curl user poll 2016

"make SSL less of a nightmare"

summary and analysis by Daniel Stenberg

About curl

Curl is an open source project that produces the curl tool and the libcurl library. We are a small project, with few core contributors, with little commercial backing and yet we're over 18 years old and we have a total of over 1400 named contributors through the years. Our products run in a vast amount of internet-connected devices, tools and services on the current Internet.

See curl.haxx.se for everything not answered in this summary.

Background

We do this user survey annually in an attempt to catch trends and longer running changes in the project, its users and in curl fits into the wider ecosystem. As usual, we only reach and get responses from a small subset of users who voluntarily decide to fill in the questionnaire and the vast majority of users and curl developers never get to hear about it and never get an opportunity to respond.

This should make us ask ourselves: is this what our users think, or is it just the opinions of the subset of users that we happened to reach. We simply have to work with what we have.

This year, the poll was up 11 days from May 16 to and including May 27th. (2015: May 6 – 16)

The poll was announced on the curl mailing lists (the libcurl list, the curl users list and the announce list), on Daniel's blog (https://daniel.haxx.se/blog) and a few times on twitter by Daniel (@bagder).

Responses

We discovered a disappointingly low level of interest and we have no way to figure out why this is so. With 176 responses it is a new low:

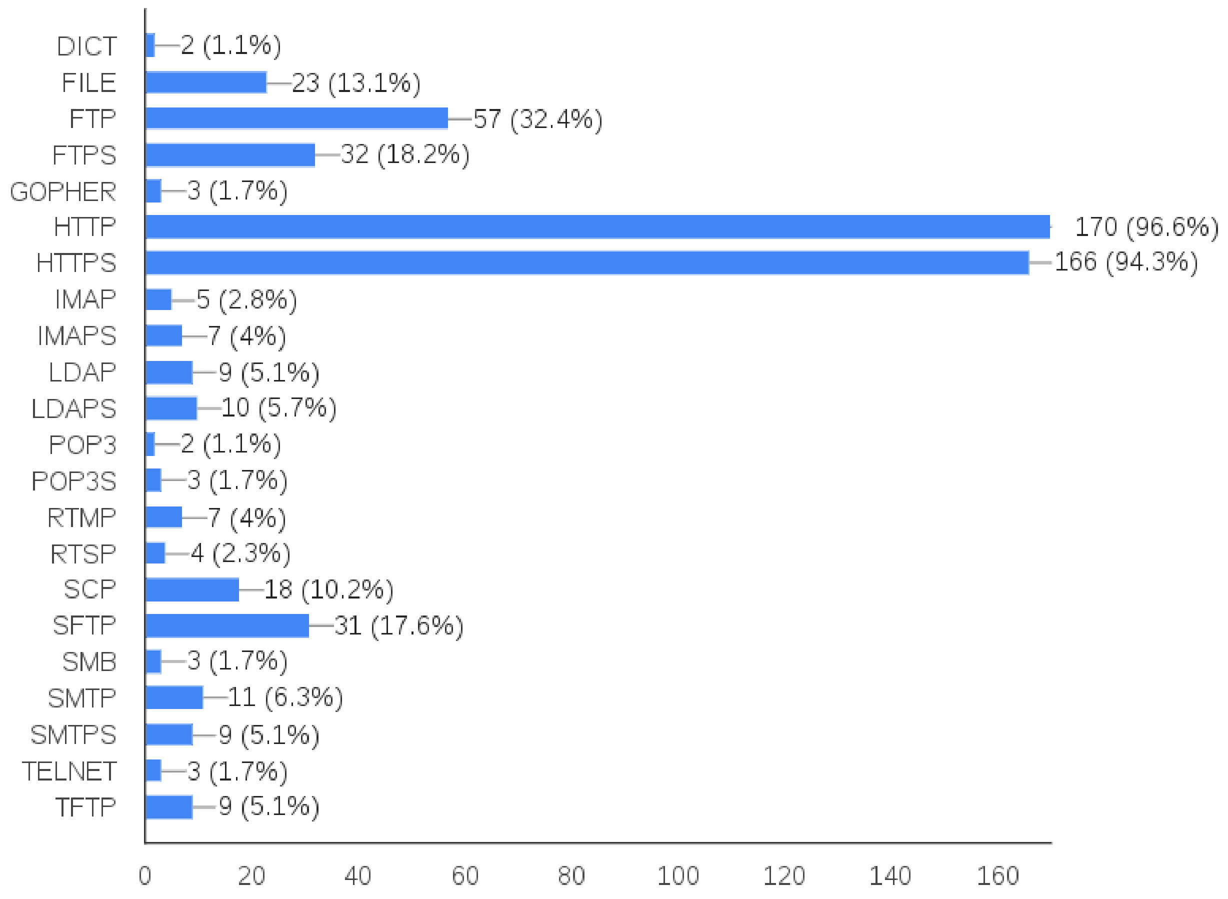
Year	**Number of responses**
2014	193
2015	1475
2016	176

Now, let's see what people answered…

Please check the protocols you use curl/libcurl for

n=176

The question that always tickles everyone's imagination since we always have a few users claiming to use even the most ancient protocols. The graph for 2016 looks like this:



Which roughly matches what we've seen other years.

On the right here you'll see the distribution over the years. The rightmost column shows how large the change has been, in percentage (growing from 9% to 13.1% equals a 45.5% growth).

The top-5 most popular protocols remain the same year after year:

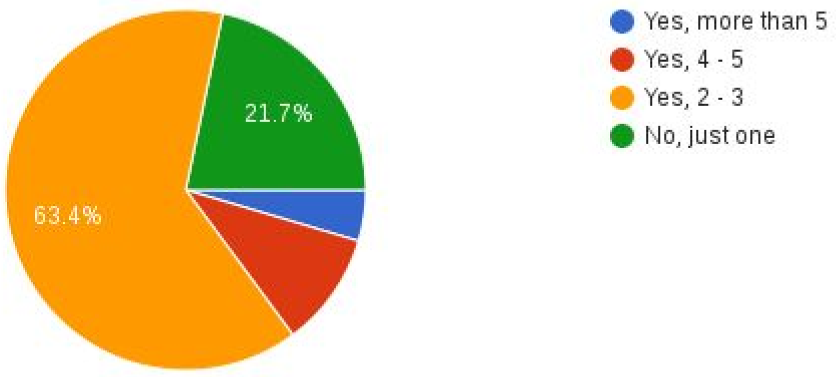
HTTP, HTTPS, FTP, FTPS and SFTP. In that order.

Protocol	2014	2015	2016	14 to 16
DICT	1.5%	0.7%	1.1%	-26.67%
FILE	9.0%	5.4%	13.1%	45.56%
FTP	43.0%	30.2%	32.4%	-24.65%
FTPS	18.0%	9.4%	18.2%	1.11%
GOPHER	1.5%	1.5%	1.7%	13.33%
HTTP	91.0%	98.1%	96.6%	6.15%
HTTPS	90.0%	94.3%	94.3%	4.78%
IMAP	5.0%	1.6%	2.8%	-44.00%
IMAPS	4.0%	1.8%	4.0%	0.00%
LDAP	4.0%	1.8%	5.1%	27.50%
LDAPS	3.6%	1.5%	5.7%	58.33%
POP3	3.0%	1.4%	1.1%	-63.33%
POP3S	2.0%	1.0%	1.7%	-15.00%
RTMP	2.5%	1.3%	4.0%	60.00%
RTSP	1.5%	0.9%	2.3%	53.33%
SCP	9.0%	5.7%	10.2%	13.33%
SFTP	18.0%	8.6%	17.6%	-2.22%
SMB			1.7%	n/a
SMTP	10.0%	2.2%	6.3%	-37.00%
SMTPS	7.0%	1.6%	5.1%	-27.14%
TELNET	3.6%	2.5%	1.7%	-52.78%
TFTP	4.0%	3.0%	5.1%	27.50%

The changes among the protocols were all so minor in comparison to the popular protocols to draw any conclusions from. We keep getting all supported protocols marked, which is a fascinating fact in itself. There's no particular sign of any single protocol just having died, even if most protocols apart from the top six are below ten percent usage.

Do you use curl/libcurl on multiple platforms?

n=175



This distribution is virtually the same as it has been in the past. We could possibly deduce that the amount of users using more than 5 platforms is shrinking. This question is a bit vague too so it may very well be so that what is a "platform" isn't universably agreed upon. Still, curl's ability to run (the same) on many different systems remains a key feature.

