

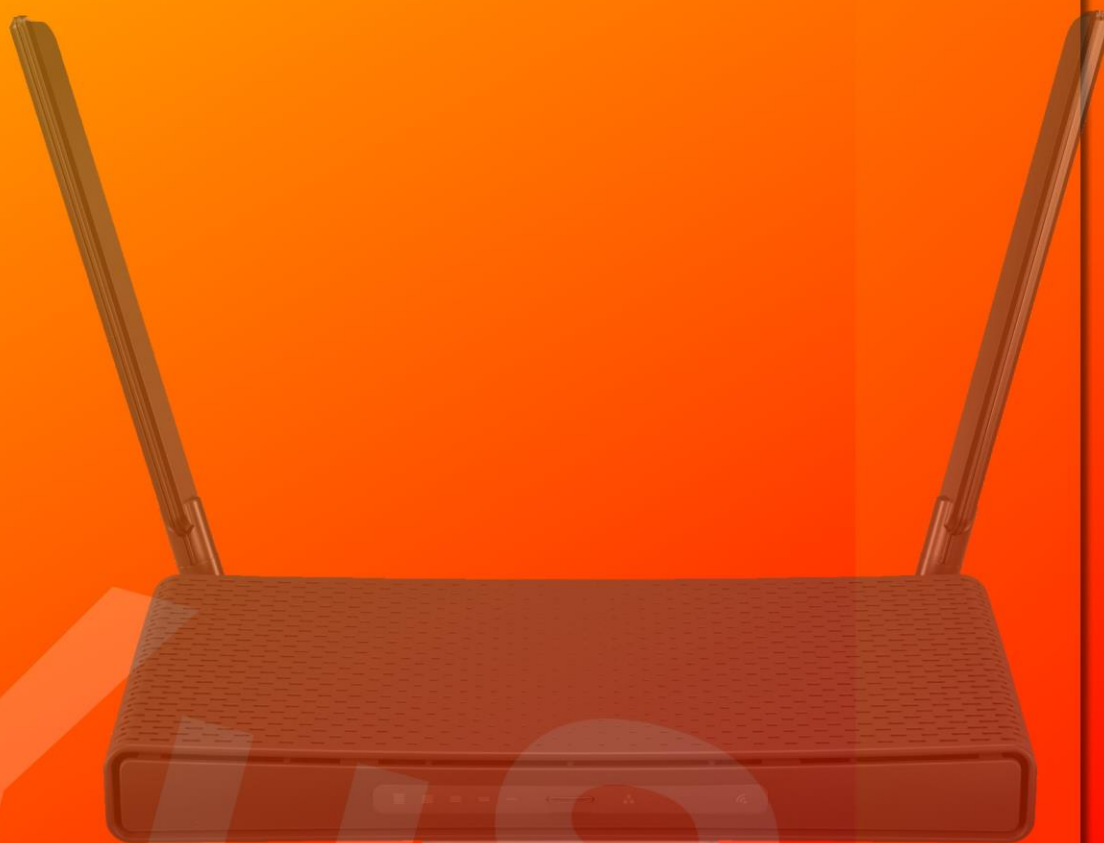


RouterOS入门到精通 v7.4e

- RouterOS v7中文教程

#include "RouterOS入门到精通v6"

#include "MikroTik CRS3系列交换机配置手册"



编：余松

www.irouters.com

前言

关于《RouterOS 入门到精通 v7e》教程，是独立介绍 RouterOS v7 版本的功能，大部分功能和基础内容是建立在《RouterOS 入门到精通 v6e》版本，大多的功能需要单独参考《RouterOS 入门到精通 v6》版本！

v7 主要对路由性能和配置做了重大改动，引入了其他一些新增功能如 Wireguard、Zerotier、L2TPv3、Docker、User Manager v5 和 GPIO 等。《RouterOS 入门到精通 v7e》教程除了介绍新功能外，我个人计划将引入更多的配置案例介绍。

版本: v7.4e
适用: RouterOS v7
作者: 余松
Web: www.irouters.com
E-mail: athlon_sds@163.com
如有更新，恕不通知！
该手册由本人多年整理编写而成，请勿非法篡改或其他商业用途！

目 录

前 言	2
目 录	3
RouterOS v7 升级内容	8
第一章 RouterOS v7 路由	10
1.1 路由协议多任务支持	10
BGP Sub-Tasks.....	10
1.2 Routing Tables.....	11
1.3 查看路由	12
1.4 ECMP 配置改动	14
1.5 v7 策略路由配置实例.....	15
1.6 v7 官方 PCC 配置	17
1.7 RIP	19
RIP 配置实例	20
1.8 OSPF	22
Redistribution 路由重分布.....	22
OSPF 配置实例	23
PPTP 隧道的 OSPF 建立	25
多种隧道协议建立 OSPF 多路径等价路由	27
1.9 BGP.....	32
v7 BGP 改进说明	32
CPU 亲和参数.....	32
BGP 建立.....	33
BGP 路由过滤.....	34
BGP 路由通过 address-list 发布路由到 routing table	35
1.10 Routing Filters	39
Routing Filter 语法.....	40
第二章 WireGuard.....	42
2.1 WireGuard 配置实例.....	44
防火墙配置.....	47
2.2 Windows 与 RouterOS 的 Wireguard 连接,.....	48

基本配置	48
Windows 配置	49
RouterOS 代理上网配置	55
2.3 基于 WirGuard 的 EoIP 隧道	56
2.4 与 Ubuntu20.04 建立 WireGuard	59
RB5009 配置	60
Ubuntu 20.04 配置	63
2.5 Wireguard 的多节点组网	68
基础网络配置.....	69
Wireguard 连接	70
OSPF 配置	72
BGP 配置.....	73
第三章 Zerotier	75
3.1 本地网络与网关配置建议.....	75
3.2 RouterOS 配置实例	77
第四章 Container（Docker）	80
4.1 启用 Container 模式	80
4.2 创建网络	81
路由模式	81
桥接模式	82
4.3 添加环境变量和 mount.....	83
4.4 添加 container 镜像	83
1、外部库获取镜像.....	84
2、从 PC 导入镜像	84
3、在 PC 创建镜像	85
4.5 container 启动与设置	86
Docker 的端口映射	86
Shell.....	87
RAM 限制.....	87
开机启动	87
4.6 安装 nginx（v7.4beta4）	88
4.7 hAP ac3 安装 DNSMasq.....	91
第五章 IoT	97

5.1 GPIO	97
analog.....	98
digital	99
控制继电器.....	100
监测输入信号.....	101
第六章 User Manager v5.....	102
6.1 Database（数据库）	102
6.2 Attributes（字段属性）	103
6.3 User Manager v5 配置实例	104
启用 RADIUS	104
配置 User Manager.....	107
6.4 WiFi 创建 802.1x EAP 认证	113
cAP ac 端配置	113
RB5009 配置	119
Windows 连接 WiFi	122
第七章 Certificate 证书.....	124
7.1 证书简单操作	124
证书模板	124
签发证书	125
导出证书	125
7.2 Let's Encrypt 证书.....	126
7.3 创建 SSTP 证书.....	127
生成证书	128
配置 SSTP-server 证书.....	134
导出证书	135
导入 windows.....	136
第八章 Dot1x（基于以太网 802.1x）	144
8.1 客户端	144
8.2 服务器	145
8.3 功能与实例	147
RADIUS 配置	148
基于 VLAN 的端口分配.....	148
RouterOS 客户端配置.....	149

端口 VLAN ID 分配	150
动态 switch rule 配置	151
8.4 Dot1x 组网配置实例	153
签发证书	154
RB5009 配置	155
CRS326 交换机配置	155
hEX 客户端配置	156
第九章 VXLAN	157
9.1 基本属性	157
9.2 Forwarding table	159
9.3 VXLAN 配置实例	159
第十章 L2TPv3	162
10.1 L2TPv3 以太网桥接配置实例	163
第十一章 Branding 定制	169
11.1 ASCII logo	170
11.2 默认页面	170
11.3 其他参数	171
第十二章 Device-Mode	172
12.1 Device-mode 基本配置	172
12.2 Flagged 状态	175
第十三章 ROSE-存储功能	177
13.1 RAID 配置	177
13.2 iSCSI	178
13.3 NFS	178
13.4 SMB	179
13.5 NVMe over TCP	179
13.6 RAMdisk	180
13.7 数据加密	181
Self-Encrypting Drives	181
13.8 File Sync	181
13.9 SMB 与 NFS 配置实例	182
第十一章 各类应用案例	186
14.1 RouterOS 在 VPS 上的隧道应用	186

弹性 IP 地址分配通过 proxy-arp:	186
EoIP 建立二层隧道	188
Changelog	191
参考文献:	192

RouterOS v7 升级内容

RouterOS v7 是从 2020 年开始发布，从 7.0beta 版本开始，到 7.1beta，然后到 7.1rc 版本，2021 年 12 月 2 日发布了 v7.1 版本，在 2021 年 12 月 22 日，有继续发布了 7.2rc1 版本，v6 版本基本确认到 2022 年停更，2022 年将主要更新放在了 RouterOS v7。RouterOS v7 主要升级内容包括以下几个方面：

