Redes de Computadoras

Solución – 20 de febrero 2017

(ref: solrc20170220.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.**
- **Sólo 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 (5 puntos)

Un switch que tiene dos VLANs definidas recibe una trama ethernet con dirección destino broadcast por uno de sus puertos (que pertenece a una de las VLANs definidas). Explique qué hace el switch con esta trama.

Solución

La trama ethernet será re-enviada por todos los puertos del switch que tengan asignados la VLAN del puerto de entrada, exceptuando el puerto por el que llegó la trama.
Por los puertos asignados con la otra VLAN, no envía nada.

Pregunta 2 (10 puntos)

a) Mencione los mecanismos provistos por TCP para que un emisor pueda inferir que existe congestión en la red.

b) Explique en qué consiste la etapa del algoritmo de control de congestión conocida como "arranque lento", y cuándo termina.

Solución

a) TCP implementa un control de congentión sin notificación explícita, por eso se habla de mecanismos que permiten inferir (o deducir) que está ocurriendo congestión. Los dos mecanismos provistos por TCP para este fin son:
1) la recepción de tres ACKs duplicados, que indican congestión leve.
2) timeout, que indica congestión grave.

b) En la etapa de arranque lento, la **Ventana de Congestión** del emisor se inicializa en 1 MSS, y se incrementa en 1 MSS por cada ACK recibido, lo que implica un crecimiento exponencial de la velocidad de trasmisión. Esta etapa finaliza cuando se alcanza el **umbral** de congestión, que determina el comienzo de la etapa de "evitación de la congestión". El umbral queda definido por el valor **Ventana de Congestión /2** cuando se produjo el evento de congestión anterior a la etapa actual de arranque lento.

Pregunta 3 (10 puntos)

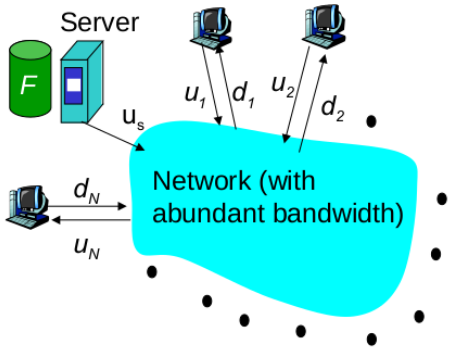
Compare el tiempo de distribución de un archivo de tamaño F desde un servidor hacia N clientes, considerando i)una arquitectura cliente-servidor, y ii) una arquitectura P2P.

Solución

`i) Cliente-servidor (transparencias del curso):`

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download



Time to distribute F to N clients using client/server approach

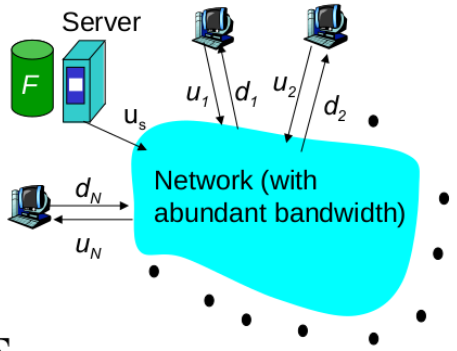
$$= d_{cs} = \max \{ NF/u_s, F/\min_i(d_i) \}$$

increases linearly in N (for large N)

`ii) P2P (transparencias del curso):`

File distribution time: P2P

- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \sum u_i$



$$d_{P2P} = \max \{ F/u_s, F/\min_i(d_i) , NF/(u_s + \sum u_i) \}$$

Pregunta 4 (10 puntos)

Considere los algoritmos de enrutamiento vistos en el curso.

a) ¿Qué es un algoritmo de vector-distancia?
b) ¿Qué es un algoritmo de estado del enlace?
c) Compare en función de la robustez de cada tipo de algoritmo.