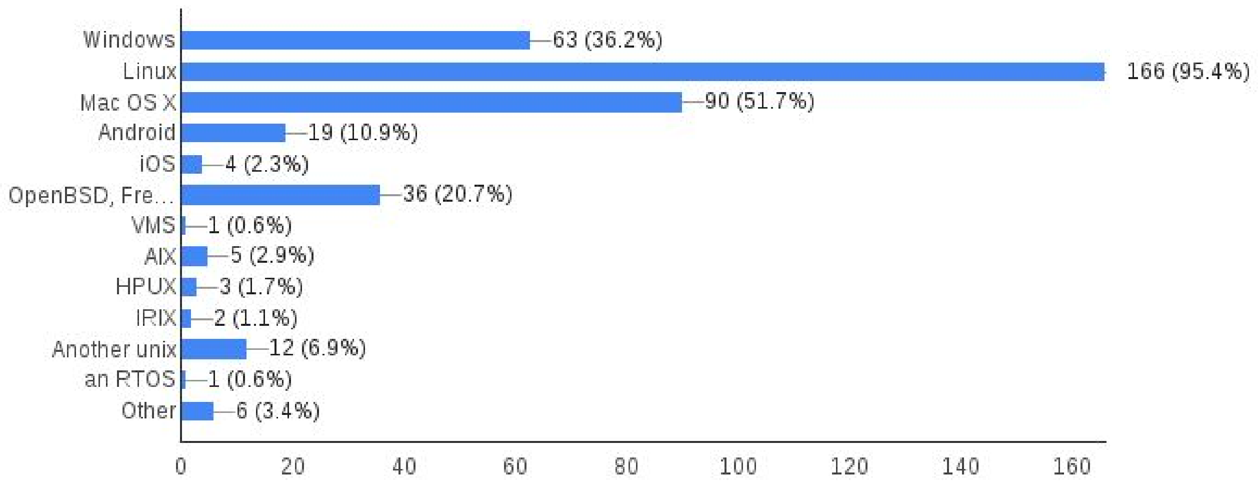
	2-3	**one**	**4-5**	**>5**
2016	63.4	21.7	10.3	4.6
2015	65	19.4	8.5	7.5
2014	58	19	12	11

You use curl/libcurl on which platforms?

n=174

Now here's a new question for the year. As the previous question suggests, most users run curl on multiple platforms and this questions answers which those platforms are.

Linux, Mac OS, Windows, BSD flavors and Android all land on over ten percent. Interestingly all offered systems were marked at least once, and the write ins for "Others" included Hurd, OpenIndiana, Solaris and Ubuntu.



Do you typically build curl/libcurl yourself?

n = 172

I ask this question partly to see how large a share of our users that actually do get confronted by our build infrastructure, but also to get an image of the humans that have responded to this question.

32.6% answered YES to this, so basically a third of the respondents build curl from source.

(This question was not included in the survey last year.)

Tell us which libcurl features you use?

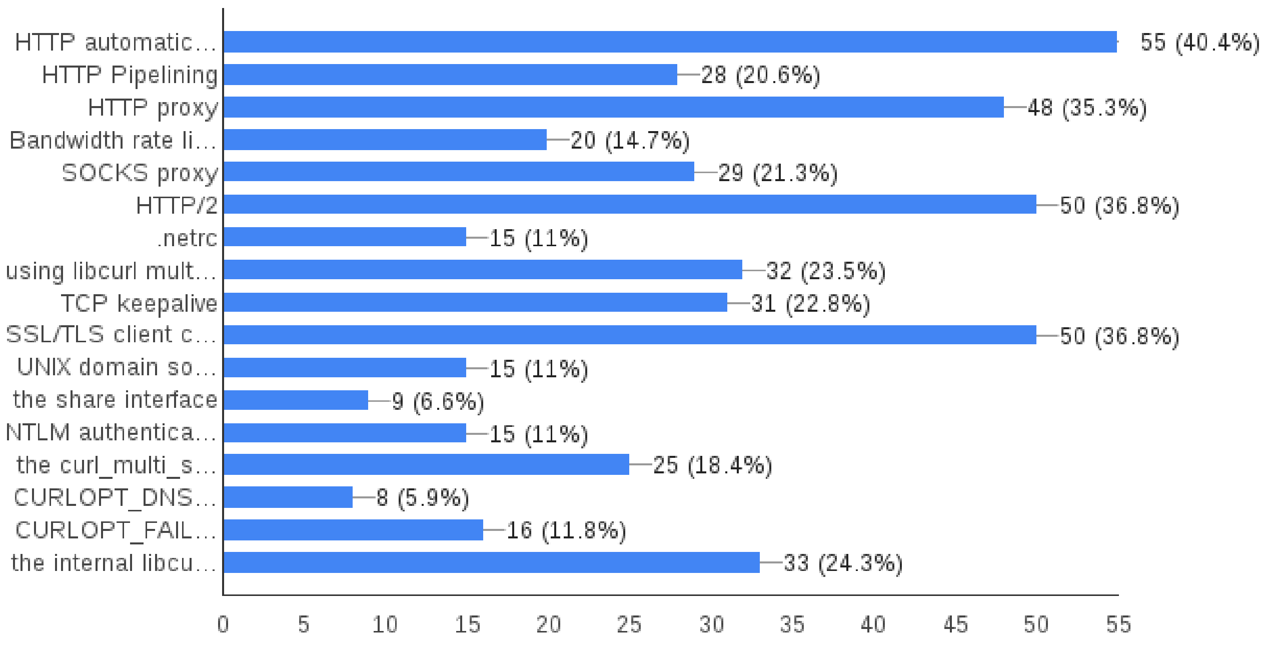
n = 136

Like many other questions, the responses to this could be use as an indication of what we could drop in the future or where we should put more emphasis. But as you can see here, just about all features we asked about have a fair usage share. Only two were below ten percent, but they were all above five.

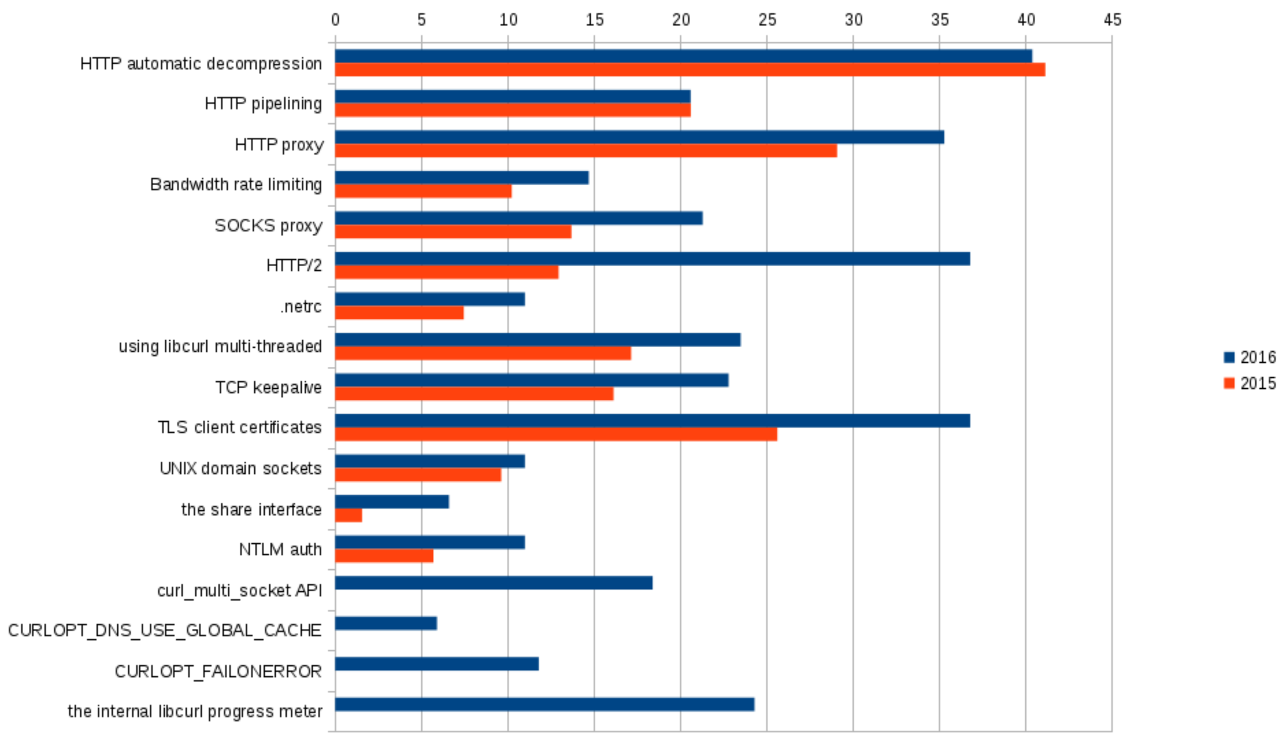
The very large numbers for HTTP Pipelining, with me at the same time knowing and acknowledging that we have plenty of problems in the pipelining code and yet we get very few bug reports about it, makes me suspect that people are checking this option either without actually using the feature or because they confuse it with HTTP persistent connections. Something I’ve seen done on numerous occasions in user communications. The number for this feature were almost identical last year.

In general most features went up a little bit, with HTTP/2 seeing an almost tripled usage amount. Seeing how HTTP/2 usage is growing on the general web that seems quite plausible. The other features that grew with more than 5 percentage points (ie statistically significantly interesting) were HTTP proxy, SOCKS proxy, multi-threading use, TCP keepalive and client certificates.

