

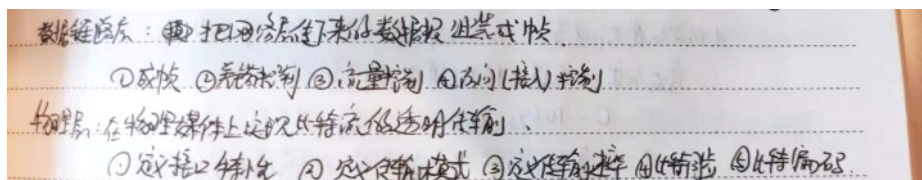
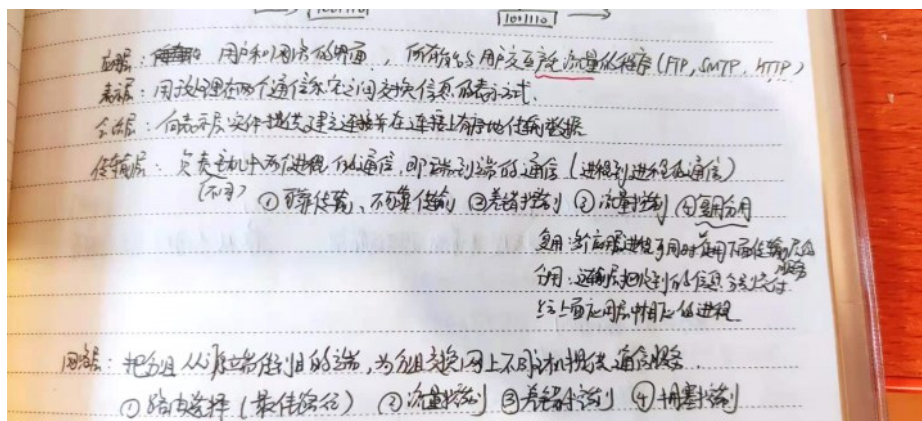
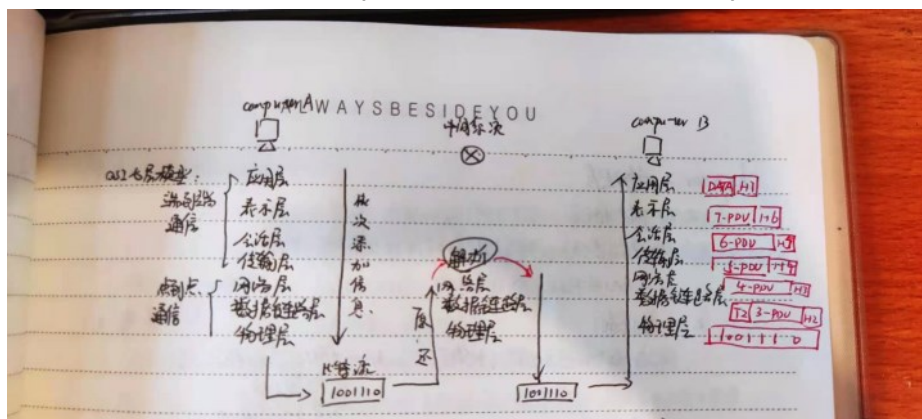
1.2-1网络基本知识

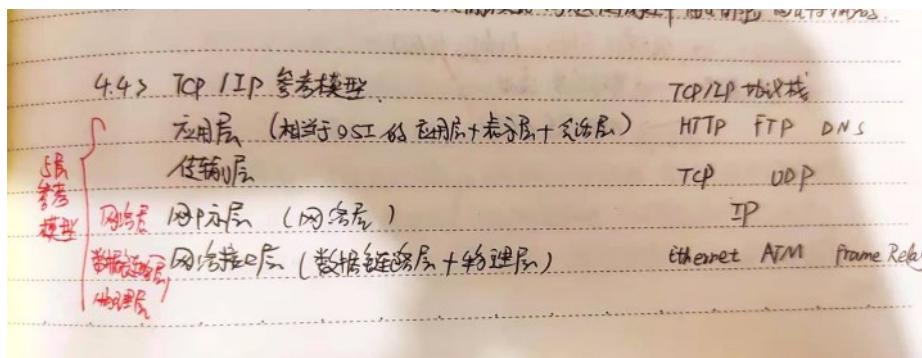
2021年4月27日 15:48

计算机网络分层结构

计算机网络分层结构常用的模型有7层OSI参考模型，4层TCP/IP参考模型以及为方便学习产生的5层参考模型。

首先是国际标准化组织的7层OSI参考模型，该模型虽然是法定标准，但因为种种原因，在实际应用中还是以TCP/IP参考模型为主（TCP/IP模型是实际上的标准）



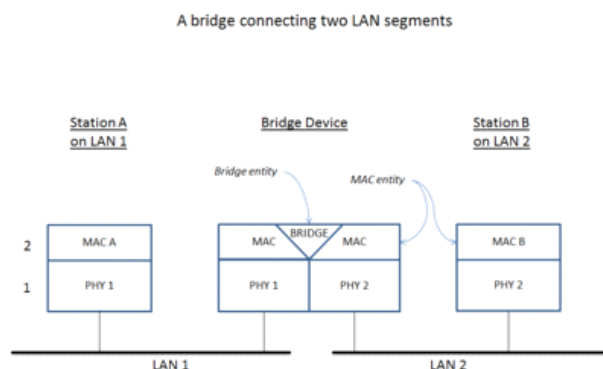


关于各层的特点、交换原理、协议等我听了网课并做了笔记，以电子版的形式放在一个pdf里，详见**计算机网络笔记.pdf**

交换机、路由器、网桥

网桥是工作在**数据链路层**的设备，它是早期的一种两端口网络交换设备。网桥可以将网络的多个网段桥接起来，扩展网络的距离和范围并提高网络的性能、可靠性。网桥能够识别数据链路层中的数据帧，并将这些数据帧重新打包后作为一个全新的数据帧转发给相连的另一个网段。

每当网桥收到一个帧时，就先暂存在其缓冲中。若此帧未出现差错，且欲发往的目的站 MAC 地址属于另一网段，则通过查找站表，将收到的帧送往对应的端口转发出去。若该帧出现差错，则丢弃此帧。



图片来源：维基百科

交换机，可以看做是一个**多端口的网桥**，在交换机工作时允许多组端口间的通道同时工作。交换机相比网桥拥有更高的数据转发效率和更强的MAC地址自动学习能力。

交换机的工作过程：

1. 当交换机从某个端口收到一个数据帧后，先读取帧头部的源MAC 地址，并与自己缓存中的映射表（CAM 表）进行比较，如果没有找到，则在CAM 表中添加一个该源MAC 地址与发送该帧的源端口映射表项。这就是交换机的MAC 地址自动学习功能。
2. 如果在CAM 表项查到了帧中源MAC 地址，则继续查看是否有帧中目的MAC 地址所对应

的映射表项。如果有，则直接把该帧转发到目的MAC 地址节点所连接的交换机端口，然后由该端口发送到目的主机。

3. 如果在交换机CAM 表中没有找到帧中目的MAC 地址所对应的表项，则把该数据帧向除源端口外的其他所有端口上进行泛洪。
4. 当MAC 地址与帧中目的MAC 地址的主机接收了该数据帧后就会向源主机产生一个应答帧，交换机获取该应答帧后从其中的源MAC 地址中获取了对应的MAC 地址和所连接端口的映射关系，并添加到CAM 表中。这样下次再有MAC 地址为这个MAC 地址的帧发送时交换机就可以直接从CAM 表中找到对应的转发端口，直接转发，不用再泛洪了。

路由器

路由器是属**网络层**的一种互联设备。用于连接多个逻辑上分开的网络。路由器具有判断网络地址和选择路径的功能，它能在多网络互联环境中，建立灵活的连接，可用完全不同的数据分组和介质访问方法连接各种子网，路由器只接受源站或其他路由器的信息，它不关心各子网使用的硬件设备，但要求运行与网络层协议相一致的软件。

路由器的主要工作就是为经过路由器的每个数据帧寻找一条最佳传输路径，并将该数据有效地传送到目的站点。路由器的基本功能是，把数据（IP 报文）传送到正确的网络，



参考链接：

<https://zh.wikipedia.org/wiki/%E6%A9%8B%E6%8E%A5%E5%99%A8>

<https://blog.csdn.net/frankarmstrong/article/details/77969699>

1.2-2Linux网桥、路由配置

2021年5月11日 18:27

linux路由配置

跨越从源主机到目标主机的一个互联网络来转发数据包的过程，称为路由。路由器根据路由表选择到达目标网络的最佳路径的过程，称为路由选择。在路由配置的过程中，有时为了能够明确控制数据包在网络中的行程或者其他原因，会需要对路由进行手动配置。

下面总结一些在linux中配置路由的命令操作。

route命令用于显示和操作IP路由表。route命令的格式与可选参数如下：

格式：

```
route [-CFvnee]
```

```
route [-v] [-A family] add [-net|-host] target [netmask Nm] [gw Gw] [metric N] [mss M] [window W] [irtt I] [reject] [mod] [dyn] [reinststate] [[dev] If]
```

```
route [-v] [-A family] del [-net|-host] target [gw Gw] [netmask Nm] [metric N] [[dev] If]
```

选项符列表：

- -C
显示路由缓存。
- -F
显示发送信息
- -v
显示详细的处理信息。
- -n
不解析名字。
- -ee
使用更详细的资讯来显示
- -V
显示版本信息。
- -net
到一个网络的路由表。

- -host
到一个主机的路由表。

参数列表：

- add
增加路由记录。
- del
删除路由记录。
- target
目的网络或目的主机。
- gw
设置默认网关。gateway 的简写，后续接的是 IP 的数值。
- mss
设置TCP的最大区块长度（MSS），单位MB。
- window
指定通过路由表的TCP连接的TCP窗口大小。
- dev
如果只是要指定由那一块网路卡连线出去，则使用这个设定，后面接 eth0 等。
- reject
设置到指定网络为不可达，避免在连接到这个网络的地址时程序过长时间的等待，直接就知道该网络不可达。

