Capitolo 1

Transmission Error Detector

In questo capitolo si vuole illustrare le fasi di progettazione e implementazione di un modulo TED per ABPS su kernel Linux 4.0.
Nel precedente capito si è descritto brevemente il funzionamento di TED e il suo ruolo nel sistema ABPS ideato come 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.

1.1 Progettazione e implementazione

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 in modo tale da poter valutare la QoS del collegamento attualmente utilizzato per la trasmissione.
Per realizzare ciò TED introduce una notifica di tipo *First-hop Transmission Notification* che sarà fatta pervenire al proxy client.
Come già precedentemente accennato TED è implementato in maniera cross-

layer su più livelli dello stack di rete del kernel di Linux.
Qui di seguito verrà descritto nel dettaglio la sua implementazione lungo tutto lo stack di rete 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.

Una volta inviato il messaggio e il Socket Interface Layer avrà passato il controllo alla funzione di livello trasporto udp_sendmsg, come spiegato nel capitolo 2, tramite una funzione appositamente sviluppata e contenuta nel file socket.c del kernel Linux, si andrà ad analizzare i dati di tipo ancillary contenuti nella struttura struct msghdr passata come parametro della udp_sendmsg. La struttura struct msghdr mantiene il messaggio passato alla system call dall'app a livello utente e gli eventuali dati di controllo: una volta analizzati gli ancillary data se l'utente ha specificato di essere interessato a ricevere l'identificativo del messaggio inviato verrà settato un apposito flag is_identifierŕequired e verrà mantenuto l'indirizzo di memoria user space specificato l'app dove TED potrà settare l'identificativo calcolato per il pacchetto in invio.
Quando il controllo viene passato momentaneamente a livello rete, come descritto nel capitolo 2, e verrà allocato il socket buffer relativo al messaggio in invio, sarà calcolato l'identificativo per quel pacchetto e assegnato alla struttura sk_buff (maggiore dettagli in seguito).
Non appena il flusso dell'esecuzione viene ripreso dalla primitiva udp_sendmsg, se l'allocazione del socket buffer è andata a buon fine, nell'indirizzo di memoria user space precedentemente ricavato dagli ancillari data del messaggio passato a livello udp tramite primitiva put_user (macro che consente di copiare un certo valore presente in kernel space a partire da un certo indirizzo user space) verrà copiato l'identificativo.

1.1.2 Network Layer

A livello rete sono state implementate alcune modifiche in particolare verrà calcolato l'identificativo di ciascun pacchetto e assegnato a una struttura di tipo sk_buff appena viene allocata.
Nella precedente versione sviluppata di TED per Kernel Linux 2.6.30-rc5 come identificativo da utilizzare a livello applicativo per monitorare i singoli messaggi spediti veniva usato il campo ID (per maggiori dettagli si veda il primo capitolo) del datagram IPv4. Una volta che veniva calcolato e settato

questo campo della struttura iphdr che rappresenta un header IPv4, il valore veniva passato a livello utente.

Nella versione sviluppata per kernel Linux 4.0 viene introdotto il supporto a IPv6.

L'header IPv6, supportando solamente al frammentazione end-to-end, non è caratterizzato dal campo Identification come l'header della precedente versione dell'Internet Protocol.

Per ovviare a questo problema si è definito