The 2016 results:



The 2016 numbers laid out next to the 2015 numbers:



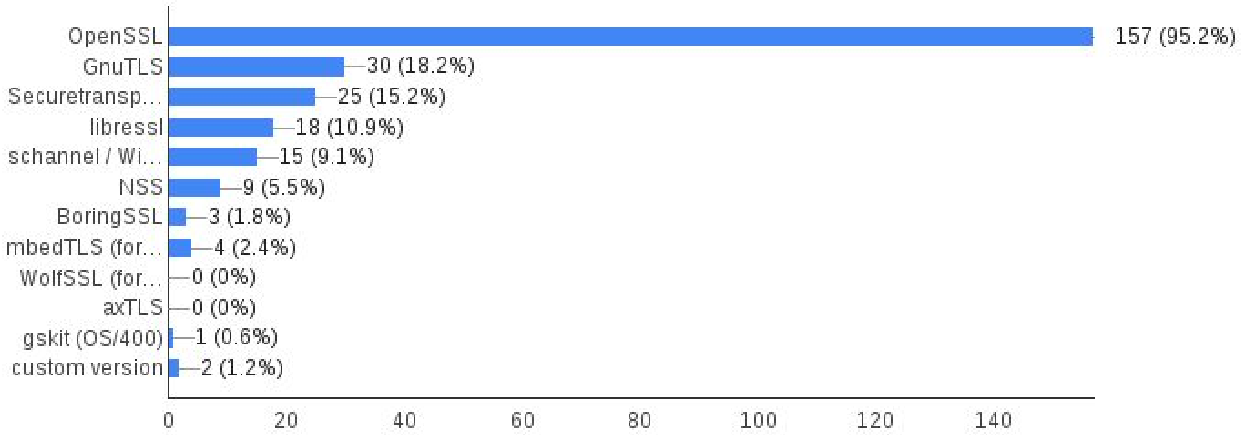
Which SSL backend(s) do you typically use the most?

n = 165

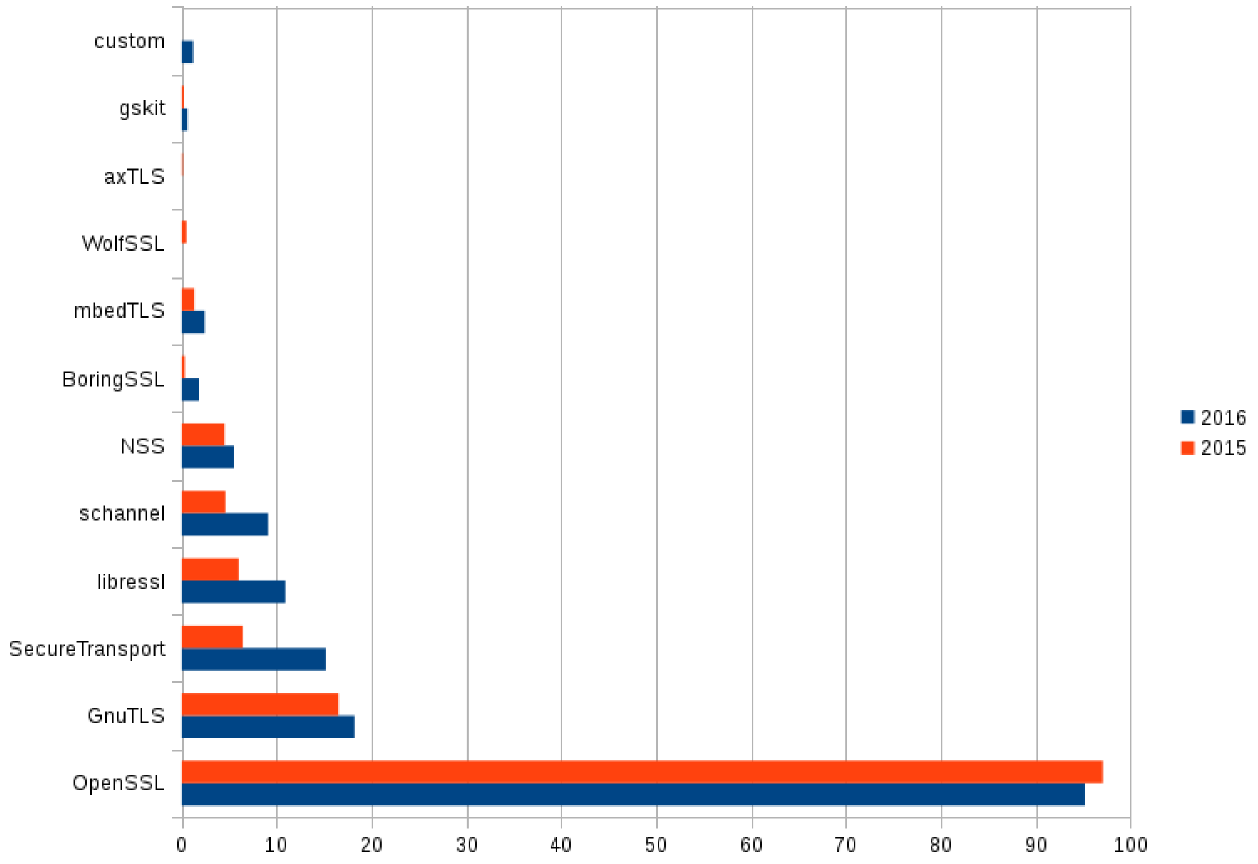
Supporting the vast amount of backends that we do takes a significant effort. Are they used?

Yes, most them are but OpenSSL is the main TLS library now just as well as has always been. The unthreatened number one curl TLS backend.

A little interesting fact is that 1.2% marked “custom version”, which then would imply that there are users putting in their own TLS backend magic!



Comparing to 2015 there are some interesting numbers that either is an indication of a trend or just shows that we had a different user set that answered the question this year.



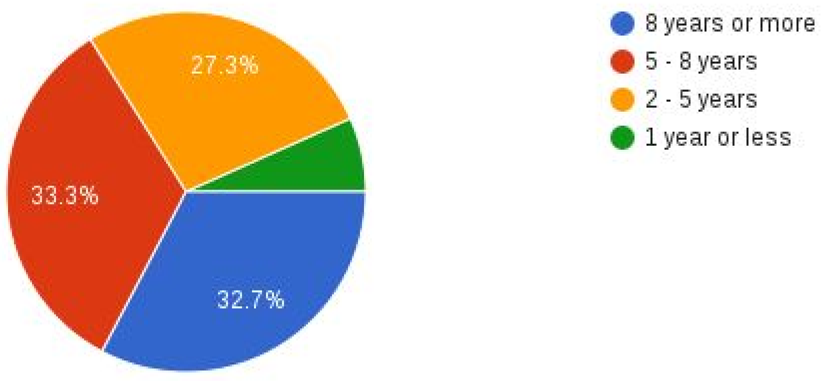
Three TLS backends usage grew significantly. SecureTransport (the native Mac OS and iOS TLS backend) went from 6.4 to 15.2% usage, libressl (the OpenSSL fork) bumped up from 6 to 10.9% and schannel (the native Windows API) roughly doubled its share from 4.6% to 9.1% use.

The numbers of the more rarely used libs were so small so its hard to judge what their differences mean. Like BoringSSL went from 0.3 to 1.8. Possibly an indication that it is a library that has matured?

How many years have you been using libcurl?

n = 165

The first libcurl release was made 2001. Thus we can conclude that nobody has used it for more than 15 years! Still, our users are quite often old-time users. The question is also meant to gently check how we manage to reach out and get new users.



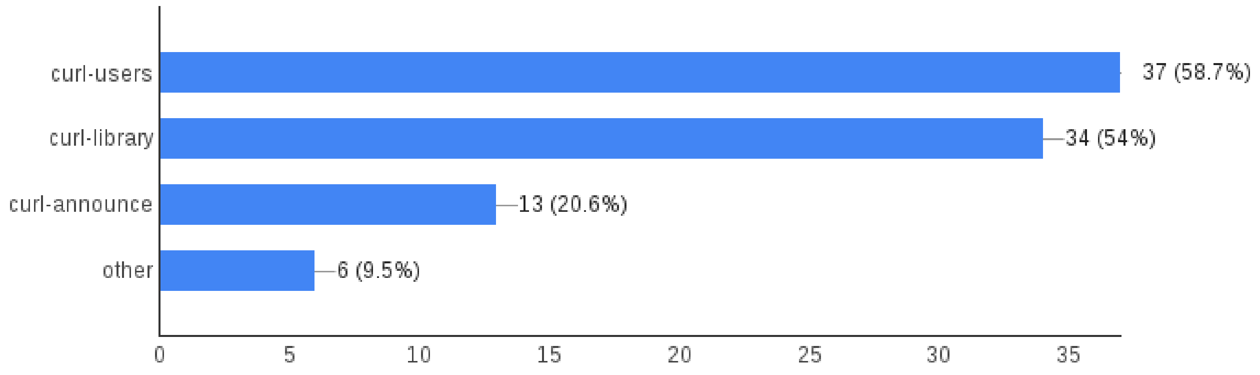
At a share of 6.7% users on one year or less, I think it seems fine. Yet that user category has shrunk over the years from the 17% in 2014 via 10% in 2015…

The distribution among the different answers remain roughly the same as in the past and I think it looks like a healthy mix too. We remain a solid choice for the old-timers and yet we are apparently good enough to attract new users as well as the ones who found us a few years ago.

