6-streisand

小组成员及贡献

一项目基本背景和发展历程 张欣然

学号	姓名	贡献
51215903038	张欣然	项目基本背景和发展历程；历史轨迹分析(2，5)；项目归档原因（3.3）
51215903005	俞融	历史轨迹分析（4，6，7，8.1.2, 8.1.3, 8.2）项目归档原因（3.2）
51215903032	高歌（组长）	历史轨迹分析（1，3，7，8.1.2） 项目归档原因（3.1）

一、项目基本背景和发展历程

1. 项目基本背景

Streisand基于Ansible开发，只需要运行脚本即可在远程服务器完成自动配置，打包等工作。是一个将远程服务器配置成为多种VPN服务并科学上网的工具，提供多种协议作为选择，包括OpenSSH，OpenConnect/Cisco AnyConnect，OpenVPN，Shadowsocks，WireGuard。Streisand原生支持多个VPS供应商，包括亚马逊EC2，微软云服务，DigitalOcean，Google云计算，Linode，Rackspace。Streisand仅支持运行在BSD，Linux或MacOS上，不支持Windows系统。

Streisand部署过程简易，耗时短。部署完毕后易于使用。该项目为每一种工具都提供了文档和详细的描述，便于使用者操作，且工具的种类众多带来了较强的灵活性与为稳定性，若一种连接方式遭到封锁，还有其他方式可供选择。

至今Streisand项目获得了22.8k的Star数量，2k的Fork数量。

2. 版本发布历史

Streisand项目并没有明确的版本发布，在项目归档前不断有PR与Issue的提出。项目的更新多为针对某一种VPN服务进行。值得注意的是，从2020年开始，项目的PR与Issue数量变得较少。

3.主要贡献者的构成

Streisand项目的核心贡献人员有六位，分别是

- Jay Carlson (@nopdotcom)
- Nick Clarke (@nickolasclarke)
- Joshua Lund (@jlund)
- Ali Makki (@alimakki)
- Daniel McCarney (@cpu)
- Corban Raun (@CorbanR)

其中Joshua Lund是Streisand项目仓库最初的持有者。

3.1 Jay Carlson (@nopdotcom)

Jay Carlson并未表明自己的国家与公司。所属组织仅有Streisand Effect，推测为自由开发者。2017-2019年较为活跃，绝大部分贡献都在Streisand项目上。

3.2 Nick Clarke (@nickolasclarke)

Nick Clarke居于Berkeley, California，属于WeaveGrid公司。他为Streisand项目的介绍性文档做出了许多贡献，包括Readme与安装指南。当他发现许多国人使用了Streisand之后，他提出了增添中文版Readme的Issue。

3.3 Joshua Lund (@jlund)

Joshua Lund于2014年建立Streisand仓库。参与了许多与Ansible相关的项目的贡献。2018年之后几乎不再活跃。Joshua Lund也并未表明自己的国家与公司。

3.4 Ali Makki (@alimakki)

Ali Makki居住于Montreal,Canada。并未表明所属公司与组织。2016年开始参与贡献Streisand，2017-2018年在Streisand项目中较为活跃。

3.5 Daniel McCarney (@cpu)

Daniel McCarney居住于the woods, Québec。并未表明所属公司，但是在许多仓库中均高度活跃。主要于2017-2019年期间为Streisand进行贡献。

3.6 Corban Raun (@CorbanR)

Corban Raun工作于Raunco Corporation，并属于Bridge（https://www.getbridge.com）组织。同Joshua Lund一样参与了许多与Ansible相关的项目的贡献。偏好使用ruby语言。

4. CI/CD的使用

Streisand专门设置了CI Testing用于测试Ansible的语法正确与否。完整的脚本将会运行三个yml文件：development-setup.yml，syntax-check.yml，run.yml。

