Donut 项目结题报告

[一、成员介绍 1](#_Toc518559548)

[1. 小组成员 1](#_Toc518559549)

[2. 分工 1](#_Toc518559550)

[二、功能简介 1](#_Toc518559551)

[1. 概述 1](#_Toc518559552)

[2. 整体架构 2](#_Toc518559553)

[3. 工作流程 2](#_Toc518559554)

[4. 本地处理 3](#_Toc518559555)

[三、具体实现 3](#_Toc518559556)

[1. 入网 3](#_Toc518559557)

[2. 网络更新 5](#_Toc518559558)

[3．ID表块的管理 5](#_Toc518559559)

[4．下载服务ID表冗余机制 6](#_Toc518559560)

[5．从系统中重构软件包与切块 7](#_Toc518559561)

[6．从数据块恢复出deb包 10](#_Toc518559562)

[7．Donut的使用 11](#_Toc518559563)

# 一、成员介绍

## **小组成员**

我们组一共有五名队员：柴磊、陈俊羽、归舒睿、顾健鑫、吴豫章。其中柴磊

担任组长。

## **分工**

我们的项目从整体上可以分为本地包处理和网络拓扑构建管理这两部分，前者

由柴磊和顾健鑫负责好，后者由余下的三个人负责。同时PPT的制作主要有柴磊和顾健鑫完成。

# 二、功能简介

## **概述**

Donut是一个轻量的分布式的包分发工具，是一次打破传统的勇敢尝试。它旨在解

决当前集中式的包保管体系所存在的一系列问题：下载速度受限，大量存储，对中心服务器过度依赖等。它借鉴分布式网络的思想，让包分发不再由中心服务器来担任，而是有参与构建Donut网络的用户来共同承担。利用分布式网络的并行下载的有点提高速度，同时探索从用户系统中直接恢复出可用的包的方法以此来解决占用用户存储空间的问题。从目前实现的情况来看，Donut可以解决大规模的同时装机所带来的对单一源进行高负载请求的问题，同时Donut有着高度的可拓展性，它甚至可以传输任何形势的文件。

## **整体架构**

Donut的整体架构是这样的，所有的节点在这个网络中都是平等的，在逻辑上，这

些节点以它进入网络时的被分配的ID作为唯一的标识。ID可以将Donut的结构抽象为一个所有节点按一定规则围成的圈，如图-1所示，我们的项目名称”Donut”也由此得名。



图--1

每个节点享有想其他节点请求下载的权利，同时也有响应其他节点的请求并提供下

载服务的义务。每个节点作为Donut的组成部分，他们之间的距离由其ID的绝对差决定。每个节点存储的额外信息是一部分IP-ID表以及可以提供某个特定包下载服务的ID名单，它存储那个包的ID名单是根据包名字和版本号等计算出来的，当然存在着一个用户不止存储一份名单的情况。

## **工作流程**

整个系统的工作流程是：如果一个用户需要某个包，以gcc-8-base为例，那么会有

下面的一系列行为。至于细节将在下一节中说明。

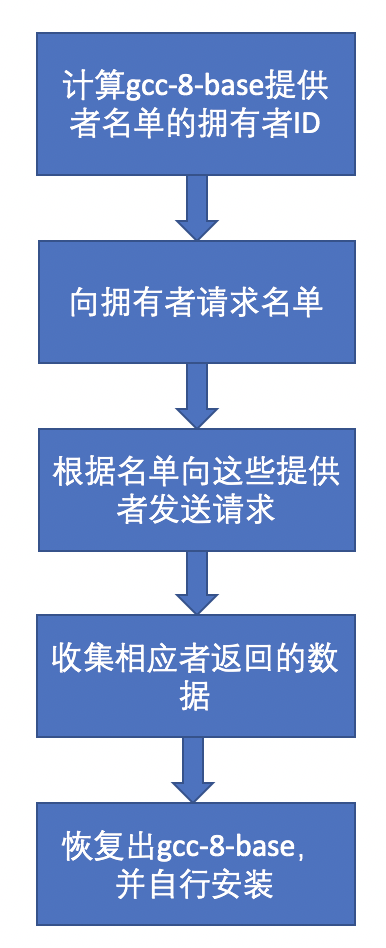


图--2

## **本地处理**

本地处理的主要任务是：

1. 在用户接收到下载请求时，从本地提取出请求者所需要的东西
2. 在用户已经收集到足够多的包块，将这些块整合成可用软件包

**4.1 系统恢复：**

由于时间有限，我们目前只针对deb包实现了从系统中出可用包的工作。

通过调研，我们发现deb包在安装之后，虽然包会被丢弃，但是系统中保留了足够多的信息，我们可以使用dpkg的一系列子命令得到足够的信息，根据这些信息我们就可以恢复出所需的包。