路由配置过程中可能用到的一些命令格式：

- 查看linux内核路由表

`route -n`

```
reed@ubuntu:~$ route -n
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0        192.168.116.2  0.0.0.0         UG    100    0      0 ens33
169.254.0.0    0.0.0.0        255.255.0.0     U    1000   0      0 ens33
192.168.116.0  0.0.0.0        255.255.255.0   U    100    0      0 ens33
reed@ubuntu:~$
```

其中Destination指的是目标网段或者主机，Gateway是网关地址，Genmask是网络掩码，Metric是路由距离。

Flags是标记，标记这条路由的类型或者状态。**主机路由**是路由选择表中指向单个IP地址或主机名的路由记录。主机路由的Flags字段为**H**。**网络路由**是代表主机可以到达的网络。网络路由的Flags字段为**N**。当主机不能在路由表中查找到目标主机的IP地址或网络路由时，数据包就被发送到**默认路由**

(默认网关) 上。默认路由的Flags字段为**G**。

- 路由配置命令

```
route [add|del] [-net|-host] target [netmask Nm] [gw Gw] [[dev] If]
```

参数意义：

add	添加一条路由规则
del	删除一条路由规则
-net	目的地址是一个网络
-host	目的地址是一个主机
target	目的网络或主机
netmask	目的地址的网络掩码
gw	路由数据包通过的网关
dev	为路由指定的网络接口

- 静态路由配置

```
ip route [destination_network] [mask] [next-hop_address]
```

参数意义：

ip route	用于创建静态路由的命令。
Destination_network	需要发布到路由表中的网段。
Mask	在这一网络上使用的子网掩码。
Next-hop_address	下一跳路由器的地址。

- route命令使用举例

添加到主机的路由

```
# route add -host 192.168.1.2 dev eth0:0  
# route add -host 10.20.30.148 gw 10.20.30.40
```

添加到网络的路由

```
# route add -net 10.20.30.40 netmask 255.255.255.248 eth0  
# route add -net 10.20.30.48 netmask 255.255.255.248 gw 10.20.30.41  
# route add -net 192.168.1.0/24 eth1
```

添加默认路由

```
# route add default gw 192.168.1.1
```

删除路由

```
# route del -host 192.168.1.2 dev eth0:0  
# route del -net 10.20.30.40 netmask 255.255.255.248 eth0  
# route del default gw 192.168.1.1  
//route del default 删除所有的默认路由
```


添加一条静态路由

```
# route add -net 192.168.2.0/24 gw 192.168.2.254
```

添加到一台主机的静态路由

```
# route add -host 192.168.2.2 gw 192.168.2.254
```

- 开启主机的路由转发功能

临时开启路由功能：

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

?永久开启路由功能

```
# vim /etc/sysctl.conf
```

```
net.ipv4.ip_forward = 1
```

```
# sysctl -p
```

- 使用route命令所添加的路由，在主机重启后就丢失了，如果要添加永久路由的话，需要修改配置文件。

```
vim /etc/sysconfig/network-scripts/route-ethN
```

或者

```
vim /etc/sysconfig/network
```

可以通过vim编辑器修改network-scripts中的配置文件或者network配置文件，其中ethN是网卡名。

在文件中添加一条这样的信息：*10.211.6.0/24 via 192.168.3.1 dev eth3*就代表在访问地址10.211.6.0时通过192.168.3.1这个网关。

linux网桥配置

? 什么是linux网桥?

linux网桥是一种**虚拟的**网络设备，功能与物理交换机相同，可以绑定多个以太网接口设备，从而将他们桥接起来，实现报文的相互转发。

网桥的配置

1.安装bridge-utils (brctl)

```

reed@ubuntu:~$ su root
Password:
root@ubuntu:/home/reed# apt-get install -y bridge-utils
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
linux-headers-4.15.0-45 linux-headers-4.15.0-45-generic
linux-headers-4.15.0-47 linux-headers-4.15.0-47-generic
linux-headers-4.15.0-50 linux-headers-4.15.0-50-generic
linux-headers-4.15.0-52 linux-headers-4.15.0-52-generic
linux-image-4.15.0-45-generic linux-image-4.15.0-47-generic
linux-image-4.15.0-50-generic linux-image-4.15.0-52-generic
linux-modules-4.15.0-45-generic linux-modules-4.15.0-47-generic
linux-modules-4.15.0-50-generic linux-modules-4.15.0-52-generic
linux-modules-extra-4.15.0-45-generic linux-modules-extra-4.15.0-47-generic
linux-modules-extra-4.15.0-50-generic linux-modules-extra-4.15.0-52-generic
Use 'apt autoremove' to remove them.
The following NEW packages will be installed:
bridge-utils
0 upgraded, 1 newly installed, 0 to remove and 81 not upgraded.
Need to get 28.6 kB of archives.
After this operation, 102 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu xenial/main amd64 bridge-utils amd64 1
.5-9ubuntu1 [28.6 kB]
Fetched 28.6 kB in 2s (11.5 kB/s)
Selecting previously unselected package bridge-utils.
(Reading database ... 398138 files and directories currently installed.)
Preparing to unpack .../bridge-utils_1.5-9ubuntu1_amd64.deb ...
Unpacking bridge-utils (1.5-9ubuntu1) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up bridge-utils (1.5-9ubuntu1) ...
root@ubuntu:/home/reed#

```

```

root@ubuntu:/home/reed# brctl
Usage: brctl [commands]
commands:
    addbr          <bridge>          add bridge
    delbr          <bridge>          delete bridge
    addif          <bridge> <device>  add interface to bridge
    delif          <bridge> <device>  delete interface from bridge
    hairpin        <bridge> <port> {on|off}  turn hairpin on/off
    setageing      <bridge> <time>    set ageing time
    setbridgeprio  <bridge> <prio>    set bridge priority
    setfd          <bridge> <time>    set bridge forward delay
    sethello       <bridge> <time>    set hello time
    setmaxage      <bridge> <time>    set max message age
    setpathcost    <bridge> <port> <cost> set path cost
    setportprio    <bridge> <port> <prio> set port priority
    show           [ <bridge> ]       show a list of bridges
    showmacs       <bridge>           show a list of mac addrs
    showstp        <bridge>           show bridge stp info
    stp            <bridge> {on|off}  turn stp on/off
root@ubuntu:/home/reed#

```

2. 建立一个逻辑网桥bridge1

`brctl addbr bridge1`

```

root@ubuntu:/home/reed# brctl addbr bridge1
root@ubuntu:/home/reed# ^C
root@ubuntu:/home/reed# brctl show
bridge name      bridge id      STP enabled  interfaces
bridge1          8000.000000000000  no
root@ubuntu:/home/reed#

```

此时的bridge还没有分配任何接口。

1.2-3网络仿真系统ns2

2021年5月12日 20:10

? 前言

网络仿真技术就是应用仿真方法对现有网络或待建网络在计算机上 进行虚拟仿真的技术。利用数学建模和统计方法模拟网络行为，从而获得 特定网络参数。网络仿真获取的网络特性参数包括网络全局性能统计量、 网络节点的性能统计量、网络链路的流量和延迟等。

网络仿真的目的主要有：

- 学习：学习协议和算法的实现，包括它们的行为和性能。
- 测试：对未实现和未投入实际应用的协议和算法进行测试。
- 比较：对各种研究结果、协议和算法的优缺点进行比较直观和客 观的比较。
- 观察：对各种网络行为进行仿真模拟，观察网络现象的产生发展。

网络仿真的实现方法

网络是由节点和链路组成的。用户通 过终端机访问网络，其信息通过具有交换功能的节点与链路在网络中传输。 信息网络是一个离散系统，不论是IP网络还是其他通信网络，在任意时刻进入网络的分组数、分组长度、以及分组到达网络入口的时间间隔等参数都是随机变量，引起网络系统状态的变化是在数据信息产生与接收的离散时刻出现的；系统状态的变化是由于随机事件的发生而引起的，而且变化是时间上可数(或有限) 的。离散系统的状态变化只在离散时刻发生，且往往是随机的，通常即用“**事件**”来表示这种变化，所以又称离散事件系统。

所以，网络仿真是基于对离散事件系统仿真扩展开来的。离散事件系统主要由四种因素组成，即实体、设备、事件和活动。对信息网络进行仿真，参照OSI模型的分层概念，建立相应于各层功能/性能描述的仿真模型，再用计算机语言编制成对应仿真模型的计算机程序，应用程序可以模拟各的通信和行为，然后通过运行这些程序来模拟计算机网络的工作，从而对网络的各种工作性能进行研究。

网络仿真的基本流程

- 根据需求分析，确定仿真对象。
- 根据研究对象，编写仿真脚本：利用网络仿真系统提供的基本函数和功能，描述被仿真网络，即确定被仿真网络的拓扑、协议、承载的数据流量和控制参数等，编写可具体执行的仿真脚本。
- 运行仿真脚本
- 数据采集和分析

ns-2是一种面向对象的网络仿真器，常常被用于网络课程的教学以及网络相关的研究中。它本质上是一个离散事件模拟器， 其本身有一个虚拟时钟，所有的仿真都由离散事件驱动的。

目前NS-2可以用于仿真各种不同的通信网络已经实现的一些仿真 模块有：网络传输协议，如TCP和

UDP；业务源流量产生器，如FTP和Telnet；路由队列管理机制以及路由算法，如Droptai、RED和CBQ；以及无线通信网络如Ad hoc路由，移动IP和卫星通信网络等。

在ns-2脚本配置时使用的是**TCL脚本语言**，在这里对该语言做一些必要的了解。

- Tcl是一种可扩展的命令语言。它只支持一种数据结构：字符串。所有的命令、参数和变量都是字符串，这也是TCL简化学习难度的一种特色。

- Tcl命令的基本语法：

```
Command arg1 arg2 arg3
```

其中Command代表内置命令的名称或者Tcl过程，arg1和arg2是该命令的参数，命令以及参数之间采用空格或者TAB键来分开。换行号或者分号（;）标志一条命令的结束。

