curl user survey analysis 2022

“This is such an awesome project and the gold standard for how to run an open source project. It was so nice getting in contact and participating when I found some issues. I just love the approach to keep doing everything better, all the time, and over the years get to this level.”

summary and analysis by Daniel Stenberg

Jun 16, 2022

About curl

Curl is an established and mature open source project that produces the curl tool and the libcurl library. While is a small project, with few maintainers its products run in several billion Internet connected devices, applications, tools and services. curl is without doubt one of the world's most widely used software components.

Survey Background

We run a curl user survey annually in an attempt to catch trends, views and longer running changes in the project, its users and in how curl fits into the wider ecosystem. This year, the survey was up 14 days from May 18 to and including May 31. This was the 9th annual survey as the first one ran in 2014.

The survey was announced on the curl-users and curl-library mailing lists (with reminders), numerous times on Daniel's twitter feed (@bagder), on LinkedIn and on Daniel's blog (https://daniel.haxx.se/blog). The survey was also announced on the curl web site with an "alert style" banner on most pages on the site that made it hard to miss for web visitors.

Survey Bias

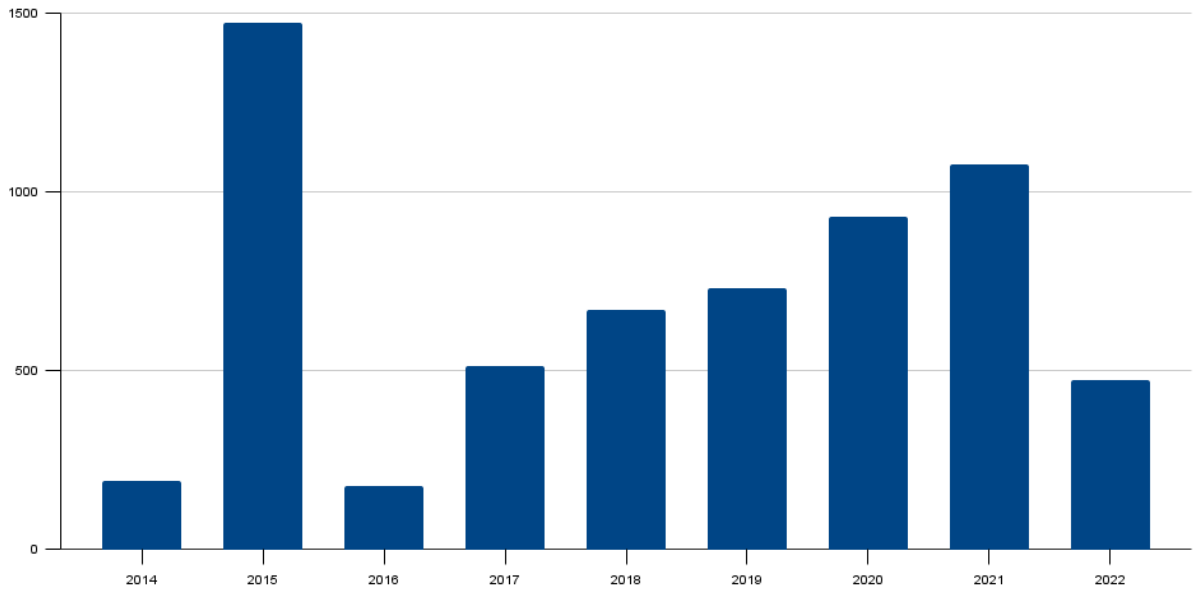
We only reach and get responses from a small subset of users who voluntarily decide to fill in the questionnaire while the vast majority of users and curl developers never get to hear about it and never get an opportunity to respond. Self-selected respondents to a survey makes the results hard to interpret and judge. 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.

Hosted by Google

We use a service run by Google to perform the survey, which leads to us losing the share of users who refuse to use services hosted by them. We feel compelled to go with simplicity, no cost and convenience of the service rather than trying to please everyone. We have not found a compelling and competitive alternative provider for the survey.

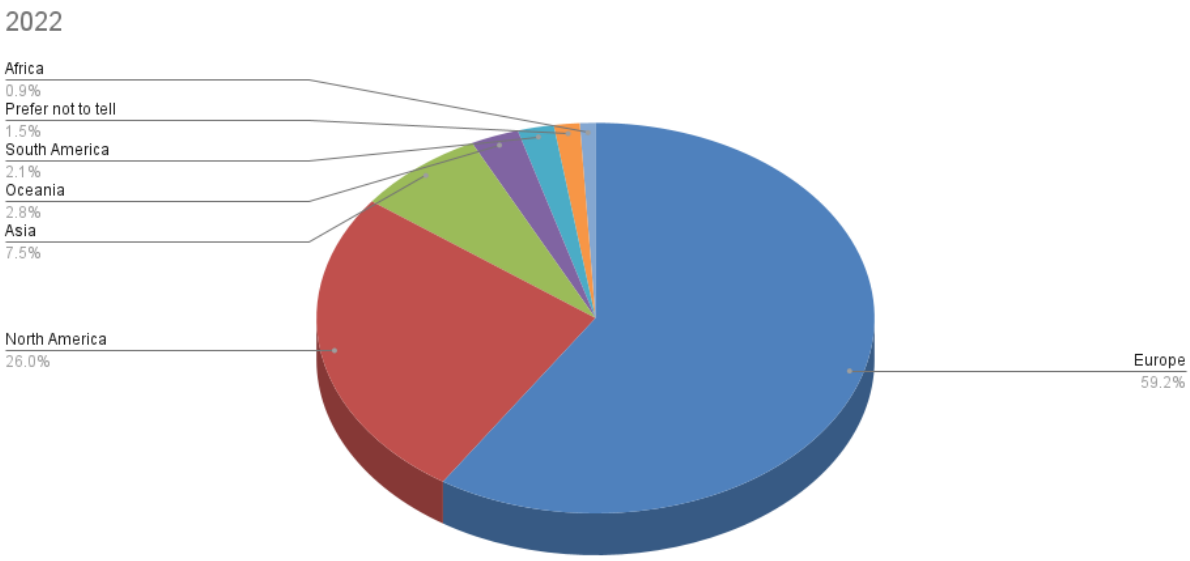
Responses

This year's survey results match previous years to a fascinating degree - which is a general pattern over the years. We seem to get strikingly similar answers year over year, even though only 18.2% of the users this year say they answered the survey last year (up from 12% last year).
One detail that broke the trend this year was the number of responses. We have seen a slow increase in attention to this survey in previous years, but in 2022 that took a real nosedive. This year's 473 responses is just 43.9% of last year's 1078 responses and the lowest participation level in six years. I have no idea why.



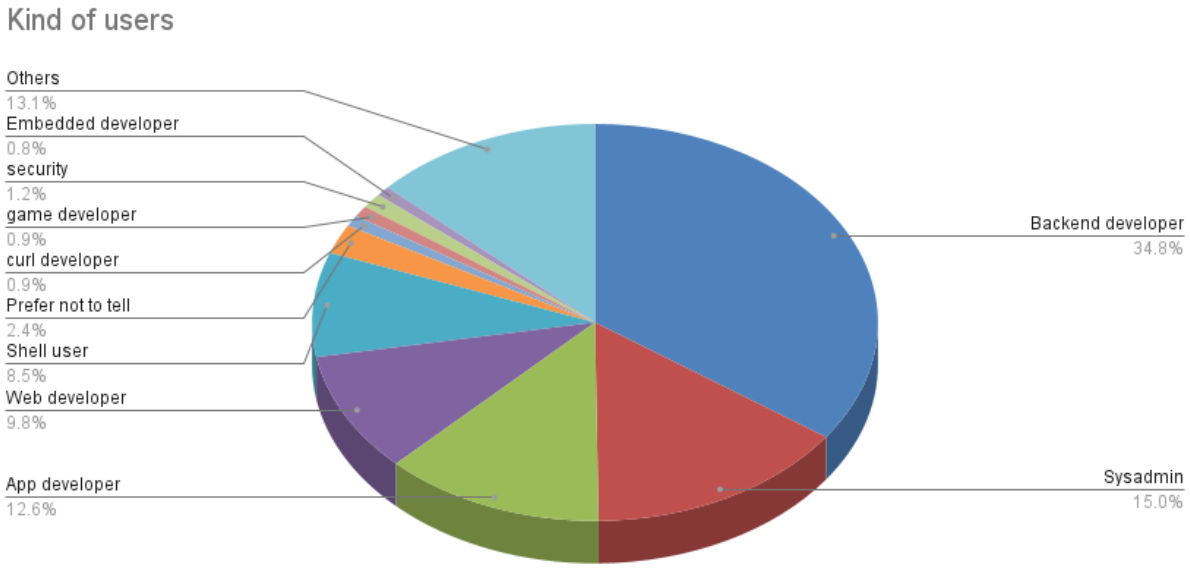
Continents

The users who answer the survey remain European to a large degree (59%). The second largest continent is North America at 26%.
The distribution across the world roughly matches the results we have seen for all five years we have asked this question.



Kind of users

To get a clue who everyone is that answers this survey, we added this question a few years ago and the response distribution is amazingly similar year-to-year. More than half identify as backend developers, sysadmins or app developers. The *other* category is notably big at 13.1%, probably because people don't quite like the provided options well enough.