主要改变：

1. 使用 5.6.3 的 Linux 内核
2. 重新编写的 NTP Client 与 Server 服务器；
3. 全新的 User Manager v5；
4. 将更多的功能合并到一个基础 **routeros** 功能包（应该是更多功能被写入到 Linux 内核中），仅将较少的功能包保留在外部选择安装，如 User Manager、zerotier、GPS 和 IoT 等功能包需要单独安装（扩展包放在 Extra packages）；
5. 新的命令行格式（仍然兼容 v6 的命令）；
6. 支持 Let's Encrypt 证书生成；
7. 支持 REST API；
8. 支持 x86 的 UEFI 引导模式
9. 支持 Container 容器
10. 新增 ROSE 存储功能

网络部分

1. CHR FastPath 支持“vmxnet3”和“virtio-net”驱动；
2. 支持“Cake”和“FQ_Codel”类型队列；
3. 支持 IPv6 NAT；
4. 在所有 CRS3 系列交换机支持三层硬件加速
5. 支持 MBIM 驱动程序的基本功能，支持所有 MBIM 模式的调制解调器；
6. CRS3 系列交换机支持 MLAG 功能；
7. 支持 VRRP 分组和节点间连接跟踪数据同步；
8. 支持 VxLAN

路由

1. 全新的 BGP，并对性能进行改进；
2. 全新的 IPv6 stack
3. 全新的 MPLS，接口列表，多路径和 LDPv6 支持；
4. 全新的 OSPF，并对性能进行改进；
5. 全新的路由过滤（routing filter），类似脚本的规则语法，支持 RPKI 和扩展 community 过滤；
6. 支持 IPv6 ECMP 和 VRF(包括 VRF-lite)；
支持 IPv6 递归路由和策略路由；

VPN

1. 支持 L2TPv3；
2. 支持 OpenVPN UDP 传输协议；
3. 支持 WireGuard；

4. 支持基于 ARM 和 ARM64 设备的 ZeroTier;

无线

1. 全新的 wireless 无线功能包（替代版本）支持 802.11ac Wave2、WPA3 和 802.11w 管理帧保护支持(需要 ARM CPU 和 256MB RAM);

RouterOS 经过 20 多年的发展，给我的感觉越来越像是一个 Linux 的工具整合平台，各种 Linux 的网络应用都在往里面加，好似披着路由器外壳的 Linux 工具箱！

虽然 2021 年 12 月官方强行将 bug 很多的 v7.1 作为发行版本，目的也是想 2022 年开启 RouterOSv7 的天下，我个人对 RouterOS v7 将来的发展还是很看好的，首先从安装功能包上看，已经没有像 v6 那样有很多功能包需要进行独立安装，而是集成到了一个 routeros 基础功能包中，也就是很多以前的功能很可能被写入了 Linux 内核，性能会得到提升！v7 当前在外的 npk 功能包只有 7 个，分别是 container、calea、gps、iot、tr069、user-manager 和 zerotier。而 v6 功能包算上 system 一共有 20 个。

其次在官方在其 [v7 宣传视频](#)中，透露了使用 Linux 内核 5.6.3(相对当前 Linux 版本算是较新的内核)，因此在内核方面有更多硬件支持和新功能扩展，为将来实现更多功能打下基础。

关于 CRS3 系列交换机，L3-hardware-offload 和相关的其他 v7 功能，请参阅 [MikroTik CRS3 系列交换机操作配置手册](#)

第一章 RouterOS v7 路由

路由部分是 RouterOS v7 最大的改动，和之前的所有版本都不同。不仅仅是路由表的创建方式，还包括路由表查询、RIP、OSPF、BGP 和 Routing filter 路由过滤等功能，几乎全部被改动，特别是 OSPF、BGP 和路由过滤配置方式和之前版本完全不同。

1.1 路由协议多任务支持

RouterOS v7 能够从一个路由进程中拆分多个任务。通过一个“main”任务，它可以启动/停止子任务，并在这些子任务之间进行处理。每个子任务可以分配“私有”(只有该任务可以访问)和“共享”内存(所有路由任务可以访问)。

可拆分的任务如下列表：

- "print"命令处理；
- 整个 OSPF 协议处理；
- 整个 RIP 协议处理；
- 静态路由配置处理；
- 策略路由配置；
- BGP 连接和配置处理；
- BGP 接收(每个对等体一个任务或按特定参数分组)；
- BGP 发送 (每个对等体一个任务或按特定参数分组)；
- FIB 更新处理。

BGP Sub-Tasks

BGP 的接收和发送可以根据特定的参数划分为子任务 sub-tasks，例如，为每个输入 Peer 运行，或是将所有输入的 Peer 分组在主进程中运行。

在/routing/bgp/template 中，通过“input.affinity”和“output.affinity”参数配置来控制子任务。这可以通过在内核较少的设备上使用亲和值来提高性能，因为在任务之间共享数据比在一个任务中处理相同的数据要慢。例如，在单核或双核心设备上，在主进程或实例进程中运行输入和输出将提高性能。

当前的任务及其分配的私有/共享内存可以使用以下命令查看：

```
[admin@MikroTik]/routing/stats/process/print
```

输出样本：

```
[admin@BGP_MUM] /routing/stats/process> print interval=1
Columns: TASKS, PRIVATE-MEM-BLOCKS, SHARED-MEM-BLOCKS, PSS, RSS, VMS, RETIRED, ID, PID, RPID, PROCESS-TIME, KERNEL-TIME, CUR-I
# TASKS PRIVATE-M SHARED-M PSS RSS VMS R ID PID R PROCESS- KERNEL-TI
0 routing tables 11.8MiB 20.0MiB 19.8MiB 42.2MiB 51.4MiB 7 main 195 0 15s470ms 2s50ms
  rib
  connected networks
1 fib 2816.0KiB 0 8.1MiB 27.4MiB 51.4MiB fib 255 1 5s730ms 7m4s790ms
2 ospf 512.0KiB 0 3151.0KiB 14.6MiB 51.4MiB ospf 260 1 20ms 100ms
  connected networks
3 fantasy 256.0KiB 0 1898.0KiB 5.8MiB 51.4MiB fantasy 261 1 40ms 60ms
4 configuration and reporting 4096.0KiB 512.0KiB 9.2MiB 28.4MiB 51.4MiB static 262 1 3s210ms 40ms
5 rip 512.0KiB 0 3151.0KiB 14.6MiB 51.4MiB rip 259 1 50ms 90ms
  connected networks
6 routing policy configuration 768.0KiB 768.0KiB 2250.0KiB 6.2MiB 51.4MiB policy 256 1 70ms 50ms
7 BGP service 768.0KiB 0 3359.0KiB 14.9MiB 51.4MiB bgp 257 1 4s260ms 8s50ms
  connected networks
8 BFD service 512.0KiB 0 3151.0KiB 14.6MiB 51.4MiB 12 258 1 80ms 40ms
  connected networks
9 BGP Input 10.155.101.232 8.2MiB 6.8MiB 17.0MiB 39.1MiB 51.4MiB 20 270 1 24s880ms 3s60ms
  BGP Output 10.155.101.232
10 Global memory 256.0KiB global 0 0
```

1.2 Routing Tables

RouterOS v7 与之前版本在路由操作上最大的不同在于创建新路由表，之前的版本创建新路由表（路由标记）可以在 **ip route**、**ip route rules** 和 **ip firewall mangle** 三处位置，而 **v7** 只能在 **/routing/table** 下创建，如果路由表需要推送给 **FIB**，需设置 **fib** 参数。根据官方的意思，统一在 **/routing/table** 下创建，原因是避免误操作，引起的配置错乱，这个做法我非常认可。

路由表在 Linux 系统中，会有一个默认路由表，可以称作“main”主路由表，同时还能创建多个路由表（**/etc/iproute2/rt_tables**），为指定的 IP 和网络协议提供独立的路由查询。同样使用 linux 系统的 RouterOS 在 **/ip/route** 中可查看到 **main** 路由表，创建新的路由表目的是实现相关的策略路由，将指定的网络路由参数放到新建的路由表查询，与 **main** 路由表独立开，特别记住路由器的直连路由（**connect route**）的路由查询是在 **main** 表完成的。