- 执行一条命令后，会有两个结果：一个返回值和一个字符串。返回值标志着命令是否正确执行，字符串给出附加信息。以下：

TCL_OK 命令正确执行，字符串给出了命令的返回值。

TCL_ERROR 表示有一个错误发生，字符串给出了错误描述。

- Tcl中变量名可以采用任意字母、数字和下划线，长度也没有限定，但是区分字符大小写。在使用变量前不需要实现声明，因为解释器会在第一次使用变量的时候创建变量，使用变量时在变量前加\$符号，例如\$exp表示使用变量exp。set命令和unset命令用来创建和取消变量。

- 数组是变量的集合。在Tcl中数组索引可以是任意的字符串。数组同样由set命令创建。

```
Set arr(index) value
```

可以使用

```
puts $qian(123);
```

的形式来获得数组的内容，也可以使用parray命令获取数组的所有信息，如：

```
parray arr;
```

- 字符串操作：string 命令基本语法如下：

```
string option string1 string2;
```

option的操作选项：

compare按照字典的排序方式进行比较。根据string1 <, =, >string2分别返回-1, 0, 1

first返回string2中第一次出现string1的位置，如果没有出现string1则返回-1

last和first相反

trim从string1中删除开头和结尾的出现在string2中的字符

tolower返回string1中的所有字符被转换为小写字符后的新字符串

toupper返回string1中的所有字符串转换为大写后的字符串

length返回string1的长度

- 数字操作：因为tcl中只有一个string类型的变量，所以当要操作数字进行运算的时候，tcl提供了

incr和expr两个操作数。

incr的基本用法为：

```
incr variable integer; # (variable必须为数字)
```

incr的作用是将variable增加integer的值，如果integer为负数，则相当于减。默认的incr a等同于a++的意思，即本身自增1。

expr的基本语法为：

```
expr function number;
```

expr是为了提供更加复杂的操作而设计的一个语法，比如运算乘除法等等。在执行算术操作的时候必须将expr命令放在算术操作之前。例如：

```
set a 20; set b 4;  
set c [expr $a/$b]; ?#此时的c的值为5
```

除此之外，expr还能够识别一些函数及其返回值如下：

abs(x) x的绝对值

round(x) x舍入后得到的整数值

sin(x) x的正弦

、

详细的tcl语言介绍在**TCL脚本入门教程**文件中。

📖 NS的网络组件

NS中所有的网络组件可分为分类器 (Classifier)和连接器(Connector)。分类器的派生类组件对象包括地址分类器(AddrClassifier)和多播分类器(McastClassifier)等；连接器的派生类组件对象包括队列(Queue),延迟(Delay)各种代理(Agent),和追踪对象类(Trace)。由一个或多个由基本组件复合派生的组件称为复合组件，NS有许多这样的组件来支撑对网络的仿真。对IP网络比较重要的有：

- 拓扑节点Node

是由一个节点入口对象和若干个分类器(Classifier) 组成的一个复合对象。结点的功能是分配数据包，建立网络拓扑的第一步就是建立网络结点。

NS中node的**操作命令**可分为4类，控制功能、地址和端口管理、点到点 路由功能、代理管理和添加邻居。

- a)控制功能

```
$node entry
```

返回node对象的入口对象，对单播结点而言，通常是某AddrClassifier

- b) 地址和端口管理，点到点路由功能

```
$node id
```

返回node对象标志号

```
$node agent port
```

返回与指定端口的相连的代理

- c) 代理管理

`$node attach`

为结点添加代理

`$node detach`

将代理从结点删除

d) 添加邻居

`$node add-neighbor`

为结点添加邻居

- 拓扑结点连接类Link

一个结点和另一个结点之间的连接(simpex-link)是单向的。一个最基本的简单连接由 一个连接入口、包缓冲队列、延迟处理对象、废弃处理对象和时间处理对象 (TTL)组成。一个节点的输出队列，是通过和这个结点相连的Link中的缓冲队列来实现的。一个数据包进入link后，首先到达link head，送到由它指向的数据缓冲队列。

- 代理网络组件Agent

代表了在网络层中数据包的产生地和运输源头，同时也是各层网络中各种协议的实现，是NS-2中最重要的一个组件，可大致认为它是传输层的仿真，NS-2中定义了大量的Agent类型，每种Agent都有自己的数据包格式以及对数据包的操作。

- packet

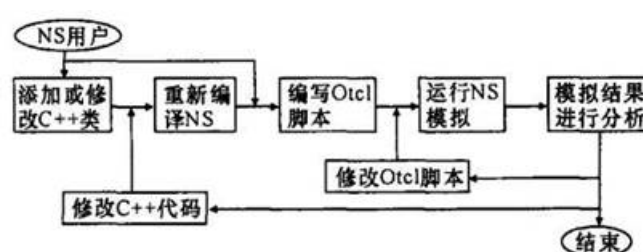
在NS-2中，一个“包”是由一个报头(header)堆栈(Stack)和一个可选择的数据空间构成。报头的格式在模拟器对象(Simulator)创建的过程中被创建，所有注册过的报头，比如一般的属性说明、IP报头、TCP报头都在包的报头堆栈中存储。不同网络组件可以通过报头堆栈中的偏移量来访问包中的不同报头。通常在仿真研究，包只有报头堆栈部分，并没有数据空间。这是因为在仿真实验时，传输实际的数据没有意义。

- 其他构件

队列 (Queue)和延迟 (Delay)构件是基于链路的，是链路的重要性质。跟踪对象(Trace)是NS中用于存储仿真结果的一个对象，用于配置需要跟踪的一些参数，并将其写入跟踪日志文件 (Trace file)中。应用 (Application) 对象可用**来仿真**各种应用层对象如FTP、HTTP等。

NS的仿真流程

使用NS仿真涉及到两个层次：一个是基于Otccl编程的层次，利用NS具有的网络元素实现仿真，只要编写Otccl脚本就可以，而无需对NS本身进行修改；另一个层次是基于C++和Otccl编程的层次，如果NS中没有所需的网络元素，则要对NS扩展修改，添加所需的网络元素，重新编译，然后进行仿真。



Nam

nam是基于Tcl/Tk的动画演示工具，用于观察网络模拟的trace行为和现实世界中数据包的trace数据，例如网络拓扑、包传输和队列管理等。Nam是基于仿真完成后的nam文件进行演示的，该文件其中包含了网络的拓扑信息，如节点、链路以及数据包的数据。nam文件由NS创建。生成nam文件之后，就可以用Nam进行演示。演示开始时，nam会读取trace文件，创建网络拓扑，显示窗界面，进行必要的布局设置，然后停止在时间标记为0的地方。

NS中可以对节点、链路、队列和Agent等对象进行nam动画显示方面的控制：

- 节点

对于节点对象，NS可以设定动画显示时节点的形状、颜色、表示名称和注释等，常用命令有：

`$node label [label];` #设定节点的名称

`$node shape [shape];` #设定节点的形状

`$node color [color];` #设定节点的颜色

`$node label-color [lcolor];` #设定节点显示名称的颜色

`$node label-at [ldirection];` #设定名称的显示位置

`$node add-mark [name][color][shape];` #增加注释

`$node delete-mark [name];` #删除注释

- 链路

NS中一般通过下面的命令来指定链路的显示属性。

`$ns duplex-link <attribute> <value>`

其中attribute可以是这些值：orient, color, queuePos和label。Orient指定了链路的的方向，可以用角度或文字来定义，如right、right-up、right-down, left, left-up, left-down, up, down; label定义链路显示的名称；

- Agent

在NS中，Agent都是绑定在节点上的，在Nam中可以显示出来某个节点上绑定了哪些Agent。下面命令可使想要显示的Agent以AgentName出现在节点附件的方框内：