Are you subscribed to a curl mailing list?

n = 63

Look, only 63 out of the 176 responded to this question. Roughly a third. It shows that the survey at least reached out fairly good to people outside of the mailing lists.

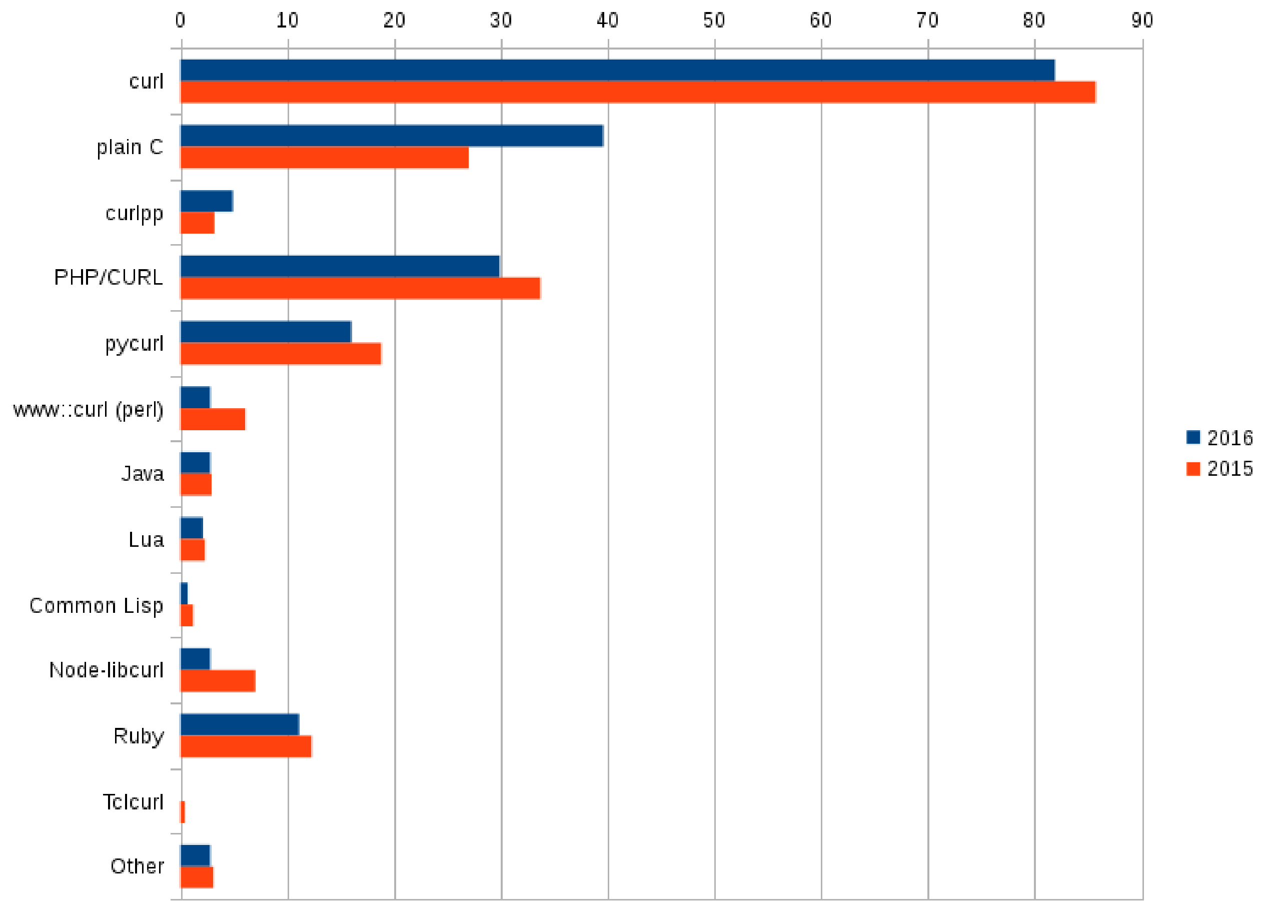


If you're using a libcurl binding, tell us which!

n = 144

It is interesting to learn which the popular bindings are, and it is also interesting to learn which bindings the users have been using when they've responded to these questions.

This year was almost identical to last year. The only notable change was that more people marked the plain C API this year. The curl command line tool, the C API and the PHP/CURL binding remain our top entries to the world of curl, with Python and Ruby bindings following.

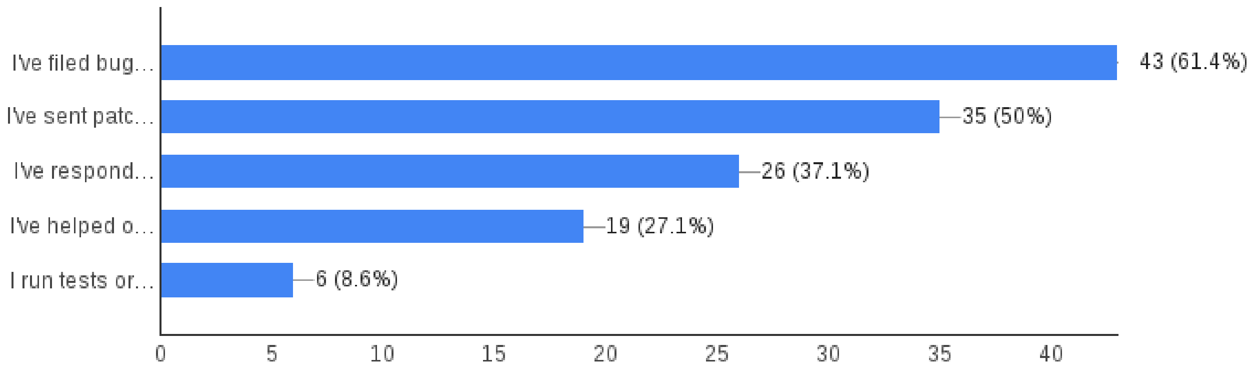


How have you contributed to the curl project?

n = 70

So 70 out of the 176 who responded are contributors. About 40%. We have over 1400 named contributors so far which then tells us that a sad 5% of our contributors answered the survey!

Those who contributed, did the following. Two thirds filed bugs, half sent patches, a little over a third responded on mailing lists/forms, roughly a forth help “in other ways” and out of the 40% who has helped us about every 11th respondent run tests or infrastructure for us!

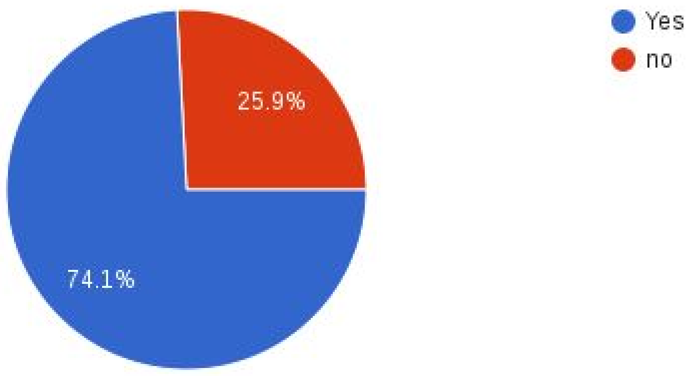


(Due to deficiencies in how I recorded the answers the previous two years, I cannot do any reliable comparisons with other years.)

