable de almacenar los datos y responder a solicitudes de los clientes y otros servidores.

El servidor responde a las consultas de los Clientes y otros servidores utilizando el protocolo **DATOS**. Los datos son almacenados entre las instancias activas de los servidores de forma distribuida y transparente para los clientes. Los servidores se encuentran conectados a una misma red local y se descubren entre ellos usando el protocolo **DESCUBRIMIENTO**.

En cada servidor se cuenta implementada la función *getValue(<key>)* que permite consultar localmente el valor asociado a una clave. En caso de no encontrar la clave solicitada esta función devuelve el string "*NONE*". El servidor, en caso de no encontrar la clave localmente, debe consultar (usando el protocolo DATOS) a todos los demás servidores que se tenga *descubiertos* hasta que alguno le de una respuesta. El alta y baja de valores (*clave-valor)* no es parte de lo solicitado.



El sistema utiliza dos protocolos:

a) ***DESCUBRIMIENTO***, este protocolo es implementado por los servidores. Está basado en mensajes UDP de broadcast en la LAN, utilizando el puerto 2022. Cada 30 segundos envía un mensaje de texto plano, con el siguiente texto: ANNOUNCE**\n**, indicando que el servidor atiende en el puerto 2023 de TCP donde acepta conexiones para el protocolo ***DATOS***.

b) ***DATOS***, protocolo utilizado para realizar las consultas de datos. Está implementado sobre TCP utilizando el puerto 2023. Es orientada a texto, con mensajes separados por caracteres de linea nueva (\n). Las comunicaciones tienen la forma pregunta/respuesta, donde el intercambio es siempre iniciado por el nodo que abrió la conexión. Utiliza mensajes:

- GET <key>**\n** para consulta del valor asociado a la clave *<key>*. La respuesta es el resultado de la función *getValue()* para la clave consultada (con un \n al final).

Se pide:

Implemente en un lenguaje de alto nivel, usando la API de sockets del curso **el servidor** que atiende el servicio. Considere que el mismo puede recibir consultas que se encuentran almacenadas en otro servidor, debiendo resolverlas y responderlas.ente que lo solicita.

Solución Problema 2

Se presenta una solución donde la consulta es diferenciada cuando es solicitada por un cliente (en caso que el solicitante no se encuentre en la lista de servidores), o de otro servidor, cuando se presenta en la lista. Para cada solicitud de un cliente recibida, se realizarán conexiones con todos los demás servidores para determinar si el valor se encuentra en él, y en caso de encontrarlo devuelve el valor y en caso de no tenerlo sigue con la búsqueda en los otros.

SERVIDOR DE DBsearch

```
def DBsearch()
 # Para mituoexcluir acceso al diccionario
 mutex = semaphore()
 # Diccionario para agregar IP:timestamp de los servidores activos
 # el timestamp se utiliza para medir los 30 segundos
 Servidores = new Diccionario()
 #creo socket para protocolo DESCUBRIMIENTO
 sktUdp = socket.udp()
 sktUdp.bind(*, 2022)
 # Inicio hilo de envío de escucha de DESCUBRIMIENTO
 new.thread(listenDescubrimiento, sktUdp)
 # Inicio hilo de chequeo de vencimiento de DESCUBRIMIENTO
 new.thread(checkDescubrimiento)
 #creo socket para atender clientes de DATOS por TCP
 server = socket.tcp()
 server.bind(*, 2023)
 server = server.listen()
 # Espero nuevos clientes para DATOS
 while true:
   client, err = server.accept()
   new.thread(atenderCliente, client)
 end
 server.close()
 sktUdp.close()
end

def  checkDescubrimiento()
   while true do
       # Ejecuto cada 60 segundos
       sleep 60
       # para cada servidor chequeo vencimiento de tiemstamp
       mutex.acquire()
       for k, v in Servidores.items():
           # chequeo timestamp es mayor a 60 segundos
           if now() - v > 60 then
               remove.Servidores(ip)
           end
       end
       mutex.release()
    end
end

def  listenDescubrimiento( socket desc)
   sendUdp = now()
   while true do
        # Espero datagrama con ANNOUNCE,
        # si no llega nada espero hasta 10 segundos
        datagram, ip, port = desc.receive(10)
        # chequeo si no se envio por mas de 29 segundos
        # broadcast_addr contiene la dirección de broadcast red local
        if now() - sendUdp > 28 then
           desc.sendto("ANNOUNCE\n", broadcast_addr, 2222)
           sendUdp = now()
        end
        # chequeo si el receive salio por timeout o recibio algo
        pos = string.pos(datagram, "ANNOUNCE\n")
        if pos > 0 then
            # descarto mensajes que no contengan ANNOUNCE\n
             next
```

```
        end
        # Bloqueo la lista para proceso de nuevo servidor
        mutex.acquire()
        if ip in Servidores then
           # modifico Servidores timestamp
            modify.Servidores(ip,now())
        else
           # agrego Servidores ip,timestamp
           add_dictionario.Servidores(ip,now())
        end
       mutex.release()
end

def  atenderCliente(socket client)
   # espero solicitud de get
   repeat
     r, err = client.receive()
     if err=="closed" then return end
     buff = buff + r
     # busco primer \n en datos recibidos
     pos = string.pos(buff, "\n")
   until (pos > 0)
   dato = buff.split(' ')
   # En la posicion 0 esta GET y en la 1 el clave a buscar
   if ( dato[0] == 'GET' ) then
      # consulto si es local y si lo es devuelvo resultado
      result == getValue(dato[1])
      if result != 'NONE' then
         remain, err = client.send(result + '\N')
         client.close()
         return
      end
      # obtengo IP origen de la conexión TCP
      ip_orig, port = client.gethost()
      # Veo si cliente no es uno de los servidores
      # en ese caso consulto demas servers
      if not (ip_orig in Servidores)  then
         # Bloqueo la lista de clientes
         mutex.acquire()
         for k, v in Servidores.items():
             # consulto por dato[1]
              if k != MY_IP and result == 'NONE' then
                  # busco en servidor k
                  result=askValue(k,dato[1])
             end
          end
          # Libero la lista de clientes
          mutex.release()
      end
      # envio resultado
       remain, err = client.send(result + '\N')
       client.close()
  end
end

def  askValue(ipServer,key)
 # creo socket para lista con servidores y obtener datos
 clientAux = socket.tcp()
 client, err = clientAux.connect(ipServer, 2023)
 if err then
    close(clientAux)
    return('ERROR')
```

```
 end
 # envio consulta
 data = 'GET ' + key +'\n'
 remain, err = clientAux.send(data)
 repeat
     r, err = clientAux.receive()
     if err=="closed" then return('ERROR') end
     buff = buff + r
     pos = string.pos(buff, "\n")
  until (pos > 0)
  # devuelvo hasta \n en datos recibidos
  dato = buff[:pos]
  close(clientAux)
  return(dato+'\n')
end
```