In our info hash, four values correspond to unreserved characters if you look at the ASCII chart: `0x4c` is L, `0x54` is T, `0x68` is h and `0x71` is q
So instead of `%4c`, we can use L, similarly instead of `%54`, we can use T etc. to have a shorter string
This way, the URL encoded value for our info hash would be 52 characters long instead of 60: %d6%9f%91%e6%b2%ae**LT**%24**h**%d1%07%3a**q**%d4%ea%13%87%9a%7f
or `%d6%9f%91%e6%b2%aeLT%24h%d1%07%3aq%d4%ea%13%87%9a%7f` with code formatting

19 reply

sarp challenge author 9 months ago

How to parse peers response
In the compact representation of peers response, *peers* field is of type *string*, but its content is *binary data*. You need to treat it as a sequence of bytes, and split it into groups of 6.

For example, the byte sequence `[178, 62, 82, 89, 201, 14, 165, 232, 33, 77, 201, 11]` would correspond to IP:port pairs of
`178.62.82.89:51470`
`165.232.33.77:51467`

In each group, the first 4 bytes correspond to the IP address, where each byte represents a number in the IP address
The last 2 bytes represent the port number, in big-endian order, meaning that we can interpret a group of bytes as an integer by just putting them together left to right.

In our example, 201 = `0xc9` in hexadecimal and 14 = `0x0e` in hexadecimal
When put together left to right, `0xc90e` is `51470` ((16^0) x 14 + (16^1) x 0 + (16 ^2) x 9 + (16^3) x 12)

Programming languages usually have library functions to convert byte arrays into integers, so you don't have to do this math yourself. You can use:

- `binary.BigEndian.Uint16(byteSlice)` in Go
- `int.from_bytes(byte_array, byteorder='big', signed=False)` in Python
- `u16::from_be_bytes(byte_slice)` in Rust

10 reply

ollien 7 months ago