**4.2切块：**

为了发挥饭不是网络并行下载的威力，Donut中的数据在进行一致性压缩之后采用erasure code进行分开，然后不同的用户将特定的包传给同意个用户。

**4.3块验证与解码：**

随着包一起发送给用户的不仅有包切块后的数据，还有整个压缩文件以及块的

哈希。利用群体决策之后，选择出符合标准的块之后，再解码就可以恢复出包，在本地进行打包操作之后，就可以安装了。

# 三、具体实现

## **入网**

* 1. **执行命令：**

donut init

* 1. **入网的详细步骤为：**

a. 节点从DNS服务器获取随机ID，并向服务器请求下载ID-IP映射表。

b. 节点自我筛选需要的ID保留。

c. 遍历系统中已有的软件包，生成相应的ID表。

d. 根据生成的ID表，通知相应的节点更新自己维护的ID表。

e. 保留自己筛选的ID-IP映射，删除其他表。

f. 开启常驻服务，响应其他节点的更新、下载请求。

* 1. **DNS服务器：**

其功能是为新节点加入网络提供必要的信息，并且会随着节点的下线而自动更

新。它将ID—IP对作为一个字典类型存储在本地，使用SimpleXMLRPCServer库中的远程调用，在此基础上DNS\_server的操作有：

a. ADD: 增加一对ID—IP关系，发生冲突时报错。

b. DELE: 根据ID删除一对ID—IP关系，在ID不存在时报错。

c. CREATE\_ID: 在新节点加入时，为其创建一个ID，并调用ADD方法。

d. DOWNLOAD\_ALL: 收到下载请求时，返回其存储的所有ID—IP关系表。

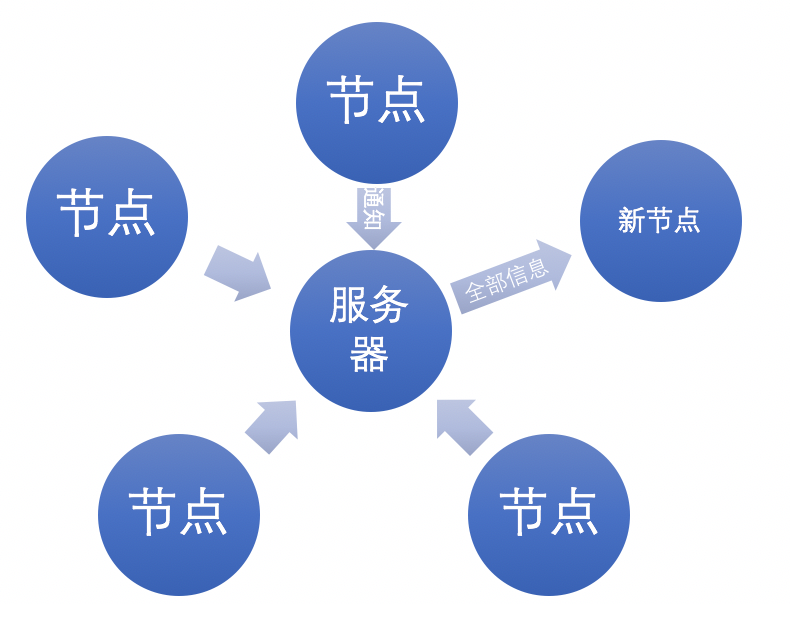
****

图--3

* 1. **节点的筛选**

当一个新加入的节点得到所有节点的ID—IP映射表之后，将距离(抽象概念，指ID之差的绝对值，设L是节点的总个数)恰为的节点作为理想节点，对每个理想节点，取离其最近且实际存在的节点**，**对其进行ping操作。若其不在线，则通知DNS服务器更新ID—IP表，相应节点更新ID表。若在线就将其作为最优节点将其ID—IP映射保存在本地。而其他的信息就可以在init动作成之后抛弃掉。

## **网络更新**

* 1. **节点上线**

一个节点上线就当作新节点加入处理，具体操作参看“1. 入网”

* 1. **节点下线**

节点下线的信息在Donut中也是被动触发的，一个节点被发现下载的情况有：

1. 新节点加入时，删选最优节点的过程中，参见“1.4 ID表的筛选”
2. 节点在请求下载时，参见“4. 请求下载过程”
3. 近处节点进行长连接时，发生异常，参见“2.3 下载服务ID表冗余机制”