其中syntax-check.yml的代码如下：

```
# Include each playbook in order to check syntax of each playbook
- name: Include amazon.yml
  import_playbook: ../playbooks/amazon.yml

- name: Include azure.yml
  import_playbook: ../playbooks/azure.yml

- name: Include cloud-status.yml
  import_playbook: ../playbooks/cloud-status.yml

- name: Include digitalocean.yml
  import_playbook: ../playbooks/digitalocean.yml

- name: Include existing-server.yml
  import_playbook: ../playbooks/existing-server.yml

- name: Include google.yml
  import_playbook: ../playbooks/google.yml

- name: Include lets-encrypt.yml
  import_playbook: ../playbooks/lets-encrypt.yml

- name: Include linode.yml
  import_playbook: ../playbooks/linode.yml
```

```
- name: Include localhost.yml
  import_playbook: ../playbooks/localhost.yml

- name: Include provider-detect.yml
  import_playbook: ../playbooks/provider-detect.yml

- name: Include python.yml
  import_playbook: ../playbooks/python.yml

- name: Include rackspace.yml
  import_playbook: ../playbooks/rackspace.yml

- name: Include ssh-setup.yml
  import_playbook: ../playbooks/ssh-setup.yml

- name: Include streisand.yml
  import_playbook: ../playbooks/streisand.yml

- name: Include test-client.yml
  import_playbook: ../playbooks/test-client.yml

- name: Include vagrant.yml
  import_playbook: ../playbooks/vagrant.yml

- name: Include run.yml
  import_playbook: run.yml

# Explicity include each role to ensure all roles are tested
- hosts: localhost
  remote_user: root
  roles:
    - azure-security-group
    - certificates
    - common
    - diagnostics
    - dnsmasq
    - download-and-verify
    - ec2-security-group
    - gce-network
    - genesis-amazon
    - genesis-azure
    - genesis-digitalocean
    - genesis-google
    - genesis-linode
    - genesis-rackspace
    - i18n-docs
    - ip-forwarding
    - lets-encrypt
    - nginx
    - openconnect
    - openvpn
    - service-net
    - shadowsocks
```

```
    - ssh
    - ssh-forward
    - sslh
    - streisand-gateway
    - streisand-mirror
    - stunnel
    - sysctl
    - test-client
    - tinyproxy
    - tor-bridge
    - ufw
    - validation
    - wireguard
```

完整的测试需要在Github Runners上进行，但该测试框架也可以在本地运行。进行测试时需要对Streisand提供的每一部分功能都进行测试，以保证所有服务都可以正常运行。

5. 其余信息

从几位核心开发者的贡献榜中可以看出，在2018年之后，核心开发者不再专注于该项目。而核心开发者中的自由开发者在2018，2019年之后甚至鲜少在Github上进行活动。

补充：

该项目在整个过程中出现过名字的变更，从`jlund/streisand` 到`StreisandEffect/`，再到`StreisandEffect/streisand` 。

二. 项目历史轨迹分析

2.1 数据处理

Q1-Q7部分见.ipynb文件

2.2 原因分析

Q8:根据你观察到的仓库的历史数据，找几个你认为关键或值得注意的时间节点

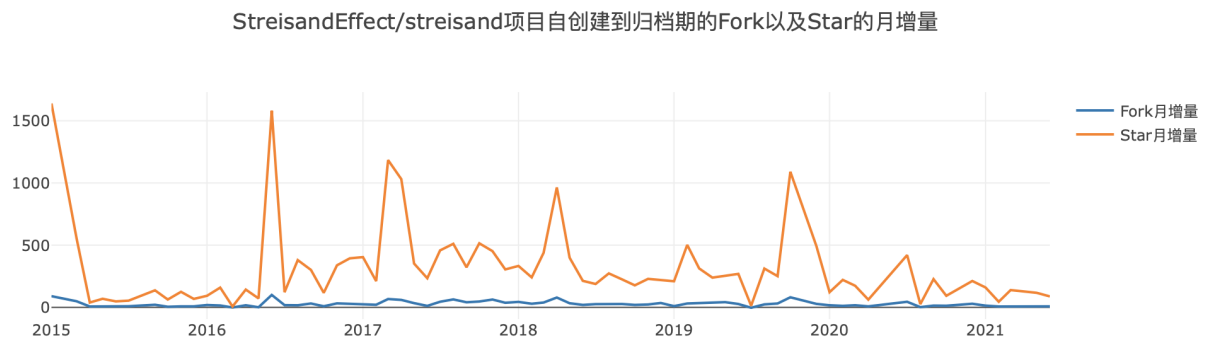
将结合二中历史轨迹数据及其分析结果，从数量、频度峰值以及事件时间段两个角度讨论其特殊的时间节点。