Are you involved in other open source projects?

n = 174

A steady ¾ says yes:



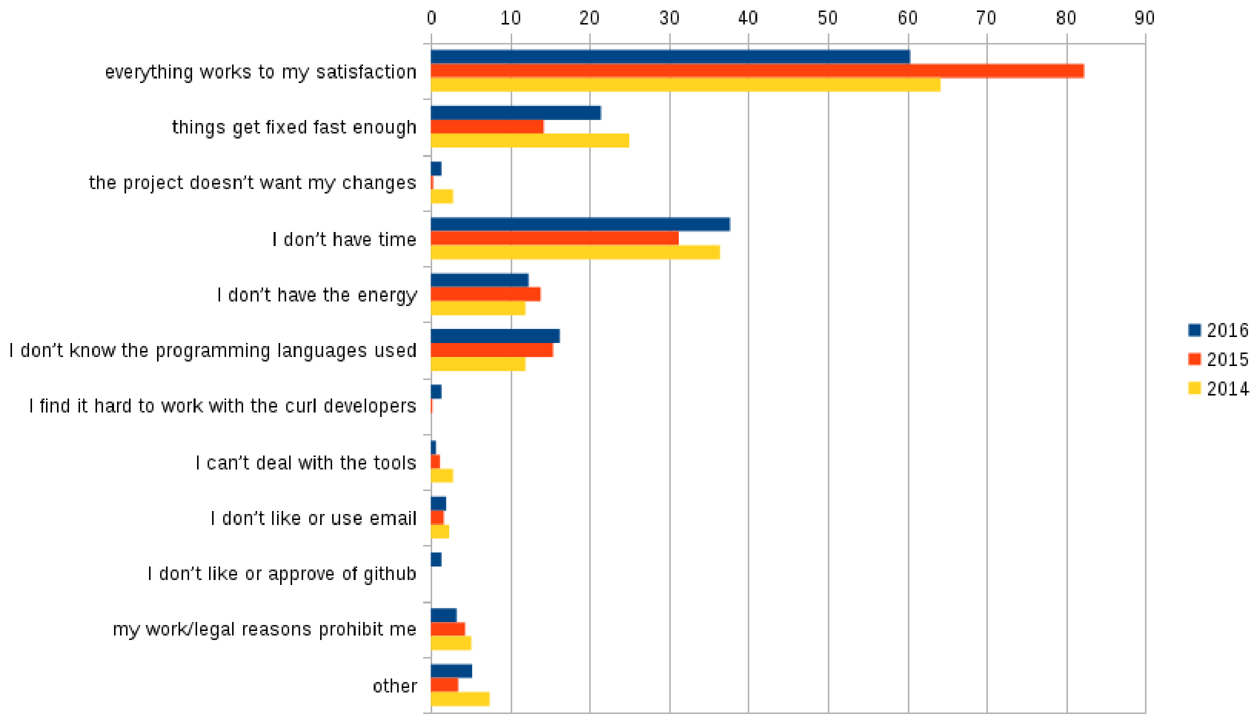
This was 77.7% in 2014 and 69.9% in 2015. Pretty stable.

What are the primary reasons you haven't contributed or don't contribute more to the project?

n = 154

So where are our obstacles for users to contribute more? Note that this was a multiple choice question so people may have checked several ones.

The reasons closely match the pattern from previous years, which could indicate we haven’t made things a lot worse. Some numbers moved a little though:



The "everything works to my satisfaction" answer went from 82% last year to 60% 2016, which is a notable decrease. I suspect it can be because of a slightly different demographics among the responders 2015. 64% marked that answer in 2014 with a similar total amount of responder as this year.

The "I don't have time" explanation is the second most selected at 37.7% this year. A typical sensation in today's modern life, no doubt. At least a only a mere 12.3% says it is because they lack energy.

The new answer "I don't like or approve of github" was gladly not getting a lot of traction: 1.3%, and some of the other answers that are there in an attempt to catch negative feelings also haven't changed much from the previous years.

The amount of users who claim they can't contribute due to legal/work reasons are now at 3.2%, which is slightly lower than before but seems to hover around that level.

How good is the project and its members to handle…

n = between 48 and 80

A very small sample size. This is 7 questions (2 new for this year) where the respondent rates the project from 1 very bad to 5 very good. We then calculate the average score for each question and then compare the questions with previous years.

The exact questions were:

1. Patches and pull-requests
2. Bug reports
3. Female contributors and other minorities

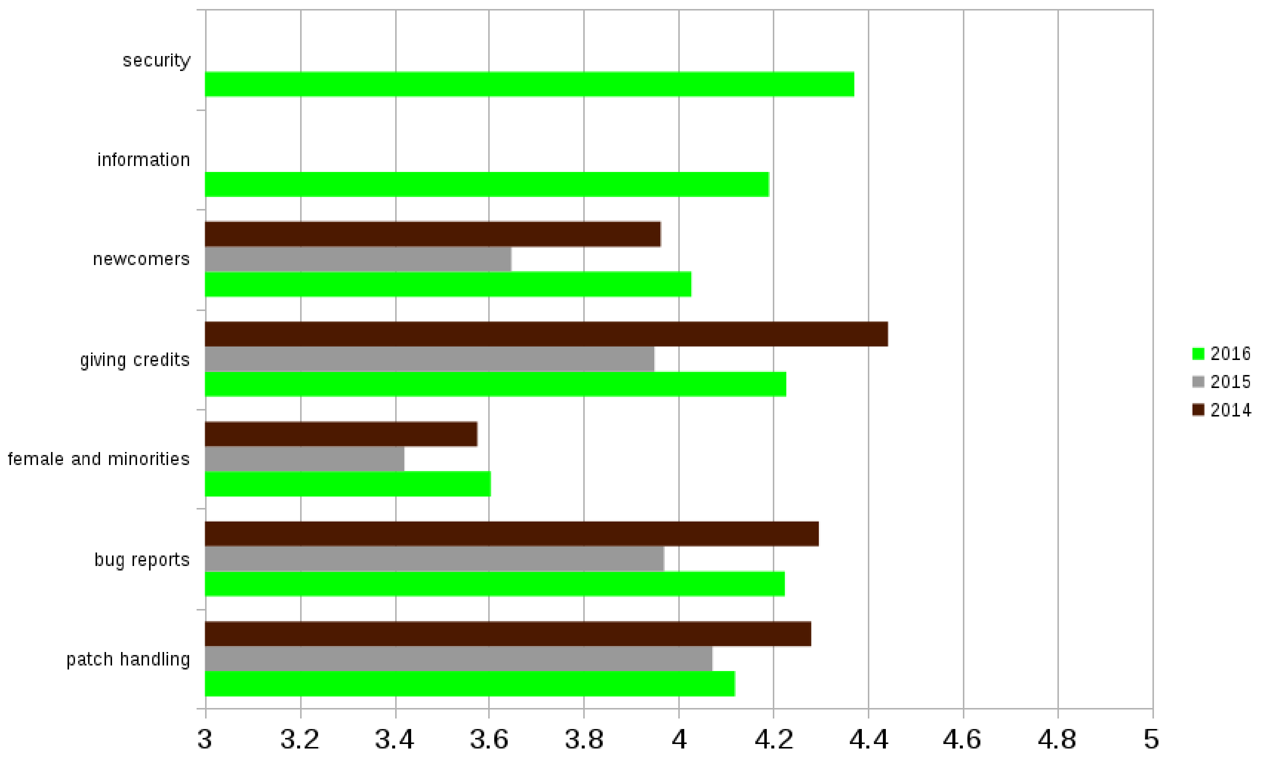
4. Attribution and giving credits
5. Helping newcomers to the project
6. Information about what's going on (new)
7. Handling security-related issues

The total average score for 5 questions was 4.11 in 2014 and 3.81 in 2015, and it was again bumped up to 4.11 again in 2016 (this year based on 7 questions). Over all very good scores I think. As the graph below shows, our users have not detected much change over the years in these aspects.

It is also a question how long time it takes before our changes are visible. For example, in March 2015 we switched track and are since then *welcoming* pull-requests and issues on github, lowering the bar for more contributors – and looking by the numbers I believe it has increased the number of contributions. Still our bug reports and patch handling scores did not change much.

Compared to last year all scores were better, compared to 2014 we're worse at giving credits, handling bug reports and patch handling. It's just completely oblivious to me how so. But then again the changes are so small and based on so few answers that it would be wrong to draw any major conclusions.

Note that the graph below shows the bars from 3 to 5 as no score was lower, to increase the resolution in the displayed results.



Which are the curl project's best areas?

