# Gestione di reti 16/17

Davide Pizzato 521991

## Introduzione

Lo scopo del progetto è di intercettare il traffico DNS per poterne poi ricavare statistiche di utilizzo. All'interno della cartella sono presenti i file sgr.c (sorgente), relazione.pdf (questo), makefile (con le istruzioni per l'esecuzione) e il file results.txt (esempio di output).

## Esecuzione

Il lavoro è stato sviluppato e testato su Ubuntu 17.04 64bit in macchina virtuale con host Windows 10 64bit con virtualbox. Pertanto è stato possibile testare un'unica interfaccia di connessione, ovvero quella diretta tra macchina virtuale e host, che utilizza un unico server DNS.

Per eseguire il programma è necessario prima installare libpcap, quindi compilare il programma ed infine eseguirlo. Tutte queste azioni sono eseguibili tramite il comando make.

```
make [help]
```

descrive, intuitivamente, come utilizzare il comando make. Di seguito riporto un esempio di comandi per eseguire correttamente il programma

Installazione libpcap

```
make dependencies
```

Compilazione:

```
make compile
```

Esecuzione (è possibile indicare manualmente l'interfacia di cattura, altrimenti ne verrà utilizzata una valida tramite la funzione pcap_lookupdev):

```
make test [i=interface_name]
```

Pulizia cartella dai risultati e dall'eseguibile:

```
make clean
```

Tutte queste operazioni sono descritte anche all'interno del makefile.

È possibile inoltre, una volta compilato il progetto, utilizzare l'help tramite

```
./sgrout -h
```

In quanto presenti commenti utili e una variabile di configurazione (impostata ad un valore) si consiglia di aprire il file sorgente prima di eseguire il file generato con i comandi sopra elencati.

# Funzionamento

Il funzionamento del progetto è molto semplice e verrà di seguito spiegato sfruttando le funzioni che lo compongono, partendo dalla funzione principale

### *main*

La funzione main() contiene la parte di programma più generale, che serve all'inizializzazione dei dati strettamente necessari per tutta la durata del progetto e alla gestione globale del programma. La parte però più interessante di gestione dei pacchetti è contenuta nella funzione packet_handler che vedremo in seguito. La main quindi, in sequenza, esegue queste operazioni:

- Ricava un devices disponibile dal quale verranno catturati i pacchetti (nel caso di test la scheda di rete che collega la macchina virtuale al sistema host);
- Apre il 'flusso' dal quale catturare i pacchetti;
- Applica i filtri necessari (filtro i pacchetti sulla porta 53. È possibile applicarli manualmente in un secondo momento semplicemente leggendo l'header udp, ma per semplicità è stato fatto qui);
- Inizia ad intercettare i pacchetti. Questi vengono quindi fatti gestire come detto precedentemente alla funzione packet_handler;
- Al termine chiude il flusso e stampa i risultati

### *packet_handler*

Questa funzione come detto è la responsabile della gestione dei pacchetti intercettati. Difatti ne fa una prima lettura. Dall'header del pacchetto fornito da libpcap ricava il timestamp di invio/ricezione, dall'header ethernet il tipo di traffico (che se non è UDP viene scartato in quanto il DNS lavora generalmente sulla porta UDP 53), dall'header ip del pacchetto ne ricava ip-mittente e ip-destinatario e dall'header dns il tipo di messaggio (se query o risposta). Infine basandosi su quest'ultimo campo, il tipo di messaggio, aggiunge una richiesta dns alla struttura dati principale o ne aggiunge una risposta.

### *addData*

Essendo la struttura dati principale una lista (contenente un'altra lista che ne contiene a sua volta un'altra) viene allocato, se necessario lo spazio per l'aggiunta di un nuovo elemento (un nuovo client) in base all'ip ricavato dall'header ip. Quindi aggiunge alla lista dei DNS il server contattato, tramite la funzione

### *addDns*

Come detto aggiunge un nuovo server DNS alla lista dei server contattati (se non è presente, altrimenti lo aggiorna). Aggiunge l'ip del server, il numero di richieste effettuate a quest'ultimo e inserisce la nuova richiesta nella lista dedicata, con la funzione

### *addRequest*

L'unico scopo è aggiungere la richiesta alla lista. Ogni richiesta è identificata dall'id, che viene appunto utilizzato come chiave.

### *addResponse*

Questa funzione invece gestisce l'arrivo di una risposta da un server DNS. È possibile sapere se un messaggio DNS è risposta dal campo corrispondente dell'header dns. Cerca quindi prima l'ip del destinatario all'interno della lista di client, quindi l'ip sorgente (del server DNS) all'interno della lista dei dns contattati dal client e quindi tramite l'id può aggiornare lo stato della richiesta. Se l'id non è presente la richiesta viene scartata come sconosciuta e il programma continua la sua esecuzione. Inoltre tramite il campo dell'header dns relativo, controlla se la risposta ha generato errori o se la risposta è una risposta valida (che contiene effettivamente qualcosa) e la tratta come tale, aggiornando il tempo di ricezione della risposta e le varie statistiche.

### *printList*

Quest'ultima funzione è la responsabile della stampa dei risultati dell'esecuzione del programma e viene eseguita una volta che il programma principale ha terminato di catturare pacchetti.

## Output

L'output del progetto è contenuto nel file

```
results.txt
```

nel caso sia stato utilizzato il comando make per eseguire il programma. Il contenuto del file è auto-descrittivo e contiene tutte le statistiche oltre che le richieste effettuate. Per ogni client presente nella lista vengono stampati i server dns contattati e le singole richieste effettuate, con l'aggiunta del tempo di risposta nel caso ne abbiano avuta una. Quindi al termine di ogni server DNS contattato da un client vengono stampate le statistiche relative alla coppia client-server. Terminati i server DNS utilizzati dal client vengono quindi stampate le statistiche globali relative al client (senza contare quindi le eventuali differenze di server e quindi di velocità o affidabilità).