2.2.1 峰值，转折点

说明：由于StreisandEffect/streisand项目起始时间在2014年，归档于2021年，而数据库中可使用的是2015~2022年的8张数据表，受限于数据的不完整性，我们的分析需要过滤不完整部分的影响，只考虑从2015年到2021年。

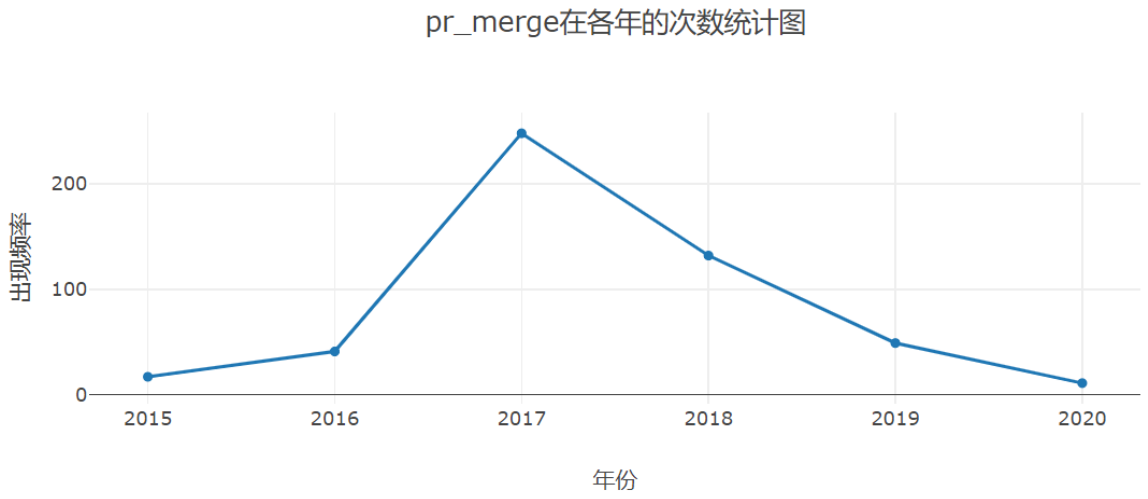
(1) star vs. fork 项目活跃度

star和fork的新增曲线变化（涨幅情况）一致，高峰期每年有一到两个，集中在2016.6, 2017.3, 2018.4,2019.2(小峰值)，2019.10,2020.7