`$ns attach-agent $node $Agent ;` #Agent和节点绑定

`$ns add-agent-trace $Agent AgentName;` #nam*\$node附近将出现AgentName

参考资料

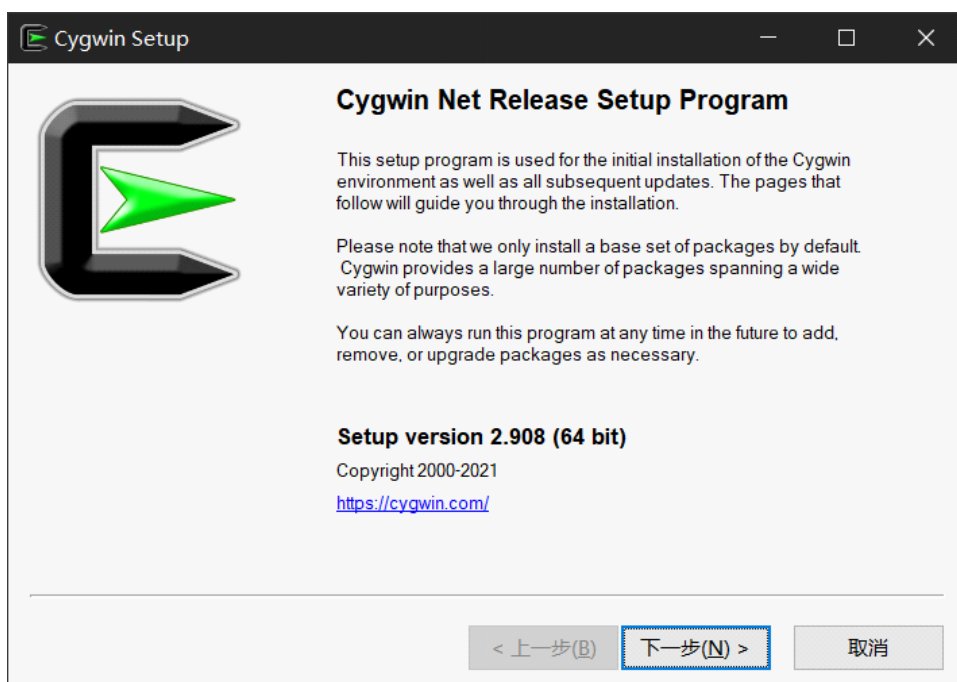
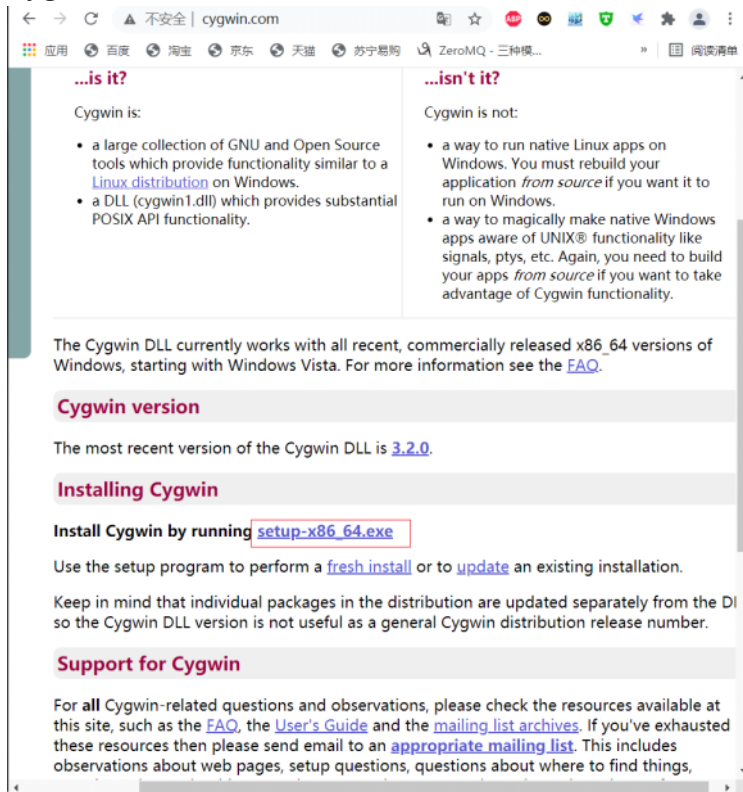
- https://download.csdn.net/download/panyajing/4314135?utm_medium=distribute.pc_relevant.none-task-download-2%7Eddefault%7EBlogCommendFromMachineLearnPai2%7Eddefault-1.control&depth_1-utm_source=distribute.pc_relevant.none-task-download-2%7Eddefault%7EBlogCommendFromMachineLearnPai2%7Eddefault-1.control
- <https://zh.wikipedia.org/w/index.php?search=tcl%E8%84%9A%E6%9C%AC%E8%AF%AD%E8%A8%80&title=Special%3A%E6%90%9C%E7%B4%A2&go=%E5%89%8D%E5%BE%80&ns0=1>
- <https://blog.csdn.net/zengxiantao1994/article/details/78136536>
- [https://zh.wikipedia.org/wiki/Ns_\(%E6%A8%A1%E6%8B%9F%E5%99%A8\)](https://zh.wikipedia.org/wiki/Ns_(%E6%A8%A1%E6%8B%9F%E5%99%A8))
- [1]牛沛琛. 基于网络模拟软件 (NS-2) 的IP网络性能仿真技术研究[D].厦门大学,2007.。

NS2的安装、环境搭建与使用

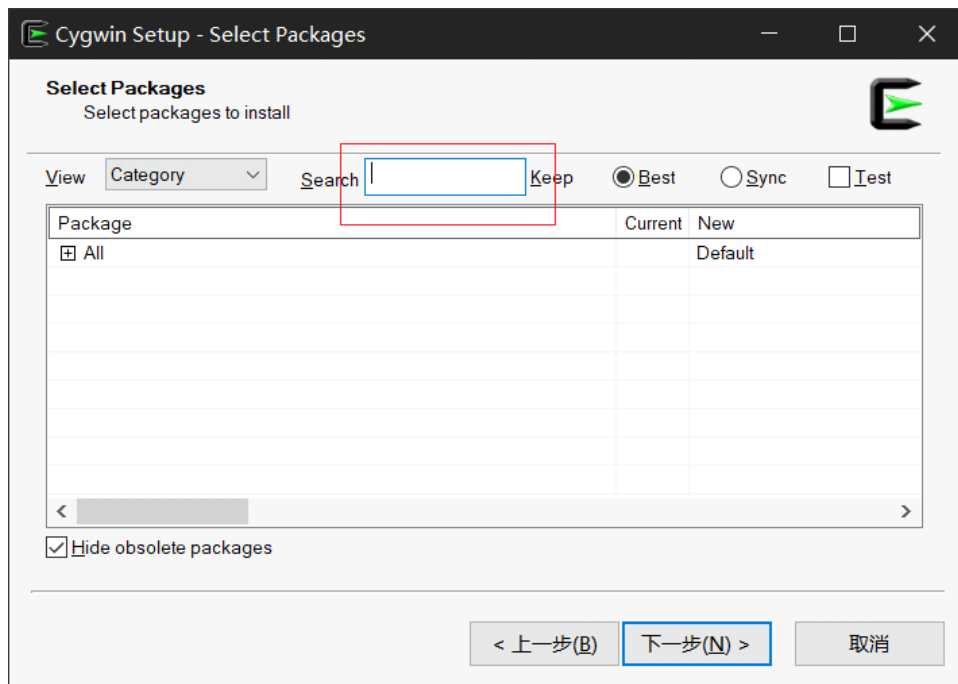
因为ns2是必须运行在UNIX/Linux环境中的，所以想在windows中使用NS2，只能通过虚拟机或者Cygwin模拟平台进行。

首先通过Cygwin模拟平台使用NS2的方式如下

Cygwin官网下载安装包并安装：



在这个界面进行安装包选择：



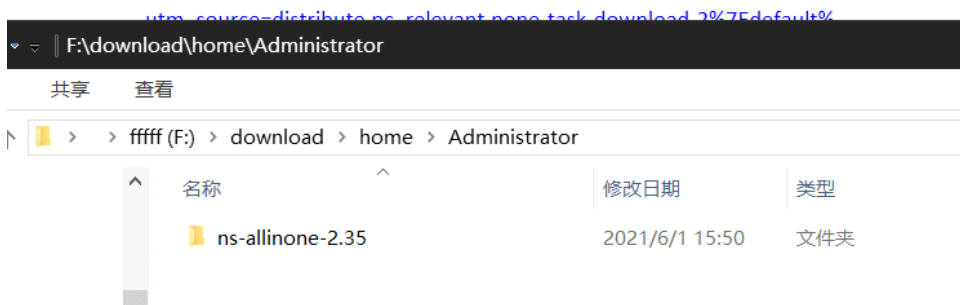
安装cygwin时候必选的安装包有：

1. gcc
2. gcc-g++
3. gnuplot
4. make
5. patch
6. perl
7. tar
8. X-startup-scripts
9. xorg-x11-base
10. xorg-x11-bin
11. xorg-x11-devel
12. xorg-x11-bin-dlls
13. xorg-x11-bin-Indir
14. xorg-x11-etc
15. xorg-x11-fenc
16. xorg-x11-fnts
17. xorg-x11-libs-data