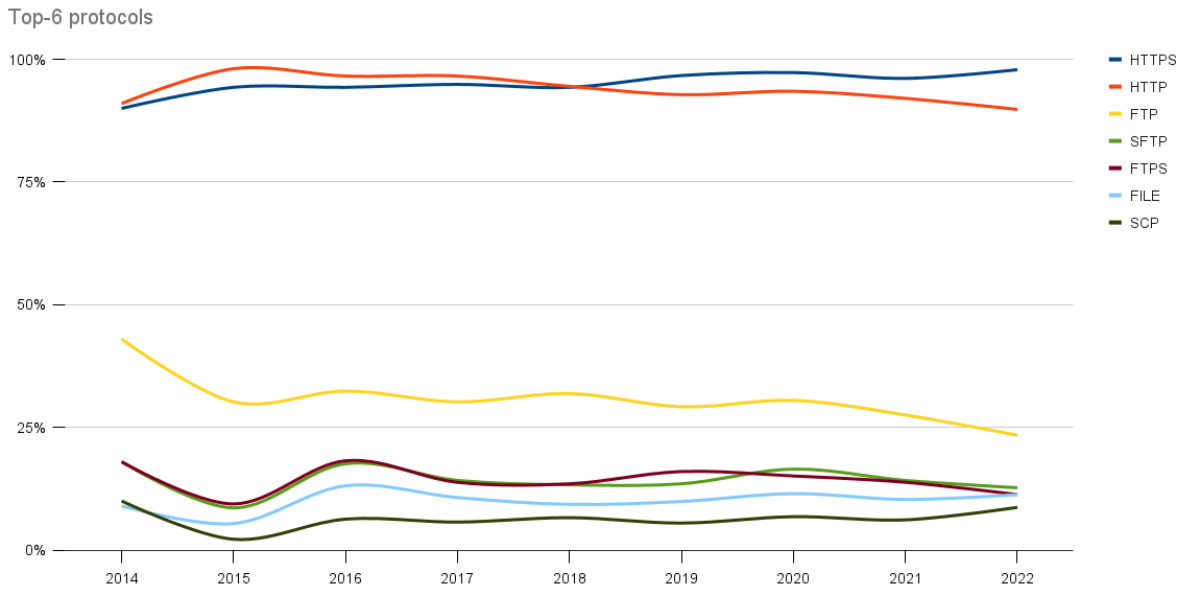


Protocols

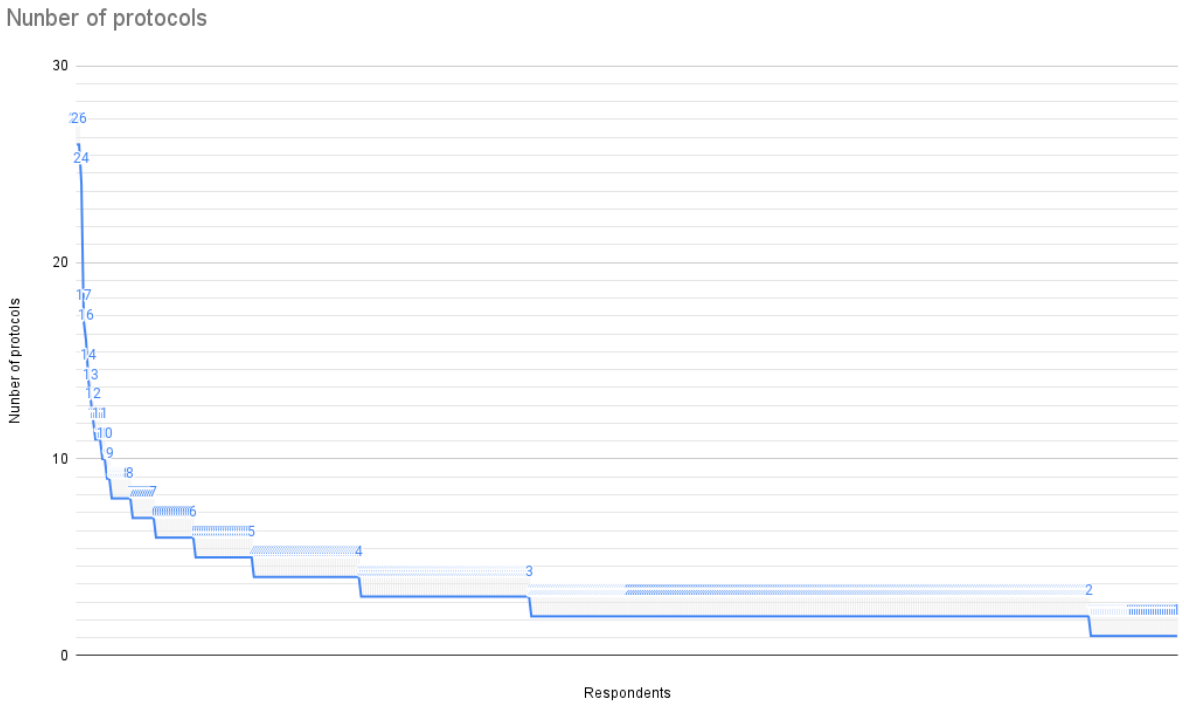
One of the most popular questions and answers from this survey every year. What protocols are people using? How have they changed from last year? This year follows the trend from previous years. HTTPS and HTTP are by far the most commonly used curl protocols. This chart is often referred back to in subsequent discussions when discussing if there are support protocols ripe for getting ripped out or otherwise deprecated.

HTTPS	97.90%
HTTP	89.80%
FTP	23.40%
SFTP	12.70%
FTPS	11.30%
FILE	11.30%
SMTP	8.70%
SCP	7.20%
SMTPS	6.20%
GOPHER	5.10%
LDAP	4.90%
LDAPS	4.90%
IMAPS	4.70%
TELNET	4.00%
IMAP	4.00%
TFTP	3.80%
SMB	3.20%
POP3	3.20%
POP3S	2.50%
MQTT	2.10%
SMBS	1.90%
RTMPS	1.70%
RTMP	1.50%
DICT	1.50%
GOPHERS	1.50%
RTSP	1.10%

Looking at the six most popular protocols this year (the ones with 10% or more usage) and how they have developed over the years, we can possibly see a decreased use of FTP - the yellow line in the graph below. HTTPS became the number one protocol in 2018 and has remained in top since then.



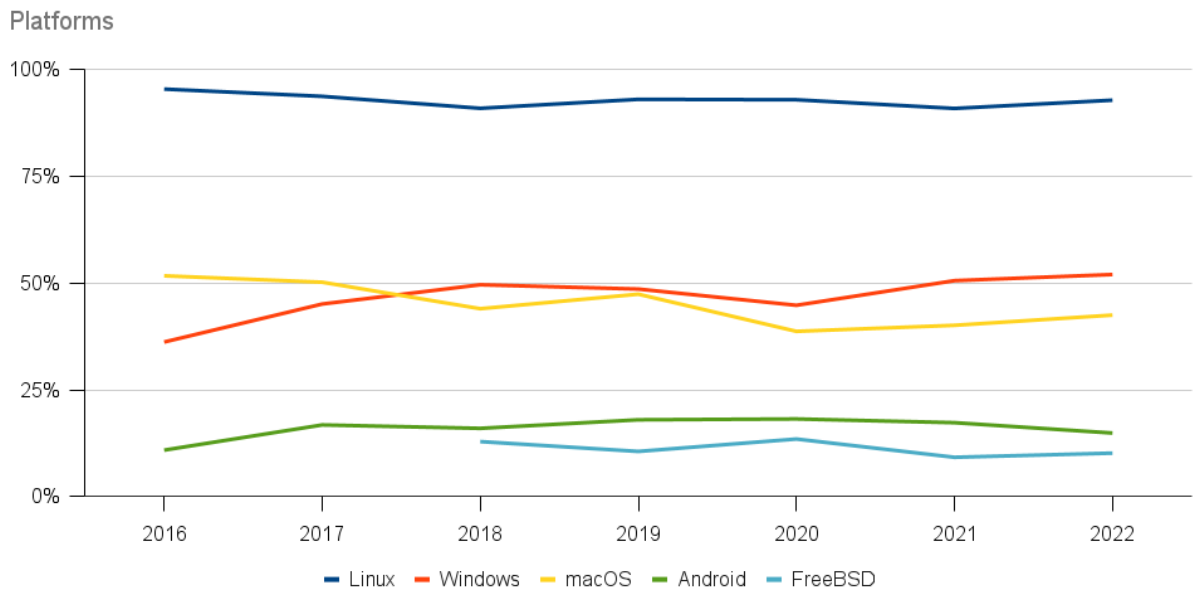
On average each respondent selected 3.2 protocols. The median number of protocols selected were two. Two users selected 26 protocols.
41.4% of users use **three** or more protocols. 26.0% use **four** or more. 16.2% use **five** or more protocols. 10.9% use **six** or more protocols. 2.9% use **ten** or more protocols!



Platforms

We have records of curl running on **86** different operating systems over the years. Several of those were probably custom modified with changes we never got upstreamed, and many of the users on niche systems most likely did not respond to this survey.

The top-5 platforms remain the same over time. Windows surpassed macOS a few years ago and it seems to reliably have established itself as the number two platform. On average, users selected 2.4 platforms in the answer.