对于a情况，发现下线情况时，只需通知DNS服务器更新其ID—IP映射表即

可。

对于b情况，发现下线情况时，需要通知“主”节点更新其下载服务ID表，并且通知DNS服务器更新其ID—IP映射表。

对于c情况，如果长连接异常，认为该异常节点下线，更新自己的联络ID表，并通知DNS服务器更新其ID—IP映射表。

## **3．ID表块的管理**

**3.1 概述**

一个节点拥有两种ID表块，一是路由ID表，即“1. 入网”中通过筛选得来的ID列表，另一个即下载服务ID表，其是根据包名字以及版本号映射到此节点上，要求该节点存储的为其他需要该包的节点提供查询服务的。ID表的管理提供的方法有：

1. AddMyID: 节点根据自己可以提供下载服务的包计算出哪些节点应该存自己的ID，向这些节点发送添加请求。
2. AddID: 处理AddMyID的请求，将请求方的ID加入到自己维护的下载服务ID表中，如果没有则创建。(服务函数)
3. Create\_IDBlock: 处理AddID中没有对应ID表的情况，其创建的格式为：

{

<Package name and Version>,

<ID list>

}

1. DeleID: 当对应某ID对应的节点下线时，参见“2.2 节点下线”，改节点可能收到发现该节点下线的节点发来的下线通知，在对此通知进行验证(PING)并确认为真时，将此ID中下载服务ID表块中删除。

**3.2 下载服务ID表的存储**

下载服务ID表块的存储方案是：我们根据某个包的名字和版本号进行哈希，将其映射到Donut网络中的某一个ID上，但是这个ID是一个理想值。我们找到离这

这个“理想”ID最近的节点，此节点负责维护此下载服务ID表。我们称之为此软件包的“主”节点。其主要的指责是：a. 响应其他节点的请求，将自己维护的某下载服务ID表块发送给用户；b. 响应并验证节点的上下线通知，更新自己维护的某下载服务ID表。

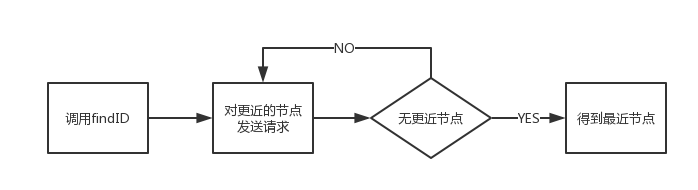
确定“主”节点的规则是：如果一个节点目前存储了某一个下载服务ID表块，如果它的ID恰好就是“理想”ID，那么它就维持现状。否则的，它向比自己离“理想”ID更新的节点发送权利移交请求，该节点变为“主”节点。如果没有离自己更新的节点在线，那就暂时维持现状。

图--4

## **4．下载服务ID表冗余机制**

上节已经说过，下载服务ID表中的内容是可以提供某种软件包下载服务的节点

的ID。也就是说每一个软件包都会对应一个下载服务ID表。也就是说如果一个节点如果需要下载某个软件包，那么它就必须先拿到此软件包对应的下载服务ID表，考虑到分布式网络的动态特定(节点随时都可能下线)，一个下载服务ID表肯定不能只存一份。这里就引出了下载服务ID表的冗余机制。