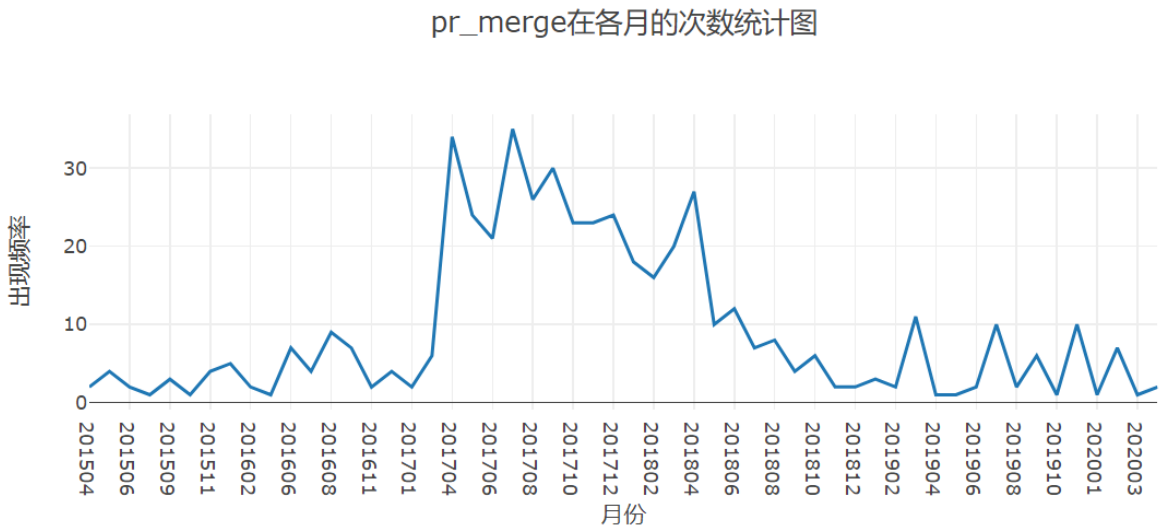


(2) issue vs PR 用户高峰期

- 月打开issue和关闭issue数量
- PR合并峰值和活跃期



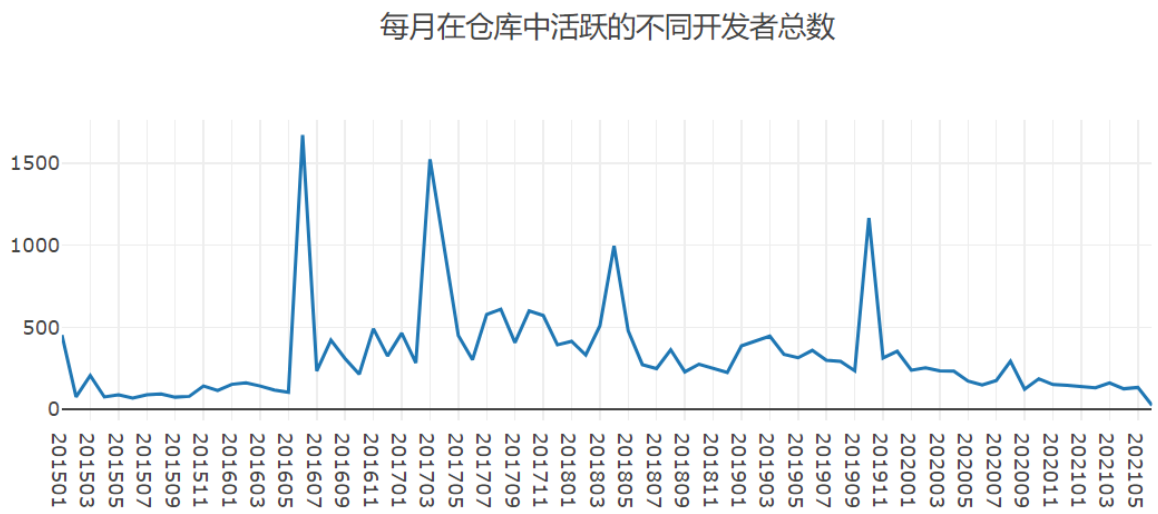
对于github来说，当创建一个PR时，会自动为其创建一个issue，因此PR打开时间即为issue创建时间。从各年份pr合入频数统计图中，我们开源发现，StreisandEffect/streisand项目的频数分布在整个生命周期中呈现一个峰值分布。以年为单位来看，参与到项目贡献的pr成功合并数在2017最多，2017-2018年请求合并成功的热度较高。



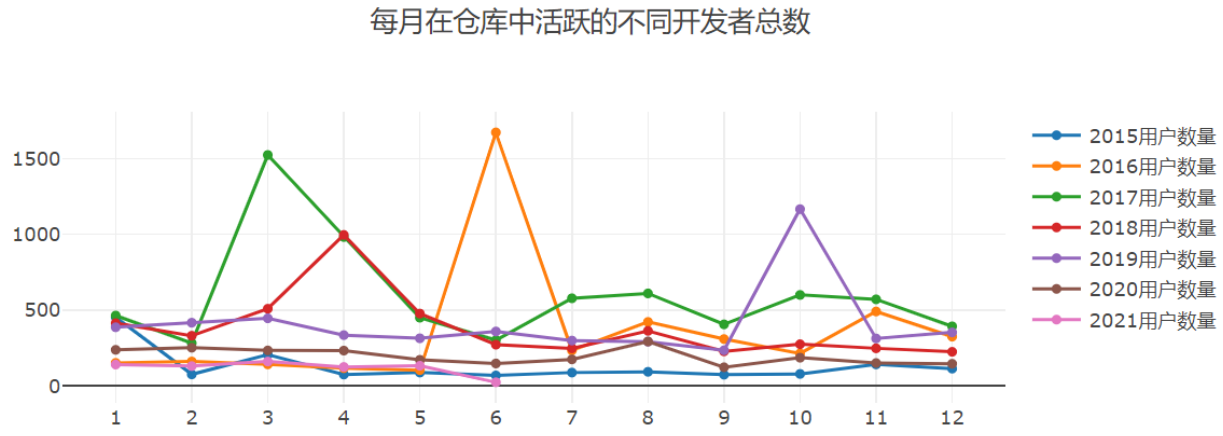
经过对不同年份PR合并数量的讨论分析后，我们又以月为单位发现，2017年4月是合并量开始激增的转折点，由此至2018年4月期间，整体处在高位波动，且存在2017.6和2018.2这两个局部极小值，事件时间上与高校学生寒暑假时间段重合，推测是受此影响。从2018.12开始处在低位波动。