n = 150

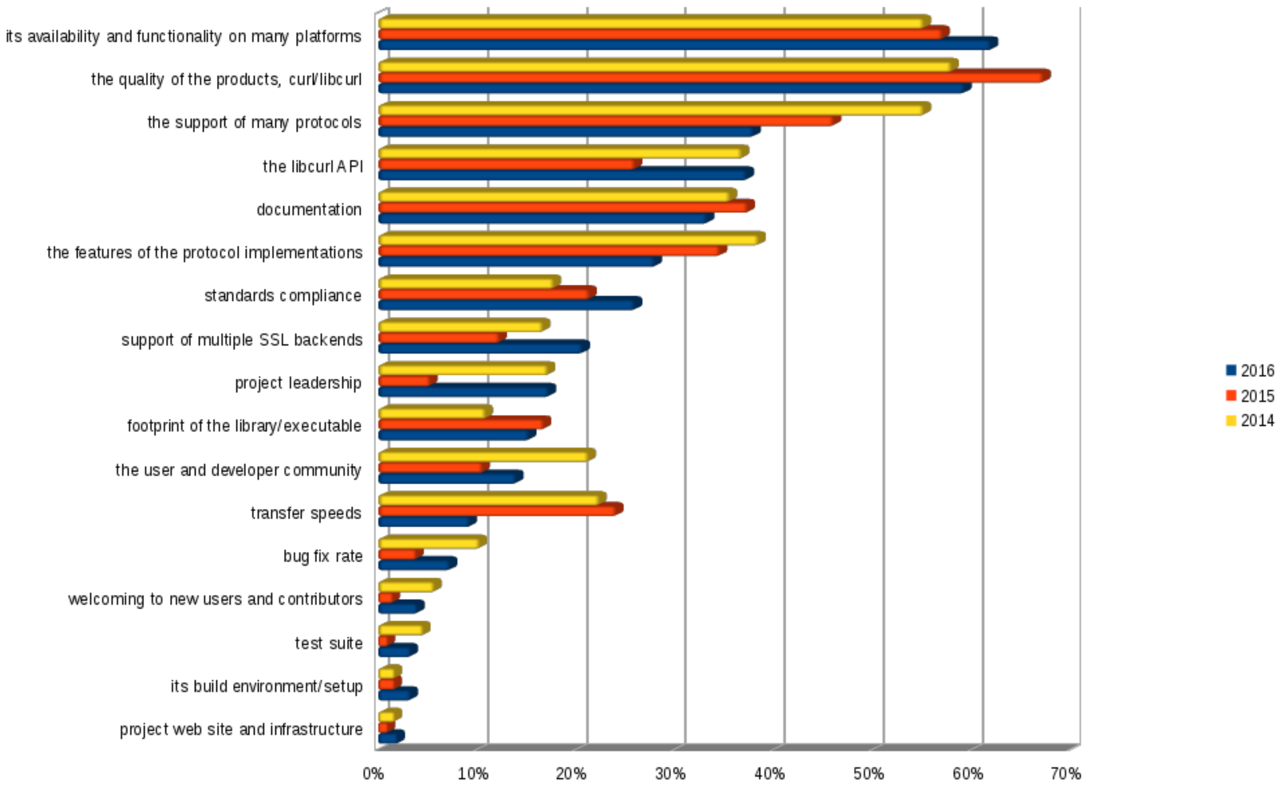
Here users could select up to 5 areas he/she thinks are the curl project's best ones, out of 17 listed. In no particular order.

An interesting detail here is that this year we got a new number one result, as the “quality” area was moved down to second place (59.3%), the “availability on many platforms” (62%) area become the new top quality. Something I personally find interesting as we live in times where I think the platform diversity is rather shrinking than growing in general.

The “support of many protocols” remains the number three best area, which serves as good input and counter-argument to those who argue we should cut off support for all protocols except HTTP(S).

In spite of our hard efforts on improving the documentation since last year, it seems to only have fallen down a tad bit more. It landed at forth place 2015, and is fifth this year. “the libcurl API” climbed from a 6th place last year to a 4th place here.

The graph below shorts the three last years compared, and the order is sorted based on how they scored this year (2016)



The least selected as best area are the same ones over the years as you can see above: project web site and infrastructure (2%), its build environment/setup (3.3%), test suite (3.3%), welcoming to new users and contributors (4%), bug fix rate (7.3%).

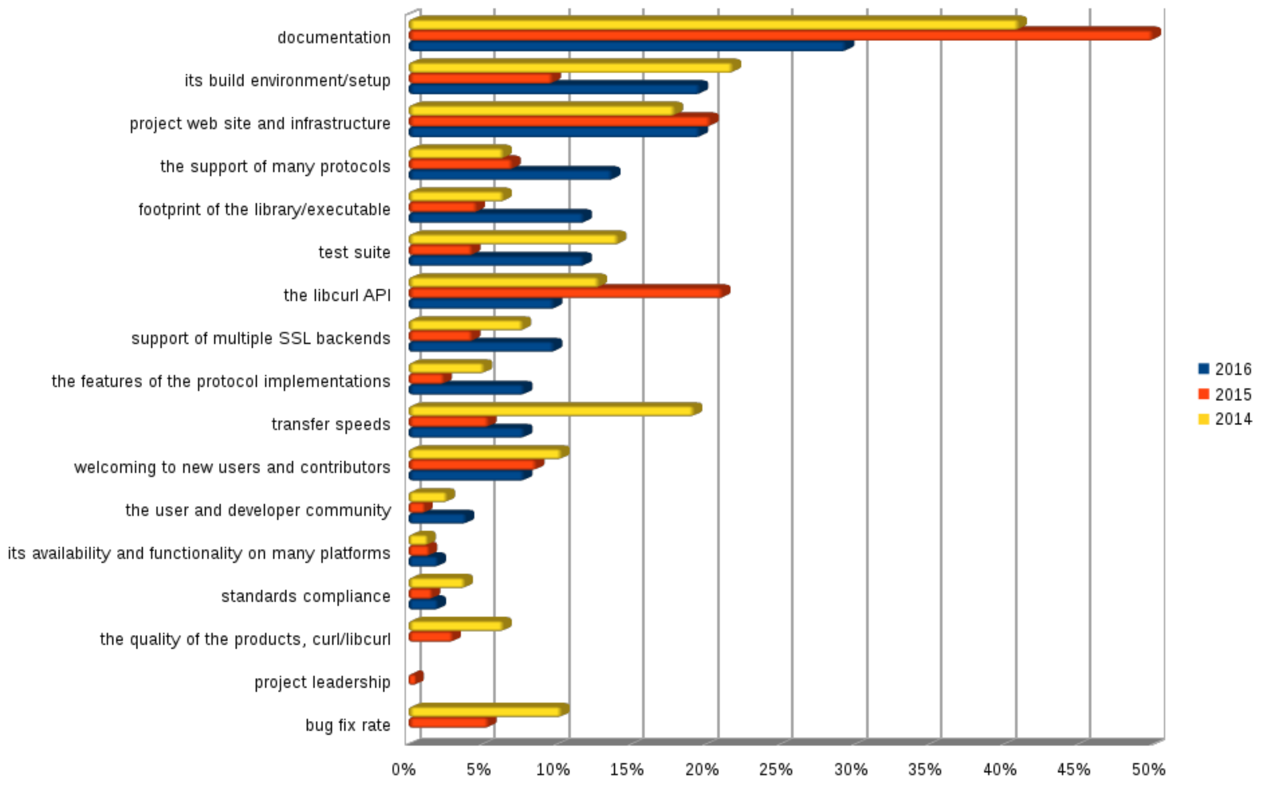
Which are the curl project’s worst areas?

n = 51

Three times as many filled in answers for best areas as answered the question about worst areas. This was another question that allowed the respondent to fill in 5 areas and the selection is the same 17 areas from above.

This is always very interesting to compare with the former question. A naive thinking would be that this list would be virtually the former one up-side-down, but no, it never is.

Let’s see them sorted by the score of 2016:



Maybe this is how our improve-the-documentation effort can be seen? Documentation as “worst area” is still the number one worst, but now down to being marked by 29.4% of the users, from 50% last year and 41% in 2014. A notable difference.

Other than the first listed area, there were a bunch of movements in the top list. Look at how the top-6 this year were positioned this year compared to previous years:

Position 2016	**Area**	**Position 2015**	**Position 2014**
1	documentation	1	1
2	its build environment/setup	4	2
3	project web site and infrastructure	3	4
4	the support of many protocols	6	10
5	footprint of the library/executable	9	11
6	test suite	10	5

(The 2nd and 3rd place got the same score 2016: 19.6%)

I fear that we can blame the low number of answers here. Possibly it is also an indication that the web site reorg we did in June 2015 did not improve the web site much for people.

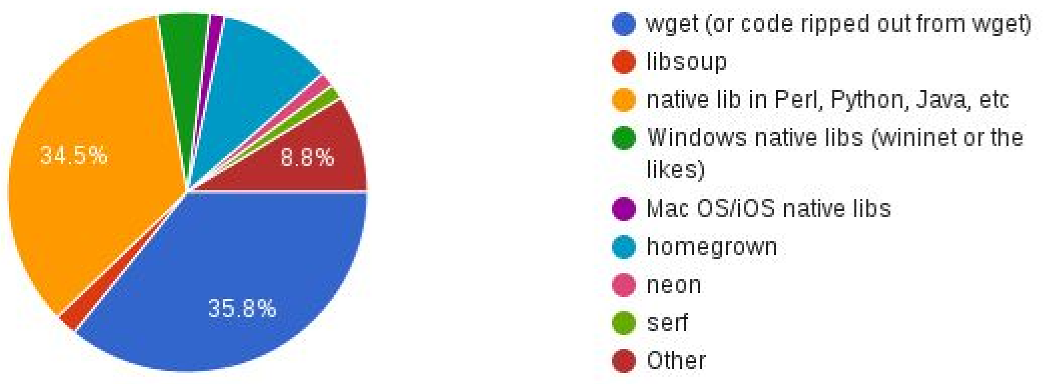
The libcurl API was the second worst area last year and ended up on place 7 this year!

Transfer speed was checked 7.8% down from the whopping 19.2% in 2014. I still don’t think we’ve changed anything significantly in curl/libcurl that has changed transfer speeds. So what made people change their perception? In the same time period, transfer speed got marked as “best area” 9.3% in 2016 down from 22.4% in 2014. So basically half the amount as before thought it was a best area and half the amount thought it is a worst area! Confusing.

