

Discussion

The application is designed to be a simplified implementation of the *curl* command line utility, which issues a request to a given url and prints the HTML response from the webpage to the terminal on successful retrieval, or one of an assortment of error messages on unsuccessful retrieval. The file is run with either of the following formats:

- (1) `python tjinovakMyCurl.py [url]`
- (2) `python tjinovakMyCurl.py [url] [hostname]`

In (1), the user must provide a full url containing the server's hostname, such as <http://www.google.com>. In (2), the url argument is an IP address, in which the hostname must also be provided as a second argument for proper behavior. An example of this would be using the url: <http://93.184.216.34>, and the hostname: www.example.com. On successful retrieval, the program will write the HTML content received from the web server to a file, HTTPoutput.html, and append an entry with information about the connection to another file, Log.csv. On unsuccessful retrieval, the program does not write to HTTPoutput.html, but still appends an entry to Log.csv with connection information. Entries in Log.csv have the following format:

Successful or Unsuccessful, Server Status Code, Requested URL, hostname, source IP, destination IP, source port, destination port, Server Response line (including code and phrase)

As noted at the beginning of the document, the program is not a perfect replica of *curl*'s capabilities. Our program only operates with the HTTP protocol, and cannot handle HTTPS connections to web servers. It also requires that urls begin with the *http://* prefix, designating that HTTP connection should be used, and aborts execution prematurely if this prefix is not present. The program also does not work with chunk-encoded webpages, such as <http://www.google.com>, and will print an error message to the terminal specifying that chunk-encoding is not supported. The *curl* command also does not abide by our restrictions on command line argument input, and can work with a single IP address argument without supplying a hostname. This is because *curl* can issue a DNS query to find the hostname if omitted, which our program does not support.

As part of the project deliverables, we were tasked with testing two successful downloads, two unsuccessful downloads, one redirect, and a request to port 443. These tests are provided in the submitted Log.csv file with the appropriate connection data.

Successful Downloads:

(1) <http://www.example.com>

1	0.00000000	10.0.2.15	75.75.75.75	DNS	77	Standard query 0x1813 A www.example.com
2	0.01946500	75.75.75.75	10.0.2.15	DNS	93	Standard query response 0x1813 A 93.184.216.34
3	0.01959800	10.0.2.15	93.184.216.34	TCP	76	39671 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4294914195 TSecr=0 WS=128
4	0.03598200	93.184.216.34	10.0.2.15	TCP	62	http > 39671 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
5	0.03593100	10.0.2.15	93.184.216.34	TCP	56	39671 > http [ACK] Seq=1 Ack=1 Win=29200 Len=0
6	0.03664300	10.0.2.15	93.184.216.34	HTTP	96	GET / HTTP/1.1
7	0.03689300	93.184.216.34	10.0.2.15	TCP	62	http > 39671 [ACK] Seq=1 Ack=1 Win=65535 Len=0
8	0.05267000	93.184.216.34	10.0.2.15	HTTP	1647	HTTP/1.1 200 OK (text/html)
9	0.05268900	10.0.2.15	93.184.216.34	TCP	56	39671 > http [ACK] Seq=41 Ack=1592 Win=32120 Len=0
10	0.05345600	10.0.2.15	93.184.216.34	TCP	56	39671 > http [FIN, ACK] Seq=41 Ack=1592 Win=32120 Len=0
11	0.05365300	93.184.216.34	10.0.2.15	TCP	62	http > 39671 [ACK] Seq=1592 Ack=42 Win=65535 Len=0
12	0.07372800	93.184.216.34	10.0.2.15	TCP	62	http > 39671 [FIN, ACK] Seq=1592 Ack=42 Win=65535 Len=0
13	0.07376700	10.0.2.15	93.184.216.34	TCP	56	39671 > http [ACK] Seq=42 Ack=1593 Win=32120 Len=0

6 0.036643000 10.0.2.15 93.184.216.34 HTTP 96 GET / HTTP/1.1

Frame 6: 96 bytes on wire (768 bits), 96 bytes captured (768 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 93.184.216.34 (93.184.216.34)
Transmission Control Protocol, Src Port: 39671 (39671), Dst Port: http (80), Seq: 1, Ack: 1
Hypertext Transfer Protocol
GET / HTTP/1.1
Host: www.example.com
Full request URI: http://www.example.com/
[HTTP request 1/1]
[Response in frame: 8]

8 0.052670000 93.184.216.34 10.0.2.15 HTTP 1647 HTTP/1.1 200 OK (text/html)