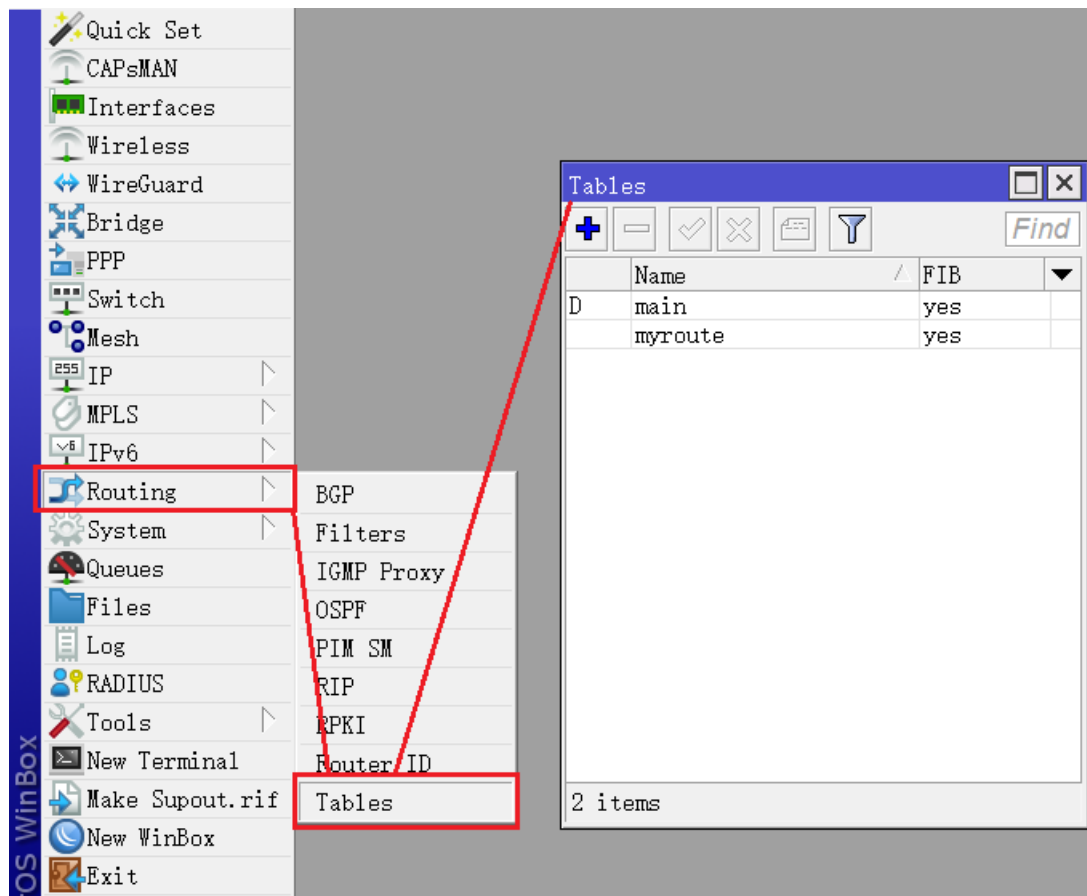
新建路由表无法查询直连路由导致的结果就是：路由器配置了多个 IP 地址段时，被策略的 IP 无法访问到路由器本地 IP 地址段（包括 PCC）。华为路由交换设备配置了策略路由（PBR）也是如此！

RouterOS v7 创建策略路由，如将 8.8.8.8 的 DNS 指定到路由表 myTable 查询，网关为 172.16.1.1

```
[admin@MikroTik] /routing/table add name=myTable fib
[admin@MikroTik] /routing/rule add dst-address=8.8.8.8 action=lookup-only-in-table table=myTable
[admin@MikroTik] /ip/route add gateway=172.16.1.1 routing-table=myTable
```

在 **/routing/table** 创建的路由表后，在 **mangle** 创建路由规则 **action=mark-routing**，就可以被 **new-routing-mark** 调用，仍然和 RouterOS v6 中一样。

在 Winbox 中，配置路径现在固定在 **/routing/table** 下



1.3 查看路由

v7 还引入了新的操作路径 `/routing/route`，这里显示了所有的路由协议的路由和路由属性。`/ip/route` 和 `/ipv6/route` 下仅用于静态路由的配置和显示。也就是 `/routing/route` 下用于静态和动态路由的显示和查询。

在 RouterOS 中，你有三个菜单可以查看路由表中的路由当前状态：

- `/ip route` – 显示 IPv4 路由的基本信息
- `/ipv6 route` – 显示 IPv6 路由基本信息
- `/routing route` – 显示所有路由的扩展信息

`/routing route` 目录当前只读属性，添加和删除路由仍然在 `/ip(ipv6) route` 目录下操作。

路由查看

```
[admin@MikroTik] /ip/route> print
```

Flags: D - dynamic; X - disabled, I - inactive, A - active; C - connect, S - static

c, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn

Columns: DST-ADDRESS, GATEWAY, Distance

#		DST-ADDRESS	GATEWAY	DI
0	XS	10.155.101.0/24	1.1.1.10	
1	XS		11.11.11.10	
	D d	0.0.0.0/0	10.155.101.1	10

```

2  AS  0.0.0.0/0          10.155.101.1      1
3  AS  1.1.1.0/24         10.155.101.1      10
4  AS  8.8.8.8            2.2.2.2           1
   DAC 10.155.101.0/24    ether12           0
|  |||  |                |                |
|  |||  |                |                \---- 路由距离
|  |||  |                \--网关
|  |||  \-- 目标前缀
|  ||\----- 协议标志(bgp, osf,static,connected.)
|  |\----- 路由状态标志(active, inactive, disabled)
|  \----- 如果动态路由显示 D
\----- 终端控制台序号(仅显示静态可编辑路由条目)

```

`/routing/route` 非常类似于 `/ip/route`，除了显示基本的路由信息，还会列出过滤路由信息

```

[admin@MikroTik] /routing/route> print
Flags: X - disabled, I - inactive, F - filtered, U - unreachable, A - active; c - connect, s - static,
r - rip, b - bgp, o - ospf, d - dhcp, v - vpn, a - ldp-address, l - ldp-mapping
Columns: DST-ADDRESS, GATEWAY, DISTance, SCOpe, TARget-scope, IMMEDIATE-GW
      DST-ADDRESS      GATEWAY      DIS  SCO  TAR IMMEDIATE-GW
Xs  10.155.101.0/24
Xs
d  0.0.0.0/0          10.155.101.1 10  30  10  10.155.101.1%ether12
As  0.0.0.0/0          10.155.101.1 1  30  10  10.155.101.1%ether12
As  1.1.1.0/24         10.155.101.1 10  30  10  10.155.101.1%ether12
As  8.8.8.8            2.2.2.2      1  254 254 10.155.101.1%ether12
Ac  10.155.101.0/24    ether12       0  10      ether12
lc  2001:db8:2::/64    ether2        0  10
lo  2001:db8:3::/64    ether12       110 20  10
lc  fe80::%ether2/64    ether2        0  10
Ac  fe80::%ether12/64    ether12       0  10      ether12
Ac  fe80::%bridge-main/64 bridge-main    0  10      bridge-main
A   ether12              0  250
A   bridge-main          0  250

```

在 `/routing/route` 目录下使用 `print detail`，会显示更多的信息用于调试

```

[admin@MikroTik] /routing route> print detail
Flags: X - disabled, I - inactive, F - filtered, U - unreachable, A - active;
c - connect, s - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn, a - ldp-address, l - ldp-ma>
+ - ecmp
Xs  dst-address=10.155.101.0/24
Xs
d  afi=ip4 contribution=best-candidate dst-address=0.0.0.0/0 gateway=10.155.101.1
   immediate-gw=10.155.101.1%ether12 distance=10 scope=30 target-scope=10

```

```
belongs-to="DHCP route" mpls.in-label=0 .out-label=0 debug.fwp-ptr=0x201C2000
```

```
As afi=ip4 contribution=active dst-address=0.0.0.0/0 gateway=10.155.101.1
   immediate-gw=10.155.101.1%ether12 distance=1 scope=30 target-scope=10
   belongs-to="Static route" mpls.in-label=0 .out-label=0 debug.fwp-ptr=0x201C2000
...
```

1.4 ECMP 配置改动

ECMP (Equal Cost Multi-Path)等价多路径路由，是到达相同目标地址有多个网关，在三层网络的多路径下路由较常用（非 nat 网络）。

在 RouterOS v6 配置，例如到达目标地址 192.168.2.0/24 有 3 条相同网关，可以在 winbox 中指定 2 个 Gateway:

在 RouterOS v7 的 winbox 中这个配置被取消，一条规则只能配置一个网关，相同目标 IP 需要创建多条路由规则。

这些路由既可以通过手工添加，也可以通过动态路由协议(OSPF、BGP、RIP)动态创建。对于 ECMP 到同一目的地址的多个相同优先级的路由，将被分配+标志，如下的例子：

```
[admin@MikroTik] /ip/route>add dst-address=192.168.2.0/24 gateway=10.155.125.1
[admin@MikroTik] /ip/route>add dst-address=192.168.2.0/24 gateway=172.16.1.2
[admin@MikroTik] /ip/route> print
Flags: D - DYNAMIC; I - INACTIVE, A - ACTIVE; C - CONNECT, S - STATIC, m - MODEM; + - ECMP
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#       DST-ADDRESS      GATEWAY      D
```