Solución

Todo algoritmo de enrutamiento busca calcular las rutas de costo mínimo entre cualquier pareja origen-destino.

a) Los algoritmos de vector-dictancia (Distance-Vector, DV) utilizan información propia y la aportada por los vecinos para calcular un vector de estimaciones de costo hacia los nodos conocidos de la red. Son algoritmos distribuidos y asíncronos, ya que cada nodo ejecuta una instancia del algoritmo independientemente del resto de los nodos; es también iterativo, ya que cada nodo, durante el proceso de cómputo, comunica a sus vecinos su vector de distancias si el mismo sufre modificaciones. El algoritmo termina cuando no se reciben más actualizaciones, y puede recomenzar ante la ocurrencia de un evento, por ejemplo el cambio en un costo de un enlace. Un ejemplo típico es el algoritmo de Bellman-Ford.

b) Los algoritmo de estado del enlace (Link State, LS) utilizan información topológica completa de la red, aprendida dinámicamente, para resolver el problema de los caminos de costo mínimo. Estos algoritmos pueden ejecutarse en forma distribuida o centralizada; en el caso de IP (OSPF, IS-IS), cada nodo ejecuta su instancia sobre su vista de la información global. Un ejemplo típico es el algoritmo de Dijkstra, que para cada nodo construye un árbol de cubrimiento mínimo cuya raíz es el nodo en el que se ejecuta.

c) Los algortimos LS comunican costos de enlaces, y cada nodo computa los costos mínimos independientemente de los demás, mientras que en DV se comunican (y propagan) los costos de los caminos al resto de los nodos, iterando sobre estos cambios. Por lo tanto un fallo en DV puede potencialmente afectar a toda la red, mientras que en LS el efecto está acotado.

Pregunta 5 (5 puntos)

Explique qué servicios ofrece la Capa de Transporte del modelo TCP/IP.

Solución

La capa transporte (capa 4 del modelo) proporciona una comunicación lógica entre los procesos de las aplicación que se ejecutan en distintos sistemas finales (“hosts” o “end systems”), independiente de las redes físicas que los conectan.

Los protocolos de transporte se ejecutan en los sistemas finales e implementan:

- del lado del transmisor: genera segmentos a partir de los mensajes de la Aplicación, y los pasa a la capa de red.
- del lado del receptor: a partir de los segmentos, los mensajes son pasados a la capa de Aplicación.

En Internet la capa de transporte ofrece dos alternativas: TCP y UDP. Mientras TCP ofrece un transporte orientado a conexión y confiable (entrega de segmentos en orden y sin errores), UPD ofrece un servicio no orientado a conexión, libre de errores.

Problema 1 (30 puntos)

Solución

Se necesitan los siguientes prefijos:
Sitio #1 - /23
Sitio #2 - /22
Sitio #3 - /25
Además se necesitan 3 prefijos /30 para los enlaces.

i) A partir de un prefijo más grande que el /22, los prefijos se pueden asignar de la siguiente forma:

```
/21 _ _ /22 (Sitio #2)
     |_ /22 _ _ /23 (Sitio #1)
              |_ /23 _ _ /24
                       |_ /24 _ _ /25 (Sitio #3)
                                |_ /25 - se puede subdividir para los enlaces
```

Si tomamos como ejemplo 50.50.8.0/21, los prefijos que nos interesan son:
50.50.8.0/22 (Sitio #2)
50.50.12.0/22
50.50.12.0/23 (Sitio #1)
50.50.14.0/23
50.50.14.0/24
50.50.15.0/24
50.50.15.0/25 (Sitio #3)
50.50.15.128/25 - a partir de este último /25 se pueden generar los enlaces:
50.50.15.252/30 (Enlace R1-R2)
50.50.15.248/30 (Enlace R1-R3)
50.50.15.244/30 (Enlace R2-R3)

Dado que el prefijo asignado es **independiente de los proveedores**, se pude implementar redundancia publicando idénticos prefijos en la configuración BGP de R1, R2 y R3: el prefijo /21. En el ejemplo, todos anunciarán 50.50.8.0/21.

ii) En este caso el proveedor 1 asigna el prefijo /23 del Sitio #1, mientras que el proveedor 2 deberá asignar un prefijo /21 para los sitios #2 y #3. El cálculo para estos sitios es igual que antes:

```
/21 _ _ /22 (Sitio #2)
     |_ /22 _ _ /23
              |_ /23 _ _ /24
                       |_ /24 _ _ /25 (Sitio #3)
                                |_ /25 - se puede subdividir para los enlaces
```

Se puede usar cualquier prefijo para los enlaces internos, porque no necesitan ser alcanzables desde la internet global; por sencillez se puede usar el /25 señalado.