（3）开发活跃度

- 月活跃开发者



从对各月活跃开发者的分析中发现，被仓库日志记录到的用户存在四个峰值，2016.6、2017.4、2018.5和2019.10。



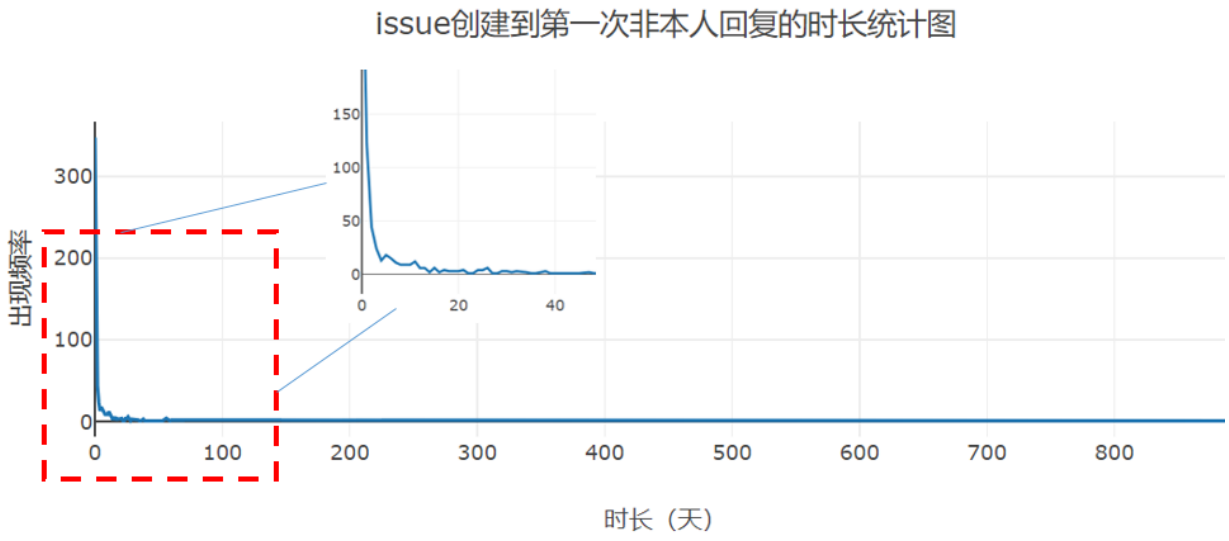
从上图中对每年12个月的活跃用户研究中，发现在3、6、10月会有小高峰，值得开发者分配更多的时间关注用户体验。