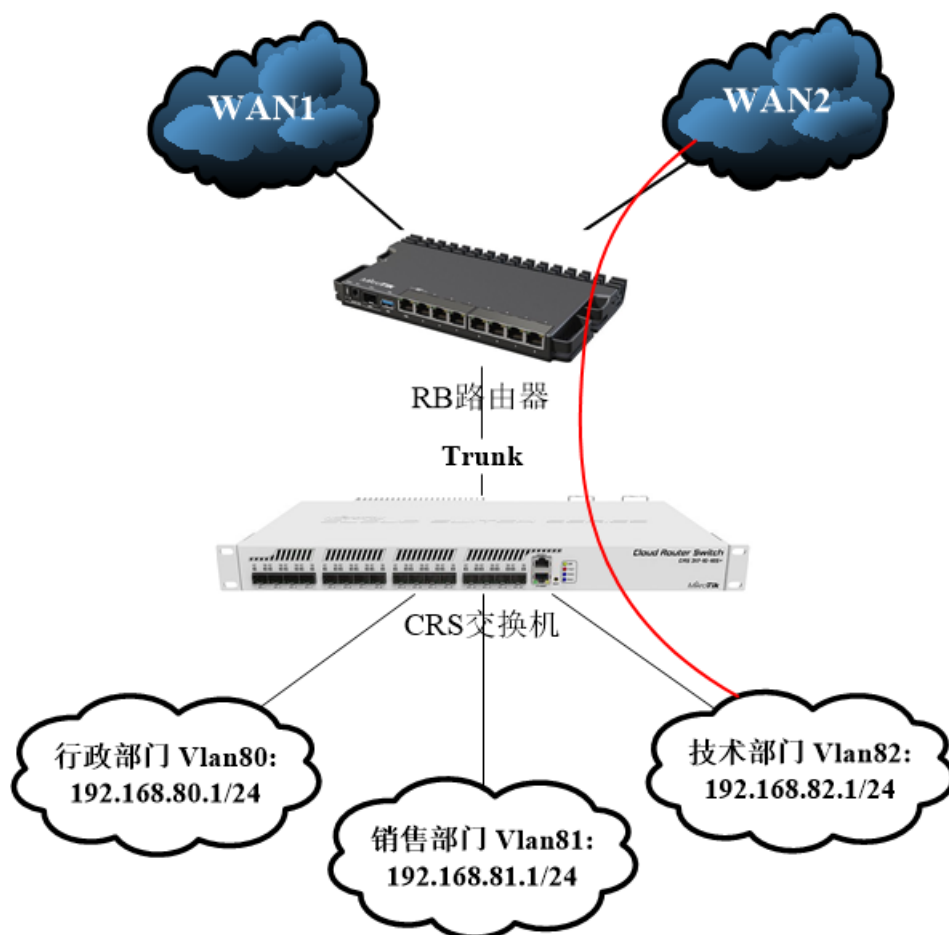
0	AS+ 192.168.2.0/24	10.155.125.1	1
1	AS+ 192.168.2.0/24	172.16.1.2	1

1.5 v7 策略路由配置实例

对于 RouterOS v7 创建策略路由，需新建路由表，在 /routing table 下进行操作，通过下面一个实例进行说明，在当前网络有多个内网 IP 地址的划分：

- 行政部门：192.168.80.1/24 接入 vlan80
- 销售部门：192.168.81.1/24 接入 vlan81
- 技术部门：192.168.82.1/24 接入 vlan82

外网有两条线路接入 wan1 和 wan2，采用 pppoe 拨号上网，需要将技术部门地址是 192.168.82.0/24，走地址段的走 wan2 出口



首先，为三个部门创建内网的三层 vlan 接口

```
[admin@MikroTik] > interface vlan
[admin@MikroTik] /interface/vlan > add name=vlan80 interface=sfp-sfpplus1 vlan-id=80
[admin@MikroTik] /interface/vlan > add name=vlan81 interface=sfp-sfpplus1 vlan-id=81
[admin@MikroTik] /interface/vlan > add name=vlan82 interface=sfp-sfpplus1 vlan-id=82
```

在 vlan80-82 接口上创建对应的内网 IP 地址

```
[admin@MikroTik] > ip add
[admin@MikroTik] /ip/address> add address=192.168.80.1/24 interface=vlan80
[admin@MikroTik] /ip/address> add address=192.168.81.1/24 interface=vlan81
[admin@MikroTik] /ip/address> add address=192.168.82.1/24 interface=vlan82
[admin@MikroTik] /ip/address> print
Flags: I, D - DYNAMIC
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS NETWORK INTERFACE
0 192.168.80.1/24 192.168.80.0 vlan80
1 192.168.81.1/24 192.168.81.0 vlan81
2 192.168.82.1/24 192.168.82.0 vlan82
3 D 10.10.10.2/32 10.10.10.1 pppoe-out1
4 D 10.20.20.2/32 10.20.20.1 pppoe-out2
```

创建内网的用户地址列表，取名为 lan

```
[admin@MikroTik] /ip/address> /ip firewall/address-list/
[admin@MikroTik] /ip/firewall/address-list> add list=lan address=192.168.80.0/24
[admin@MikroTik] /ip/firewall/address-list> add list=lan address=192.168.81.0/24
[admin@MikroTik] /ip/firewall/address-list> add list=lan address=192.168.82.0/24
[admin@MikroTik] /ip/firewall/address-list> print
Columns: LIST, ADDRESS, CREATION-TIME
# LIST ADDRESS CREATION-TIME
0 lan 192.168.80.0/24 dec/06/2021 10:22:15
1 lan 192.168.81.0/24 dec/06/2021 10:22:19
2 lan 192.168.82.0/24 dec/06/2021 10:22:21
[admin@MikroTik] /ip/firewall/address-list>
```

创建两条出口的 nat 规则

```
[admin@MikroTik] /ip/firewall/address-list> /ip firewall/nat/
[admin@MikroTik] /ip/firewall/nat> add src-address-list=lan out-interface=pppoe-out1 \
action=masquerade
[admin@MikroTik] /ip/firewall/nat> add src-address-list=lan out-interface=pppoe-out2 \
action=masquerade
```

进入 routing table 创建新的路由表 wan2

```
[admin@MikroTik] /ip/firewall/address-list>/routing table
[admin@MikroTik] /routing/table > add name=wan2 fib
```


在 mangle 通过 new-routing-mark 标记技术部门 IP 地址 192.168.82.0/24，并排除到目标地址是 LAN 地址列表，避免技术部门的 IP 在新 wan2 路由表无法查询到本地直连路由，排除后会到 main 路由表查询。

```
[admin@MikroTik] /routing/table >/ip firewall mangle
[admin@MikroTik] /routing/table >/ip firewall mangle
[admin@MikroTik] /ip/firewall/mangle>add chain=prerouting src-address=192.168.82.0/24
dst-address-list=!lan new-routing-mark=wan2
```

进入 ip route 指定 wan2 标记的路由表，走 pppoe-out2 出口

```
[admin@MikroTik] /ip/firewall/mangle>/ip route
[admin@MikroTik] /ip/route >add gateway=pppoe-out2 routing-table=wan2
```

通过 print where 过滤，查看 wan2 路由表，

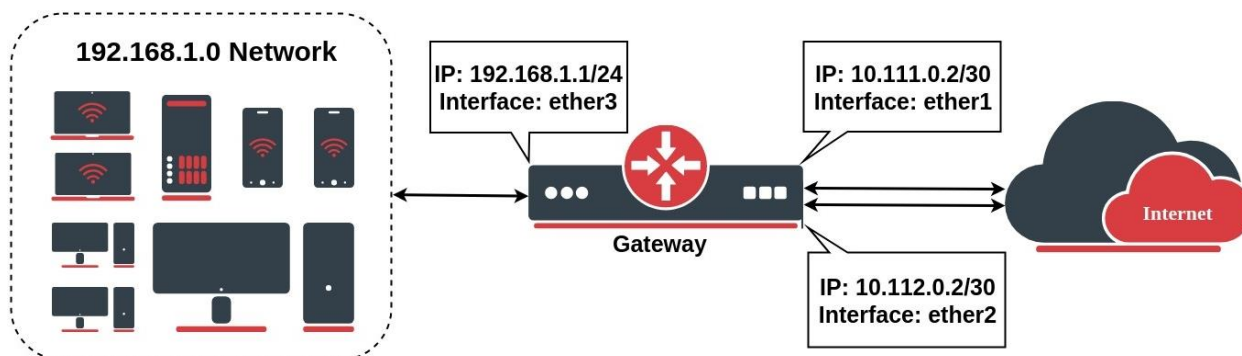
```
[admin@MikroTik] /ip/route> print where routing-table="wan2"
Flags: A - ACTIVE; s, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#   DST-ADDRESS  GATEWAY      DISTANCE
3 As 0.0.0.0/0    pppoe-out2    1
[admin@MikroTik] /ip/route>
```

这样 192.168.82.0/24 将走 pppoe-out2 出口

1.6 v7 官方 PCC 配置

提示：PCC 负载均衡在国内，仅适用于相同运营商的多线均衡！不同运营商受到不同路径和 DNS 影响会导致上网故障。

RouterOS v7 在路由策略配置方面有所变动，新建路由表，不能直接在 ip route rule 和 mangle 下创建，只能通过 routing table 配置，下面是网络拓扑结构图，接入两个外网固定 IP，一个内网 IP 段



IP 地址配置:

路由外网接入 ether1 和 ether2，地址分别是 10.111.0.2/24 和 10.112.0.2/24，ether3 作为内网 IP 地址为 192.168.1.1/24。

```
/ip address
add address=192.168.1.1/24 interface=ether3
add address=10.111.0.2/30 interface=ether1
add address=10.112.0.2/30 interface=ether2
```