Como ejemplo, el prefijo asignado por el proveedor 1 podría ser:
1.1.12.0/23 (Sitio #1)

El prefijo asignado por el proveedor 2 podría ser 2.2.8.0/21, que se subdivide de la siguiente forma:

2.2.8.0/22 (Sitio #2)
2.2.12.0/22
2.2.12.0/23
2.2.14.0/23

2.2.14.0/24
2.2.15.0/24
2.2.15.0/25 (Sitio #3)
2.2.15.128/25 - a partir de este último /25 se pueden generar los enlaces:
2.2.15.252/30 (Enlace R1-R2)
2.2.15.248/30 (Enlace R1-R3)
2.2.15.244/30 (Enlace R2-R3)

En este caso los prefijo son propios de cada proveedor, y por lo tanto no es posible implementar redundancia entre proveedores. En este caso la configuración de BGP de R1 anunciará el prefijo 1.1.12.0/23, mientras que la configuración BGP de R2 y R3 es idéntica y anuncia el prefijo 2.2.8.0/21 (hay redundancia de enlaces con el proveedor 2).

Problema 2 (30 puntos)

Solución

Cliente:

```
server_ip = arg[1]
command   = arg[2]
file_name = arg[3] --opcional

client_fd = socket.connect(server_ip, 3001)

--esta función sabe como recibir un archivo. Al terminar cierra la aplicación
function accept_transfer ()
  master_fd = socket.tcp()  --master
  master_fd:bind('*', 3002) --escuchamos en todas las interfaces
  file_fd = io.open(file_name, 'wb') --abrimos archivo para escritura
  receiver_fd = master_fd.accept()
  --consumimos del socket mientras llegan datos
  buffer = receiver_fd:read() --bloqueante, devuelve nil cuando lo cierran
  while buffer do
    file_fd:write(buffer)
    buffer = receiver_fd:read()
  end
  --al descargar todo, finaliza la aplicación
  file_fd:close()
  client_fd:close()
  os.exit()
end

if command == 'GET' then
  --si solicitamos un archivo, creamos un servidor donde escucharlo (en un hilo)
  Thread.new(accept_transfer)
end

--enviamos parámetros al servidor
client_fd:write(command .. '\n')
if file_name client_fd:write(file_name .. '\n') end

--quedamos escuchando respuesta
buffer = receiver_fd:read_line() --bloqueante, devuelve nil cuando lo cierran
while buffer do
  print(buffer)
  buffer = receiver_fd:read_line()
end
client_fd:close()
```

Servidor:

```
master_fd = socket.tcp()  --master
master_fd:bind('*', 3001)

function process_client (connection_fd)
  while true do --con este while soportamos clientes “persistentes”
    line = connection_fd:read_line() --devuelve linea, nil si cierran conexión
    if not line then
      client_fd:close()
      return --terminamos hilo porque cerraron la conexión
    end
    command, filename = parse_command(line) -- extraemos patametros del string.
    if command == 'LIST' then
      connection_fd:write(get_available_files())
    end
    if command == 'GET' then
      file_fd = io.open(file_name, 'r') --abrimos archivo para lectura
      if not file_fd then
        connection_fd:write('ERROR\n') --archivo inexistente
      else
        transfer_fd = socket.connect(connection_fd.get_ip(), 3002)
        transfer_fd:write(file_fd:read('all'))
        transfer_fd:close()
      end
    end
  end
end

while true do
  connection_fd = master_fd.accept()
  Thread.new( process_client, connection_fd)
end
```