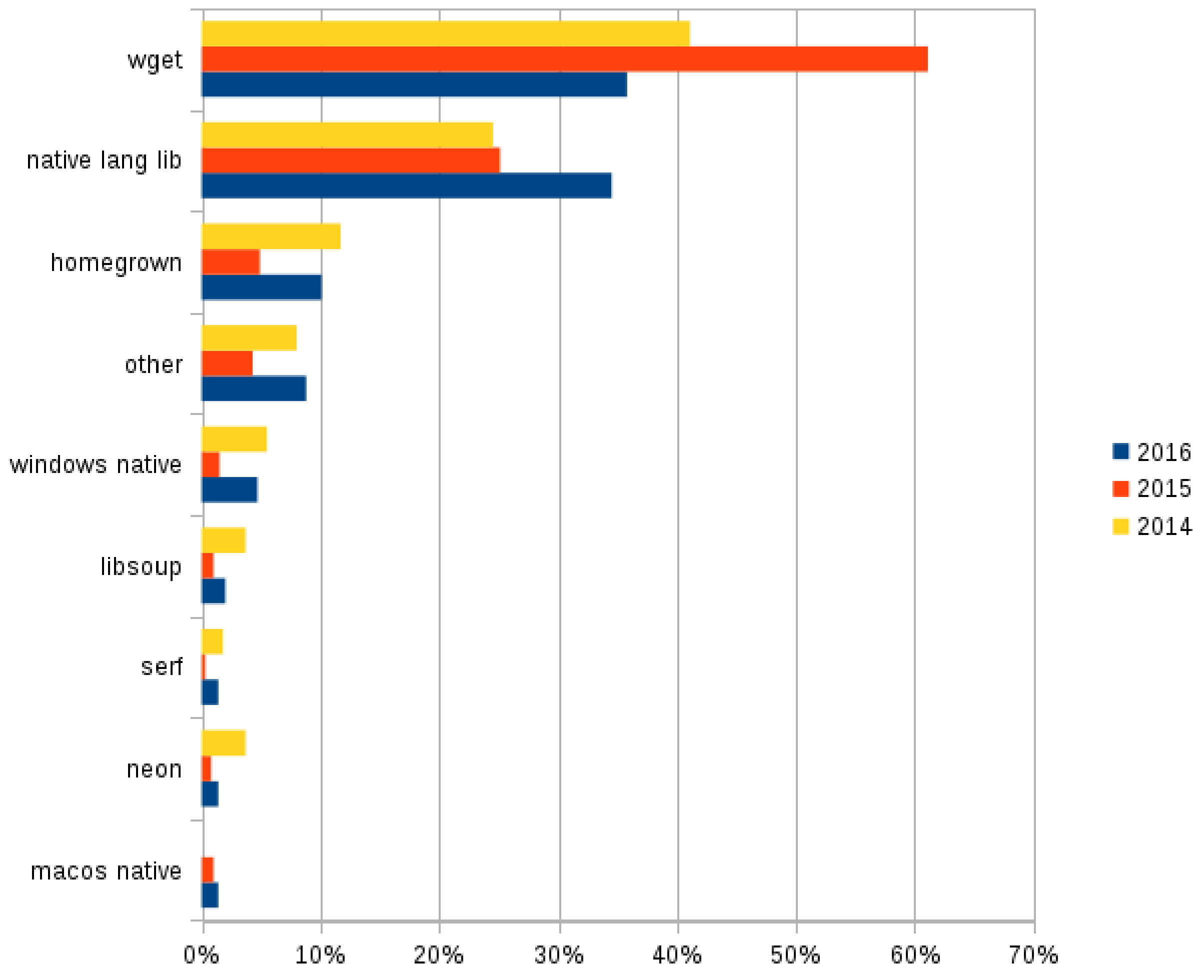
If you couldn't use libcurl, what would be your preferred alternative?

n = 148

This is a question to see what people consider the alternatives or the main competition to be. The pattern in the answers matches that of previous years:



Something based on wget (code) or a native library are what most people would go for without libcurl.

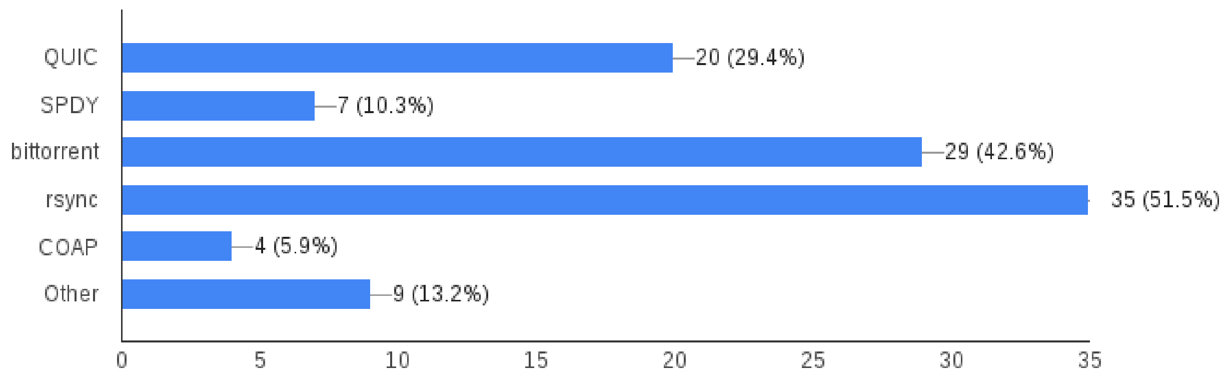


If you miss support for a protocol, tell us which!

n = 68

Okay, now we're getting into the blue sky visions and ideas of what curl should and shouldn't do in the future. Curl supports an excessively large amount of protocols already so I find this question very interesting. Do folks want more protocol support added? I think the answer is a clear *yes!*

This year I offered a few protocols to check and there was an Other that allowed people to write in additional suggestions.



QUIC, the transfer protocol that's been made and pushed by Google is just now taken to the IETF and it will go through that procedure. I'm already very keen on adding support for this to curl/libcurl so I don't think there's any doubt that we should do this. It's more just a question of when and how it will happen. The way it worked out with HTTP/2, with an external library for all the nitty gritty binary details is ideal and allows more projects to share the work and reap the benefits.

To my great surprise, both bittorrent and rsync got even higher numbers than QUIC. I guess our TODO list won't have to dry up any time soon. I think in general people aren't much aware of the small "permanent" development team we have… We are in great need of more hands to be able to pull through adding protocols such as these. Preferably developers who have a use case and personal need for them.

COAP has been mentioned and discussed a little on the mailing list before, and I've had my own personal general doubts about it. I still think it could be done if someone just is interested and devoted enough to pull it through.

SPDY is the protocol that was the precursor to HTTP/2 and is still supported on 7% of the Internet's top 10-million web sites. There's existing libraries for it that we could use much like we use nghttp2 for HTTP/2, so implementing support for this could be easier than many other of the mentioned protocols. Still it is mostly a protocol of former glory. Chrome just removed support for it in their latest release and the future is probably going to be much more HTTP/2 than SPDY – then again, nothing ever dies on the Internet so I'm sure we'll see SPDY remaining on X percent of servers for a very long time to come.

Other protocols people suggested we should add in the free form text field:

- websockets (websockets really is a hard match for curl, in the same league as TELNET really)
- "email other than imap or pop3" (I have no idea what this might entail)
- git
- nntp
- ssh (I suppose SCP and SFTP aren't enough then? SSH is more like TELNET and isn't a very good match for libcurl APIs etc IMHO)
- h2 (HTTP/2 has been supported for several years already!)
- RFC 7574 - Peer-to-Peer Streaming Peer Protocol, PPSPP
- "please keep gopher!"

Looking back at answers to this question, we see a pattern: SPDY, QUIC, COAP, websockets, bittorrent and rsync keep getting mentioned.

What feature/bug fix would you like to see the project implement next?

n = 31

Opening up the table to listen in what ideas and thoughts people have on what we should do next.

New stuff

- DANE support
- COAP Support
- support for ZSYNC or similar
- Websocket support
- Unicode support

Better

- Better RTMP support
- Detecting remote server out of space, better transfer progress stats, etc.
- Better logging for connection issues with SSH (e.g., for SFTP). Better login for auto creation of directories (for SFTP).

