Printing DNS messages

The goal of this assignment is to understand the DNS message compression scheme.

Background:

A DNS message has a DNS header and four sections. The message starts with a DNS header followed by the questions, answers, authority, and additional records sections. The structure of a **DNS header** is as follows.

```
struct dns_header {
 __be16 id;
 __be16 flags;
 __be16 num_ques; // # questions
 __be16 num_ans; // # answer RRs
 __be16 num_auth; // # authority RRs
 __be16 num_add; // # additional RRs
};
```

Here, `num_ques, num_ans, num_auth`, and `num_add` correspond to the number of entries in the questions, answers, authority, and additional records sections. Each entry in the question section begins with a ***name*** followed by a structure shown below.

```
struct dns_q {
 __be16 type;
 __be16 class;
};
```

Each entry in the answers, authority, and additional record sections begins with a `name` followed by a structure shown below. Here, `rdata` is a variable-length byte stream whose length is stored in the field `rlen`.

```
struct dns_rr {
 __be16 type;
 __be16 class;
 __be32 ttl;
 __be16 rlen;
 unsigned char rdata[];
};
```

The names are stored in a compressed format.
Let's first discuss how an uncompressed name is stored. Let's say we want to store www.example.com. In a DNS message, this string is divided into labels, where each label is a substring separated by a dot. In www.example.com there are three labels: www, example, and com.

The labels are stored as follows.

3	w	w	w	6	e	X	A	M	p	l	e	3	c	o	m	0

Each label is preceded by its length. A length of zero indicates the end of the labels. The length of a label is restricted to 63. Therefore, at most six bits are required to store the length of a label; however, 8 bits (1 byte) are used to store the length.

Compression:

During storing a name, if the remaining labels already exist (in the same sequence terminated by a label of zero length) in the DNS message starting at offset X from the start of the DNS message, then instead of storing the remaining labels, X is stored.

Let's consider the following example. We want to store the following names. Let's say the offset of the first name with respect to the start of the message is 0x10.

l.gtld-s.net.
j.gtld-s.net.
gtld-s.net.

The first name is stored as

1	l	6	g	t	l	d	-	S	3	n	e	t	0

The second name is stored as

1	j	htons(0xC012)

Notice that the rest of the labels are already available at offset 0x12. The size of an offset in a DNS message is 16 bits. In order to distinguish between an offset and the length of a label, the top two bits of the offset are set to one (remember that the two bits of the length of a label are always zero because the length can fit in six bits). The offset is stored in the Big endian format. htons converts host order to the network order (which is Big-endian).

The third name is stored as

htons(0xC012)

Notice that the entire name is available at offset 0x12.

In this homework, you need to write a routine, print_helper, which will read labels corresponding to a name, add dots between labels, and store them in a buffer. The routine will return the address of the rest of the record after skipping the bytes corresponding to the name. For example, for the first name, 14 bytes will be skipped to get the address of the rest of the record. Similarly, for the second and third names, four and two bytes will be skipped. The skeleton code provided to you calls the print_name routine, which uses print_helper to read the name in a buffer and to get the address of the rest of the record (say X). print_name returns X to the caller after printing the string.

After getting the address after skipping the name, the skeleton code prints the various fields in the record, except **`rdata`**. After the above print statement, you need to initialize the variable **`start`** with the address of the next record in the corresponding section. Notice that the length of `rdata` is `rlen,` which also needs to be skipped for the records in the answers, authority, and additional records sections.

Skelton code:

Clone the git repository using

git clone https://github.com/Systems-IIITD/CN_DNS

Run make to compile pcap.c.

Run **sudo ./pcap** to run the tool.

The pcap application uses the pcap library to read all packets from the network interface card. It then filters the UDP packets corresponding to the DNS messages. The DNS server uses port 53, which is used to filter DNS messages. You need to implement the `print_helper` routine and the skipping logic as discussed before.

Submission:

Follow the naming format for homework and assignments. You need to submit the pcap.c file along with a design documentation that prints the DNS record printed by your modified pcap application, when you run **dig @198.41.0.4 www.example.com** from another terminal.

A correct output would look something like this:

```
-------------------- START -----------------------

id: 26682 flags: 33024 num_questions: 1 num_answers: 0 num_authority: 13
num_additional: 27

printing question section
www.example.com. type: 1 class: 1

printing answer section
```

```
 printing authority section
com. type: 2 class: 1 ttl: 2a300 len: 14
com. type: 2 class: 1 ttl: 2a300 len: 4
com. type: 2 class: 1 ttl: 2a300 len: 4
com. type: 2 class: 1 ttl: 2a300 len: 4
com. type: 2 class: 1 ttl: 2a300 len: 4
com. type: 2 class: 1 ttl: 2a300 len: 4
com. type: 2 class: 1 ttl: 2a300 len: 4
com. type: 2 class: 1 ttl: 2a300 len: 4
com. type: 2 class: 1 ttl: 2a300 len: 4
com. type: 2 class: 1 ttl: 2a300 len: 4
com. type: 2 class: 1 ttl: 2a300 len: 4
com. type: 2 class: 1 ttl: 2a300 len: 4
com. type: 2 class: 1 ttl: 2a300 len: 4

 printing additional section
 l.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
l.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
j.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
j.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
h.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
h.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
d.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
d.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
b.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
b.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
f.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
f.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
k.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
k.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
m.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
m.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len:
10 i.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
i.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
g.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
g.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
a.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
a.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
c.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
c.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
e.gtld-servers.net. type: 1 class: 1 ttl: 2a300 len: 4
e.gtld-servers.net. type: 1c class: 1 ttl: 2a300 len: 10
type: 29 class: 1000 ttl: 0 len: 0

 ------------------- END ----------------------
```