Mangle 标记

通过 mangle 标记，并配置策略路由，可以强制指定流量到特定的网关，策略路由是创建新的路由表，但这样会导致路由器本地外网 IP 路由无法被查询，因为本地外网 IP 的路由查询是在 main 路由表，这样会导致内网 IP 段无法访问到路由器本地公网 IP 地址，路由器的 IP 仍然被发现外网出口的网关，这种方式将产生路由环路。为了避免这种情况，将允许外网接口 IP 地址使用 main 路由表进行路由查询：

```
/ip firewall mangle
add chain=prerouting dst-address=10.111.0.0/30 action=accept in-interface=ether3
add chain=prerouting dst-address=10.112.0.0/30 action=accept in-interface=ether3
```

需要注意上面两条规则必须是在所有规则之前优先处理，也就是在 mangle prerouting 链表里面的规则序列必须是 0 和 1

创建两组 PCC 策略，方式和之前 v6 一样，特别说明下：加入了一个参数 **connection-mark=no-mark**，用于标记未被标记的数据流，创建第一组 PCC 策略：

```
/ip firewall mangle
add chain=prerouting in-interface=ether1 connection-mark=no-mark action=mark-connection
new-connection-mark=ISP1_conn
add chain=prerouting in-interface=ether2 connection-mark=no-mark action=mark-connection
new-connection-mark=ISP2_conn
```

注意：定义 IP 数据包从那个接口进入，就按原路从那个接口回去，即保证每个外网口的数据能得到正确的路由，这里采用的是 prerouting 链表，不再使用 input 链表，有助于多线端口映射的配置。

RouterOS v7 不同于之前的所有版本，在标记路由之前，必须在 /routing/table 创建路由表

```
/routing/table
add fib name=to_ISP1
add fib name=to_ISP2
```

Action=mark-routing 只能用于链表（chain）为 output 和 prerouting，但 prerouting 用户抓取所有流量包括到路由器自身的，因此需要使用 dst-address-type=!local 来排除本地流量。按照源地址和目标地址来建立 PCC 规则，并标记 2 组连接，官方的配置采用的是 in-interface 为内网口 ether3，这个配置可以替换为源地址 src-address/src-address-list

```
/ip firewall mangle
add chain=prerouting in-interface=ether3 connection-mark=no-mark dst-address-type=!local
per-connection-classifier=both-addresses:2/0 action=mark-connection new-connection-mark=ISP1_conn
add chain=prerouting in-interface=ether3 connection-mark=no-mark dst-address-type=!local
per-connection-classifier=both-addresses:2/1 action=mark-connection new-connection-mark=ISP2_conn
```

给所有来自之前标记连接的数据包打上路由标记。入接口为内网口 **ether3**

```
/ip firewall mangle
add chain=prerouting connection-mark=ISP1_conn in-interface=ether3 action=mark-routing
new-routing-mark=to_ISP1
add chain=prerouting connection-mark=ISP2_conn in-interface=ether3 action=mark-routing
new-routing-mark=to_ISP2
add chain=output connection-mark=ISP1_conn action=mark-routing new-routing-mark=to_ISP1
add chain=output connection-mark=ISP2_conn action=mark-routing new-routing-mark=to_ISP2
```

NAT 配置，配置流量通过 **ether1** 和 **ether2** 外网口出去的 nat 伪装规则：

```
/ip firewall nat
add chain=srcnat out-interface=ether1 action=masquerade
add chain=srcnat out-interface=ether2 action=masquerade
```

添加 2 条 PCC 标记的策略路由

```
/ip route
add gateway=10.111.0.1@main routing-table=to_ISP1 check-gateway=ping
add gateway=10.112.0.1@main routing-table=to_ISP2 check-gateway=ping
```

如果你是 PPPoE 拨号的出口，拨号接口是 **pppoe-out1** 和 **pppoe-out2**，配置如下：

```
/ip route
add gateway=pppoe-out1@main routing-table=to_ISP1
add gateway=pppoe-out2@main routing-table=to_ISP2
```

多条线路的 PCC 配置规则较多，可以参考 [PCC 脚本生成器](#)（使用需注意 RouterOSv6，还是 v7 版本的 PCC 脚本）。

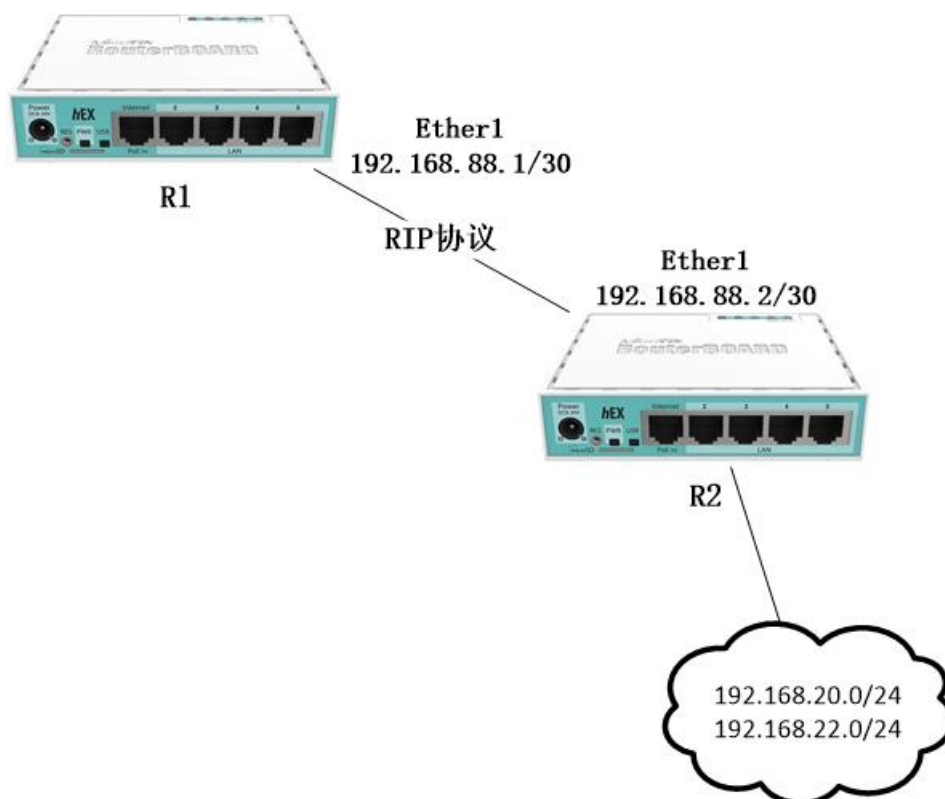
1.7 RIP

RIP 协议，在 RouterOS 被重新编写，采用了 **interface 模板+instance** 配置方式，因此与 v6 配置不同，在论坛上部分用户认为应该被 MikroTik 淘汰，而用 IS-IS 取代，但官方回复保留 RIP 是因为还有很多旧设备还在支持并使用 RIP 协议。

RIP 我就不多参数说明，通过一个实例给大家介绍配置方式，网络中有 3 台 RouterOS 设备，R1 和 R3 是 RouterOS v7 版本，R2 为 RouterOS v6 版本，7.1 和 v6.49.2 的 RIP 实际测试中发现，v6 向 v7 路由发布 RIP 的静态路由存在问题，v7 收到的路由条目网关不是对端路由器的直连 IP 地址，而是对端路由器静态路由的实际网关 IP，导致路由失效。

RIP 配置实例

如下网络拓扑图，两台路由器 R1 和 R2 通过 RIP 路由协议互联，R2 路由器的 ether2 下对接了 192.168.20.0/24 和 192.168.22.0/24 两网段，需要重分布给 R1：



R1 路由器创建 ether1 接口互联 IP 地址，互联 IP 地址建议使用/30 的子网掩码。

```
[admin@R1] >/ip address
[admin@R1] /ip address>add address=192.168.88.1/30 interface=ether1
```

进入 RIP 路径，配置 instance 实例接口

```
[admin@ R1] /ip/route>/routing rip instance
[admin@ R1] /routing/rip/instance >add disabled=no name=rip-instance-1
```

进入 interface 模板，添加 ether1 接口，并使用 rip-instance-1

```
[admin@ R1] /routing/rip/instance >/routing rip interface-template
[admin@ R1] /routing/rip/interface-template >add disabled=no instance=rip-instance-1 interfaces=ether1
```

R2 路由器配置

添加 R2 路由器的 ether1 和 ether2 的 IP 地址

```
[admin@R2] > ip address
[admin@R2] /ip/address> add address=192.168.88.2/30 interface=ether1
[admin@R2] /ip/address> add address=172.16.10.1/24 interface=ether2
```

配置静态路由

```
[admin@R2] /ip/address>/ip route
[admin@R2] /ip/route> add dst-address=192.168.20.0/24 gateway=172.16.10.2
[admin@R2] /ip/route> add dst-address=192.168.22.0/24 gateway=172.16.10.3
```

进入 rip 接口配置

```
[admin@R2] /ip/route>/routing rip
[admin@R2] /routing/rip>
```

配置 RIP 的 instance 实例，并配置重分布静态路由

```
[admin@R2] /routing/rip> /routing rip instance
[admin@R2] /routing/rip/instance >add disabled=no name=rip-instance-1 redistribute=static
```

