Pràctica 4. Implementació de Protocols de transport. Pràctica de multiplexat / demultiplexat de les comunicacions de transport. Laboratori d'Aplicacions i Serveis Telemàtics

Josep Cotrina, Marcel Fernandez, Jordi Forga, Juan Luis Gorricho, Francesc Oller

1. Introducció

Es vol implementar la funcionalitat de multiplexat de les comunicacions a la capa de transport. A diferència de la pràctica anterior, a on només es treballava amb una única comunicació de transport entre dos nodes, ara es vol gestionar més d'una comunicació de transport entre dos nodes concrets.

Per aconseguir aquest objectiu s'introdueix el concepte de port associat a una comunicació, de tal forma que cada comunicació queda completament identificada mitjançant la quàdrupla:

`<IP_origen, port_origen, IP_destí, port_destí>.`

Més d'una comunicació establerta entre els mateixos dos nodes extrems tindran les mateixes adreces: `IP_origen` i `IP_destí`. Fins i tot, més d'una comunicació pot tenir el mateix `port_origen` ó el `mateix port_destí` (aquest últim és el cas habitual quan s'està accedint al mateix servidor web des de diferents navegadors que pengen d'Internet). Però dues comunicacions establertes entre els mateixos dos nodes, mai podran tenir simultàniament els mateixos: `port_origen` i `port_destí`, llavors serien indistingibles.

Conseqüentment, el demultiplexat dels paquets IP que arriben a destinació, per lliurar el contingut d'aquests (segment de transport) a la comunicació de transport corresponent, es fa en base a la comparació de la quàdrupla definida anteriorment amb totes les quàdruples guardades en local al node de destí, quàdruples corresponents a totes les comunicacions de transport obertes en aquell moment.

En el nostre cas en particular, amb motiu de la necessària simplificació de la pràctica, no fem servir adreces IP per identificar els nodes, ja que la pràctica es desenvolupa mitjançant l'ús d'un canal simulat d'enviament de segments de transport entre dos nodes únics i establerts a priori. Conseqüentment, a la pràctica es demanarà el demultiplexat dels segments de transport que ens arriben pel canal simulat, pertanyents a comunicacions diferents, fent servir únicament la dupla: `<port_origen, port_destí>`.

A la pràctica anterior només es treballava amb una única comunicació; i per tant, tota la funcionalitat de la capa de transport quedava recollida dins de la classe Java: `TSocket`, en les seves dues modalitats: `TSocketSend` i `TSocketRecv`. Ara, amb la necessitat de multiplexar varies comunicacions de transport entre dos nodes, es diferencien els conceptes de **Socket** i de **Protocol**. El **Socket** és propi d'una comunicació de transport en particular, mentre que el **Protocol** ha de gestionar la operativa conjunta de més d'una comunicació de transport alhora. Conseqüentment, la implementació de les classes Java: `ProtocolSend` i `ProtocolRecv` hauran de gestionar les llistes de Sockets oberts de transmissió i recepció respectivament, implementant la funcionalitat de multiplexat/demultiplexat de les comunicacions de transport.

2. Exercicis.

Per la consecució d'aquesta pràctica es demana:

1. Analitzar el codi de les classes Java:

   - `Main`
   - `Sender`
   - `Receiver`

   per entendre la proposta de pràctica a on es treballa amb dues comunicacions de transport sobre el mateix canal de transport simulat. Conseqüentment es fan servir 4 fils d'execució a nivell d'aplicació, dos fils al node emissor i dos fils al node receptor. **Completa el codi de la classe Main**.

2. Analitzar la proposta de codi de les classes Java:

   - `ProtocolSend`
   - `ProtocolRecv`

   per entendre l'arquitectura necessària que fa possible el multiplexat/demultiplexat de varies comunicacions de transport operant sobre el mateix canal simulat. **Més concretament entendre perque en les dues classes hi ha una llista de `TSockets`**.

3. Completar el codi de les classes Java:

   - `ProtocolSend`
     - El mètode `openForOutput(int localPort, int remotePort)` prepara un nou `TSocketSend` per una nova connexió.
   - `ProtocolRecv`
     - El mètode `openForInput(int localPort, int remotePort)` prepara un nou `TSocketRecv` per una nova connexió.
     - El mètode `TSocketRecv getMatchingTSocket(int localPort, int remotePort)` busca el `TSocketRcv` a qui va dirigit un determinat segment.
     - El mètode `ipInput(Segment segment)` fa que el `TSocketRcv` adequat processi el segment rebut.

   per aconseguir la implementació de la funcionalitat de multiplexat/demultiplexat desitjada.