而且考虑到网络达到一定规模后，每个包对应的下载服务ID表就会很大，所以此表采用分块存储的方式。

* 1. **左右邻居**

在建立路由ID表的基础上，我们在理由ID表上增加离本“主”节点最近的左右两个节点，称之为左右邻居。“主”节点与其左右邻居之间会保持一个长连接。

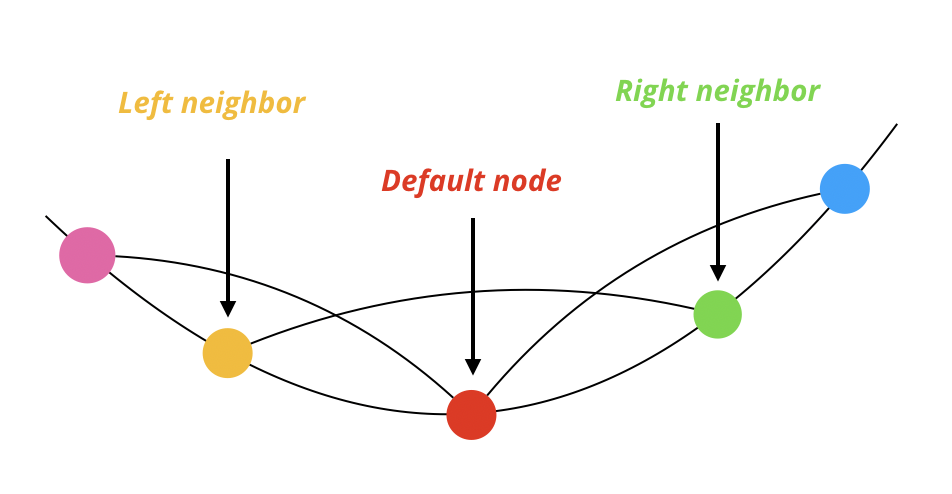


图--5

* 1. **下载服务ID表的备份**

在规定了左右邻居之后，每个节点就会与它的左右邻居进行相互通信，通信的

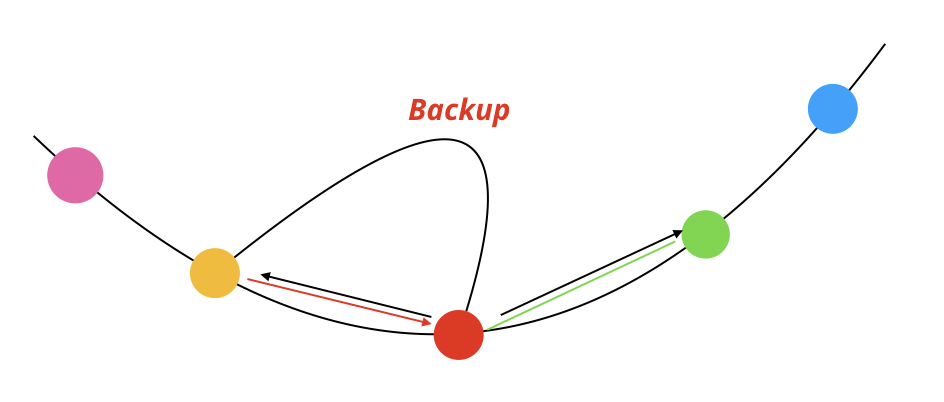
内容为自己目前正在维护的下载服务ID表，如果某节点检查到自己是该节点的邻居，它就会将此表作为备份存储下来，并在接下来的长连接过程中不断根据交流信息进行备份表的更新。

如果某个时刻，邻居节点发现到“主”节点下线，那么网络就会自动将原“主”节点的左邻居作为新的“主”节点，暂时代理服务。此节点同样进行上述的备份机制。

同时代理节点会不断寻找是否有比自己更适合作为“主”节点的节点。如果有就会有权力的移交。其具体原理同“3.2 下载服务ID表的存储”中主节点的选择部分。

具体算法为:

1. 获取自身与自身两侧最近节点的ID。
2. 与左右邻居交换自己维护的下载服务ID表。
3. 验证左右邻居的身份，即是否是某些包的“主”节点。
4. 对左右邻居作为“主”节点的包的下载服务ID表进行备份。
5. 如果发现左右邻居是比自己更优的“主”节点，发生权力移交



图—6

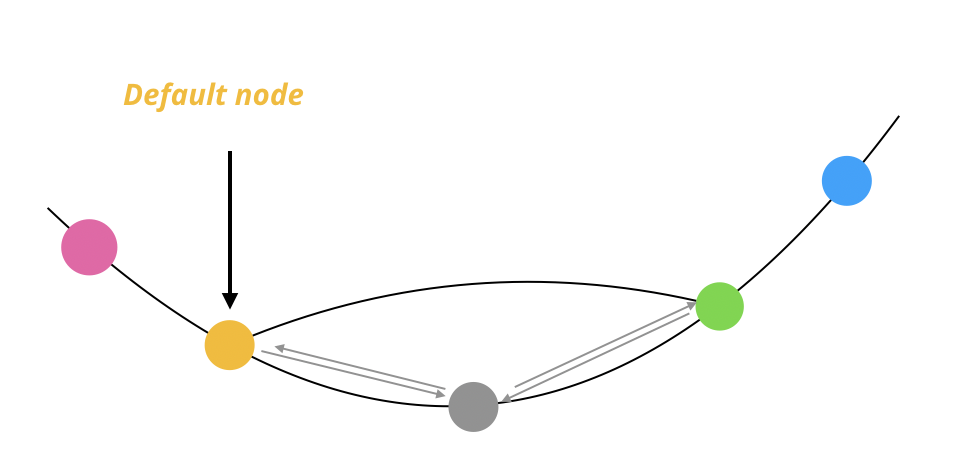


图--7

## **5．从系统中重构软件包与切块**

为了不占用用户的额外内存空间，如果一个用户要下载某软件包，在想可提供此软件包的节点发送下载请求之后，收到请求的节点会从自己的系统中把做需要的软件包恢复出来。具体的操作，以gcc-8-base为例说明。

**5.1 deb包的组成**

可以在/Donut/docs/SurveyReport中看到详细的deb包调查资料。

首先，deb包在解包之后，会包含两部分的内容。usr文件夹下的内容可以直接通过dpkg命令得到；DEBIAN中的内容是一些安装信息和控制脚本，md5信息可以通过计算得到，control可以从系统中读取恢复。