配置接口模板

```
[admin@R2] /routing/rip/instance >/routing rip interface-template
[admin@R2] /routing/rip/interface-template >add disabled=no instance=rip-instance-1 interfaces=ether1
source-addresses=""
[admin@R2] /routing/rip/interface-template >
```

查看 R1 路由表

```
[admin@R1] /ip/route> print
Flags: D - DYNAMIC; A - ACTIVE; c, s, r, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
```

#	DST-ADDRESS	GATEWAY	DISTANCE
DAc	192.168.88.0/30	ether1	0
DAc	172.16.10.0/24	ether2	0
DAr	192.168.20.0/24	192.168.88.2	120
DAr	192.168.22.0/24	192.168.88.2	120

```
[admin@MikroTik] /ip/route>
```

1.8 OSPF

OSPFv3 和 OSPFv2 在 RouterOS v7 被合并到 `/routing ospf` 下管理。要同时启动 OSPF v2 和 OSPF v3 的 instance 实例，需要分别创建 instance 实例，如下：

```
/routing ospf instance
add name=v2inst version=2 router-id=1.2.3.4
add name=v3inst version=3 router-id=1.2.3.4
```

然后在 area，添加区域和对应的 instace 实例，配置如下

```
/routing ospf area
add name=backbone_v2 area-id=0.0.0.0 instance=v2inst
add name=backbone_v3 area-id=0.0.0.0 instance=v3inst
```

以上配置完成后，在 interface 中引用 area 的 instace 模板，连接对端的 OSPF。模板用于匹配运行 OSPF 协议的接口，可以通过指定 IP 网络或直接指定接口来实现（OSPF 会自动检测接口），下面是相关的配置示例：

```
/routing ospf interface
add network=192.168.0.0/24 area=backbone_v2
add network=2001:db8::/64 area=backbone_v3
add network=ether1 area=backbone_v3
```

v7 使用模板（templates）将接口与模板进行匹配，并应用匹配模板中的配置。OSPF 下的 interface 和 neighbor 目录包含只读条目，仅用于状态监控。

Redistribution 路由重分布

从 v7.1beta7 的重分布（redistribution）选项被重新加入（包括 RIP、BGP 协议），至于路由开销和类型的设置过滤仍然是通过 routing filter 完成。

```
[admin@MikroTik] /routing/ospf/instance> print
Flags: X - disabled, I - inactive
0 name="ospf-instance-1" version=2 vrf=main router-id=main routing-table=main
[admin@MikroTik] /routing/ospf/instance> set 0 redistribute=
bgp connected copy dhcp fantasy modem ospf rip static vpn
```

重分布包含多种类型，包括 bgp、connected、copy、dhcp、fantasy、modem、ospf、rip、static 和 vpn。

例如，只想重分布 IPv4 静态路由，地址范围为 192.168.0.0/16。首先在 `/routing/filter/rule` 定义 ospf_out 的过滤规则，

```
/routing filter rule add chain=ospf_out rule="if (dst in 192.168.0.0/16) {accept}"
```

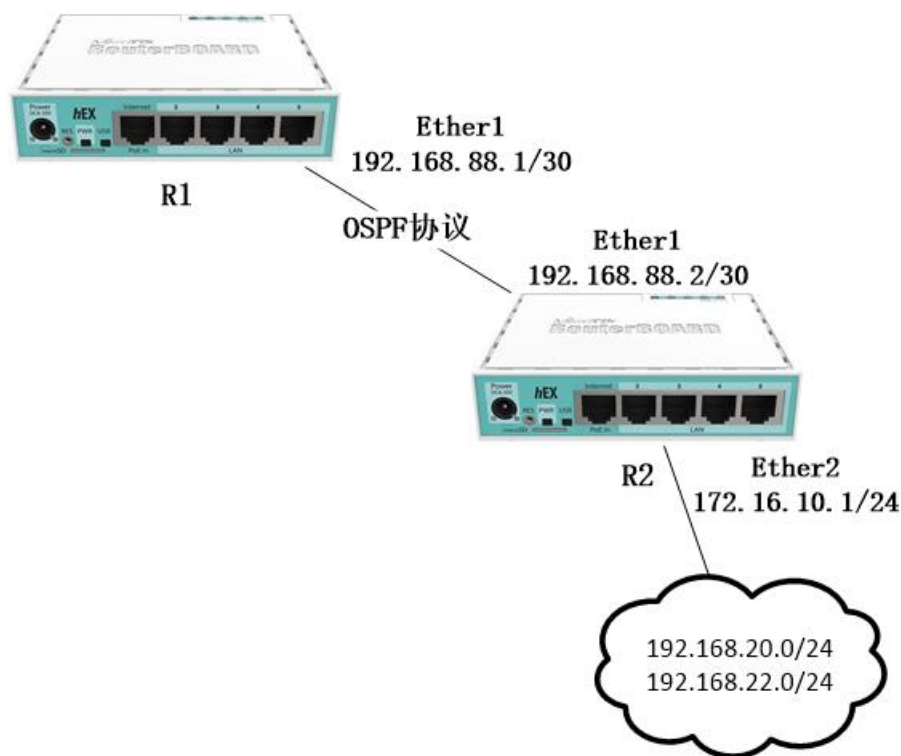
然后在 instance 中使用 redistribute 重分布静态路由，out-filter-chain 设置过滤规则 ospf_out

```
/routing ospf instance
set backbone_v2 out-filter-chain=ospf_out redistribute=static
```

"if (dst in 192.168.0.0/16) {accept}"，filter 的 rule 采用条件语法，默认的过滤 action 执行是“reject”。

OSPF 配置实例

还是继续使用之前 RIP 网络实例，作为 OSPF 实例介绍，仍然是 R2 重分布 192.168.20.0/24 和 192.168.22.0/24 两个网段



R1 路由器配置 OSPF

首先配置 instance 实例

```
[admin@R1]/routing/ospf/instance> add name=ospf-instance-1
```

配置 ospf area 的区域 0

```
[admin@R1]/routing/ospf/instance> /routing ospf area
[admin@R1] /routing/ospf/are > add instance=ospf-instance-1 name=ospf-area-0
```

创建 ospf 模板, 配置方式有两种, 一种基于接口, 一种基于 network, 同时设置也可以, 下面采用 network 方式配置 ospf 模板:

```
[admin@R1] /routing/ospf/are >/routing ospf interface-template
[admin@R1] /routing/ospf/interface-template > add area=ospf-area-0 networks=192.168.88.0/30
```

配置 R2 路由器 OSPF

首先配置 instance 实例

```
[admin@R2]/routing/ospf/instance> add name=ospf-instance-1 redistribute=static
```

配置 ospf area 的区域 0

```
[admin@R2]/routing/ospf/instance> /routing ospf area
[admin@R2] /routing/ospf/are > add instance=ospf-instance-1 name=ospf-area-0
```

创建 ospf 模板, 选择接口:

```
[admin@R2] /routing/ospf/are >/routing ospf interface-template
[admin@R2] /routing/ospf/interface-template > add area=ospf-area-0 networks=192.168.88.0/30
```

在 R1 的 neighbor 查看 ospf 状态:

```
[admin@R1] >/routing ospf neighbor
[admin@R1] /routing/ospf/neighbor> print
Flags: V - virtual; D - dynamic
 0  D instance=ospf-instance-1 area=ospf-area-0 address=192.168.88.2 priority=1
    router-id=192.168.88.1 dr=192.168.88.1 bdr=192.168.88.2 state="Full"
    state-changes=5 adjacency=29m3s timeout=39s
[admin@R1] /routing/ospf/neighbor>
```

可以看到 state 为 full, 代表完全邻接状态, OSPF 关系建立完成。

这里补充下 ospf 的 8 种状态分布是:Down 状态、Attempt 状态、Init 状态、Towway 状态、Exstart 状态、Exchange 状态、Loading 状态和 Full 状态。在 RouterOS 中一般只会显示 Init、Towway、Exstart 和 Full:

- **Init:** 表明收到了 Hello 包, 但是 2-Way 双向通信还未建立;
- **Towway:** 双向会话通信建立, RID 出现在对方的邻居列表中。(若为广播网络: 例如: 以太网。在这个时候应该选举 DR,BDR。)
- **ExStart:** 信息交换初始状态
- **Full:** 完全邻接状态, 邻接间的链路状态数据库同步完成

查看 R1 路由情况，从路由表可以看到两条 DAo 标记的路由 192.168.20.0/24 和 192.168.22.0/24，表明已经收到 R2 路由器发布的路由。

```
[admin@R1] /ip/route> print
Flags: D - DYNAMIC; A - ACTIVE; c, s, d, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#    DST-ADDRESS    GATEWAY    DISTANCE
DAc 192.168.88.0/30    ether1      0
DAc 172.16.10.0/24    ether2      0
DAo 192.168.20.0/24    192.168.88.2%ether1    110
DAo 192.168.22.0/24    192.168.88.2%ether1    110
```