- 提交 高峰期

可以看出来，2017年中到2018年中是一个提交高峰期，和PR合入的高峰时间段基本吻合。可以确定这段时间是主要开发期。

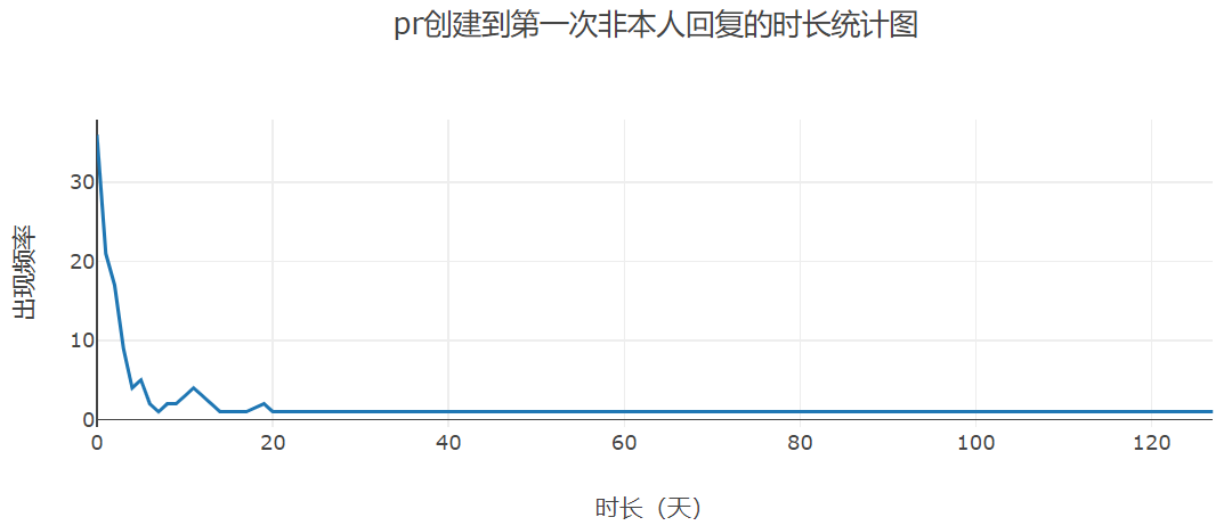
2.2.2 特殊时间段

- PR 在17-18年 处于活跃期
- issue 在 17-18年处于活跃期
- issue打开到关闭的时间段
- issue 打开到第一次回复的时间段



上图是Issue被提出到有非本人回复的时长统计，整体呈现zipfi分布，大部分都是20天以内，时长较短，极少部分是很长时间才被回复的，甚至超过两年，因此，在issue时长的统计上，最大是895天，仅出现1次，中位数为1，平均数为20，所以使用平均数是没有意义的，中位数更有代表性。