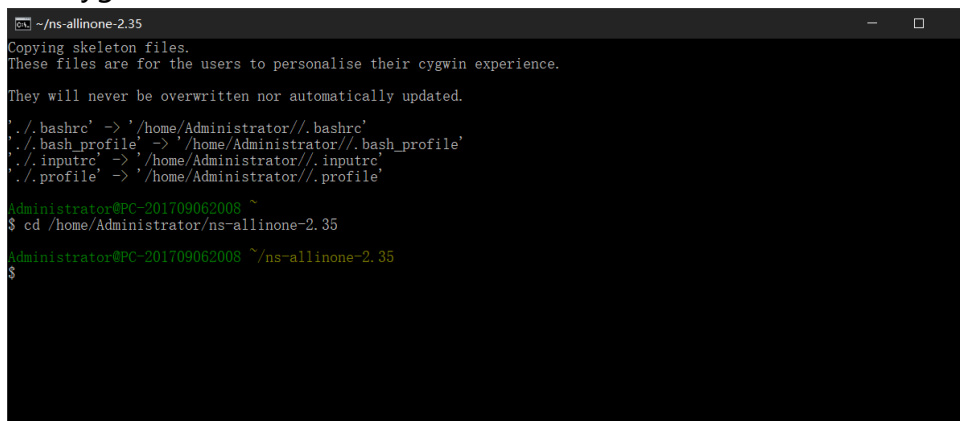
如上图，可以在红框框出来的搜索栏中搜索安装包并选择安装。

在安装过程中发现很多软件包找不到，查找资料后发现**安装时务必下载32位的安装包，否则会缺少很多软件包！同时记得取消左下角隐藏选项**

下载安装NS2：下载NS2压缩包并解压到以下位置：

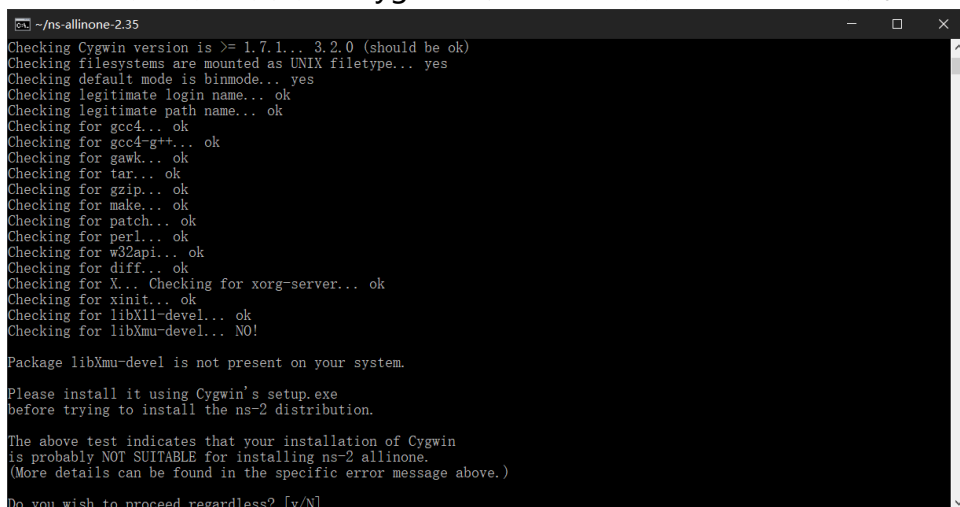


打开cygwin命令行，切换到ns目录下：

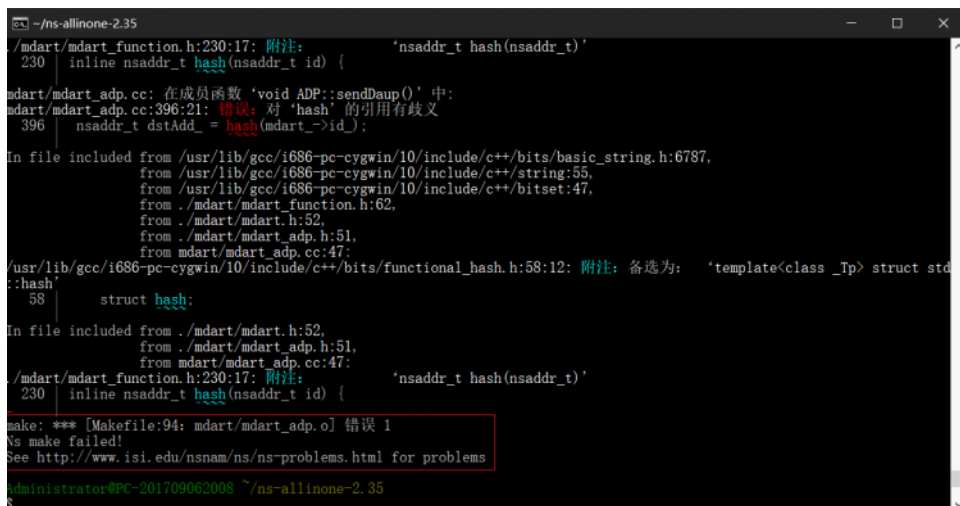


输入./install开始安装

在安装过程中可能还会提示有缺少的包，此时只要重新运行cygwin安装程序，在之前的安装目录下进行安装，就可以实现对cygwin的更新，选择所需的包进行安装即可。



在安装了所有所需软件包之后正式开始安装ns2，结果安装失败...



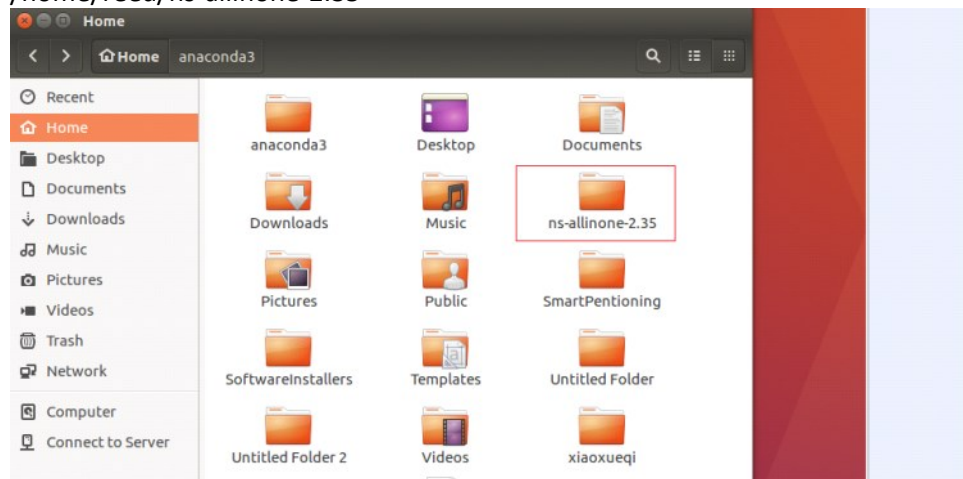
google搜索得知可能是因为当前的gcc版本太高了，但是这版本的cygwin安装程序中找不到更低版本的gcc了。

折腾了一下午，还是没有解决...感觉cygwin这个东西有点过时了，还是先在虚拟机中安装NS2吧

在ubuntu虚拟机中安装NS2

首先将下载的ns2压缩包解压到用户目录下面。

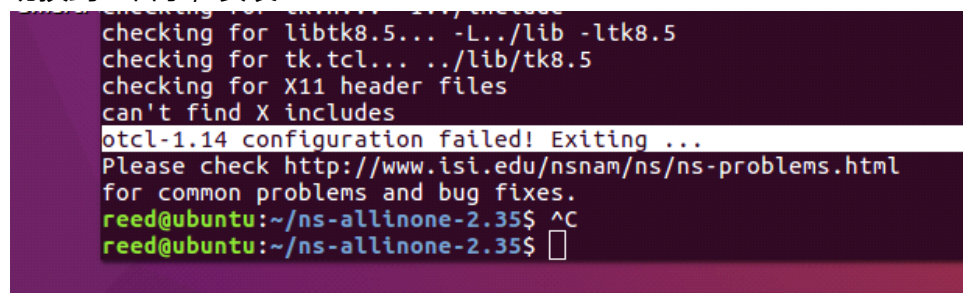
/home/reed/ns-allinone-2.35



安装所需的依赖包

```
sudo apt-get install build-essential
sudo apt-get install tcl8.5 tcl8.5-dev tk8.5 tk8.5-dev
sudo apt-get install libxmu-dev libxmu-headers
```

切换到ns目录，安装



安装出现的问题，解决方法：安装libxt-dev

再次进行ns2安装，继续报错

```
reed@ubuntu: ~/ns-allinone-2.35
/include -I/usr/include/pcap -I./tcp -I./sctp -I./common -I./link -I./queue -I./
adc -I./apps -I./mac -I./mobile -I./trace -I./routing -I./tools -I./classifier
I./mcast -I./diffusion3/lib/main -I./diffusion3/lib -I./diffusion3/lib/nr -I./d
ffusion3/ns -I./diffusion3/filter_core -I./asim/ -I./qs -I./diffserv -I./satell
te -I./wpan -o linkstate/ls.o linkstate/ls.cc
In file included from linkstate/ls.cc:67:0:
linkstate/ls.h: In instantiation of 'void LsMap<Key, T>::eraseAll() [with Key =
int; T = LsIdSeq]':
linkstate/ls.cc:396:28:   required from here
linkstate/ls.h:137:25: error: 'erase' was not declared in this scope, and no de
clarations were found by argument-dependent lookup at the point of instantiation
[-fpermissive]
    void eraseAll() { erase(baseMap::begin(), baseMap::end()); }
                        ^
linkstate/ls.h:137:25: note: declarations in dependent base 'std::map<int, LsIdS
eq, std::less<int>, std::allocator<std::pair<const int, LsIdSeq> > >' are not fo
und by unqualified lookup
linkstate/ls.h:137:25: note: use 'this->erase' instead
Makefile:93: recipe for target 'linkstate/ls.o' failed
make: *** [linkstate/ls.o] Error 1
Ns make failed!
See http://www.isi.edu/nsnam/ns/ns-problems.html for problems
reed@ubuntu: ~/ns-allinone-2.35$
```

解决方法：修改ns安装包中的ls文件

```

LsList(const Tp & x) : baseList(1, x) {}
void eraseAll() {
    baseList::erase(baseList::begin(), baseList::end());
}
LsList<Tp>& operator= (const LsList<Tp> & x) {
    return (LsList<Tp> &)baseList::operator= (x);
}

template<class Key, class T>
LsMap : public map<Key, T, less<Key> > {
public:
    typedef less<Key> less_key;
    typedef map<Key, T, less_key> baseMap;
    LsMap() : baseMap() {}

    // this next typedef of iterator seems extraneous but is required by gcc-2.96
    typedef typename map<Key, T, less<Key> >::iterator iterator;
    typedef pair<iterator, bool> pair_iterator_bool;
    iterator insert(const Key & key, const T & item) {
        typename baseMap::value_type v(key, item);
        pair_iterator_bool ib = baseMap::insert(v);
        return ib.second ? ib.first : baseMap::end();
    }

    void eraseAll() { this->erase(baseMap::begin(), baseMap::end());}
    T* findPtr(Key key) {
        iterator it = baseMap::find(key);
        return (it == baseMap::end()) ? (T *)NULL : &(*it).second;
    }
}

C/C++/ObjC Header  Tab Width: 8  Ln 137, Col 73
```