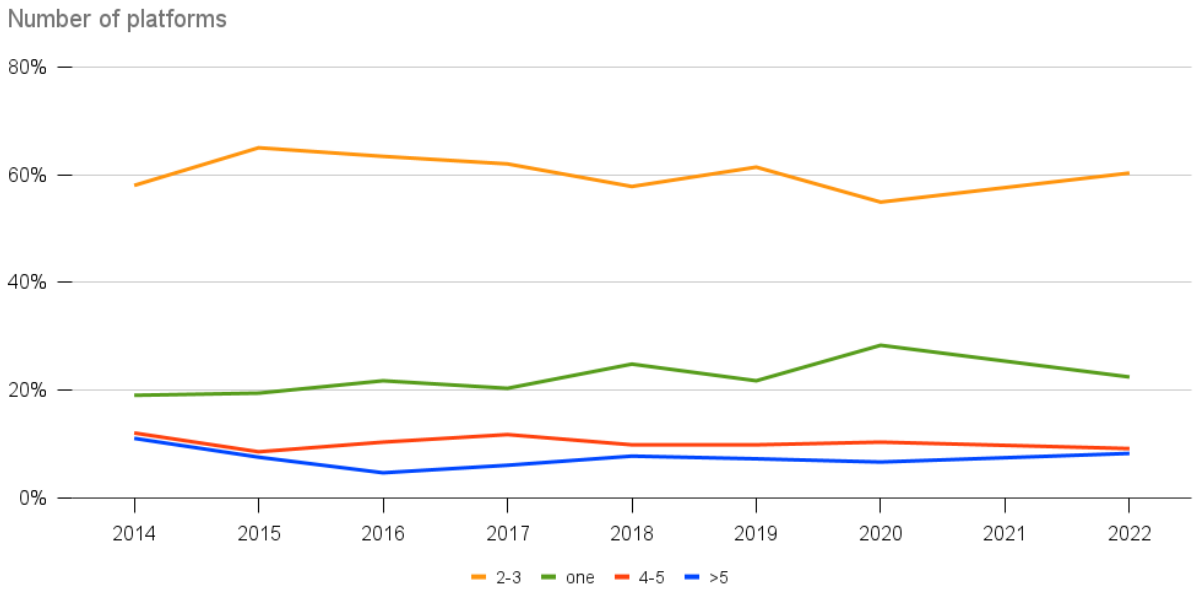


The complete results for 2022

Linux	92.80%
Windows	52.00%
macOS	42.50%
Android	14.90%
FreeBSD	10.20%
iOS	5.90%
OpenBSD	5.70%
NetBSD	2.50%
Game console	2.10%
Solaris	1.90%
IBM I	1.90%

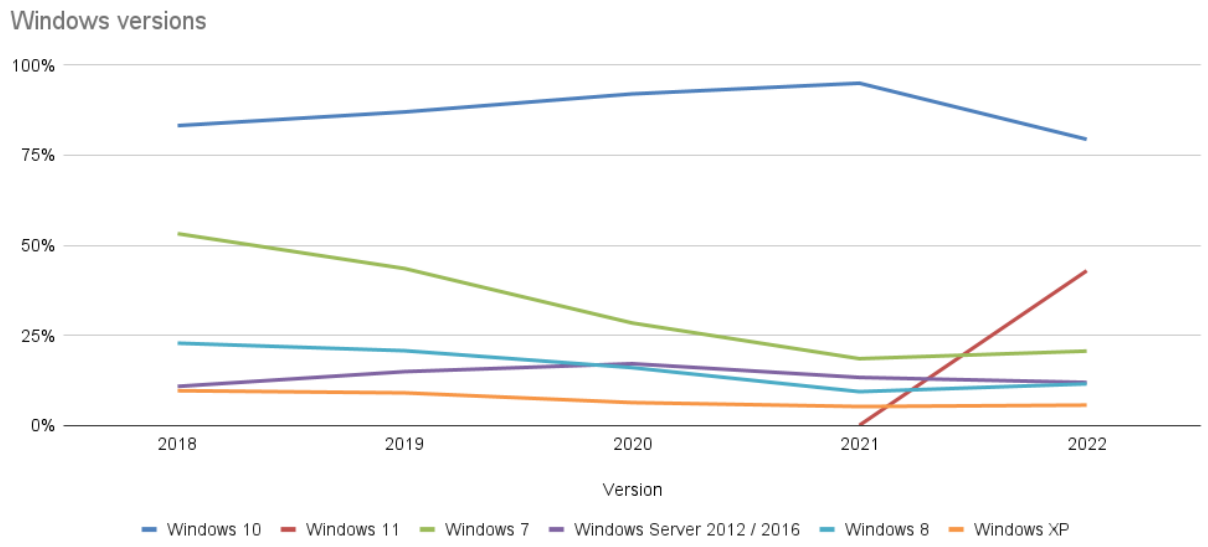
Another unix	1.30%
OpenIndiana	1.30%
AIX	1.30%
RTOS	0.80%
VMS	0.80%
MS-DOS	0.60%
AmigaOS	0.40%
HPUX	0.40%
IRIX	0.20%

60% of users remain using 2-3 platforms and the distribution of single and multi platform users remains astonishingly fixed over the years.



Windows versions

This question stands out this year as the answers are significantly different compared to last year. This was hardly any surprise since Windows 11 was released since last year's survey. It raced up to the second-most used Windows version at once with 42.9%, and correspondingly Windows 10 shrunk from 95% to 79.4%.

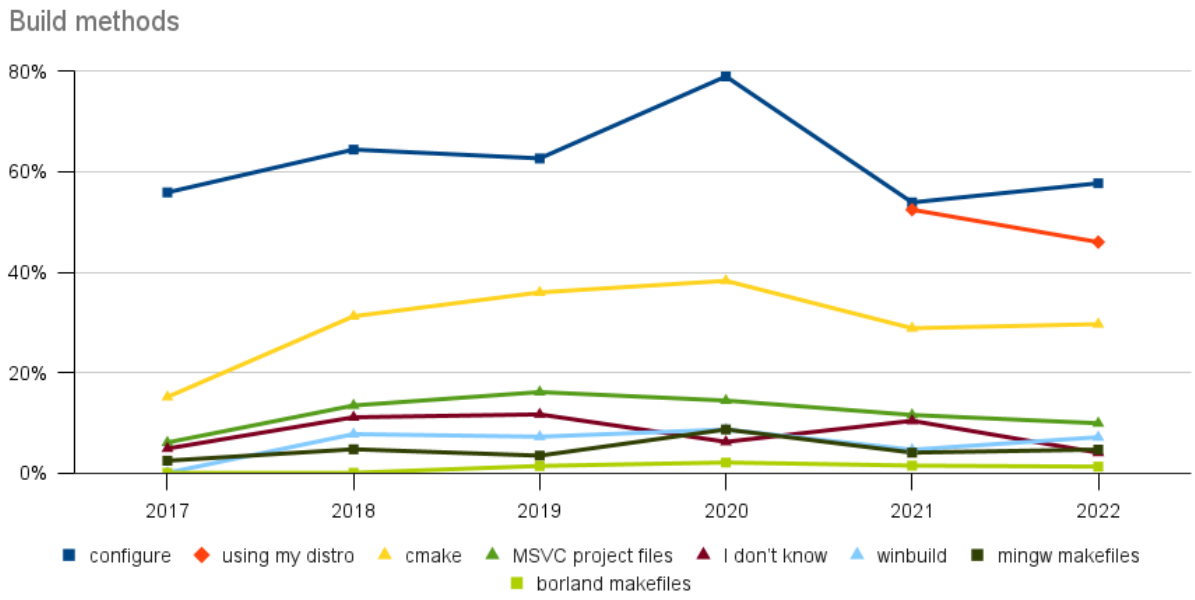


The full distribution 2022. The average response selected 1.86 versions.

Windows 10	79.4%
Windows 11	42.9%
Windows 7	20.6%
Windows Server 2012 / 2016	11.9%
Windows 8	11.5%
Windows XP	5.6%
Windows Server 2008	4.0%
Windows Vista	3.2%
Windows Server 2019	2.0%
Windows CE/Embedded	1.2%
Windows Server 2003	0.8%
Windows 2000	0.8%
Windows Server 2022	0.8%
Windows 98	0.8%
Windows 95	0.8%

Building curl

67.6% of the respondents say they don't build curl at all which certainly lowers the number of people who actually answered this question with their preferred method. configure remains the top build method this year at 57.7%.



