Instructions  Code Examples  Screencasts  Forum  Stage #FI9  Pending

**sarp** challenge author  8 months ago

**Info hash is a sequence of 20 bytes, so how do we represent binary data in the URL for our GET request?**

This is where URL encoding comes into play. The article says:

> *Since the publication of RFC 1738 in 1994 it has been specified that schemes that provide for the representation of binary data in a URI must divide the data into 8-bit bytes and percent-encode each byte. Byte value 0x0F, for example, should be represented by %0F, but byte value 0x41 can be represented by A, or %41. The use of unencoded characters for alphanumeric and other unreserved characters is typically preferred, as it results in shorter URLs.*

Programming languages usually have library functions for URL encoding, so you don't have to do this manually. If you create your request using "net/url" package in Go or "requests" package in Python, this will be done for you under the hood.

It can be useful to understand how it works for debugging or for creating your own implementation.

Let's say the hexadecimal representation of our info hash is `d69f91e6b2ae4c542468d1073a71d4ea13879a7f`
This 40 character long string was representing 20 bytes, so each character pair corresponds to a byte
We can just put a % before each byte so the URL-encoded representation would be: `%d6%9f%91%e6%b2%ae%4c%54%24%68%d1%07%3a%71%d4%ea%13%87%9a%7f`
The result is 60 characters long. This works, but there's a more efficient way of doing it

Remember that each character has a corresponding hex value in ASCII encoding.
URL encoding specifies a set of unreserved characters: (0-9, A-Z, a-z, hypen, underscore, full stop, tilda) that have no special meaning in URLs
If our hex value corresponds to an unreserved character, we don't need to use url encoding for it
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 ins
60: %d6%9f%91%e6%b2%ae**LT**%24**h**%d1%07%3a**q**%d4%ea%13%87%9a%7f

Ready to run tests... Show logs