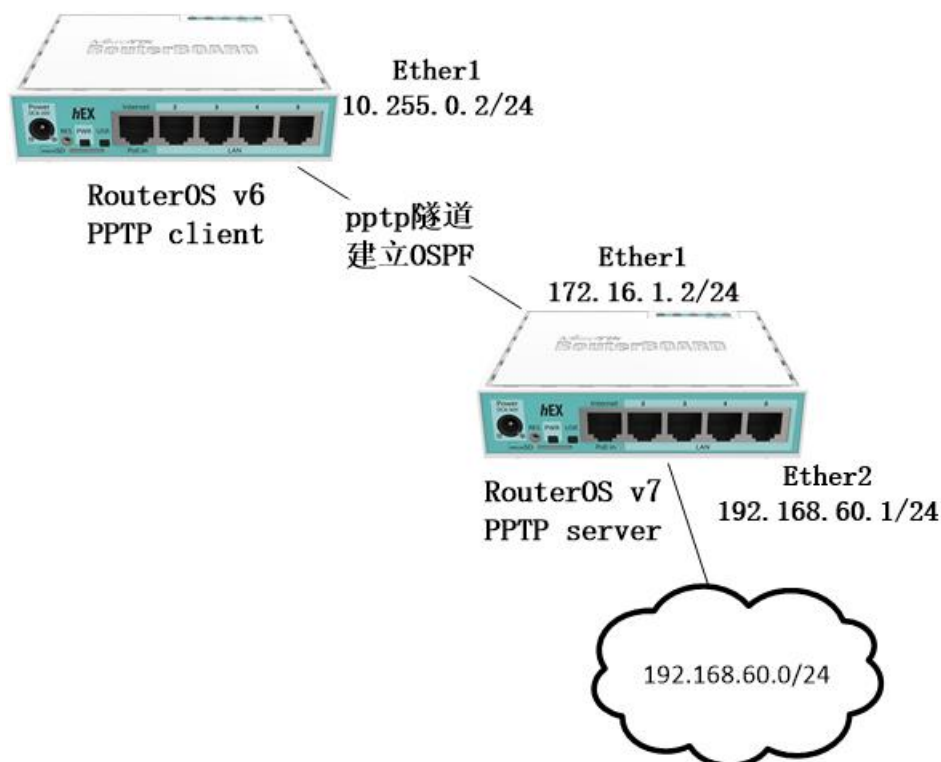
补充下 OSPF 模板使用 interface 创建配置方式如下（使用接口有点类似 juniper 的方式）：

```
[admin@R1] /routing/ospf/interface-template > add area=ospf-area-0 interfaces=ether1
[admin@R2] /routing/ospf/interface-template > add area=ospf-area-0 interfaces=ether1
```

R2 路由器也做相同的配置，当然 networks 和 interface 两个参数在 RouterOS 也可以同时设置。

PPTP 隧道的 OSPF 建立

基于 PPTP 隧道建立 OSPF 动态路由，通过 VPN 隧道打通远程网络的三层网络，一般采用静态路由方式，如果采用 OSPF 则可以动态发布地址，PPTP 客户端采用 RouterOSv6 和 PPTP 服务端使用 RouterOS v7，如下面的拓扑图：



查看 v7 的 IP 地址配置

```
[admin@v7] /ip/address > print
```

Flags: X - disabled, I - invalid, D - dynamic

#	ADDRESS	NETWORK	INTERFACE
0	172.16.1.2/24	172.16.1.0	ether1
1	10.10.10.1/29	10.10.10.0	ether2

RouterOS v7 启用 PPTP 服务

```
[admin@v7]> /interface/pptp-server/server
```

```
[admin@v7] /interface/pptp-server/server> set enabled=yes
```

创建 pptp 账号，并分配 PPTP 隧道的本地 IP 地址和远端 IP 地址

```
[admin@v7] /interface/pptp-server/server> /ppp secret
```

```
[admin@v7] /ppp/secret > add name=yus password=yus local-address=1.2.3.1 remote-address=1.2.3.2
```

配置 OSPF，首先创建 ospf 实例，并重分布直连路由

```
[admin@v7] /ppp/secret > /routing/ospf/instance
```

```
[admin@v7] /routing/ospf/instance> add name=ospf-instance-1 redistribute=connect
```

进入 area 创建区域 0

```
[admin@v7] /routing/ospf/instance>/routing ospf area
```

```
[admin@v7] /routing/ospf/area> add instance=ospf-instance-1 name=ospf-area-0
```

然后创建 ospf 模板，networks 填写对端 IP 地址，并配置 ospf 连接类型为 ptp（点对点）

```
[admin@v7] /routing/ospf/area>/routing ospf interface-template
```

```
[admin@v7] /routing/ospf/interface-template >add area=ospf-area-0 networks=1.2.3.2 type=ptp
```

RouterOS v6 客户端配置

创建 pptp 客户端

```
[admin@v6] /interface pptp-client> /interface pptp-client
```

```
[admin@v6] /interface pptp-client>add connect-to=172.16.1.2 disabled=no name=pptp-out1  
password=yus user=yus
```

RouterOSv6 已经默认创建了 instance 和 area 的模板，这里只需添加 network 对端 IP

```
[admin@v6] /interface pptp-client>/routing ospf network
```

```
[admin@v6] /routing ospf network> add area=backbone network=1.2.3.1
```

在 instance 路径下查看默认实例配置，并设置 redistribute-connected=yes 重分布直连路由，选择 type-1。

```
[admin@v6] /routing ospf network>/routing ospf instance
[admin@v6] /routing ospf instance> print
Flags: X - disabled, * - default
0  * name="default" router-id=0.0.0.0 distribute-default=never redistribute-connected=no
    redistribute-static=no redistribute-rip=no redistribute-bgp=no redistribute-other-ospf=no
    metric-default=1 metric-connected=20 metric-static=20 metric-rip=20 metric-bgp=auto
    metric-other-ospf=auto in-filter=ospf-in out-filter=ospf-out
[admin@v6] /routing ospf instance> set 0 redistribute-connected=as-type-1
```

查看 v6 ospf 的邻居状态：

```
[admin@v6] /routing/ospf/neighbor> print
Flags: V - virtual; D - dynamic
0  D instance=ospf-instance-1 area=ospf-area-0 address=1.2.3.2 router-id=100.64.0.3
    state="Full" state-changes=6 adjacency=2m31s timeout=39s
```

查看 v6 路由表，可以看到从 PPTP 服务端的 v7 学习到了标记为 ADo 的 ospf 路由 192.168.60.0/24

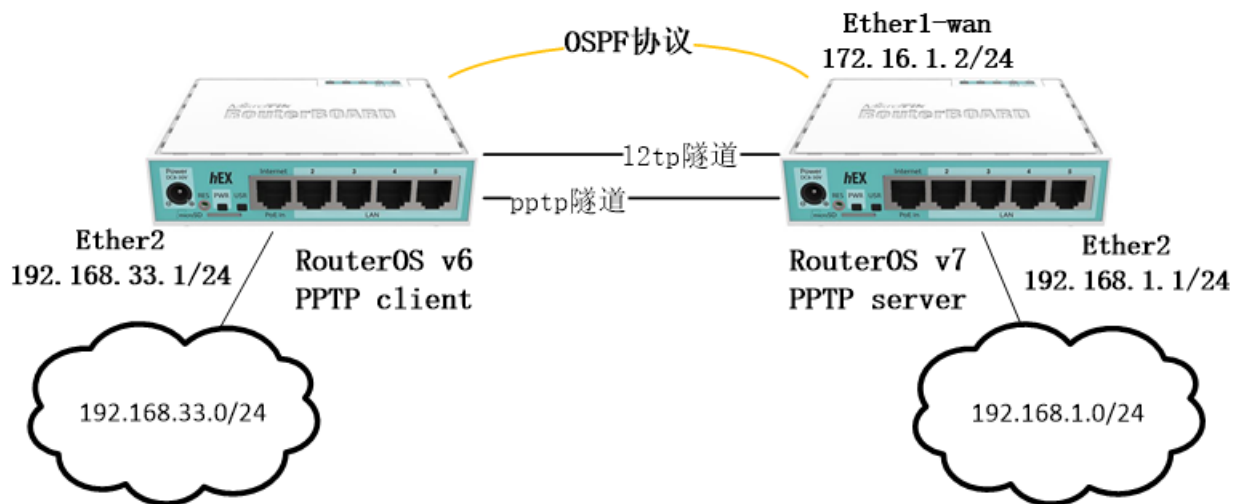
```
[admin@v6] /ip route> print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
```

#	DST-ADDRESS	PREF-SRC	GATEWAY	DISTANCE
0 A S	0.0.0.0/0		172.16.0.1	1
1 ADC	1.2.3.1/32	1.2.3.2	pptp-out1	0
2 ADo	192.168.60.0/24		1.2.3.1	110
3 ADC	10.255.0.0/24	10.255.0.2	ether1	0
4 ADC	192.168.80.0/24	192.168.80.1	ether2	0

同理 v7 也会学习到 v6 的直连路由。

多种隧道协议建立 OSPF 多路径等价路由

该案例是在之前 PPTP 隧道基础上演变而来，对于两个异地办公网络，需要建立隧道实现办公网络互通，并需要隧道协议多样性，避免一种隧道协议中断，使用另外一条隧道建立备份，同时通过 OSPF 的多路径保持两条隧道实现负载均衡。如下拓扑：