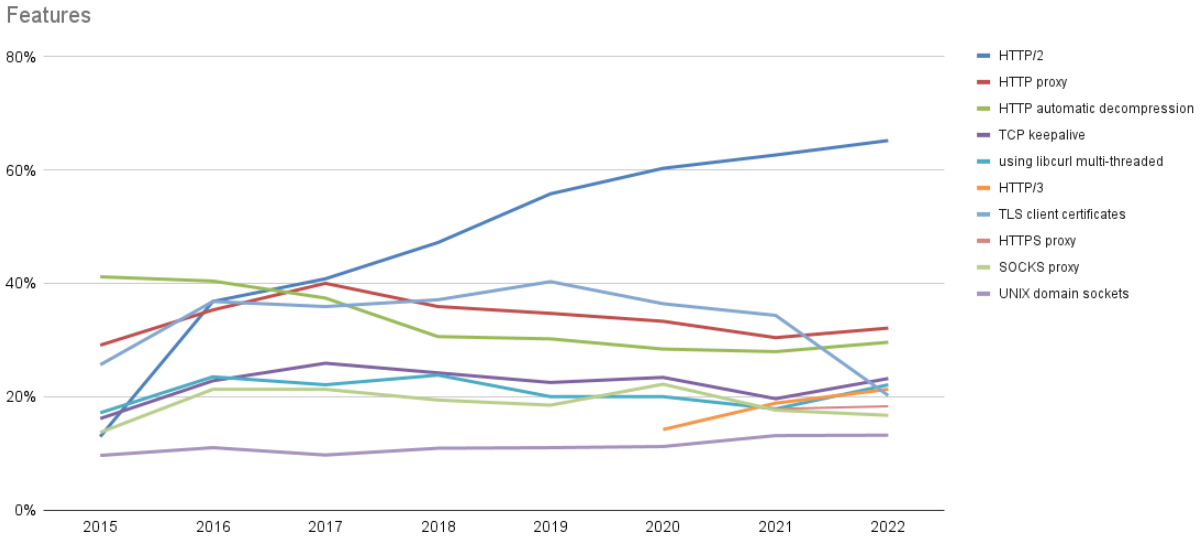
Features

What do people do with curl? We can see some interesting trends here that continue this year.

HTTP/2 use continues to grow. 65.2% say they use it. HTTP/3 use also increased a lot up to 21.3% (from 18.9% last year), which I believe is a significant number since the feature is still experimental in curl and you need to explicitly enable it in the build.

TLS Client certificates are down by a lot to 20,.% from 34.3% last year.

The graph below shows the trend for the features ranked top-10 this year.



Here's the full distribution. On average, users selected 3.47 features each.

HTTP/2	65.2%
HTTP proxy	32.1%
HTTP automatic decompression	29.6%
TCP keepalive	23.2%
using libcurl multi-threaded	22.1%
HTTP/3	21.3%
TLS client certificates	20.2%
HTTPS proxy	18.3%
SOCKS proxy	16.7%
UNIX domain sockets	13.2%

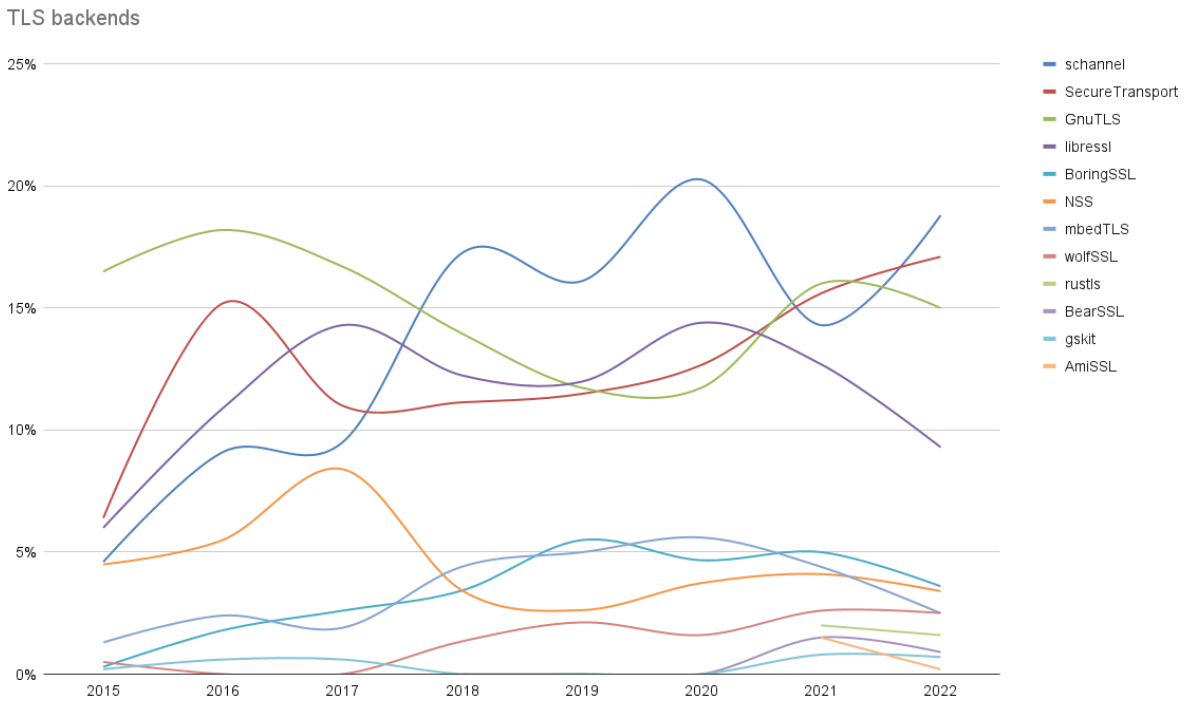
Bandwidth rate limiting	12.4%
HSTS	11.3%
DNS-over-HTTPS (DoH)	11.1%
curl_multi_socket API	10.8%
.netrc	10.0%
HTTP/0.9	8.6%
CURLOPT_FAILONERROR	6.5%
NTLM auth	5.9%
the share interface	5.7%
Alt-svc	3.2%

TLS backends

At the time of the survey, curl supported 13 different TLS backends, down from 14 last year (MesaLink support is gone). This year 25.1% said they did not know their TLS backend, and I think in general it might be a good thing that not all users know this, as it is not supposed to be an important factor for users.

OpenSSL remains the undisputed king of TLS backends for curl users at 98.1% of the ones who did know. It means it continues at roughly the same level it has been at in every curl user survey since 2015. The second most used backend was Schannel at a mere 18.8%.

The real fight the last few years have been between Schannel, Secure Transport and GnuTLS for second place. To better illustrate this, the graph below is made without OpenSSL to allow us to zoom in a little.



The complete distribution in 2022 looks like this:

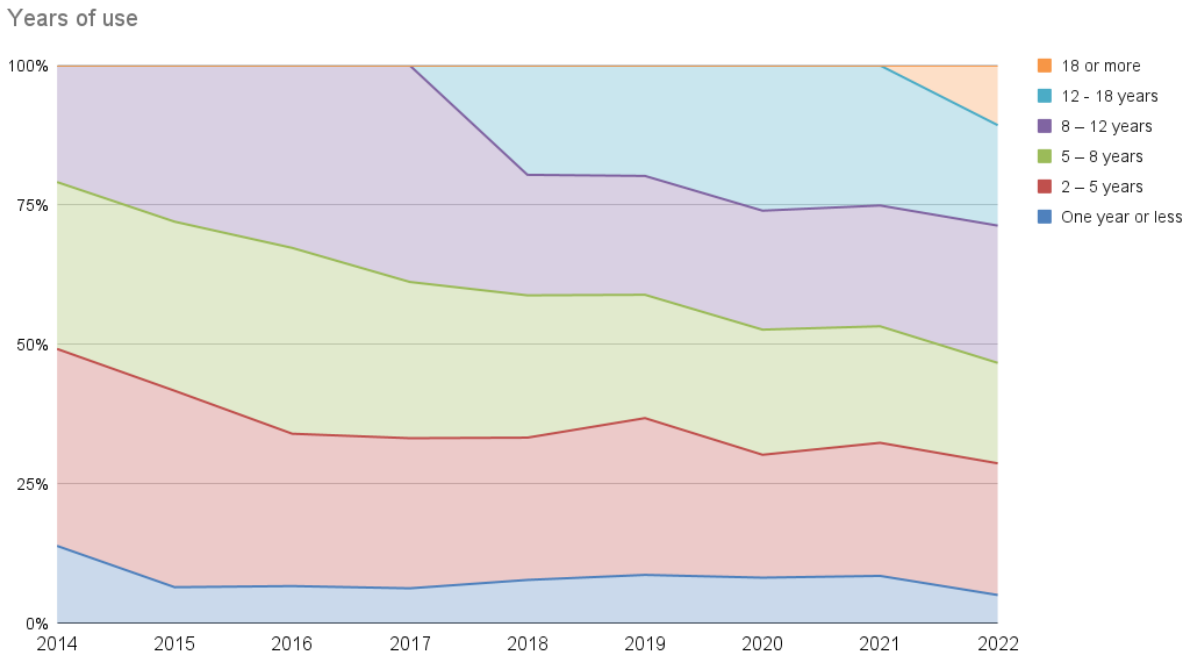
OpenSSL	98.1%
schannel	18.8%
SecureTransport	17.1%

GnuTLS	15.0%
libressl	9.3%
BoringSSL	3.6%
NSS	3.4%
mbedTLS	2.5%
wolfSSL	2.5%
rustls	1.6%
BearSSL	0.9%
gskit	0.7%
AmiSSL	0.2%

Years of curl use

The first curl shipped in the spring 1998 and the general sense I get is that we have happy and loyal users. curl still delivers on its promise. It is a solid and trusted tool.

This question is a (rough) method to see if we manage to gain new users and how well we retain old users. New for this year is that we introduced an answer alternative for "18 years or more" as previously "12 years or more" was the top alternative.