- PR打开到第一次回复的时间段

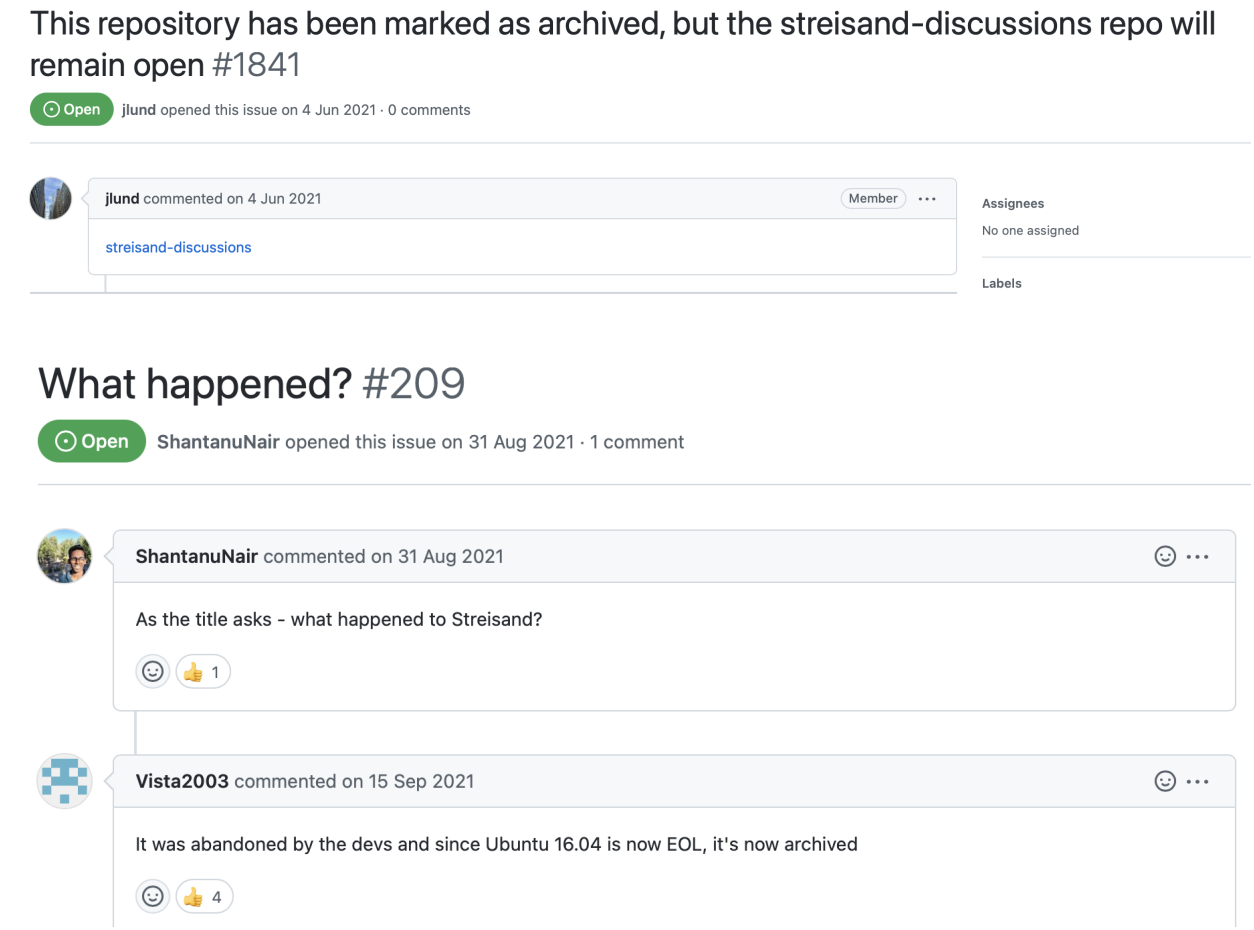


PR的创建到第一次非本人回复，这个作为issue创建到第一次非本人回复的子集，相对效果更加明显，最大时长是127天，大多数在0~20天以内，平均数是7.5，中位数是2。

三. 洞察项目被归档的可能原因

3.1 结合一和二中得到的信息和分析结果，尝试总结项目可能的归档原因

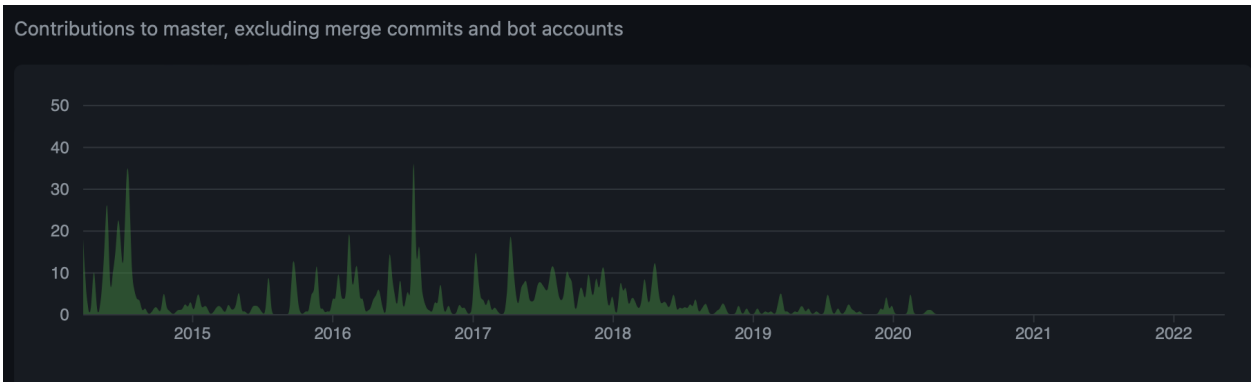
3.1.1 Issue中获取归档原因



从上图可以看出该项目在Ubuntu在16.04EOL之后该项目不再开发维护，应该是归档的主要原因。

3.1.2 根据日志数据分析归档原因

贡献曲线图



从贡献曲线图以及项目历史轨迹分析中可以看出，无论是从项目活跃度，开发高峰期，还是用户使用量来看，17-18年是一个高峰，18年中过后热度逐渐减退，不仅参与开发的贡献者数量减少而且代码维护量也减少。项目关注度减退，核心成员开发重心转移（少数几个不再活跃在开源社区）可能是一个主要原因。