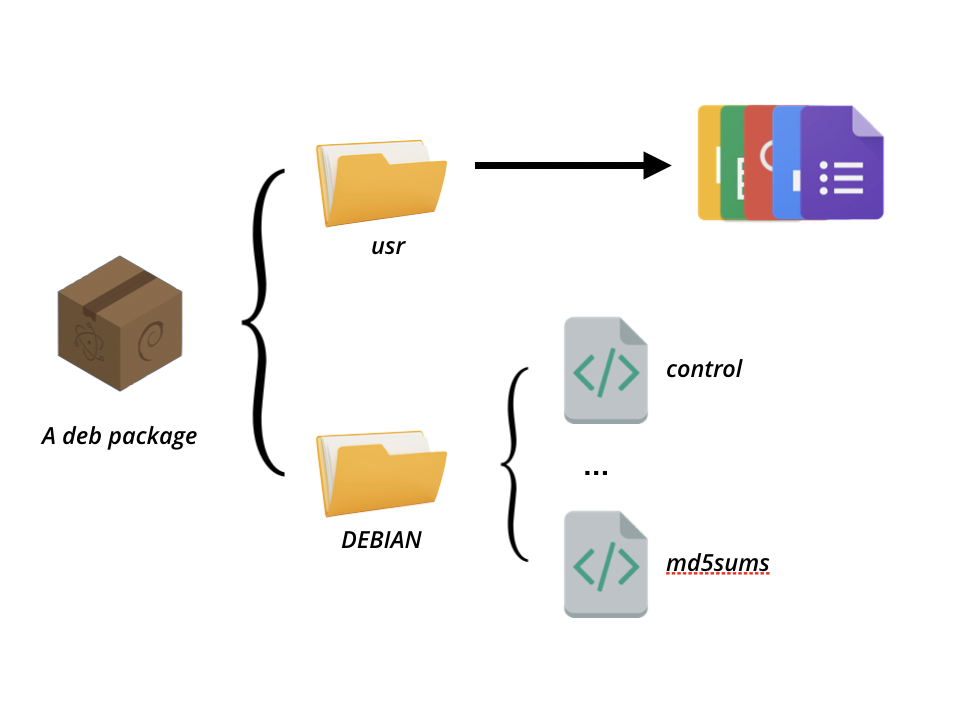


图--8

**5.2 deb包的重构**

由于debian类软件包的构成分为内容和控制文件两部分，因此恢复阶段的工作

便是从本地将这两部分恢复出来。

首先进行的是content部分的处理。调用dpkg-query命令，可以将获取软件包

content部分的信息。但这部分信息有所冗余，从中筛选出不必要的信息后，将剩

余路径中的拷贝复制软件包的信息到暂时的文件夹content下，同时将上级目录命