Frame 8: 1647 bytes on wire (13176 bits), 1647 bytes captured (13176 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 93.184.216.34 (93.184.216.34), Dst: 10.0.2.15 (10.0.2.15)
Transmission Control Protocol, Src Port: http (80), Dst Port: 39671 (39671), Seq: 1, Ack: 41
Hypertext Transfer Protocol
HTTP/1.1 200 OK
Age: 383015
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Tue, 01 Mar 2022 21:30:27 GMT
Etag: "3147526947-ident"
Expires: Tue, 08 Mar 2022 21:30:27 GMT
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Server: ECS (sab/5693)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 1256
[HTTP response 1/1]
[Time since request: 0.016027000 seconds]
[Request in frame: 6]
Line-based text data: text/html

0000 00 04 00 01 00 06 08 00 27 27 c6 3a 00 00 08 00T.....
0010 45 00 00 50 93 96 40 00 40 06 65 28 0a 00 02 0f E..P..@.e(....
0020 5d b8 d8 22 9a f7 00 50 f5 2d a3 08 00 02 ee 02]..*..P.....
0030 50 18 72 10 42 2c 00 00 47 45 54 20 2f 20 48 54 P.r.B... GET / HT

Khufu — -bash — 78x5
date(base) Theos-MacBook-Pro-4:~ Khufu\$ date
Tue Mar 1 13:32:42 PST 2022

This is the first of our successful downloads, whose output matches that of *curl*.

(2) <http://www.pudim.com.br.com>

Wireshark packet capture showing a successful HTTP GET request to <http://www.pudim.com.br.com>. The packet list shows a GET request in frame 6 and a 200 OK response in frame 8. The packet details for frame 8 show the full HTTP response, including headers like Date, Server, Last-Modified, ETag, and Content-Type. The packet bytes show the raw data of the response.

Frame 6: 93 bytes on wire (744 bits), 93 bytes captured (744 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 54.207.20.104 (54.207.20.104)
Transmission Control Protocol, Src Port: 46659 (46659), Dst Port: http (80), Seq: 1, Ack: :
Hypertext Transfer Protocol
GET / HTTP/1.1
Host: pudim.com.br
[Full request URI: http://pudim.com.br/]
[HTTP request 1/1]
[Response in frame: 8]

Frame 8: 1182 bytes on wire (9456 bits), 1182 bytes captured (9456 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 54.207.20.104 (54.207.20.104), Dst: 10.0.2.15 (10.0.2.15)
Transmission Control Protocol, Src Port: http (80), Dst Port: 46659 (46659), Seq: 1, Ack: :
Hypertext Transfer Protocol
HTTP/1.1 200 OK
Date: Tue, 01 Mar 2022 21:39:07 GMT
Server: Apache/2.4.34 (Amazon) OpenSSL/1.0.2k-fips PHP/5.5.38
Last-Modified: Wed, 23 Dec 2015 01:18:20 GMT
ETag: "353-527867f65e8ad"
Accept-Ranges: bytes
Content-Length: 851
Content-Type: text/html; charset=UTF-8
[HTTP response 1/1]
[Time since request: 0.191155000 seconds]
[Request in frame: 6]
Line-based text data: text/html

Khufu — -bash — 78x5
date(base) Theos-MacBook-Pro-4:~ Khufu\$ date
Tue Mar 1 13:32:42 PST 2022

This is the second of our successful downloads, whose output matches that of `curl`.

Unsuccessful Downloads:

(1) <http://www.example.com>

1	0.00000000	10.0.2.15	75.75.75.75	DNS	76	Standard query 0xc822 A www.example.com
2	0.04917900	CadmusCo 27:c6:3a		ARP	44	Who has 10.0.2.2? Tell 10.0.2.15
3	0.04974700	RealtekU 12:35:02		ARP	62	10.0.2.2 is at 52:54:00:12:35:02
4	0.05055800	75.75.75.75	10.0.2.15	DNS	92	Standard query response 0xc822 A 104.247.82.13
5	0.05067800	10.0.2.15	104.247.82.13	TCP	76	54470 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=148590 TSecr=0 WS=128
6	0.13146200	104.247.82.13	10.0.2.15	TCP	62	http > 54470 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
7	0.13148700	10.0.2.15	104.247.82.13	TCP	56	54470 > http [ACK] Seq=1 Ack=1 Win=29200 Len=0
8	0.13179800	10.0.2.15	104.247.82.13	HTTP	95	GET / HTTP/1.1
9	0.13192300	104.247.82.13	10.0.2.15	TCP	62	http > 54470 [ACK] Seq=1 Ack=40 Win=65535 Len=0
10	0.29732900	104.247.82.13	10.0.2.15	HTTP	350	HTTP/1.1 403 Forbidden (text/html)
11	0.29735300	10.0.2.15	104.247.82.13	TCP	56	54470 > http [ACK] Seq=40 Ack=295 Win=30016 Len=0
12	0.29778500	10.0.2.15	104.247.82.13	TCP	56	54470 > http [FIN, ACK] Seq=40 Ack=295 Win=30016 Len=0
13	0.29803700	104.247.82.13	10.0.2.15	TCP	62	http > 54470 [ACK] Seq=295 Ack=41 Win=65535 Len=0
14	0.38641700	104.247.82.13	10.0.2.15	TCP	62	http > 54470 [FIN, ACK] Seq=295 Ack=41 Win=65535 Len=0
15	0.38646900	10.0.2.15	104.247.82.13	TCP	56	54470 > http [ACK] Seq=41 Ack=296 Win=30016 Len=0
16	1.02713200	10.0.2.15	91.189.89.198	NTP	92	NTP Version 4, client
17	1.17554200	91.189.89.198	10.0.2.15	NTP	92	NTP Version 4, server