将第137行修改为图中所示内容

再再次进行安装。

终于安装成功了

```

reed@ubuntu: ~/ns-allinone-2.35
make[4]: Leaving directory '/home/reed/ns-allinone-2.35/dei80211mr-1.1.4/src'
make[3]: Leaving directory '/home/reed/ns-allinone-2.35/dei80211mr-1.1.4/src'
make[2]: Leaving directory '/home/reed/ns-allinone-2.35/dei80211mr-1.1.4/src'
make[1]: Leaving directory '/home/reed/ns-allinone-2.35/dei80211mr-1.1.4/src'
make[1]: Entering directory '/home/reed/ns-allinone-2.35/dei80211mr-1.1.4'
make[2]: Entering directory '/home/reed/ns-allinone-2.35/dei80211mr-1.1.4'
make[2]: Nothing to be done for 'install-exec-am'.
make[2]: Nothing to be done for 'install-data-am'.
make[2]: Leaving directory '/home/reed/ns-allinone-2.35/dei80211mr-1.1.4'
make[1]: Leaving directory '/home/reed/ns-allinone-2.35/dei80211mr-1.1.4'
Please compile your nam separately.

Ns-allinone package has been installed successfully.
Here are the installation places:
tcl8.5.10:      /home/reed/ns-allinone-2.35/{bin,include,lib}
tk8.5.10:      /home/reed/ns-allinone-2.35/{bin,include,lib}
otcl:          /home/reed/ns-allinone-2.35/otcl-1.14
tclcl:         /home/reed/ns-allinone-2.35/tclcl-1.20
ns:            /home/reed/ns-allinone-2.35/ns-2.35/ns
xgraph:        /home/reed/ns-allinone-2.35/xgraph-12.2
gt-itm:        /home/reed/ns-allinone-2.35/itm, edriver, sgb2alt, sgb2ns, sgb2comns,
sgb2hierns

-----
--

Please put /home/reed/ns-allinone-2.35/bin:/home/reed/ns-allinone-2.35/tcl8.5.10
/unix:/home/reed/ns-allinone-2.35/tk8.5.10/unix
into your PATH environment; so that you'll be able to run itm/tclsh/wish/xgraph.

IMPORTANT NOTICES:

(1) You MUST put /home/reed/ns-allinone-2.35/otcl-1.14, /home/reed/ns-allinone-2
.35/lib,
    into your LD_LIBRARY_PATH environment variable.
    If it complains about X libraries, add path to your X libraries
    into LD_LIBRARY_PATH.
    If you are using csh, you can set it like:
        setenv LD_LIBRARY_PATH <paths>

```

```

Please put /home/reed/ns-allinone-2.35/bin:/home/reed/ns-allinone-2.35/tcl8.5.10
/unix:/home/reed/ns-allinone-2.35/tk8.5.10/unix
into your PATH environment; so that you'll be able to run itm/tclsh/wish/xgraph.

IMPORTANT NOTICES:

(1) You MUST put /home/reed/ns-allinone-2.35/otcl-1.14, /home/reed/ns-allinone-2
.35/lib,
    into your LD_LIBRARY_PATH environment variable.
    If it complains about X libraries, add path to your X libraries
    into LD_LIBRARY_PATH.
    If you are using csh, you can set it like:
        setenv LD_LIBRARY_PATH <paths>
    If you are using sh, you can set it like:
        export LD_LIBRARY_PATH=<paths>

(2) You MUST put /home/reed/ns-allinone-2.35/tcl8.5.10/library into your TCL_LIB
RARY environmental
    variable. Otherwise ns/nam will complain during startup.

After these steps, you can now run the ns validation suite with
cd ns-2.35; ./validate

For trouble shooting, please first read ns problems page
http://www.isi.edu/nsnam/ns/ns-problems.html. Also search the ns mailing list ar
chive
for related posts.

```

配置环境变量。

```
sudo vim /home/.bashrc
```

文件末尾添加：

```
export PATH=" $PATH:/home/reed/ns-allinone-2.35/bin:/home/reed/ns-
allinone-2.35/tcl8.5.10/unix:/home/reed/ns-allinone-2.35/tk8.5.10/unix"
```

```
export LD_LIBRARY_PATH=" $LD_LIBRARY_PATH:/home/reed/ns-
allinone-2.35/otcl-1.14:/home/reed/ns-allinone-2.35/lib"
```

```
export TCL_LIBRARY=" $TCL_LIBRARY:/home/reed/ns-
```



```
allinone-2.35/tcl8.5.10/library"
```

其中，reed是我的用户名。

添加环境变量后，通过

```
source ~/.bashrc
```

使得修改立刻生效。

运行测试例子

首先切换到例子所在目录下，之后

```
ns *.tcl
```

```
reed@ubuntu:~$ cd ns-allinone-2.35
reed@ubuntu:~/ns-allinone-2.35$ cd ns-2.35
reed@ubuntu:~/ns-allinone-2.35/ns-2.35$ cd tcl
reed@ubuntu:~/ns-allinone-2.35/ns-2.35/tcl$ ns example.tcl
couldn't read file "example.tcl": no such file or directory
reed@ubuntu:~/ns-allinone-2.35/ns-2.35/tcl$ cd ex
reed@ubuntu:~/ns-allinone-2.35/ns-2.35/tcl/ex$ ns example.tcl
0 delay=0
1 delay=0.075198896355555836
2 delay=0.14036090311111046
3 delay=0.18700956444444308
4 delay=0.23536785066666494
5 delay=0.28330976711111172
6 delay=0.33064953742222519
7 delay=0.37698112284444979
8 delay=0.42498596977778563
9 delay=0.47332022613335162
10 delay=0.52062097066669533
11 delay=0.56711607751114967
12 delay=0.61545838933338282
13 delay=0.66368562631117112
14 delay=0.71083429546673627
15 delay=0.75707166151118865
16 delay=0.80554185955564217
17 delay=0.85349646222230757
18 delay=0.90085145600008143
19 delay=0.94752994986674266
```

```
(file "example1.tcl" line 115)
reed@ubuntu:~/nsexamples$ ns flooding.tcl
running nam...
ns: finish: couldn't execute "nam": no such file or directory
while executing
"exec nam flooding.nam &"
(procedure "finish" line 8)
invoked from within
"finish"
```

仿真运行结束后会在tcl文件所在的目录下产生两个同名文件：*.nam和*.tr。其中tr文件是仿真过程中得到的仿真数据，nam可以用来图形化模拟过程。

首先安装nam。安装完成之后

```
nam *.nam
```

运行出现错误（segmentation fault），段错误，根据系统错误提示，尝试更新软件包看能否解决问题。


```

reed@ubuntu: ~/ns-allinone-2.35/nam-1.15
checking for gcc... (cached) gcc
checking whether we are using the GNU C compiler... (cached) yes
checking whether gcc accepts -g... (cached) yes
checking for gcc option to accept ISO C89... (cached) none needed
checking for g++... g++
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking for ANSI C header files... (cached) yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking for string.h... (cached) yes
checking for main in -lXbsd... no
checking for socket in -lsocket... no
checking for gethostbyname in -lnsl... yes
checking for dcgettext in -lintl... no
checking for getnodebyname in -ldnet_stub... no
checking that g++ can handle -O2... no
checking for zlib.h... -I../zlib-1.2.3
checking for libz1.2.3... -L../zlib-1.2.3 -lz
checking for X11 header files
checking for X11 library archive
checking for XOpenDisplay in -lX11... yes
checking for XShmAttach in -lXext... yes
checking for tcl.h... -I../include
checking for tclInt.h... -I../include
checking for libtcl8.6... no
checking for init.tcl... ../lib/tcl8.5
checking for http.tcl... ../lib/tcl8.5/http1.0
checking Tcl http.tcl library... yes
checking for tclsh8.6.5... no
checking for tclsh8.6... /usr/bin/tclsh8.6
configure: error: Installation of tcl seems incomplete or can't be found automatically.
Please correct the problem by telling configure where tcl is
using the argument --with-tcl=/path/to/package
(perhaps after installing it),
or the package is not required, disable it with --with-tcl=no.
reed@ubuntu:~/ns-allinone-2.35/nam-1.15$ sudo ./configure

```

报错：系统不能自动找到tcl。

可能是之前的环境变量添加的有问题。解决方法：

修改环境变量或者直接使用以下命令通过附加参数告知tcl和tk的位置

```

usage: sudo -v [-AknS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-AknS] [-g group] [-h host] [-p prompt] [-U user] [-u user]
[command]
usage: sudo [-AElknPS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
prompt] [-u user] [VAR=value] [-i|-s] [<command>]
usage: sudo -e [-AknS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
prompt] [-u user] file ...
reed@ubuntu:~/ns-allinone-2.35/nam-1.15$ sudo ./configure -with-tcl=/home/reed/n
s-allinone-2.35/tcl8.5.10 -with-tcl-ver=8.5.10 -with-tk=/home/reed/ns-allinone-2
.35/tk8.5.10 -with-tk-ver=8.5.10
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for ANSI C header files... yes
No .configure file found in current directory
Continuing with default options...
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking target system type... x86_64-unknown-linux-gnu
checking for gcc... (cached) gcc
checking whether we are using the GNU C compiler... (cached) yes
checking whether gcc accepts -g... (cached) yes
checking for gcc option to accept ISO C89... (cached) none needed

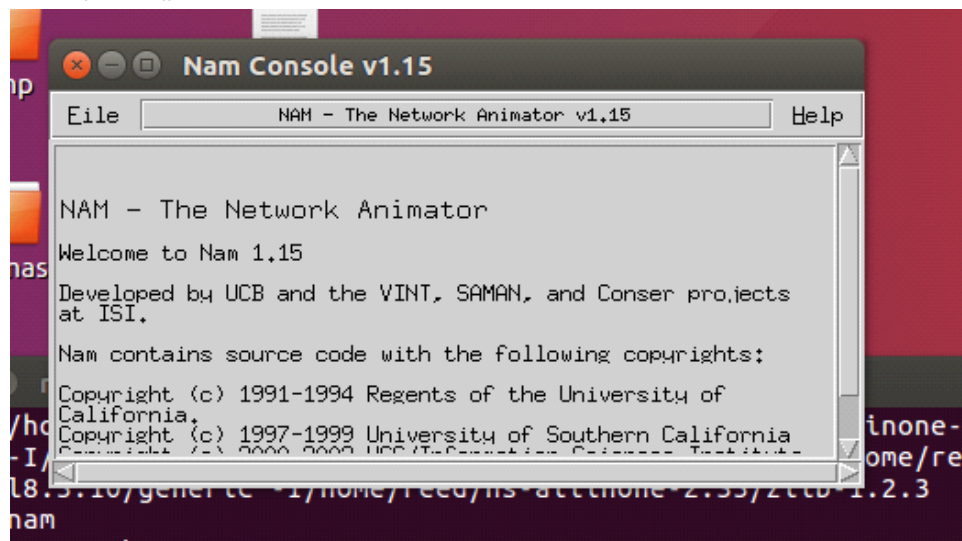
```