名为usr。

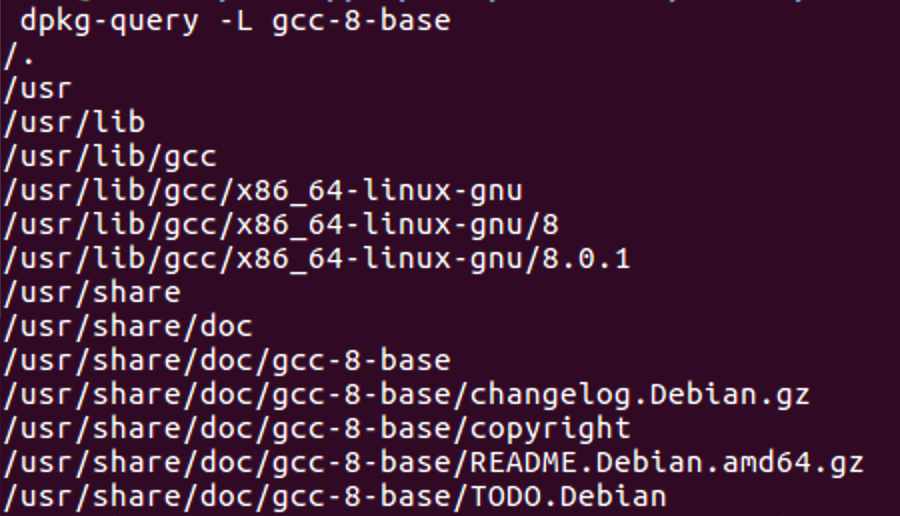


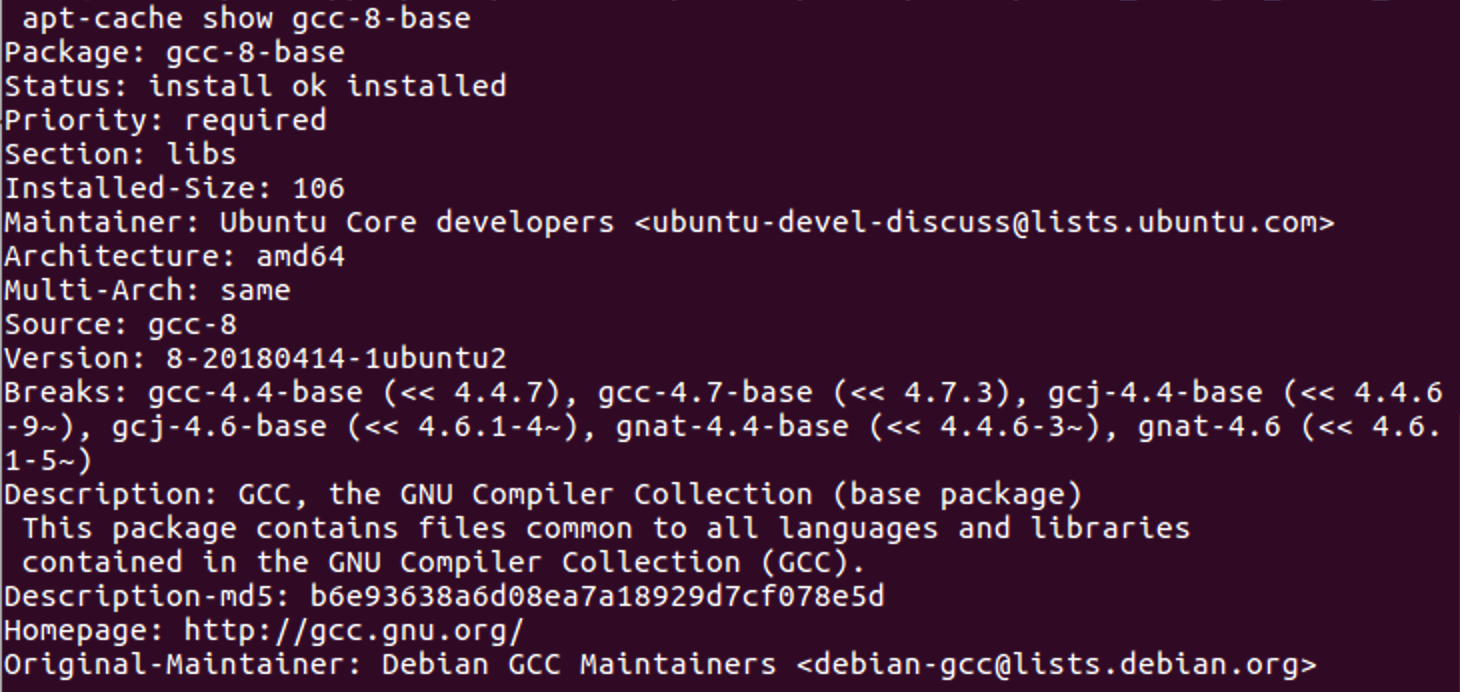
图--9

控制文件包括control和md5sums。control文件可以通过apt-cache show

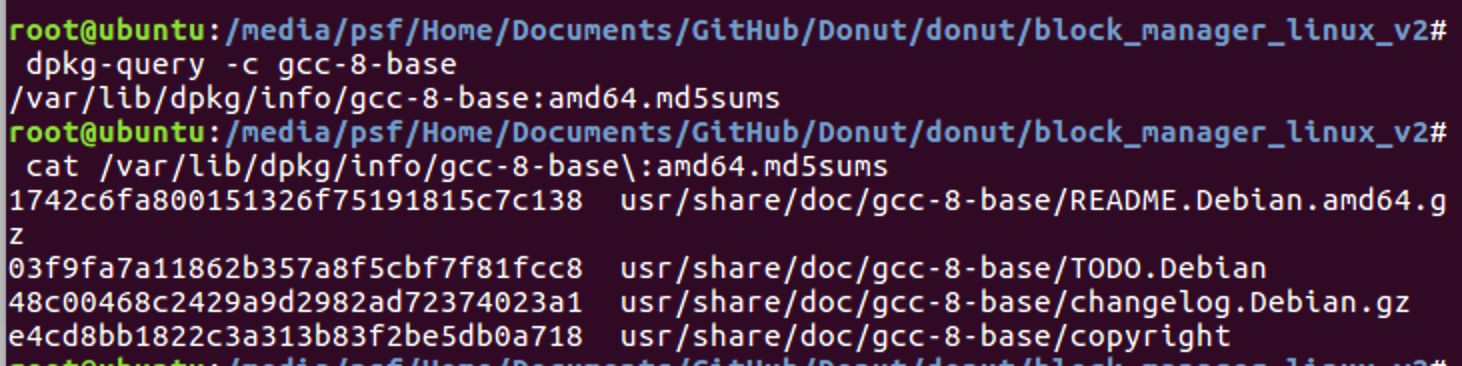
‘package name’来进行获取。而md5sums则可以通过两种方式获取，一是采取本