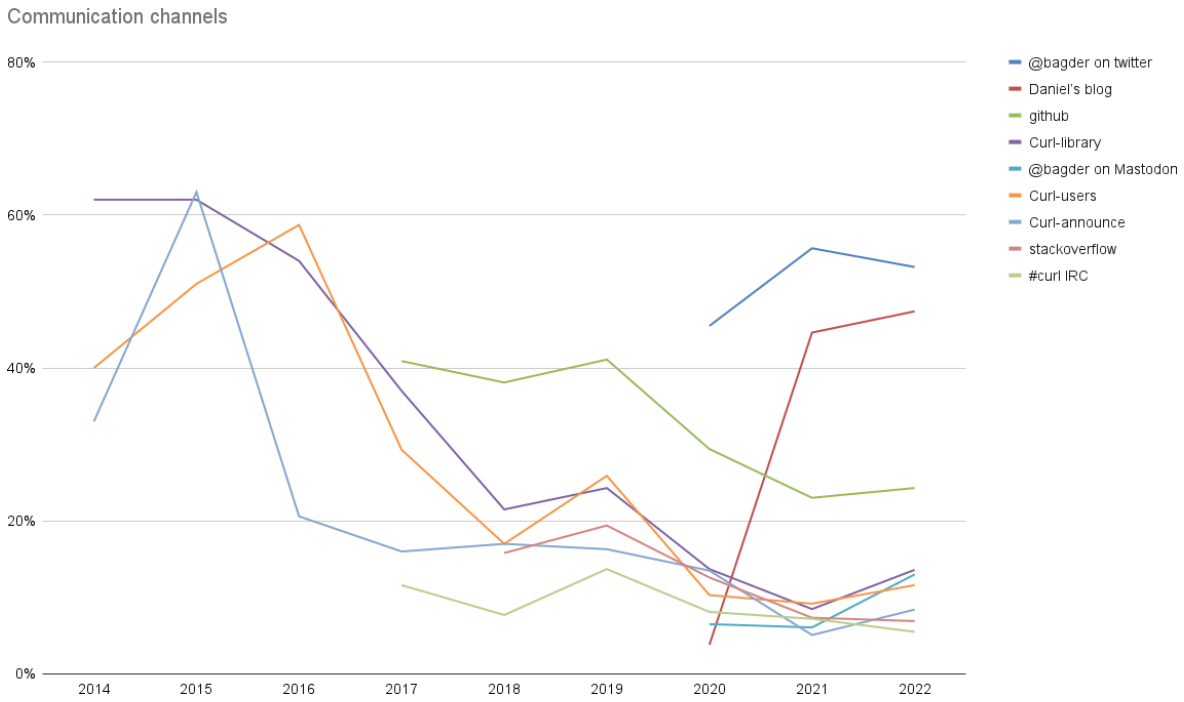


The distribution appears to be safely distributed among users of all ages.

Participating channels

Where do users learn and talk about curl? This year does not seem to reveal anything new but instead establishes the new truths we have learned: the mailing lists are diminishing communication channels among users while my Twitter feed and personal blog have become major ones.

A few people wrote in “Daniel’s youtube channel”, which might be reason enough to offer as an answer next year.



What ways of communication would users like us to use (more?) This was a new question for this year meant as a probe to learn if there is somewhere we need to be or to go in the future for more and better project communication.

My reading of the numbers (presented below), is that we are already covering the desired messaging channels fairly well. While there is no official curl Twitter account, I do a lot of curl related communication and discussion on my personal account @bagder. I think we should keep an eye on how the Slack and Discourse answers develop in the future.

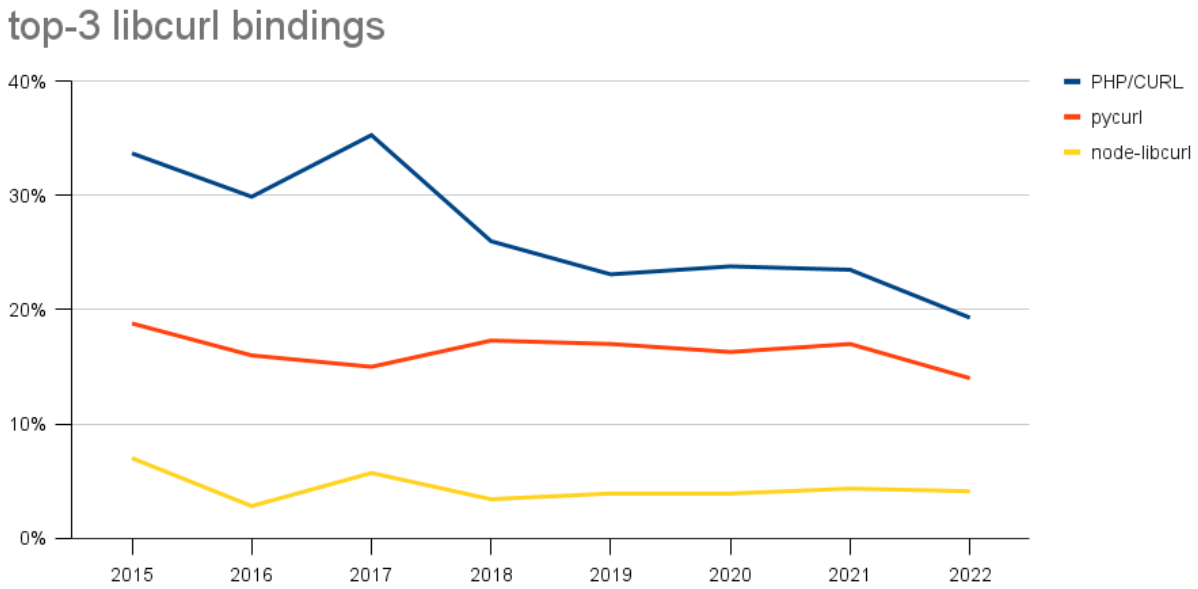
Twitter	35.70%

Mailing lists	30.50%
GitHub discussions	29%
Mastodon	19.70%
IRC	17.10%
Slack	7.10%
Discourse	5.20%
LinkedIn	3.70%
Facebook	3%
Video meetings	2.20%
Matrix	2.20%
RSS	1.50%
Discord	1.40%
Telegram	1.10%
XMPP	0.70%

Accessing libcurl

libcurl is the network transfer engine of the command line tool and is commonly accessed by users via bindings. The bindings are what makes libcurl truly accessible to almost all developers everywhere. Which ones do people use?

78.5% answered the command line tool curl, and the average response had 1.86 answers selected. 32.2% selected the native C API. Only two actual bindings got more than 10% of the answers. The graph below shows the top-3 binding's development since 2015. The PHP binding's share seems to slowly shrink over time.



Here's the complete binding distribution for 2022:

curl	78.5%
plain C	32.2%
PHP/CURL	19.3%
pycurl	14%
Node-libcurl	4.1%
curlpp	3.9%
Go-curl	3.9%
.NET core	3.9%
Rust-curl	3.6%
www::curl (perl)	3.4%
Ruby	3.4%

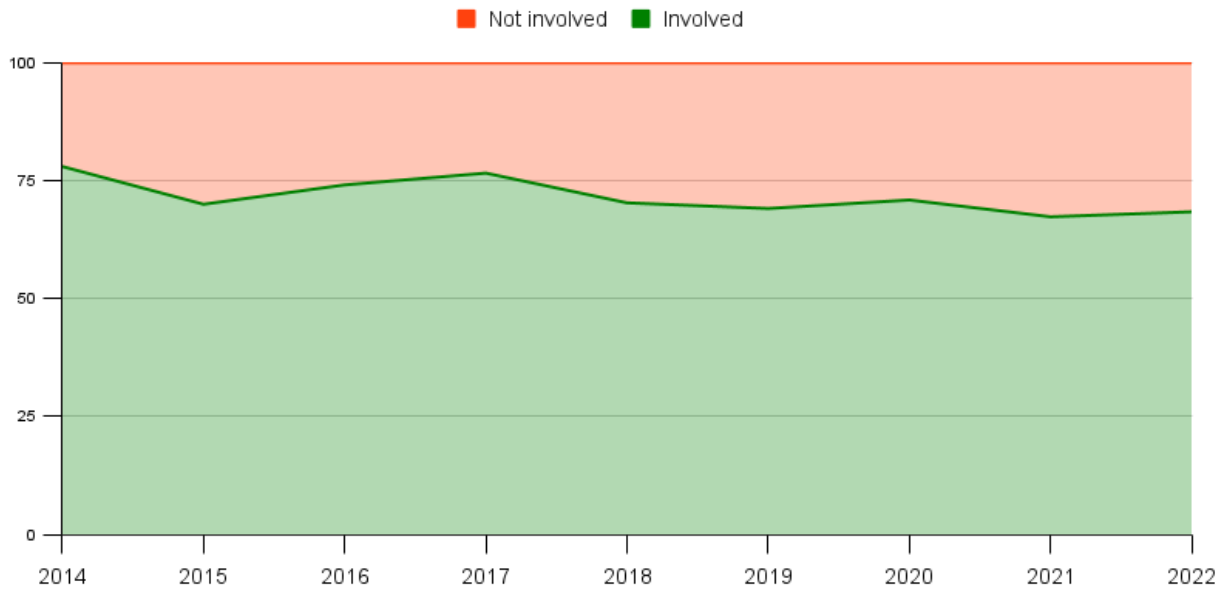
Java	2.7%
Lua	2.4%
R curl	1.7%
Common Lisp	1.2%
Tclcurl	1.2%

Contributions

Users still contribute to a large degree. Over 30% of the answers detailed at least one way they contributed.