3.2 结合你搜集到的信息，尝试分析项目归档后可能产生的影响（对开发者和用户）

- 对用户的影响

  1.用户不能通过这个项目软件的帮助去访问被封锁的网站

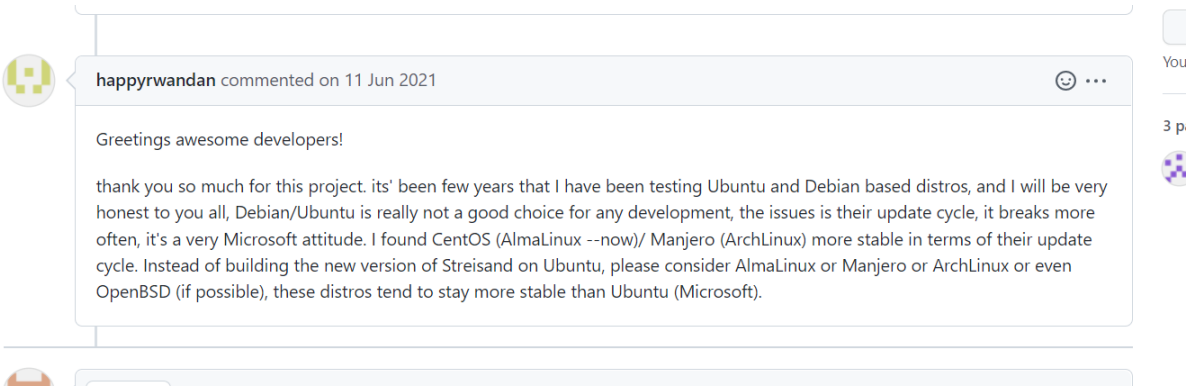
  由于该项目的归档，开发者不再对其进行维护，由于Ubuntu16.04项目中止带来的次生影响，用户想要科学上网，访问被封锁网站的渠道被阻塞，对用户来说最直接的就是不能访问这些网站了。

  2.用户的需求需要通过换成其他同类项目来替代

  在对归档后讨论区的研究中，有些用户提供https://github.com/trailofbits/algo等作为同类替代品

- 对开发者的影响

  1.对于想要做贡献的用户开发者来说，等待技术的迭代来提供新的尝试复活项目机会。原开发者虽然没有再维护该项目，但是讨论区的保留对于新的开发者来说是福音。他们开源等待其他技术软件的更新迭代，在更新的ubuntu版本上或者是新的工具上继续进行着尝试，比如有一条讨论关于ubuntu不稳定、更新缓慢，尝试转移新平台的建议。



2.原开发团队不再维护，缺少代码合并时的自动化验证，代码正确性缺乏评判。

3.3 表述你对开源项目如何可持续发展的理解

开源项目的可持续性可以被初步理解为项目能够支撑自身的发展，结合实际指项目能够得到及时的更新与各类问题的修复，项目本身能够承担项目的支出。

从项目自身开发者的角度出发，应当在开发的初期就设立好可持续性发展计划。由于开源这个特性，不论是公司开发团队还是自由开发团队，只靠开发团队承担所有的项目支出都是有难度的。因此开发者需要在开发时注意如下几点：

1.明确项目的发展方向，即项目想要达到什么样的目的。这需要项目开发者在项目领域的深刻了解，具有一定的预见性。

2.确立各项规范，建立良好项目基础以便利使用者进行贡献。尽可能保证代码规范，使用各类分析工具，撰写完善的文档，确定协议等等。

3.开源项目是面向国际的，项目开发者应当考虑各地用户是否能够便利地接触该项目。

构建有机的，可持续发展的开源社区也是开源项目实现可持续发展的重要因素之一。良好的开源社区应当设立一套完善的机制，如新人的晋升机制，项目信息的公开机制，便利的沟通机制。良好的开源社区能够吸引优质开发者的持续贡献，促进项目的传播，增进项目的生命力。

开源并不等于免费，只凭开发者的无私贡献使项目持续下去是不现实的。开源需要一定程度上的商业化。国外已经有许多较为成熟的开源项目商业化的先例。开源项目可以寻求运营模式的转变，以Docker为例，Docker分为三个版本，开源版本供使用者免费使用，而

针对商业化的版本闭源需要付费才可使用。Red Hat通过向使用他们产品的用户提供有偿服务进行收费。设立基金会对开源项目进行支持也是被证明富有成效的手段之一。

实现开源项目的可持续发展或许会是一个漫长的过程，需要对开源观念的改变和包括项目开发者，使用者，企业乃至政府的努力。