地生成的方式，而是利用dpkg-query -c来获得原本md5sums的路径，从而获取

md5sums文件。

****

图—10

****

图—11

通过上面的工作，我们就已经获取了重构一个包所需要的全部信息了，当然还

要一些细节，将得到的这些信息进行加工处理之后，就可以构建出所需要的deb包。

**5.3 deb包的一致性压缩**

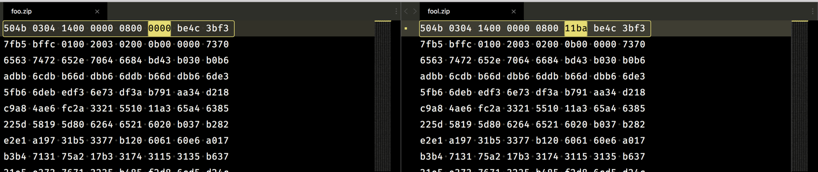
在donut中，我们用到了erasure code的概念，一个用户如果要在donut中

下载某个包，他首先得到的是对这个包进行分块之后的某些块。本节叙述这些块如何得到

为了保证用户从不同节点得到的块可以恢复出完整可用的deb包，就必须保证

在不同的机器上，进行reasure code编码的数据是一致的。经过调研我们发现对上一节中提到的回复出来的文件直接打包成deb包时，由于打包时所在机器，打包时间等很多本地信息都会参与进来，所以如果要让不同用户在不同时间打的包一致，几乎是不可能的。

但是，我们发现如果使用zip对文件进行压缩，压缩时除了文件内容只有文件的创建时间最后修改时间参与到压缩过程中。下图是我们在不同机器上对同一份文件进行压缩之后，得到的压缩文件的内容对比。



图—12

于是我们采用了如下方案：

将恢复出来的文件的修改创建时间统一设置为2018-1-1，这样一来就达到了在不同机器上不同时间点压缩出来的文件是一模一样的。即达到了deb包压缩一致性的要求。

可以在/Donut/docs/SurveyReport中看到详细的zip调研资料。

**5.4 deb压缩文件的分块和验证信息的捆绑**

对上面的deb包进行reasure code分块，目前我们采用的是指定分几块，产生多少校验块的方式，也就是说并没有保证不同的包切块的大小一致性，但此一致性并没什么特别意义，这个日后可以调整。

在对包进行erasure code编码之后，然后我们会计算此压缩文件的md5，以及所有切块的md5，将这些信息和erasure code所需的解码信息一起捆绑到用户要求发送的块上然后将此数据块发送给相关用户。

在工作完成后，将所有这些产生的文件删除

## **6．从数据块恢复出deb包**

**6.1 验证数据块的可信度**

用户接受到最先到达的N(N指恢复出完整信息所需的最小块数)个块之后，就开始解读这几个数据块中的验证信息，于此同时用户仍在接受其他数据块

验证的步骤是：

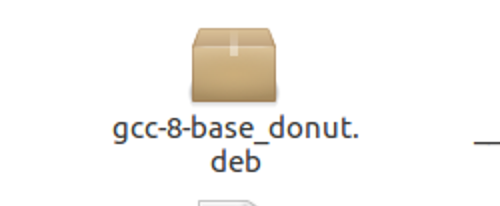
1. 首先通过群体投票的方式确定文件真正的md5信息和所有数据块的md5，详见代码
2. 根据确定的信息，选出认为正确的块