I haven't contributed yet	68.9%
filed bug reports	13.8%
sent pull requests	12.3%
I can't remember	2.7%
helped out in other ways	5.2%
curl stickers on prominent places	4.9%
responded to mailing lists / forums	6.9%
donated money	4.2%
spend time in the IRC channel	2.2%
run tests or provide infrastructure	1.7%
write documentation	1.7%

And by no surprise at all, curl users are involved in other Open Source projects to a very high degree.



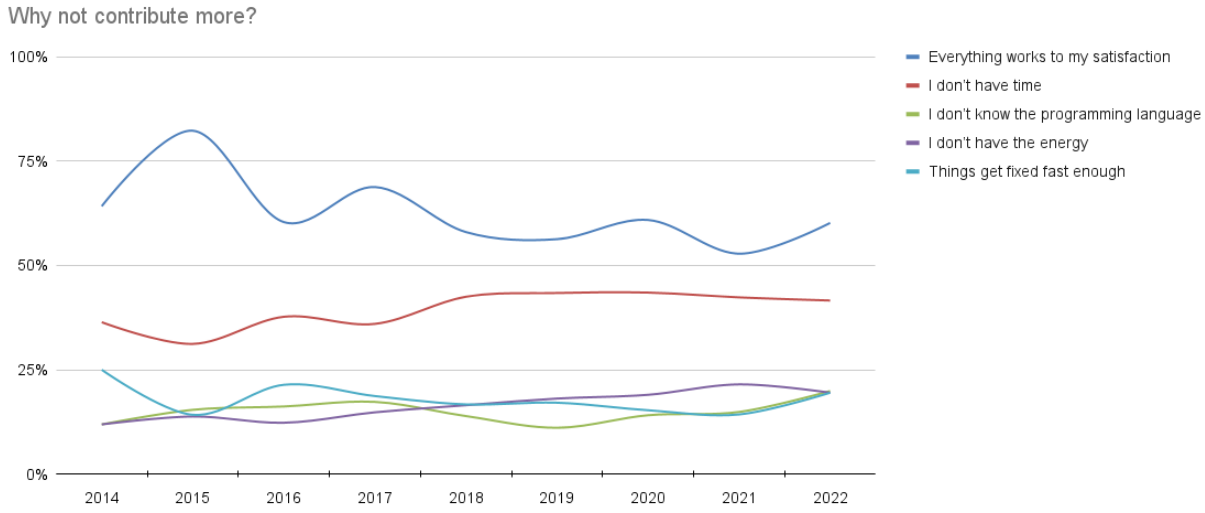
When asked for the reason why they have not contributed or not contributed more, the reasons they state are:

Everything works to my satisfaction	60.2%

I don't have time	41.6%
I don't know the programming language	19.9%
I don't have the energy	19.5%
Things get fixed fast enough	19.5%
Too hard to get started	12.4%
I don't like or approve of GitHub	4.4%
My work/legal reasons prohibit me	3.8%
I can't deal with the tools	3.5%
I don't like or use email	1.8%
The project doesn't want my changes	1.8%
I find it hard to work with the curl developers	1.3%

It is pleasing to see that among the two alternatives that are most negative for the project (highlighted with red in the table) there is a fairly low answer rate.

Looking at the top-5 reasons over the years, they remain rather solid. This could be interpreted that the project keeps performing at a similar level.



This year we also added a free-text form asking "*What could the curl project do/change to get (more) contributions from you?*". 90 respondents took their time to fill in suggestions.

It is hard to summarize 90 separate free form replies, but a few patterns could be spotted:

1. The jokes about adding bugs and creating time machines

2. The people who said there's nothing we need to do
3. Use another programming language
4. Eleven replies were variations of *list issues marked as "good for first time contributors* and other ways to make it easier to find where to contribute.

That last category there seems like the only one that can be acted upon. We have improved documentation over the last few years, including "getting started" entry points, that is meant to help newcomers to the project find where to start. We also list TODO items and known bugs that are ready to get grabbed.

We do however not provide "good first issue" labels or similar on reported issues. The reason for this is simple: all our easy issues are fixed and closed almost instantly. We take great pride in our speed and agility in acting on reported mistakes. In order to provide such beginner labels on issues we would have to deliberately keep issues open (for how long?) and somehow restrict who would be allowed to fix them. I do not see us doing this in the near term.

How good is the project and its members

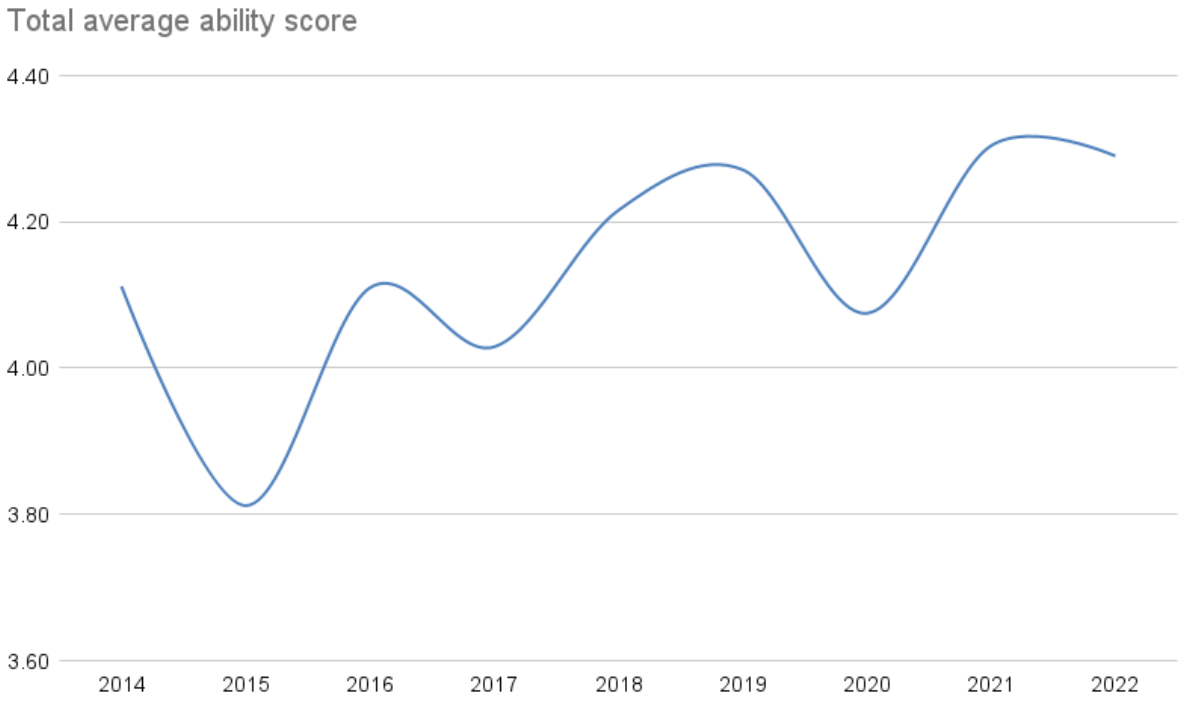
Another question meant to help us probe how we are doing in the project and we get worse or better at specific things over time.

The respondents were asked to rate how good we are at handling things in these seven different areas on a scale from 1 to 5.

1. Handling security-related issues
2. Attribution and giving credits
3. Bug reports
4. Patches and pull-requests
5. Information about what's going on
6. Helping newcomers to the project
7. Female contributors and other minorities

The order of the above areas are also the order of how good the users rank us. The order has remained remarkably similar over the years. I think the bottom two areas are what we should take away from this and dig deep to see how we can improve.

Counting the average score on all 7 areas and plotting that in a graph shows that we only vary very little year-to-year. *Possibly* a miniscule growing trend can be detected even if we went down from 4.30 last year to 4.29 in 2022.