Build

- Better build for MinGW that allows specifying openssl libzip et al better than hardcording paths in Windows
- Updated, complete, comprehensive Windows Build instructions that result in a working .exe, if someone can help me through the process, I can update the docs. Also some help to use Git so I can submit them myself for review.
- better build support for Android NDK, HTTP2 support without additional dependency

Improved examples

- better multi support and examples, better performance when using multi unthreaded (select/epoll/etc based)
- More and more example for each multi_socket API
- out of the box working examples for HTTP client with curl multi and libevent in c++

Command line

- "User daemon" mode. When curl is started in this mode it opens unix socket and current user can do `echo <command> > /var/tmp/unix-socket` a shell script/command line to put download task or retrieve progress information. (This is mostly what TODO item 17.12 specifies already)
- I use curl -OL, but I miss the support for -OL together with "-c -"
- HTTP pipelining in the command line curl. Would be used for security testing. (This requires the curl tool to switch to use the multi API, mentioned in the TODO item 17.4)

Bug fixes

- HTTPS working out of the box on Mac OS X (unclear what this means or what we don't do good enough already)
- More support for cert handling (pinning, etc) with SSPI.
- More tests for HTTP2 (An HTTP/2 test server has since been added to the test suite and we can now more easily add HTTP/2 tests)
- More non-blocking especially for proxy-related code.
- There is a bug with winssl that doesn't seem well documented and I'm struggling to nail down. Makes winssl completely unreliable for me, even though the bug is only a handful of users.
- http2 server push enhancements (unclear what enhancements this means!)

General Improvements

- validate invalid input (unclear exactly what input this refers to)
- Working on a way to send text-only email using scp (curl ?) emails using qmail inject on a site where I have privileges. (seems very specific to a particular solution)
- Callback or more likely "failure" information for client-side certificates so we can put up a chooser UI. (this sounds easier said than done, but adding ways to help applications find out what's wrong is always good)
- Cache OpenSSL contexts, #13.4 on the TODO list (yeah, "someone" should work on it!)
- Show the negotiated curve, too, in addition to the cipher suite - the ID in hex suffices
- Curl can't sync it's cookies between multiple instances, that would be nice. (yes it can)
- CMake support (we support cmake already, what's missing?)
- A nice option would be selection of methods that stick. Enable or disable the options that you really use. (This is a nifty idea – specify which options to automatically "reset" back to default after a transfer has completed!)

Blue sky

- Something to make SSL less of a nightmare. (Hehe, I don't think we have the mandate to change SSL in any particular way)
- reduce feature set/spin off some protocols into other projects

What feature/bug fix would you like to see the project REMOVE?

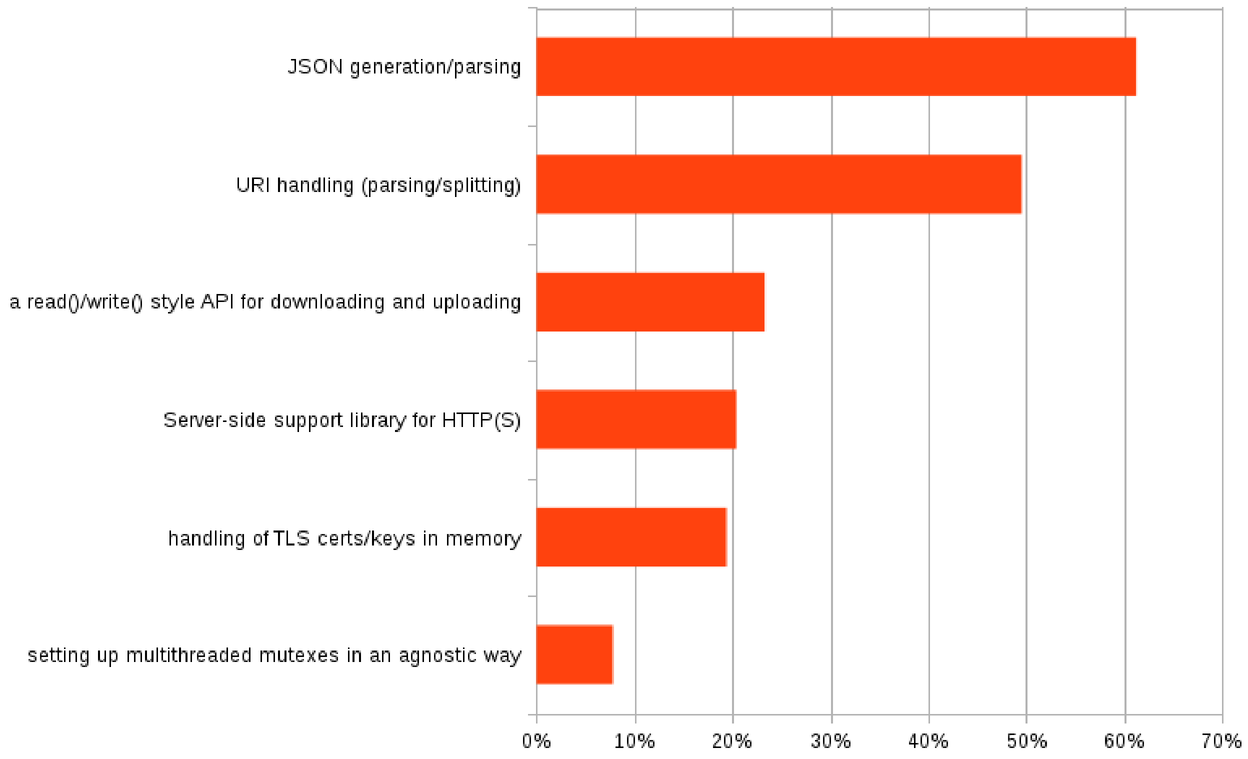
n = 13

- IPv4 support (I won't even write a witty response to that...)
- Remove confusion about Windows Build process. ;-)
- Support for ancient Windows versions (< 2000), makes writing patches a bit hard sometimes (we are more or less already on track on making XP the lowest Windows platform we will bother about as a project)
- gopher (probably one of the least impact protocols, there's really not much of a penalty to anyone to keep this)
- Automatic HTTP redirection to non HTTP protocols is almost a security vulnerability. Many programs use libcurl for only HTTP, but forget to disable protocols or use CURLOPT_REDIR_PROTOCOLS. This can allow sever side request forgery using unexpected protocols. This was one of the things I fixed in git as part of CVE-2015-7545. Maybe just better documentation? Or disable some by default?
- None really, though libcurl seems to be doing a lot, which hopefully doesn't impact development on HTTP (s/2/etc) features, which it seems to be most used for. (we spend most development time and efforts on the most commonly used protocols)
- SPDY (I think we'd first need to add support for it before we could remove it!)
- NTLM
- Legacy protocols; web protocols are enough
- Protocols other than HTTP ;-), at least the ones "nobody" uses e.g telnet.
- curl_easy_setopt (okay, that's certainly not the most elegant API but we'd need to come up with a decent alternative first before we can remove that fundamental pillar from libcurl)
- rtmp, rtsp

Which of these API(s) would you use if they existed?

n = 103

At times we discuss adding support for new things and new APIs to libcurl. This question was asked to see what the interest level for some API ideas would be in the general libcurl audience . This question has not been asked before:



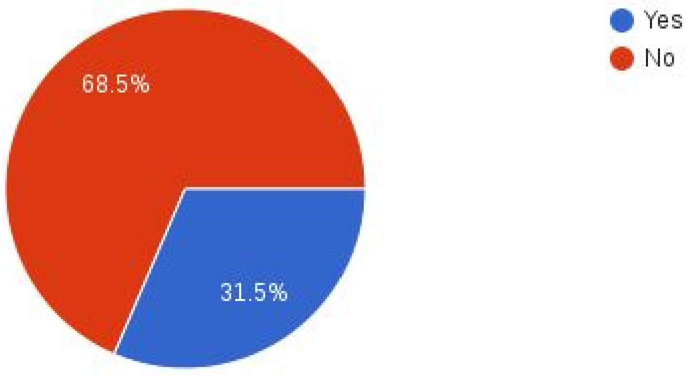
There's clearly a huge interest in JSON support and ways to parse/split URIs. I think we should investigate what we can do and what is sensible to do in these areas going forward. In regards to the read()/write() style API that reached 23%, that's what I've started playing with in a separate project I call fcurl. See this blog post for details: https://daniel.haxx.se/blog/2016/04/24/fcurl-is-fread-and-friends-for-urls/. I have not received very many comments or feedback on that project so far though, which could imply that it is easy to check "sure I'm interested" in a checkbox in a survey, but then to actually in real life do something is a complete different matter…

Server-side support and TLS keys in memory got 20.4% and 19.4%, meaning quite a few people are interested in those too. They're both sort of complicated and problematic in their own ways so even though there's interest in them, I suspect we won't see a huge undertaking in people developing API for this anytime soon. But I'm not ruling that out!

Should curl join an umbrella project?

n = 124

People generally think we should stay totally independent. I think I might add a text box to this question next year to allow people to provide a reason for their vote on this.



It might be my ego talking, but I consider this to be a small confirmation that we’re leading and driving this project in a mostly sensible and desirable way, so that people do not have reason to think that it would get a better treatment if it would land in one of the open source/umbrella organizations.

Last year, 57% voted no and the question wasn’t asked 2014.

Final words

“We’re not done yet” is the primary conclusion I draw from this year’s survey. There’s plenty of good feedback and several good ideas here that we should try to carry forward.