UNIVERSITÀ
DI PISA

RELAZIONE DI GESTIONE DI RETI

Gianluca Vizziello

Email: g.vizziello@studenti.unipi.it

24 settembre 2025

1 Introduzione

In questa relazione vengono illustrate le modifiche apportate alla libreria nDPI per l'implementazione di una nuova funzionalità che consente di individuare e conservare metadati nei pacchetti mDNS.

Grazie al protocollo mDNS, i dispositivi embedded all'interno di una rete locale possono annunciare automaticamente i servizi che offrono, senza necessità di configurazioni manuali. La libreria include già un dissector per i pacchetti DNS, in particolare il file `dns.c`, in cui sono state apportate la maggior parte delle modifiche necessarie per questa funzionalità. La struttura dati principale, `ndpi_flow_struct`, è stata estesa in modo da memorizzare i servizi annunciati nei pacchetti. Approfondiremo in seguito i dettagli di queste due principali modifiche, ma per ora riportiamo altri cambiamenti minori apportati ad altri file:

- `ndpiReader.c`: al fine di visualizzare i metadati raccolti, è stata aggiunta una sezione di codice alla funzione `printFlow()` che, dopo aver verificato la presenza di almeno un servizio individuato, aggiunge alla stringa di output le informazioni necessarie;
- `reader_util.c`: è stato aggiunta una sezione di codice in `ndpi_flow_info_free_data()` per liberare la memoria occupata dai servizi e in `process_ndpi_collected_info()` per copiarli da `ndpi_flow_struct` alla struttura d'esempio usata in `ndpiReader.c`;
- `reader_util.h`: si è estesa `ndpi_flow_info`, ovvero la suddetta struttura di esempio, con lo stesso contenuto con cui si è estesa la struttura principale della libreria;
- `ndpi_main.h`: per liberare i metadati accumulati, è stato aggiunta una sezione di codice alla funzione `ndpi_free_flow_data()`;
- `Altri file di test`: le modifiche a `printFlow()` hanno portato di conseguenza a cambiamenti nell'output e dunque i file di test necessitavano di essere sincronizzati.

Le sezioni successive approfondiranno le modifiche principali, finalizzate alla raccolta dei metadati.

2 Struttura per i metadati

La feature richiedeva di **estendere la struttura** `ndpi_flow_struct` per consentire di conservare i dati annunciati dai pacchetti mDNS. L'idea è stata di avere innanzitutto una `struct` che potesse rappresentare un solo servizio mDNS e poi una `struct` interna a `ndpi_flow_struct` che conservasse una lista di servizi e il numero.

Per il primo punto, ho notato che nDPI possedeva già in `ndpi_typedefs.h` questa struttura:

```
PACK_ON
struct ndpi_mdns_rsp_entry {
    u_int16_t rsp_type, rsp_class;
    u_int32_t ttl;
    u_int16_t data_len;
} PACK_OFF;
```

Tuttavia, ciò non era sufficiente per conservare anche il nome di un servizio mDNS, o dati a riguardo, e il dispositivo che l'avesse annunciata. Inoltre, per i record SRV, non c'era un campo per conservare la porta annunciata (che è esattamente ciò a cui serve un record di questo tipo).

Di conseguenza la struttura in questione è stata **estesa** aggiungendo tre nuove entry:

```
struct ndpi_mdns_rsp_entry {
    u_int16_t rsp_type, rsp_class;
```

```
    u_int32_t ttl;
    u_int16_t data_len;
    char *name; // hostname
    char *data; // metadata
    u_int16_t srv_port;
};
```

Si noti che il packing è stato rimosso perché causava problemi di allineamento in alcune build al momento dell'allocazione.

In questa maniera, per i tre record principali, possiamo **interpretare la semantica** che mDNS dona ai pacchetti fedelmente:

- un record **PTR** serve ad annunciare che un certo servizio è disponibile, nel nostro caso `name` annuncerà `data` mentre `srv_port` è ignorato;
- un record **TXT** contiene metadati su un certo servizio annunciato, dunque a `name` saranno associati dei dati in `data`. `srv_port` è ancora una volta ignorato;
- un record **SRV** associa ad un servizio annunciato una porta, in questo caso `data` è ignorato e `srv_port` contiene la porta annunciata.

A questo punto è possibile estendere `ndpi_flow_struct` dichiarando dentro una nuova struttura:

```
struct {
    uint8_t num_services;
    struct ndpi_mdns_rsp_entry *services;
} mdns_metadata;
```

In particolare, `services` verrà allocata dinamicamente nel momento in cui il dissector `dns.c` troverà un servizio annunciato in un pacchetto mDNS. La dimensione di questo array è indicata in una nuova costante `MAX_NUM_MDNS_ADVERTISED_SERVICES` posta indicativamente a 8. Se vengono trovati più servizi di questi, vengono scartati.