8 0.131798000 10.0.2.15 104.247.82.13 HTTP 95 GET / HTTP/1.1

Frame 8: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 104.247.82.13 (104.247.82.13)
Transmission Control Protocol, Src Port: 54470 (54470), Dst Port: http (80), Seq: 1, Ack: 1
Hypertext Transfer Protocol
GET / HTTP/1.1\r\n
Host: www.example.com\r\n\r\n
Full request URI: http://www.example.com/
[HTTP request 1/1]
[Response in frame: 10]

10 0.297329000 104.247.82.13 10.0.2.15 HTTP 350 HTTP/1.1 403 Forbidden (text/html)

Frame 10: 350 bytes on wire (2800 bits), 350 bytes captured (2800 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 104.247.82.13 (104.247.82.13), Dst: 10.0.2.15 (10.0.2.15)
Transmission Control Protocol, Src Port: http (80), Dst Port: 54470 (54470), Seq: 1, Ack: 4
Hypertext Transfer Protocol
HTTP/1.1 403 Forbidden\r\n
Server: nginx\r\nDate: Tue, 01 Mar 2022 21:39:25 GMT\r\nContent-Type: text/html\r\nContent-Length: 146\r\nConnection: keep-alive\r\n\r\n[HTTP response 1/1]
[Time since request: 0.165531000 seconds]
[Request in frame: 8]
Line-based text data: text/html

0000 00 04 00 01 00 06 08 00 27 27 c6 3a 00 00 08 00:.....
0010 45 00 00 4f 21 8a 40 00 40 06 52 0c 0a 00 02 0f E..0f.@.R....
0020 68 f7 52 0d d4 c6 00 50 0d 44 65 e7 00 22 2e 02 h.R....P..De....
0030 50 18 72 10 c7 54 00 00 47 45 54 20 2f 20 48 54 P...T...GET / HT

0000 00 00 00 01 00 06 52 54 00 12 35 02 00 00 08 00RT..5....
0010 45 00 01 4e 01 23 00 00 40 06 b1 74 68 f7 52 0d E..N.#..@..th.R.
0020 0a 00 02 0f 00 50 d4 c6 00 22 2e 02 0d 44 66 0eP...".Df..
0030 50 18 ff ff 76 af 00 00 48 54 54 50 2f 31 2e 31 P...V...HTTP/1.1

Khufu — -bash — 78x5

date(base) Theos-MacBook-Pro-4:~ Khufu\$ date
Tue Mar 1 13:32:42 PST 2022

ie: Default

This is the first of our unsuccessful downloads, whose output matches that of *curl*. Both receive a 403 Forbidden status code, meaning that both programs do not have access to the given website.