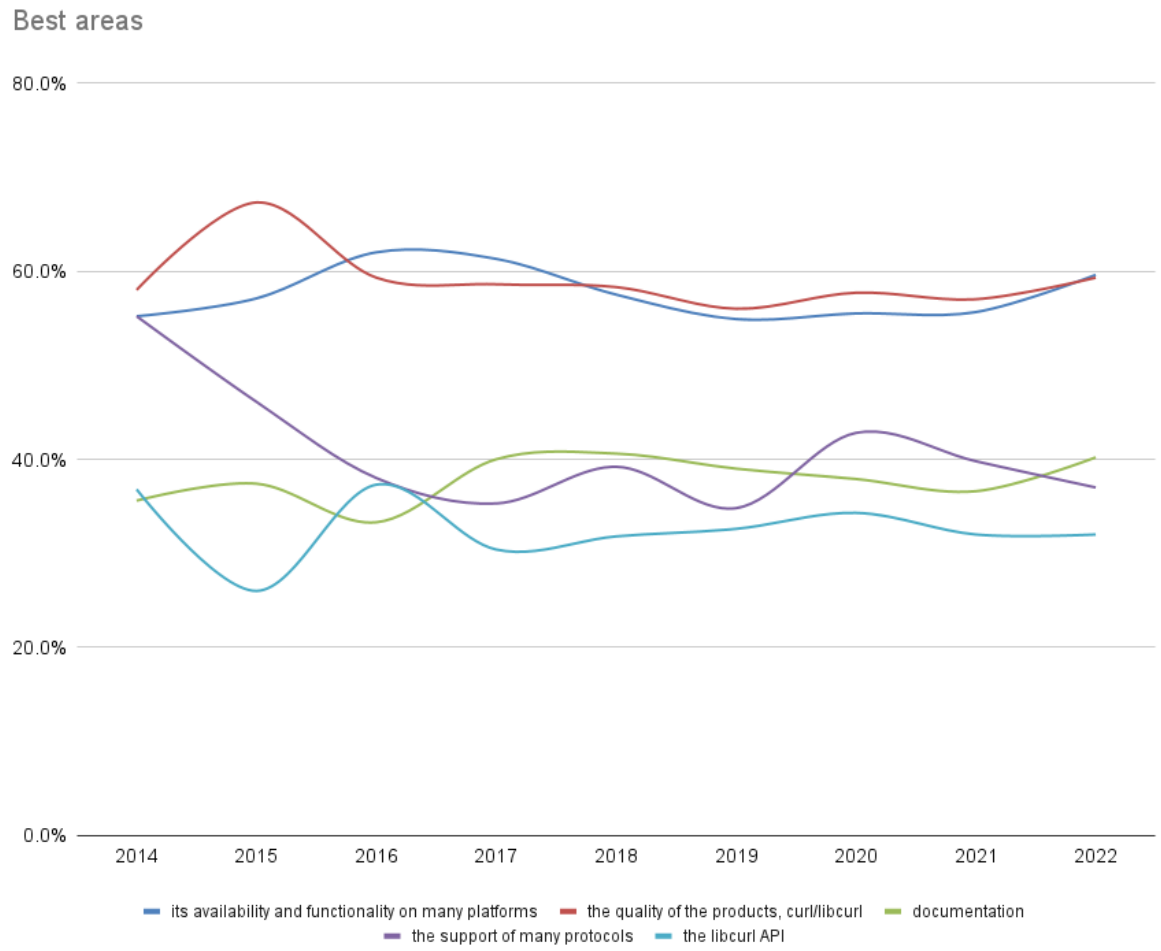


Best/worst areas

Another way to get people to help us identify what we do well and where we need to work harder is to ask them to identify the best and worst areas in the project. The users could select 5 areas, and the exact same areas were listed for both questions. On average, users selected 4.00 best areas and the full ranking is seen in the table below. I highlighted the top-5.

its availability and functionality on many platforms	59.6%
the quality of the products, curl/libcurl	59.3%
documentation	40.2%
the support of many protocols	37.0%
the libcurl API	32.0%
standards compliance	24.3%
the features of the protocol implementations	22.8%
security	22.3%
support of multiple SSL backends	20.6%
project leadership	18.1%
bug fix rate	13.4%
footprint of the library/executable	13.2%
transfer speeds	10.9%
the user and developer community	10.2%
welcoming to new users and contributors	5.7%
its build environment/setup	4.5%
project web site and infrastructure	4.2%
test suite	2.5%

The top-5 best areas has shifted like this over the years since 2014:

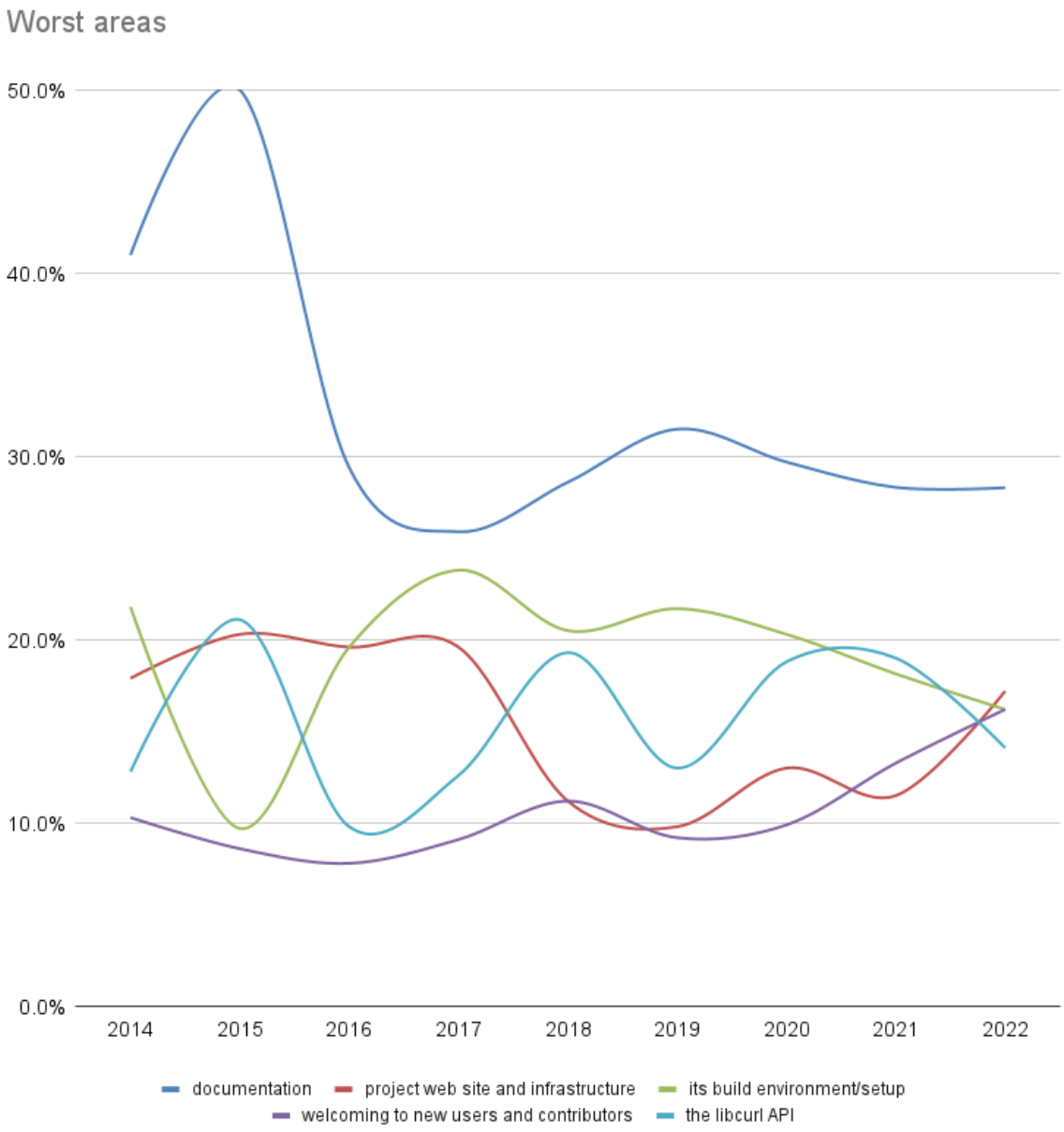


403 persons supplied “best areas” and only 99 filled in “worst areas”. On average, the ones who answered only supplied 1.57 worst areas. The ranking ended up like below. I highlighted the top-5.

documentation	28.3%
project web site and infrastructure	17.2%
its build environment/setup	16.2%
welcoming to new users and contributors	16.2%
the libcurl API	14.1%
the support of many protocols	11.1%
test suite	9.1%
the user and developer community	8.1%
footprint of the library/executable	6.1%
its availability and functionality on many platforms	6.1%
bug fix rate	5.1%

security	4.0%
project leadership	4.0%
transfer speeds	3.0%
the features of the protocol implementations	3.0%
support of multiple SSL backends	2.0%
standards compliance	2.0%
the quality of the products, curl/libcurl	1.0%

The top-5 worst areas have changed like this over the years:

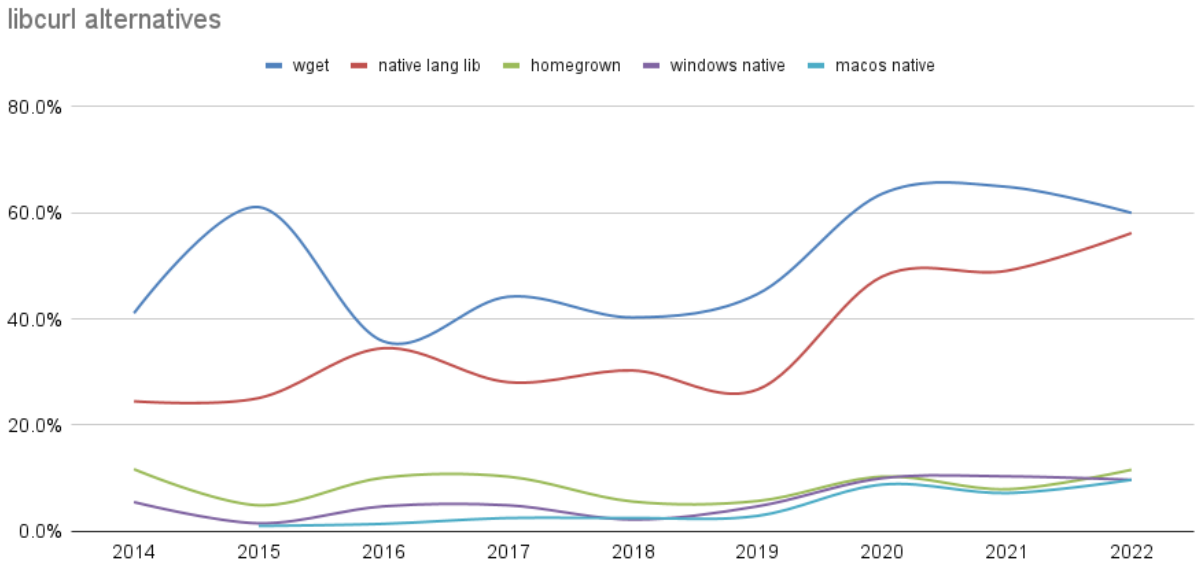


If you couldn't use libcurl, what would be your preferred transfer library alternatives?