Con queste modifiche è possibile supportare un aggiornamento di `dns.c` che richiede dei posti per poter inserire nuovi dati trovati: ci dedicheremo a questo nella prossima sezione.

3 Modifiche al dissector

`dns.c` non prevedeva molte differenze tra un pacchetto DNS classico e un pacchetto mDNS, ma la struttura sintattica dei due protocolli è la stessa. Cambia soltanto l'interpretazione del campo `rsp_type`, poiché in mDNS il bit più significativo impostato a 1 indica al ricevitore di flushare la cache, e l'interpretazione semantica di molti record, e ciò, almeno per i tipi di record supportati dalla patch, è stato discusso nella sezione precedente.

Ne consegue che, nel caso in cui mDNS sia stato individuato come protocollo, diventa sufficiente memorizzare i metadati di interesse in `mdns_metadata`.

Fatta questa considerazione, risulta un aspetto cruciale rendere la già esistente `ndpi_grab_dns_name()` in grado di **gestire correttamente qualsiasi nome**, inclusi i **nomi compressi** che non erano supportati dalla funzione. La mancanza di tale supporto rappresenta un ostacolo significativo all'obiettivo di conservare i metadati nel protocollo mDNS, per due motivi principali:

1. i record PTR tramite i quali i device annunciano la disponibilità di servizi, e che quindi costituiscono il cuore del protocollo mDNS, presentano sempre un puntatore ad un nome. Senza supporto ai nomi compressi, il resto delle modifiche avrebbe poco senso di esistere siccome nella stragrande maggioranza dei casi sarebbe impossibile capire quale servizio ha annunciato cosa;

2. anche molti nomi di device utilizzano la compressione e, per la chiarezza dell'output, è necessario averli in forma estesa.

Invece che trattarlo come un errore e ritornare un valore, ho cambiato il corpo dell'`if` che controlla se il pacchetto all'offset `off` ha un puntatore al suo interno. In questo caso diviene facile calcolarlo in questa maniera:

```
(*off)++;
u_int8_t byte2 = packet->payload[(*off)++];
u_int32_t ptr = ((cl & 0x3F) << 8 | byte2) + (packet->tcp ? 2 : 0);
```

dove ricordiamo `u_int8_t cl = packet->payload[*off]` (ossia `cl` contiene il primo byte di due del puntatore, che ha forma `11xxxxxx`), inoltre, per un pacchetto TCP, è richiesto di spostarsi di un offset di 2 byte. A questo punto diviene semplice creare una logica ricorsiva che segua la catena di puntatori per ricreare il nome intero.

Un problema ben noto, in caso di implementazione non corretta del dissector, è il rischio di incorrere in un **ciclo infinito di puntatori**. Questo può essere risolto in vari modi: impostando un numero massimo di ricorsioni oppure mantenendo, tramite una struttura dati, l'elenco dei puntatori già visitati.

In ogni caso, entrambe le soluzioni richiedono la creazione di una funzione interna, da invocare dopo una fase di preprocessing che inizializza un contatore o una struttura scelta. In questo proof of concept è stata adottata una **bitmap**, già disponibile nella libreria nDPI, utilizzata per verificare che ciascun indice non fosse stato visitato in precedenza; in caso contrario, l'indice viene registrato nella bitmap e successivamente visitato. Alternativamente, grazie alla struttura del codice, sarebbe divenuto facile adottare una qualsiasi altra metodologia (ad esempio, cambiare l'argomento `ndpi_bitmap *bitmap` di `ndpi_grab_dns_name_internal()` in un contatore inizializzato dalla funzione più esterna).

Risolto il problema dei nomi era sufficiente modificare il dissector stesso. Innanzitutto, come anche visto nel file `.pcap` di esempio nell'issue relativo alla feature da implementare, si richiedeva di estrarre i metadati nei pacchetti mDNS indiscriminatamente dalle query e dalle answer, siccome entrambe contengono servizi annunciati. Ne consegue che la funzione `search_dns()`, come concordato coi collaboratori della libreria, venisse estesa in modo tale che il branch `if(is_query)` fosse in grado di **processare anche le answer e le additionals** chiamando quindi `process_answers()` e `process_additionals()`. Si noti che questa ultima funzione, in attesa di approvazione per `process_answers()`, non è poi mai stata estesa per includere le modifiche per la feature.

4 Estensione del dissector

L'ultima cosa modificata, quindi, è la funzione `process_answers()`, in cui si trova la principale componente della patch.

