Capitolo 1

Transmission Error Detector

In questo capitolo si vuole illustrare le fasi di progettazione e implementazione di un modulo TED per ABPS per kernel Linux 4.0. Nel precedente capito si è descritto brevemente il funzionamento di TED e il suo funzionamento nel sistema ABPS pensato per il supporto alla mobilità.
Il meccanismo di Transmission Error Detection può essere applicato a qualsiasi interfaccia di rete di un dispositivo mobile. In questa capitolo verrà illustrata la progettazione di un modulo TED per Wi-Fi.
Transmission Error Detector è implementato in maniera *cross-layer* lungo lo stack di rete del kernel Linux e ha lo scopo di monitorare ciascun datagram UDP in invio da una certa interfaccia di rete e notificare ad un eventuale ABPS proxy client lo stato di consegna del pacchetto all'access point. Per far ciò TED introduce una notifica di tipo *First-hop Transmission Notification* che sarà fatta pervenire al proxy client.
Per monitorare uno specifico datagram il proxy client necessità di un identificativo che identifichi univocamente un dato datagram. A tal proposito si è estesa la system call *sendmsg* in modo tale che ritorni un id univoco per il messaggio appena spedito.

1.1 Progettazione e implementazione

Transmission Error Detector opera su più layer dello stack di rete del kernel Linux e ha il compito di monitorare ciascun datagram trasmesso da un proxy client ABPS e notificarne lo stato di consegna in modo tale da poter valutare la QoS di quel collegamento ed eventualmente inoltrare quello stesso datagram attraverso un'altra NIC del nodo mobile.
Come già precedentemente accennato TED è implementato in maniera cross-layer su più livelli dello stack di rete del kernel di Linux.

1.1.1 Livello trasporto

Le applicazioni per le quali ABPS vuole essere un supporto alla mobilità sono le applicazioni multimedia-oriented che come già trattato nei capitoli precedenti sono solitamente progettate per utilizzare UDP. Un'applicazione che utilizza UDP per le sue comunicazioni di rete sfrutta uno o più socket datagram ovvero di tipo connectionless.
Per poter valutare la ritrasmissione di un dato messaggio il proxy client necessita di un meccanismo di identificazione per ciascun datagram. A tal proposito si è esteso la system call *sendmsg* in modo tale che possa ritornare un **id** univoco per il messaggio in invio cosicché il proxy client possa mantenerlo e usarlo per riferirsi a quel preciso datagram. Tutte le notifiche ricevute poi in seguito dal proxy client e provenienti da TED faranno riferimento a un datagram utilizzando lo stesso identificativo.
La system call sendmsg consente di inviare, assieme al contenuto del messaggio, delle informazioni di controllo aggiuntive dette *ancillary data* che non saranno però trasmesse lungo la rete. Dal punto di vista implementativo i dati di tipo ancillary sono realizzati in POSIX tramite una sequenza di strutture **struct cmsghdr** contenenti le informazioni di controllo. L'estensione di sendmsg in particolare prevede l'utilizzo di ancillary data in fase di invio:

- Per segnalare a TED che l'app invocante la system call sendmsg richiede di poter ricevere l'id per il messaggio in invio. Per far ciò viene

introdotto un nuovo valore non utilizzato per il campo **cmsg_type** della struct cmsghdr.

- Per passare a TED un'indirizzo di memoria in user space dove TED potrà assegnare l'id generata per il datagram in invio.