Programación Funcional

Lucas Di Cunzolo

Febrero 2022

1 Introducción

En este trabajo se plantea la implementación parcial del protocolo DNS propuesto por la RFC 1035 [1]

2 Motivación

La motivación radica principalmente en querer aplicar los conceptos aprendidos en la materia en un proyecto real. La elección del protocolo DNS viene de la intriga de conocer un poco más en detalle la implementación del protocolo.

3 Alcance

El alcance de la implementación está acotado a los tipos de consultas más usados entre los que se encuentran A, $AAAA$, MX, TXT.

Además, nuestro servidor DNS no aplica el uso de TTLs.

4 Modelo de datos

El proyecto se inició modelando un mensaje DNS tal cual se define en la RFC, en el que se pueden distinguir 3 tipos bien definidos, el encabezado del mensaje

```haskell
data DNSHeader = DNSHeader {
  identifier            :: Word16,
  qr                    :: Bool,
  opcode                :: OPCODE,
  authoritativeAnswer   :: Bool,
  truncatedMessage      :: Bool,
  recursionDesired      :: Bool,
  recursionAvailable    :: Bool,
  z                     :: Bool,
  rccode                :: RCODE,
  qdcount               :: Word16,
  ancount               :: Word16,
  nscount               :: Word16,
  arcount               :: Word16
} deriving (Show, Eq)
```

Listing 1: data DNSHeader - *src/FDNS/Types.hs:81*

Dentro del encabezado, se pueden reconocer 2 tipos definidos que modelan el tipo de operación (OPCODE) y el tipo de respuesta (RCODE).

El segundo tipo de dato definido es la definición de una pregunta

```haskell
data DNSQuestion = DNSQuestion {
  qname                 :: String,
  qtype                 :: QTYPE,
  qclass                :: QCLASS
} deriving (Show, Eq)
```

Listing 2: data DNSQuestion - *src/FDNS/Types.hs:100*

En este caso, se cuenta con 2 tipos de datos que representan el tipo de registro DNS (QTYPE) y su clase (QCLASS)

Por último, se modeló el concepto de recurso, que aplica tanto para la sección de respuesta, de autoridad y los adicionales.

```haskell
data DNSResource = DNSResource {
  rname                 :: String,
  rtype                 :: QTYPE,
  rclass                :: QCLASS,
  ttl                   :: Word32,
  rdlength              :: Word16,
  rdata                 :: String
} deriving (Show, Eq)
```

Listing 3: data DNSQuestion - *src/FDNS/Types.hs:106*

Todos estos tipos de datos se combinan en el tipo de dato que modela el mensaje DNS como tal

```haskell
data DNSMessage = DNSMessage {
  header          :: DNSHeader,
  question        :: [DNSQuestion],
  answer          :: [DNSResource],
  authority       :: [DNSResource],
  additional      :: [DNSResource]
} deriving (Show, Eq)
```

Listing 4: data DNSQuestion - *src/FDNS/Types.hs:115*

5 Bibliotecas y módulos usadas

A continuación se van a describir las principales bibliotecas y módulos que se utilizaron en el desarrollo de este proyecto.

5.1 network [2]

Para implementar el servidor UDP se utilizaron los módulos *Network.Socket* y *Network.Socket.ByteString* pertenecientes al paquete *network*

5.2 GetOpt [3]

Para simplificar el parseo de los argumentos que recibe el programa, se utilizó el módulo *System.Console.GetOpt*. Para esto, se definió un tipo *Option* el cual define todos los argumentos, junto a sus tipos de la siguiente forma.

```haskell
data Options = Options {
  optConfig       :: FilePath,
  optBindAddress  :: String,
  optPort         :: String,
  optHelp         :: Bool
} deriving Show
```

Listing 5: data DNSQuestion - *src/FDNS/Commands.hs:14*

Junto a eso, se definió una función que define la lista de opciones, junto a su mensaje de ayuda, la lista de flags largas y cortas, y la función de bind

Referencias

[1] P. Mockapetris. Domain names - implementation and specification, 1987. URL https://datatracker.ietf.org/doc/html/rfc1035.

[2] libraries@haskell.org. network, 2001. URL https://hackage.haskell.org/package/network.

[3] libraries@haskell.org. System.console.getopt, 2002. URL https://hackage.haskell.org/package/base-4.16.0.0/docs/System-Console-GetOpt.html.

[4] Kirill Simonov Michael Snoyman, Anton Ageev. yaml, 2009. URL https://hackage.haskell.org/package/yaml.

[5] snoyberg. co-log, 2018. URL https://hackage.haskell.org/package/co-log.

[6] Simon Hengel. hspec, 2011. URL https://hackage.haskell.org/package/hspec.