configure完成之后

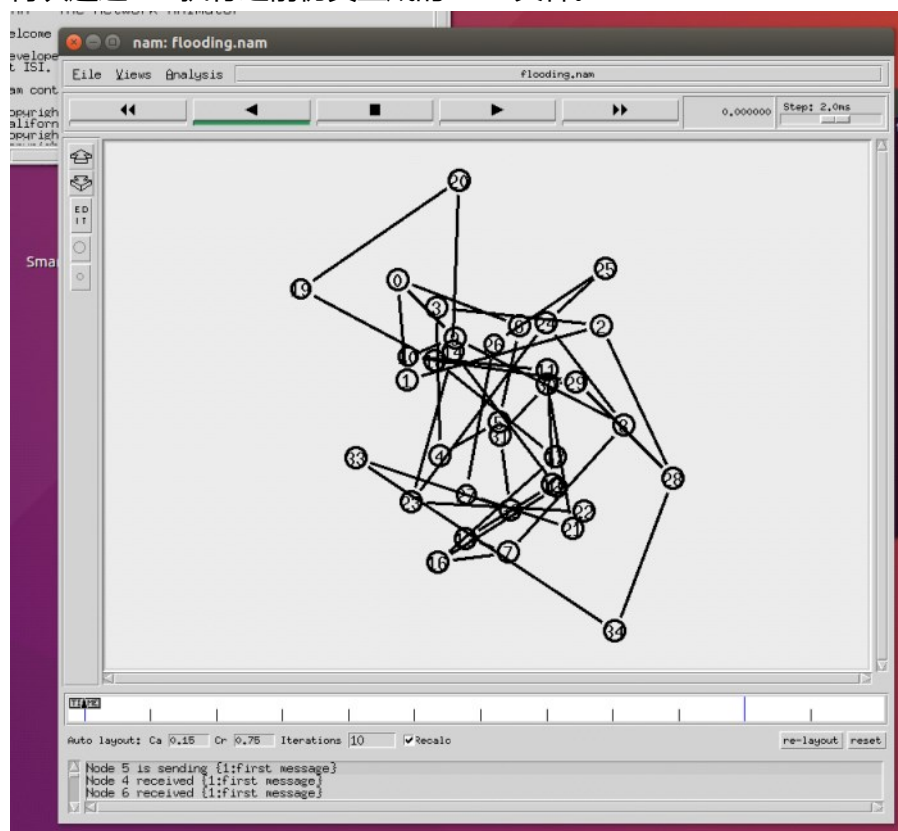
`sudo make`

```
sudo make install
```

之后命令行输入nam, 出现nam界面, 说明安装成功!



再次通过nam执行之前仿真生成的.nam文件。



成功得到演示界面。

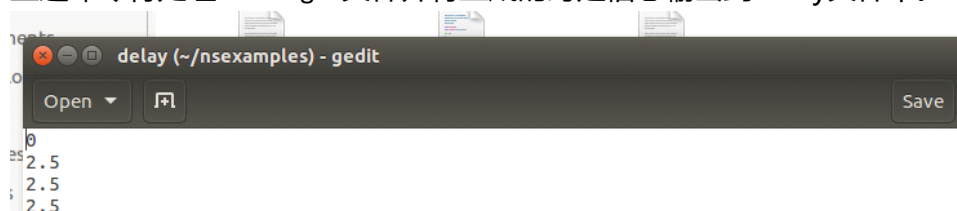
可以用gawk脚本或者python脚本处理得到的.tr文件, 并用gnuplot绘图显示。

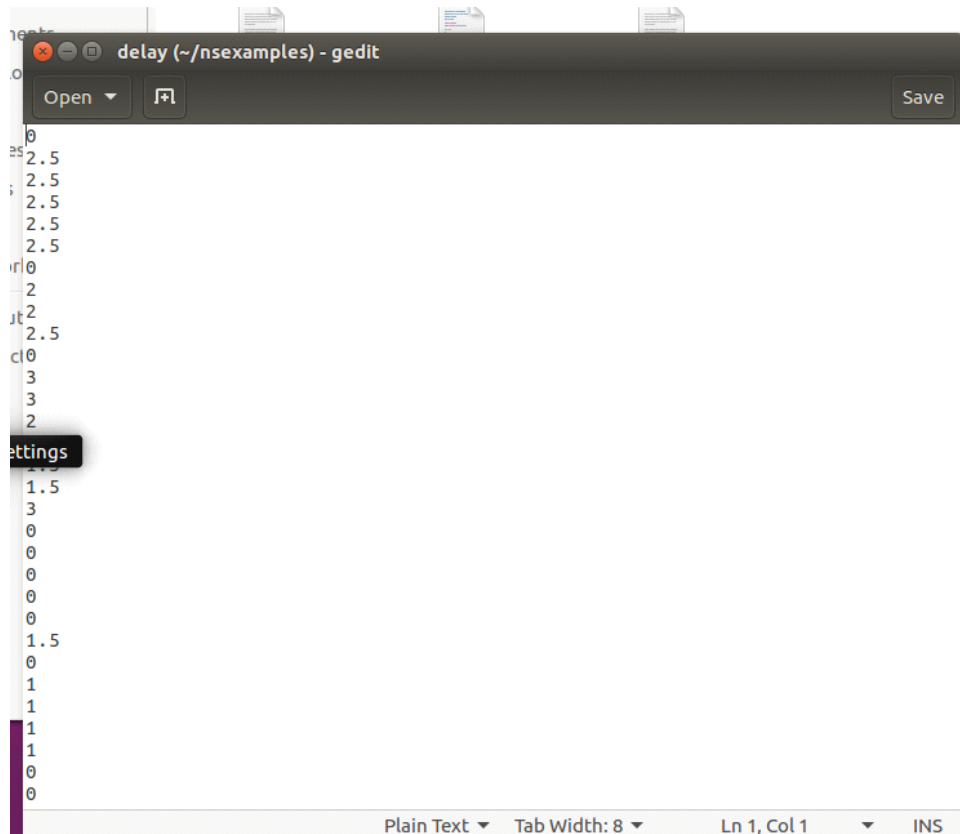
直接通过apt-get安装gawk即可。

在ns-allinone安装目录中附带有一些awk代码, 例如delay.awk可以用来分析时延信息。

```
reed@ubuntu:~/nsexamples$ gawk -f delay.awk flooding.tr>delay
```

上述命令将处理flooding.tr文件并将生成的时延信息输出到delay文件中。





也可以自行编写awk代码进行信息分析。

安装gnuplot时需要注意安装gnuplot-x11附件，否则会运行出错。

```
sudo apt-get install gnuplot gnuplot-x11
```

安装gnuplot之后

进入gnuplot进行绘图。

1.2-4iptables配置

2021年5月16日 20:11

iptables是linux平台下应用软件，是一种包过滤防火墙软件，通过控制linux内核中的**netfilter**模块，可以完成封包过滤、封包重定向和网络地址转换等功能。

Linux 的 Netfilter 可以实现如下几个功能：

- 拒绝让 Internet 的封包进入主机的某些端口
- 拒绝让某些来源 IP 的封包进入
- 拒绝让带有某些特殊旗标 (flag) 的封包进入
- 分析硬件地址 (MAC) 来决定联机与否

iptables中存在“表”、“链”、“规则”三个层次。每个“表”指的是不同类型的数据包处理流程，如filter表表示进行数据包过滤，而nat表针对连接进行地址转换操作。每个表中又可以存在多个“链”，系统按照预订的规则将数据包通过某个内建链，例如将从本机发出的数据通过OUTPUT链。在“链”中可以存在若干“规则”，这些规则会被逐一进行匹配，如果匹配，可以执行相应的动作，如修改数据包，或者跳转。跳转可以直接接受该数据包或拒绝该数据包，也可以跳转到其他链继续进行匹配，或者从当前链返回调用者链。当链中所有规则都执行完仍然没有跳转时，将根据该链的默认策略（“policy”）执行对应动作；如果也没有默认动作，则是返回调用者链。

iptables软件中有多个**表**，每个表都定义出自己的默认规则，每个表的用处都有所不同。linux中的iptables至少有三个表，包括：

- filter (过滤器)：主要跟进入 Linux 本机的封包有关，表中有多个链，包括：
 - INPUT：发往本机的数据包通过该链
 - OUTPUT：从本机发出的数据包通过该链
 - FORWARD：本机转发的数据包通过此链。
- nat (地址转换)：这个表格主要进行地址转换操作。表中的链包括：
 - PREROUTING：在进行路由判断之前通过的链，通常用于目的地址转换(DNAT/REDIRECT)
 - POSTROUTING：在进行路由判断之后通过的链，通常用于源地址转换(SNAT)
 - OUTPUT：处理本机发出的数据包
- mangle (破坏者)：这个表格主要是与特殊的封包的路由旗标有关，早期仅有 PREROUTING 及 OUTPUT 链，不过从 kernel 2.4.18 之后加入了 INPUT 及 FORWARD 链。这个表格在一般环境中较少使用。
- raw
 - raw表用于处理异常，有如下两个内建链：
 - PREROUTING
 - OUTPUT