It keeps fascinating me that so many people actually select wget *or code ripped out from wget*, as a libcurl alternative to consider. I mean, Wget is a fine product and project, but it is not a library.

wget	60.0%
native lang lib	56.2%
homegrown	11.6%
windows native	9.7%
macos native	9.7%
asio	5.9%
Qt	4.6%
libsoup	3.8%
poco	2.2%
Cpp-netlib	1.6%
neon	1.4%
serf	0.3%

The top-5 libcurl alternatives have changed over time as shown in the graph below. The “native lang lib” option seems to grow slowly.



Which other download utilities do you normally use?

This is only the third year we ask this question but the answer distribution seems to already be established and has not changed much since last year. Clearly most curl users are also familiar with and users of wget, scp/sftp and rsync.

wget	71.3%
scp/sftp	61.0%
rsync	57.0%
nc	26.2%
FileZilla	23.8%
ftp	19.6%
winscp	18.7%
Powershell	13.6%
fetch	11.9%
httpie	11.0%
lynx	9.1%
aria2c	8.6%
wget2	7.2%
lftp	6.5%
w3m	4.4%
Httrack	4.2%
axel	1.9%

Which of these features would you like to see curl support?

It is easy to say *yes please* when asked about a feature, even if you actually never thought about the feature before and eventually will hardly use it. Still, we ask this question as a way to see which things and features that might have the largest user interest and traction.

This question is hard to compare with previous years as we have added and removed options from it over time. Both because of the features that were added but also to a lesser degree because some features have been deemed not suitable.

Here are the results for 2022.

WebSockets	44.60%
JSON	40.30%
GraphQL	19.00%
gRPC	18.20%
DNS-over-TLS	17.00%
DNSSEC (DANE)	13.90%
Gemini	11.90%
Auto-detect proxy	11.10%
WHATWG URL syntax	10.80%
SRV records	10.50%
DNS-over-QUIC	10.20%
SMB v2/v3	10.20%
Thread-safe curl_global_init	9.90%
AIA (dl certs)	9.70%
DNS: URLs	8.00%
OCSP	7.70%
ECH (ex ESNI)	6.80%
dynloadable protocol handlers	6.30%
COAP	3.10%

WebSockets gets over 40% for the 6th year in a row. This fact helped me decide: work on a WebSocket API implementation for curl was recently initiated… Also, the thread-safe init feature has already landed in git.

Curiously, three content related features rated very high as 2, 3 and 4. JSON, GraphQL and gRPC are more or less what usually would be considered users of libcurl, not

features for libcurl itself. I think we should at least think about and perhaps discuss what we can and should do to make it easier to work with those protocols with curl.

Additionally, we provided a free form for users to fill in with the question *If you miss support for something, tell us what!*

Another difficult to sum up question. I filtered off the ones that repeated things we already have or do:

- When I use curl to test the multi-threaded access to HTTPS websites, it takes up a lot of CPU resources, and the error code of TLS handshake failure will appear. But the same test performs better in the winhttp object that comes with windows
- Usenet Protocols, Better Progress bar or more progress bars,
- tutorials
- Tidak
- The protocol I am working with currently plans to either move to COAP, SSE or websockets. So one of those would be nice to be able to work with. It would also be nice to have official meson support, because it would make some things easier for me, but I can understand if that is too much maintenance to add ANOTHER build system. So far I have been really happy with libcurl though! It is super easy to integrate into a different event loop and even interact with C++ coroutines and I don't have to deal with TLS myself (looking at you, beast).
- Specify a download size limit, return error if file exceeds that limit, or the download process would have exceeded that limit when size is not known initially, cut off the download at earliest moment once it is known the file is too large
- SIP
- Recursive HTML loading / spidering a la Wget. It's the only reason I don't use Curl for everything.
- Per transfer selection of TLS backend
- Native official copy API
- Multi-command support for -X, for example in pop3/pop3s: -X "RETR 1;QUIT"
- More MQTT features
- Masquerading as various web browsers in TLS handshakes
- It would be really nice to have a flag that sends basic terminal info in a header, ex. the current size, the NO_COLOR env variable, potentially the TERM env variable though that might be a bit much. This would be helpful for things like `curl https://wttr.in` and `curl "https://shork.easrng.net/$(stty size|tr ' ' _)"`

- I'd be very interested in better embedded support (bare-metal, or at least easier bring-up on new/obscure OSes)
- https by default
- Easier handling with PUT requests. As for now, the only option to create the PUT method is '--upload-file' but sometimes the PUT method don't need a upload file
- Debugging SSL connection errors
- curl --libcurl code generation that is known to be secure, that works in most cases, and that works across older versions of libcurl too. And comprehensive documentation for curl --libcurl that explains what exactly the generated code does (like is it secure by default?).
- "As a monitoring service builder (updown.io) one thing I would be interested in is more details about all requests, especially during redirects. For example timings for all requests instead of the sum, intermediary URLs. I could do that by implementing the redirect myself but then I risk having a much more brittle implementation :)
- Also something else interesting in my case would have been to be able to query without SSL validation (-k) but still know about the SSL error we would have gotten without the flag, get the certificate, etc. I had to implement this by running another separate test with the ruby http client."
- Access to underlying libssh2 errors in libcurl. For example, if a public key file cannot be decrypted, you can only get that information from the debug logs. The error code returned by curl does not help.
- 0-RTT/early-data, aria2-style multi-protocol and simultaneous-connection downloading, toolchain-level hardening measures (CFI, shadow call stacks), support for certificate revocation lists, Oblivious DoH.
- --xattr should set more attributes like creation date, download date, sha256 of the file

Which of these APIs would you use if they existed in libcurl?

The distinction here between the previous question is that this is clearly a libcurl question. About APIs provided to applications using libcurl.

JSON generation/parsing	58.2%
Establishing a websocket connection	45.6%
A read()/write() style API for downloading and uploading	27.0%
Server-side support library for HTTP(S)	24.0%
HTTP Content-Disposition header parser/helper for applications	19.4%
Pluggable async DNS resolver	14.8%
Per-multi bandwidth limitation settings	9.1%

I highlighted the WebSocket alternative at 45.6% here as this is now work in progress. In previous years, “header parsing/extracting” rated very high and as a direct result from that we have since introduced a header API to libcurl.

It is not clear to me exactly what the JSON API would do, but I think it might be worth exploring what we can do for such users seeing so many users believing they would have use for it!

Which question would you like to see in this survey next year?

Some good alternatives among the 20 provided suggestions. Not sure any of them will actually make it:

- Would you like to see any of the defaults changed?
- Which QUIC/h3 backends do you use
- what categories of software do you use that use libcurl
- Which many curl build options can deprecated
- What's your use case for curl? I don't use it to transfer files, but for debugging Http requests/responses, TLS and to explore APIs without writing code. I wonder what other people use curl for?
- Questions regarding features that might be dropped in the near future could provide some insight if somebody still depends on those features.
- Perhaps something along the line of which area/fields they use (lib)curl? E.g. embedded, medical, defense, maritime, aerospace, space, vendoring/including into other OSS projects, ... (multiple choice)
- Is there a long standing bug that you'd like to see fixed? If so, describe.
- How, specifically, would you improve an area you marked as one of the "curl project's worst areas" above? (open ended answer)

Anything else you think we should know?

Very open-ended. We got 79 responses.

Most of them were different ways of saying thank you or repeating suggestions already filled in in other sections. Here are some of the rest that stood out:

- One thing I found a bit tedious recently when I upgraded from 7.50.3 to 7.68.0 is scanning through all the changelog to find what could impact me
- Mandatory github 2FA by the end of 2023 is a no-go condition for me.
- It's hard to stop making features when a project gets really mature, because that's about the only big things that need to be done. Sometimes, I worry that the project might have reached its peak and adding additional features might degrade the quality of the project. I have nothing tangible on curl as features are very much analyzed before being integrated, but it's a pattern I have seen multiple times, especially with open source projects.

- I think -R and --xattr should be the default
- I saw that after some version curl doesn't ship with certs and instead of throwing a ssl certificate error it can throw hey we removed certs from binary go figure out and a link to docs about it
- I feel some answers on the mailing list by Daniel seem overly abrasive to people who maybe don't quite understand or grasp some aspect of what or why Daniel holds strong views on. It is uncomfortable to read some of his responses even when I understand some of his frustration or that the other party isn't necessarily communicating well. I think setting a response aside for a bit and re-reading and editing later to reduce emotion might help, but IDK, e-mail is hard and these forms of communication breed stronger responses than would be given in face-to-face encounters.

Final words

Crunching the numbers, reading the comments, digesting the meaning, filtering the feedback and generating all these graphs in this report takes a lot of time and effort, but I am happy to do it.
This user survey is in many ways the only time during the year that we can get wide and direct feedback from real-world users and I want to make the most out of it.

Enjoy this. I suspect we will do this again next year.

/ Daniel