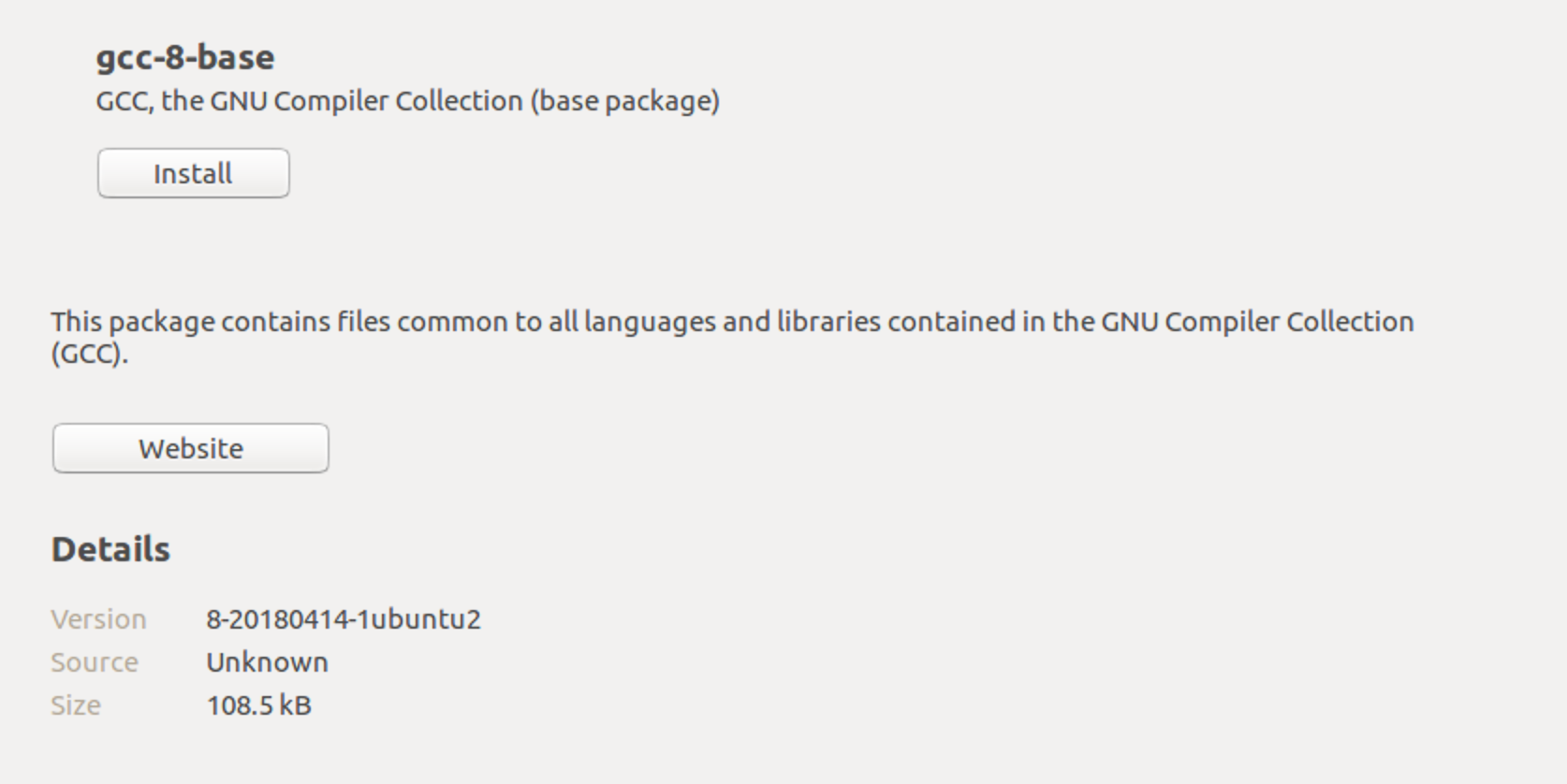
**6.2 数据恢复**

根据选出的块对数据进行恢复，达到原始的包压缩文件，对此包进行md5验证，如果失败，对参与恢复工作的包计算md5，通过对比，找出错误的数据块，如果本地还有多余的数据块未使用，就使用多余的块再次进行恢复操作，否则就重新下载。

**6.3 安装**

上面恢复出来的只是deb包的压缩文件，我们接着对此压缩文件进行解压，再

打包成deb包，使用系统命令进行安装即可。图--13



图—14

从上图可见，此包是可用的。证明了方案的可行性

## **7．Donut的使用**

**7.1 用户使用**

本程序提供命令行运行方式，用户安装之后，

首次运行

donut init

程序会自动在后台完成初始化

需要安装软件时，使用

donut install -p <package name> -v <version>

指定需要安装的软件名和版本号，其中版本号可选

**7.2 项目结构**

Models – 存放常驻服务以及rpc远程服务

Operation – 存放本地节点可以执行的操作

Config—配置文件

Donut – 程序入口

**7.3 项目结构**

[source]块代表源配置

name—源名称

maxNode—最大容纳节点

[general]块代表本地配置

path—本地用来放置下载文件的路径

retry—下载最大重试次数

threads—下载使用线程

**7.4 本地文件管理**

本程序会常驻一个服务用于扫描本地安装文件及其路径，并隔一段时间自动更新文件列表。对安装的软件包，这个服务会调用rpc通知相应的ID表块持有节点令其更新ID表

**7.5 文件说明**

**operation**

server\_task: 启用常驻服务

init\_network: 初始化

download: 下载文件

ask\_root: 请求root权限

**downloadOPs**

\_check\_block: 校验下载好的块是否正确

rpc\_download: 调用rpc进行下载

\_download\_from\_ip: 向一个节点发送下载请求，调用rpc\_download

download\_task:用于多线程下载，调用\_download\_from\_ip

download\_package:向多个节点进行多线程下载

**LocalOP**

archive: 对文件进行一致性打包操作

block: 进行erasure code编码解码

collector: 从系统中重构包

track: 追踪文件的变化