What's On The Wire?

To see why this is inefficient and rude, let's turn on the debugging features of Python's HTTP library and see what's being sent "on the wire" (i.e. over the network).

```
from http.client import HTTPConnection
HTTPConnection.debuglevel = 1                                          #①
from urllib.request import urlopen
response = urlopen('http://diveintopython3.org/examples/feed.xml')  #②
#send: b'GET /examples/feed.xml HTTP/1.1                              #③
#Host: diveintopython3.org                                            #④
#Accept-Encoding: identity                                            #⑤
#User-Agent: Python-urllib/3.1'                                       #⑥
#Connection: close
#reply: 'HTTP/1.1 200 OK'
#…further debugging information omitted…
```

① As I mentioned at the beginning of the chapter, `urllib.request` relies on another standard Python library, `http.client`. Normally you don't need to touch `http.client` directly. (The `urllib.request` module imports it automatically.) But we import it here so we can toggle the debugging flag on the `HTTPConnection` class that `urllib.request` uses to connect to the HTTP server.

② Now that the debugging flag is set, information on the HTTP request and response is printed out in real time. As you can see, when you request the Atom feed, the `urllib.request` module sends five lines to the server.

③ The first line specifies the HTTP verb you're using, and the path of the resource (minus the domain name).

④ The second line specifies the domain name from which we're requesting this feed.

⑤ The third line specifies the compression algorithms that the client supports. As I mentioned earlier, `urllib.request` does not support compression by default.

⑥ The fourth line specifies the name of the library that is making the request. By default, this is `Python-urllib` plus a version number. Both `urllib.request` and `httplib2` support changing the user agent, simply by adding a `User-Agent` header to the request (which will override the default value).

We're downloading 3070 bytes when we could have just downloaded 941.

Now let's look at what the server sent back in its response.

```
# continued from previous example
from http.client import HTTPConnection
HTTPConnection.debuglevel = 1
from urllib.request import urlopen
response = urlopen('http://diveintopython3.org/examples/feed.xml')

print(response.headers.as_string())             #①
#Date: Sun, 31 May 2009 19:23:06 GMT            #②
#Server: Apache
#Last-Modified: Sun, 31 May 2009 06:39:55 GMT   #③
#ETag: "bfe-93d9c4c0"                           #④
#Accept-Ranges: bytes
#Content-Length: 3070                           #⑤
#Cache-Control: max-age=86400                   #⑥
#Expires: Mon, 01 Jun 2009 19:23:06 GMT
#Vary: Accept-Encoding
#Connection: close
#Content-Type: application/xml

data = response.read()                      #⑦
print (len(data))
#303
```

① The `response` returned from the `urllib.request.urlopen()` function contains all the HTTP headers the server sent back. It also contains methods to download the actual data; we'll get to that in a minute.

② The server tells you when it handled your request.

③ This response includes a `Last-Modified` header.

④ This response includes an `ETag` header.

⑤ The data is 3070 bytes long. Notice what *isn't* here: a `Content-encoding` header. Your request stated that you only accept uncompressed data (`Accept-encoding: identity`), and sure enough, this response contains uncompressed data.

⑥ This response includes caching headers that state that this feed can be cached for up to 24 hours (86400 seconds).

⑦ And finally, download the actual data by calling `response.read()`. As you can tell from the `len()` function, this fetched a total of 3070 bytes.

As you can see, this code is already inefficient: it asked for (and received) uncompressed data. I know for a fact that this server supports gzip compression, but HTTP compression is opt-in. We didn't ask for it, so we didn't get it. That means we're fetching 3070 bytes when we could have fetched 941. Bad dog, no biscuit.

But wait, it gets worse! To see just how inefficient this code is, let's request the same feed a second time.

```
# continued from the previous example
from http.client import HTTPConnection
HTTPConnection.debuglevel = 1
from urllib.request import urlopen

response2 = urlopen('http://diveintopython3.org/examples/feed.xml')
#send: b'GET /examples/feed.xml HTTP/1.1
#Host: diveintopython3.org
#Accept-Encoding: identity
#User-Agent: Python-urllib/3.1'
#Connection: close
#reply: 'HTTP/1.1 200 OK'
#…further debugging information omitted…
```

Notice anything peculiar about this request? It hasn't changed! It's exactly the same as the first request. No sign of If-Modified-Since headers. No sign of If-None-Match headers. No respect for the caching headers. Still no compression.

And what happens when you do the same thing twice? You get the same response. Twice.

```
# continued from the previous example
from http.client import HTTPConnection
HTTPConnection.debuglevel = 1
from urllib.request import urlopen
response2 = urlopen('http://diveintopython3.org/examples/feed.xml')
print(response2.headers.as_string())     #①
#Date: Mon, 01 Jun 2009 03:58:00 GMT
#Server: Apache
#Last-Modified: Sun, 31 May 2009 22:51:11 GMT
#ETag: "bfe-255ef5c0"
#Accept-Ranges: bytes
#Content-Length: 3070
#Cache-Control: max-age=86400
#Expires: Tue, 02 Jun 2009 03:58:00 GMT
#Vary: Accept-Encoding
#Connection: close
#Content-Type: application/xml

data2 = response2.read()
print (len(data2)   )                            #②
3070

print (data2 == data  )                          #③
#True
```

① The server is still sending the same array of “smart” headers: `Cache-Control` and `Expires` to allow caching, Last-Modified and ETag to enable “not-modified” tracking. Even the `Vary: Accept-Encoding` header hints that the server would support compression, if only you would ask for it. But you didn’t.

② Once again, this request fetches the whole 3070 bytes…

③ …the exact same 3070 bytes you got last time.

HTTP is designed to work better than this. `urllib` speaks HTTP like I speak Spanish — enough to get by in a jam, but not enough to hold a conversation. HTTP is a conversation. It’s time to upgrade to a library that speaks HTTP fluently.