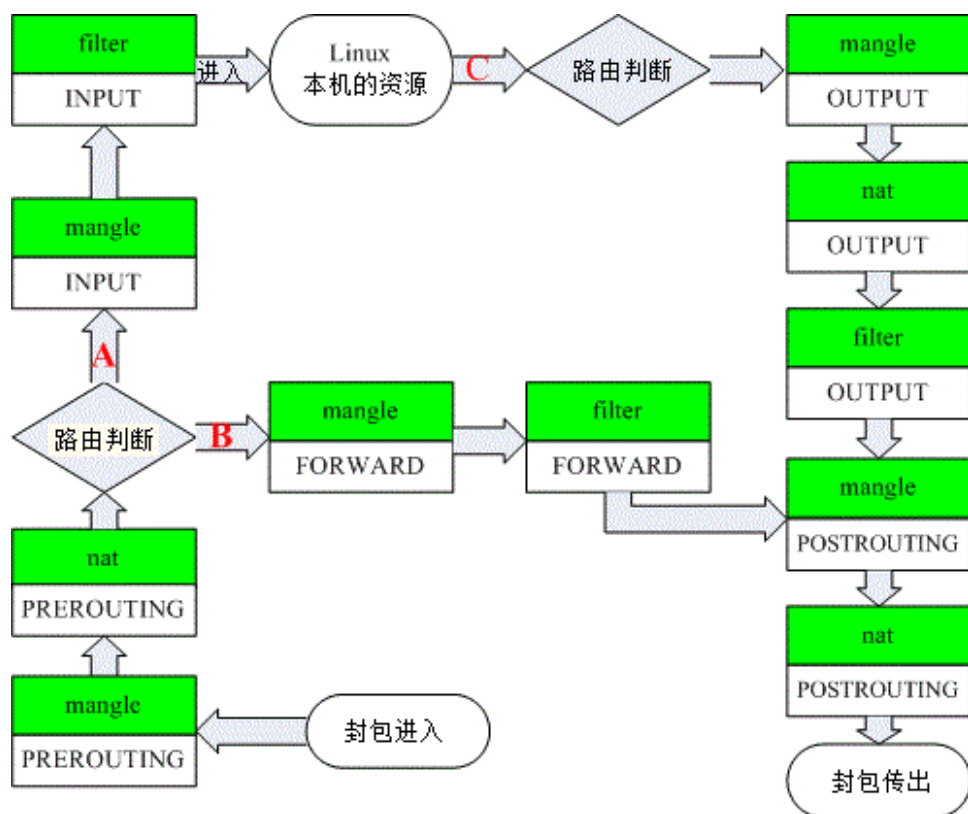


图1 -iptables 内建各表与链的相关性

接下来是实际操作的环节。

iptables的基本语法格式

```
iptables [-t 表名] 命令选项 [链名] [条件匹配] [-j 目标动作或跳转]
```

说明：表名、链名用于指定 iptables命令所操作的表和链，命令选项用于指定管理iptables规则的方式（比如：插入、增加、删除、查看等；条件匹配用于指定对符合什么样 条件的数据包进行处理；目标动作或跳转用于指定数据包的处理方式（比如允许通

命令选项包括：

- A 在指定链的末尾添加（append）一条新的规则
- D 删除（delete）指定链中的某一条规则，可以按规则序号和内容删除
- I 在指定链中插入（insert）一条新的规则，默认在第一行添加
- R 修改、替换（replace）指定链中的某一条规则，可以按规则序号和内容替换
- L 列出（list）指定链中所有的规则进行查看
- E 重命名用户定义的链，不改变链本身
- F 清空（flush）
- N 新建（new-chain）一条用户自己定义的规则链

- X 删除指定表中用户自定义的规则链 (delete-chain)
- P 设置指定链的默认策略 (policy)
- Z 将所有表的所有链的字节和数据包计数器清零
- n 使用数字形式 (numeric) 显示输出结果
- v 查看规则表详细信息 (verbose) 的信息
- V 查看版本(version)
- h 获取帮助 (help)

处理数据包的方式包括：

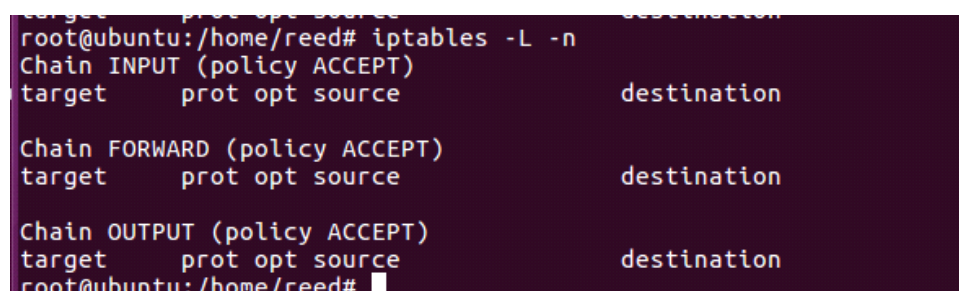
- ACCEPT 允许数据包通过
- DROP 直接丢弃数据包，不给任何回应信息
- REJECT 拒绝数据包通过，必要时会给数据发送端一个响应的信息。
- LOG在/var/log/messages文件中记录日志信息，然后将数据包传递给下一条规则

首先，**查看本机当前的防火墙规则：**

```
iptables [-t tables] [-L] [-nv]
```

选项与参数：

- t：后面接 table，例如 nat 或 filter，若省略此项目，则使用默认的 filter
- L：列出目前的 table 的规则
- n：不进行 IP 与 HOSTNAME 的反查，显示讯息的速度会快很多！
- v：列出更多的信息，包括通过该规则的封包总位数、相关的网络接口等



```

root@ubuntu:/home/reed# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@ubuntu:/home/reed#

```

目前的filter表中是没有防火墙规则的.....，显示的每个chain就代表一个链。其他项目的意义如下：

- target：代表进行的动作，ACCEPT 是放行，而 REJECT 则是拒绝，此外，尚有 DROP (丢弃) 的项目！
- prot：代表使用的封包协议，主要有 tcp, udp 及 icmp 三种封包格式；
- opt：额外的选项说明
- source：代表此规则是针对哪个『来源 IP』进行限制？
- destination：代表此规则是针对哪个『目标 IP』进行限制？

定义预设政策

预设政策指的是，当数据封包不在设定的规则的制定范围之内时，对封包的处理方式以预设政策为准。

例：设定丢弃所有发往本机的数据包，其他数据包允许通过

```
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
```

针对某条链的数据封包的处理规则设定

```
iptables [-A|链名] [-i|网络接口] [-p 协议] [-s 来源IP/网域] [-d 目标IP/网域] -j
[ACCEPT|DROP|REJECT|LOG]
```

选项与参数说明：

-A 链名：针对某的链进行规则的 "插入" 或 "累加"

-A：新增加一条规则，该规则增加在原本规则的最后面。例如原本已经有四条规则，使用 -A 就可以加上第五条规则！

-I：插入一条规则。如果没有指定此规则的顺序，默认是插入变成第一条规则。

例如原本有四条规则，使用 -I 则该规则变成第一条，而原本四条变成 2~5 号

-i|网络接口：设定封包进出的接口规范

-i：封包所进入的那个网络接口，例如 eth0, lo 等接口。需与 INPUT 链配合；

-o：封包所传出的那个网络接口，需与 OUTPUT 链配合；

-p 协定：设定此规则适用于哪种封包格式

主要的封包格式有：tcp, udp, icmp 及 all。

-s 来源 IP/网域：设定此规则之封包的来源项目，可指定单纯的 IP 或包括网域，例如：

IP：192.168.0.100

网域：192.168.0.0/24, 192.168.0.0/255.255.255.0 均可。

若规范为『不许』时，则加上 ! 即可，例如：

-s ! 192.168.100.0/24 表示不许 192.168.100.0/24 之封包来源；

-d 目标 IP/网域：同 -s，只不过这里指的是目标的 IP 或网域。

-j：后面接动作，主要的动作有接受(ACCEPT)、丢弃(DROP)、拒绝(REJECT)及记录(LOG)

例如：

设定接受所有内部循环测试接口 (lo) 的数据封包：

```
iptables -A INPUT -i lo -j ACCEPT
```

设定接受所有来自内网 (192.168.116.0/24) 的数据封包：

```
iptables -A INPUT -i ens3-3 -s 192.168.116.0/24 -j ACCEPT
```

设定丢弃来自192.168.116.110的所有数据封包：

```
iptables -A INPUT -i ens33 -s 192.168.116.110 -j DROP
```

设定丢弃所有进入端口21的TCP数据包：

```
iptables -A INPUT -i ens33 -p tcp --dport 21 -j DROP
```

设定丢弃所有来自端口1~1000的，**主动联机（带有syn标识）**进入端口1~1000的TCP数据包：

```
iptables -A INPUT -i ens33 -p tcp --sport 1:1023 --dport 1:1023 --syn -j DROP
```

通过数据封包的状态模块判断处理方式：

```
iptables -A INPUT [-m state] [--state 状态]
```

--state：一些封包的状态，主要有：

INVALID：无效的封包，例如数据破损的封包状态

ESTABLISHED：已经连接成功的封包状态；

NEW：想要新建立连接的封包状态；

RELATED：表示这个封包是主机发送出去的

例如：设定丢弃所有无效封包，接受已经连接成功的封包或者主机发送出去的封包：

```
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A INPUT -m state --state INVALID -j DROP
```

ICMP数据封包的处理规则

ICMP数据包有很多种类型，很大一部分是被用于网络监测的。其中type 8 (echo request) 包如果被设定为丢弃，远程主机就不会知道我们主机的存在（也不会相应ping命令）在设定ICMP数据封包处理规则时，可以添加ICMP类型参数。

```
iptables -A INPUT [-p icmp] [--icmp-type 类型] -j ACCEPT
```

--icmp-type：ICMP的封包类型，也可以使用代号，例如8代表echo request。

丢弃echo request数据包：

```
iptables -A INPUT -i ens33 -p icmp --icmp-type 8 -j DROP
```

```
root@ubuntu:/home/reed# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination            icmp echo-request
DROP       icmp -- anywhere          anywhere              

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

<https://zh.wikipedia.org/wiki/Iptables>

<https://www.cnblogs.com/wolfboy55-163-com/p/8149632.html>

<https://blog.csdn.net/u013733747/article/details/80875993>

<https://www.cnblogs.com/JemBai/archive/2009/03/19/1416364.html>

http://cn.linux.vbird.org/linux_server/0250simple_firewall.php>