Progetto Sistemi Incapsulamento

Lorenzo Canocchi, Emanuele Polini, Christian Falzetta

Marzo 2025

1 Introduzione

Nel contesto delle reti di calcolatori l'incapsulamento è un processo fondamentale che consente di organizzare e trasmettere i dati attraverso i vari livelli del modello TCP/IP.
Questo progetto, sviluppato in C++, ha lo scopo di simulare il processo di incapsulamento ai livelli rete e collegamento producendo un frame Ethernet conforme allo standard Ethernet 2.0.

2 Descrizione

Il programma prende in input un messaggio da un file di testo e, seguendo il principio dell'incapsulamento, lo incorpora in una struttura gerarchica di pacchetti.
Il messaggio, convertito in binario, viene prima inserito in un pacchetto IP che a sua volta viene incapsulato all'interno di un header Ethernet.
Le informazioni necessarie per compilare i vari header vengono estratte da due file di configurazione, mentre i codici di controllo degli errori (HeaderChecksum IP e CRC Ethernet) vengono calcolati dinamicamente.
L'output del programma è un file contenente la rappresentazione binaria dell'intero frame Ethernet pronto per essere trasmesso su un mezzo fisico.

3 Organizzazione

Il progetto è organizzato in diversi file, ognuno con un ruolo specifico, per garantire una chiara separazione tra la logica di elaborazione, i dati di configurazione e il messaggio da incapsulare.

3.1 File di codice sorgente

- **funzioni.h**: contiene la dichiarazione delle struct Frame e Datagram e dei prototipi delle procedure e funzioni utilizzate in funzioni.cpp.

- **funzioni.cpp**: implementa le funzioni e le procedure per la popolazione delle struct, l'incapsulamento, il calcolo dei codici di controllo degli errori e la generazione del frame binario.
- **main.cpp**: è il programma da eseguire e contiene l'inizializzazione delle struct Frame e Datagram e le chiamate alle principali funzioni e procedure.

3.2 File di configurazione

- **configurationFrame.txt**: contiene i dati per la realizzazione dell'header Ethernet.
- **configurationDatagram.txt**: contiene i dati per la realizzazione dell'header IP.

3.3 File di input e output

- **messaggio.txt**: contiene il messaggio da incapsulare nel pacchetto IP.
- **frame.txt**: file generato contenente il frame Ethernet finale in formato binario.

4 Funzioni e Procedure

In questa sezione verranno descritte tutte le funzioni e tutte le procedure utilizzate per poter realizzare questo incapsulamento.

- **string readFile(string file)**: viene chiamata passandole il nome del file da cui leggere; restituisce un stringa contenente la prima riga del file.
- **void writeFile(string mess, string file)**: viene chiamata passandole la stringa da scrivere e il nome del file; scrive la stringa su quel file.
- **string charToBin(char c)**: viene chiamata passandole un carattere; ne restituisce la conversione in binario sotto forma di stringa.
- **string stringToBin(string s)**: viene chiamata passandole una stringa; sfruttando la funzione charToBin restituisce una stringa contenente la sua conversione in binario.
- **int binToInt(string s)**: viene chiamata passandole una sequenza binaria sotto forma di stringa; ne restituisce il corrispettivo valore intero. La utilizziamo per il calcolo dell'HeaderChecksum.
- **string intToBin(int n, int dim)**: viene chiamata passandole un numero intero da convertire e il numero di bit su cui rappresentarlo; restituisce una stringa contenente la sequenza binaria. La utilizziamo per il calcolo dell'HeaderChecksum.

- **string calculateHc(Datagram ip)**: viene chiamata passandole tutta la struct Datagram; la suddivide in gruppi da 16bit, li somma tenendo conto del riporto e restituisce il complemento a 1 del risultato che corrisponde all'HeaderChecksum.
- **void setDatagram(Datagram & ip)**: viene chiamata passandole tutta la struct Datagram per riferimento; legge ogni riga del file configurationDatagram.txt e tramite l'uso di un separatore capisce a quale campo dell'header si riferisce; popola il corrispettivo campo della struct con la sequenza binaria presa da quella riga e infine popola i campi HeaderChecksum e messaggio.
- **string datagramToBin(Datagram ip)**: viene chiamata passandole tutta la struct Datagram; concatena ogni campo della struct in una stringa e la restituisce convertita in binario.
- **string calculateCRC(Datagram ip)**: viene chiamata passandole tutta la struct Datagram; divide tutto il datagramma per un polinomio generatore tramite l'utilizzo dello XOR e ne restituisce il resto.
- **void setFrame(Frame & ethernet, string ip)**: viene chiamata passandole tutta la struct Frame per riferimento e tutto il datagram in binario; legge ogni riga del file configurationFrame.txt e tramite l'uso di un separatore capisce a quale campo dell'header si riferisce; popola il corrispettivo campo della struct con la sequenza binaria presa dalla quella riga e infine popola il campo CRC e datagram.
- **string frameToBin(Frame ethernet)**: viene chiamata passandole tutta la struct Frame; concatena ogni campo della struct in una stringa e la restituisce scritta in binario.

5 Osservazioni e Conclusioni

Attualmente il progetto non considera la presenza di opzioni a livello rete né il calcolo del padding. Se volessimo potremmo renderlo completo inserendo le opzioni nel file di configurazione e calcolando il padding a partire dalla lunghezza del campo option.
La principale difficoltà l'abbiamo riscontrata nella conversione di un carattere in binario in quanto la logica era corretta ma non stavamo tenendo conto del fatto che il modulo della variabile char da convertire restituiva 1 o 0 che però nella codifica ASCII corrispondevano allo spazio e al punto esclamativo. Siamo riusciti a risolvere la cosa sommando al risultato del modulo la costante 48(valore intero del carattere '0').
In conclusione l'implementazione diretta di questi meccanismi ci ha permesso di comprendere meglio a livello pratico il funzionamento dell'incapsulamento e l'importanza dei vari campi degli header Ethernet e IP.