L'idea generale era, per ogni iterazione del ciclo `for`, di **raccogliere le informazioni** in `data` e alla fine aggiungere una nuova struttura `ndpi_mdns_rsp_entry` all'array in `mdns_metadata`. Come è possibile osservare, c'è circa all'inizio del ciclo qualche linea di codice per ottenere il nome del device che annuncerà i servizi, conservato in `char name[255]`. Ricordiamo che ora i nomi compressi sono supportati. Viene inoltre inizializzato il puntatore a `data` a `NULL`.

In seguito si ottiene il `rsp_type`. Notiamo che, per il protocollo mDNS, è necessario mascherare i bit per escludere il più significativo, che invece è contenuto in `u_int8_t cache_flush` (la patch non lo usa ma, in caso di utilizzo futuro, viene calcolato).

Di seguito vediamo come sono stati aggiornati o aggiunti i branch per i vari possibili record.

1. **PTR**: in questo caso il branch era già presente e serviva a ottenere il nome di dominio per il protocollo DNS. Tuttavia, per mDNS, l'area che questo branch va ad esplorare è di significativa importanza: qui è presente il nome di un servizio annunciato dal dispositivo memorizzato in

`name`. Dunque, se il protocollo è mDNS e il nome è valido (controllato nel codice con `len > 0`) si alloca `data` dinamicamente e ci si copiano i dati trovati. È fondamentale notare che ciò non è equivalente a copiare semplicemente il nome di dominio: quest'ultimo, memorizzato in `flow->protos.dns.ptr_domain_name`, verrebbe semplicemente riscritto $n-1$ volte se nel flusso sono presenti n servizi annunciati, non lasciando modo di memorizzare *tutti* i servizi annunciati con mDNS. La riscrittura avviene solo per quest'ultimo protocollo, siccome per gli altri la variabile `found` sarà 1 una volta che il nome di dominio è stato trovato.

2. **TXT**: il branch è stato creato da zero e le modifiche, per renderle meno impattanti possibile su altri protocolli che non fossero mDNS, sono state racchiuse in una guardia che controllasse il protocollo. Potenzialmente, tutte le modifiche sarebbero potute essere applicate ugualmente per DNS (salvo poi prevedere un altro utilizzo per `data`, che naturalmente non sarebbe dovuta entrare in `mdns_metadata`).

   In questo caso `data` viene allocata subito e c'è della semplice logica per scorrere i sottocampi del campo dei dati dei record TXT, copiandoli e separandoli con una stringa separatrice. Si noti che il codice supporta una qualsiasi modifica al separatore: per ora è stata usata `", "` ma, se la si reimposta con altro (per esempio `" - "`) funzionerebbe lo stesso. Lo scorrimento del campo è sempre controllato da opportuni check sulla lunghezza che, se falliscono, deallocano `data` e fanno procedere il dissector.

3. **SRV**: anche in questo caso il branch è creato da zero e tutto è stato racchiuso in una guardia che controllasse il protocollo, per gli stessi motivi del branch dedicato ai record TXT.

   Un campo SRV contiene parecchi metadati. Al momento della stesura di una proof of concept era interessante memorizzare il più importante: la porta del servizio. In un futuro, anche gli altri campi numerici (nel progetto saltati) sarebbero potuti essere oggetto di interesse e copiati in una `ndpi_mdns_rsp_entry` opportunamente estesa.

   Comunque, memorizzata la porta e controllata che sia diversa da zero, si procede a ottenere il nome del servizio annunciato in una maniera simile a quella usata per i record PTR.

Si noti che, soprattutto in queste ultime modifiche, è possibile individuare pacchetti malformati: in futuro si potrebbe decidere di attribuire un rischio al flow in questione.

A fine ciclo, quando le informazioni sono state raccolte, se il protocollo è mDNS viene chiamata una nuova funzione, `add_to_mdns_metadata()`. Questa prende tutti i dati raccolti e, dopo un check sui duplicati, li aggiunge a `mdns_metadata`. Nel caso in cui si sia appena trovato il primo servizio annunciato, e dunque `mdns_metadata.services` non è ancora allocato, lo si alloca all'inizio della funzione. Infine si incrementa `mdns_metadata.num_services`.

5 Conclusioni

Le modifiche introdotte alla libreria nDPI dimostrano la possibilità di estendere il dissector DNS per supportare mDNS e raccogliere i metadati relativi ai servizi annunciati. La patch presentata va intesa come un **proof of concept**: la struttura adottata e le soluzioni implementate, come l'uso della bitmap per il controllo dei nomi compressi, sono modulabili e facilmente sostituibili con metodologie alternative, ad esempio un contatore di ricorsioni o altre strutture dati.

Questa flessibilità rende la modifica adattabile a diverse esigenze di progetto e costituisce una base su cui costruire ulteriori estensioni, come l'elaborazione più completa dei record SRV o l'ottimizzazione della gestione dei servizi.