(2) <http://93.184.216.34/foo.html> www.example.com

```
1 0.000000000 10.0.2.15 93.184.216.34 TCP 76 39678 > http [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=171802 TSecr=0 WS=128
2 0.017892000 93.184.216.34 10.0.2.15 TCP 62 http > 39678 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
3 0.017949000 10.0.2.15 93.184.216.34 TCP 56 39678 > http [ACK] Seq=1 Ack=1 Win=29200 Len=0
4 0.018484000 10.0.2.15 93.184.216.34 HTTP 104 GET /foo.html HTTP/1.1
5 0.018870000 93.184.216.34 10.0.2.15 TCP 62 http > 39678 [ACK] Seq=1 Ack=49 Win=65535 Len=0
6 0.038494000 93.184.216.34 10.0.2.15 HTTP 1654 HTTP/1.1 404 Not Found (text/html)
7 0.038516000 10.0.2.15 93.184.216.34 TCP 56 39678 > http [ACK] Seq=49 Ack=1599 Win=32120 Len=0
8 0.039095000 10.0.2.15 93.184.216.34 TCP 56 39678 > http [FIN, ACK] Seq=49 Ack=1599 Win=32120 Len=0
9 0.039418000 93.184.216.34 10.0.2.15 TCP 62 http > 39678 [ACK] Seq=1599 Ack=50 Win=65535 Len=0
10 0.059263000 93.184.216.34 10.0.2.15 TCP 62 http > 39678 [FIN, ACK] Seq=1599 Ack=50 Win=65535 Len=0
11 0.059292000 10.0.2.15 93.184.216.34 TCP 56 39678 > http [ACK] Seq=50 Ack=1600 Win=32120 Len=0

4 0.018484000 10.0.2.15 93.184.216.34 HTTP 104 GET /foo.html HTTP/1.1
Frame 4: 104 bytes on wire (832 bits), 104 bytes captured (832 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 93.184.216.34 (93.184.216.34)
Transmission Control Protocol, Src Port: 39678 (39678), Dst Port: http (80), Seq: 1, Ack: 1
Hypertext Transfer Protocol
GET /foo.html HTTP/1.1\r\n
Host: www.example.com\r\n
\r\n
[Full request URI: http://www.example.com/foo.html]
[HTTP request 1/1]
[Response in frame: 6]

6 0.038494000 93.184.216.34 10.0.2.15 HTTP 1654 HTTP/1.1 404 Not Found (text/html)
Frame 6: 1654 bytes on wire (13232 bits), 1654 bytes captured (13232 bits) on interface 0
Linux cooked capture
Internet Protocol Version 4, Src: 93.184.216.34 (93.184.216.34), Dst: 10.0.2.15 (10.0.2.15)
Transmission Control Protocol, Src Port: http (80), Dst Port: 39678 (39678), Seq: 1, Ack: 49,
Hypertext Transfer Protocol
HTTP/1.1 404 Not Found\r\n
Accept-Ranges: bytes\r\n
Age: 107582\r\n
Cache-Control: max-age=604800\r\n
Content-Type: text/html; charset=UTF-8\r\n
Date: Tue, 01 Mar 2022 21:45:26 GMT\r\n
Expires: Tue, 08 Mar 2022 21:45:26 GMT\r\n
Last-Modified: Mon, 28 Feb 2022 15:52:24 GMT\r\n
Server: ECS (sab/5697)\r\n
Vary: Accept-Encoding\r\n
X-Cache: 404-HIT\r\n
Content-Length: 1256\r\n
\r\n
[HTTP response 1/1]
[Time since request: 0.020010000 seconds]
[Request in frame: 4]
Line-based text data: text/html

0000 00 04 00 01 00 06 08 00 27 27 c6 3a 00 00 08 00 .....X..@.N.....
0010 45 00 00 58 aa 22 40 00 40 06 4e 94 0a 00 02 0f E..X..@.N.....
0020 5d b8 d8 22 9a fe 00 50 d4 63 35 3b 00 24 22 02 ]...P.c5;$".
0030 50 18 72 10 42 34 00 00 47 45 54 20 2f 66 6f 6f P.r.B4..GET /foo
```

This is the second of our unsuccessful downloads, whose output does not match that of *curl*. The retrieval is unsuccessful because the page *foo.html* does not exist on that site, which responds with 404 Not Found, but also replies with the base HTML page www.example.com. The difference in outputs between our program and *curl* occurs because we do not write to HTTPoutput.html if the status code is not 200, indicating a successful download, while *curl* will instead return the base HTML page of www.example.com despite also receiving a 404 Not Found status line.

Redirect:

(1) <http://www.yahoo.com>

The image shows a Wireshark packet capture of an HTTP GET request to <http://www.yahoo.com>. The packet list shows a GET request from 10.0.2.15 to 98.137.11.163 on port 43139. The packet details pane shows the request structure, including the Host, User-Agent, and Accept headers. The packet bytes pane shows the raw data. Below the Wireshark window, a terminal window shows the command `date` being executed, with the output `Tue Mar 1 13:32:42 PST 2022`.

This is our redirect url test, whose output does not match that of *curl*. Our program will reply to the terminal with: “Unsuccessful: www.yahoo.com, 301”, as specified in the project document. *Curl* instead responds with “redirect%”. The essence of both of the responses is the same, however, as status code 301 indicates that the site has been moved permanently and is used for permanent redirecting.

Request to port 443:

(1) <http://www.example.com:443>

The image shows a Wireshark packet capture of an HTTPS request to <http://www.example.com:443>. The packet list shows a GET request from 10.0.2.15 to 93.184.216.34 on port 443. The packet details pane shows the request structure, including the Host, User-Agent, and Accept headers. The packet bytes pane shows the raw data. Below the Wireshark window, a terminal window shows the command `date` being executed, with the output `Tue Mar 1 13:32:42 PST 2022`.

This is our test using port 443 in a url, whose output closely resembles that of *curl*. Our program responds to the exception by printing: “Recv failure: connection reset by peer”, while *curl* responds with: “curl: (56) Recv failure: connection reset by peer”. This is because port 443 is explicitly for HTTPS connections, and our HTTP connection is quickly rebuffed as it is incompatible.