这里选用 PPTP 隧道和 L2TP 隧道建立双线路，RouterOS v7 搭建 server，RouterOS v6 作为 client（当然你可以换用 SSTP、OpenVPN 和 wireguard 等）。该实例的两端路由器都使用的是一个外网出口，实际应用中还会有多出口的情况，服务端需要配置多线出口的策略，而客户端需要配置静态路由指向隧道服务器的第二出口地址，避免隧道重复建立在一条出口。

基本配置与创建隧道

首先查看 RouterOS v7 路由器的 IP 地址配置

```
[admin@v7] /ip/address> print
Flags: X, D - DYNAMIC
Columns: ADDRESS, NETWORK, INTERFACE
#   ADDRESS          NETWORK    INTERFACE
0   172.16.1.2/24      172.16.1.0 ether1
1   192.168.1.1/24     192.168.1.0 ether2
```

启用 v7 的 pptp 服务

```
[admin@v7] /ip/address>/interface pptp-server server
[admin@v7] /interface/pptp-server>set enabled=yes
```

启用 v7 的 l2tp 服务

```
[admin@v7] /interface/pptp-server>/interface l2tp-server server
[admin@v7] /interface/l2tp-server> set enabled=yes
```

在 ppp secret 下创建两个账号，local-address 为隧道服务器侧地址，remote-address 为分配给客户端地址

```
[admin@v7] /interface/l2tp-server>/ppp secret
```

```
[admin@v7] /ppp secret> add name=123 password=123 local-address=1.2.3.1 remote-address=1.2.3.2
[admin@v7] /ppp secret> add name=321 password=321 local-address=2.3.4.1 remote-address=2.3.4.2
```

查看 RouterOS v6 路由器的 IP 地址配置

```
[admin@v6] /ip address> print
Flags: X - disabled, I - invalid, D - dynamic
#   ADDRESS           NETWORK           INTERFACE
0   10.20.30.2/24       10.20.30.0        ether1
1   192.168.33.1/24     192.168.33.0      ethter2
```

在 v6 路由器添加连接服务器的 pptp 和 l2tp 客户端

```
[admin@v6] /ip address>/interface l2tp-client
[admin@v6] /interface l2tp-client > add connect-to=172.16.1.2 disabled=no name=l2tp-out1
password=321 user=321
[admin@v6] /interface l2tp-client >/interface pptp-client
[admin@v6] /interface pptp-client> add connect-to=172.16.1.2 disabled=no name=pptp-out1
password=123 user=123
```

连接完成，查看 RouterOSv7 路由器 IP 地址情况，l2tp-321 和 pptp-123 都动态分配了 ip 地址

```
[admin@v7] /ip/address> print
Flags: X, D - DYNAMIC
Columns: ADDRESS, NETWORK, INTERFACE
#   ADDRESS           NETWORK           INTERFACE
0   172.16.1.2/24       172.16.1.0        ether1
1   192.168.1.1/24     192.168.1.0        ether2
2 D 2.3.4.1/32        2.3.4.2           <l2tp-321>
3 D 1.2.3.1/32        1.2.3.2           <pptp-123>
```

查看 RouterOS v6 获取 l2tp-out1 和 pptp-out1 的 IP 地址情况

```
[admin@v6] /ip address> print
Flags: X - disabled, I - invalid, D - dynamic
#   ADDRESS           NETWORK           INTERFACE
0   10.20.30.2/24       10.20.30.0        ether1
1   192.168.33.1/24     192.168.33.0      ethter2
2 D 2.3.4.2/32        2.3.4.1           l2tp-out1
3 D 1.2.3.2/32        1.2.3.1           pptp-out1
```

配置 OSPF

OSPF 配置除了建立路由关系, 还需要重分布直连路由(connect route), 在路由发布时过滤掉除 ether2 接口的内网 IP 外的其他地址。

RouterOS v6 的 OSPF 配置

添加 network, 指定 RouterOS v7 的隧道地址

```
[admin@v6] /routing ospf network> add area=backbone network=1.2.3.1/32
[admin@v6] /routing ospf network> add area=backbone network=2.3.4.1/32
```

设置默认 instance 实例重分布直连路由, 并设置为 type-1

```
[admin@v6] /routing ospf instance>/routing ospf instance
[admin@v6]/routing ospf instance>set [ find default=yes ] redistribute-connected=as-type-1
```

配置路由过滤策略, 只允许发布 ether2 接口的 192.168.33.0/24 地址段

```
[admin@v6]/routing ospf instance>/routing filter
[admin@v6] /routing filter> add action=accept chain=ospf-out prefix=192.168.33.0/24
[admin@v6] /routing filter> add action=discard chain=ospf-out
```

RouterOS v7 的 OSPF 配置

RouterOS 没有默认策略, 因此需先创建 instance 实例, 并设置重分布直连路由

```
[admin@v7]/routing ospf instance
[admin@v7]/routing/ospf/instance>add name=ospf-instance-1 redistribute=connected
```

创建 OSPF 的 area 0

```
[admin@v7]/routing/ospf/instance>/routing ospf area
[admin@v7] /routing/ospf/area >add instance=ospf-instance-1 name=ospf-area-0
```

创建 OSPF 接口模板, 添加 network, 填写分配给 RouterOS v6 的隧道地址, 并设置 type 类型为 ptp (点对点)

```
[admin@v7] /routing/ospf/area >/routing ospf interface-template
[admin@v7] /routing/ospf/interface-template>add area=ospf-area-0 networks=1.2.3.2/32 type=ptp
[admin@v7] /routing/ospf/interface-template>add area=ospf-area-0 networks=2.3.4.2/32 type=ptp
```

配置路由过滤规则, 仅发布 ether2 的 192.168.1.0/24 地址段

```
[admin@v7] /routing/ospf/interface-template>/routing filter rule
[admin@v7] /routing filter rule>add chain=ospf_out disabled=no rule="if (dst in 192.168.1.0/24) {accept}"
```

查看路由表

以上配置正确后，可以查看路由表，下面是 RouterOS v6 的路由表

```
[admin@v6] > ip route print
Flags: X - disabled, A - active, D - dynamic,
C - connect, S - static, r - rip, b - bgp, o - ospf, m - mme,
B - blackhole, U - unreachable, P - prohibit
```

#	DST-ADDRESS	PREF-SRC	GATEWAY	DISTANCE
0 A S	0.0.0.0/0		10.20.30.1	1
1 ADC	1.2.3.1/32	1.2.3.2	pptp-out1	0
2 ADC	2.3.4.1/32	2.3.4.2	l2tp-out1	0
3 ADC	10.20.30.0/24	10.20.30.2	ether1	0
4 ADo	192.168.1.0/24		2.3.4.1 1.2.3.1	110
5 ADC	192.168.33.0/24	192.168.33.1	ether2	0

通过 RouterOS v6 的路由表可以看到 ADo 标记的 192.168.1.0/24 的路由，有两个网关分别是 2.3.4.1 和 1.2.3.1，实现了多路径的负载均衡。

```
[admin@v7] /ip/route> print
Flags: D - DYNAMIC; A - ACTIVE; c, s, o, d, y - COPY; + - ECMP
Columns: DST-ADDRESS, GATEWAY, DISTANCE
```

#	DST-ADDRESS	GATEWAY	DISTANCE
DAd	0.0.0.0/0	172.16.1.1	1
DAc	1.2.3.2/32	<pptp-123>	0
DAc	2.3.4.2/32	<l2tp-321>	0
DAc	172.16.1.0/24	ether1	0
DAc	192.168.1.0/24	ether2	0
DAo+	192.168.33.0/24	1.2.3.2%<pptp-123>	110
DAo+	192.168.33.0/24	2.3.4.2%<l2tp-321>	110

RouterOS v7 路由表也同样实现了多路径的负载均衡，只是显示方式不通，标记为 DAo+。

1.9 BGP

与 RouterOS v6 相比, v7 的 BGP 配置方式被重新设计。第一个最大的区别是没有更多 **instance** 和 **Peer** 配置目录, 相对应的增加了 **connection**, **template** 和 **peer-cache**。这种结构的原因是为了严格分离用于负责连接的参数, 更改这些参数不会破坏现有的连接。

v7 BGP 改进说明

首先从 **template** (模板) 开始, 包含所有 BGP 协议相关的配置选项。它可以用作动态 **Peer** (对等体) 的模板, 模板能应用到类似的 **Peer** 连接上。请注意, 这与 Cisco 设备上的 **Peer** 分组不同, 在 **Peer** 分组中, 分组不仅仅是一个普通的配置。

BGP 的 **templat** 目录下, 提供一个默认模板配置, AS 号为 65533。

```
/routing/bgp/template set default as=65533
```

从 v7.1beta4 进行的调整, **template** 可以在“**connection**”配置中调用, 即配置 **template** 模板不再是强制性, 允许更简单 BGP 连接设置, 类似在 v6 版本中一样。

甚至可以从另一个模板继承模板参数, 例如:

```
/routing/bgp/template
add name=myAsTemplate as=65500 output.filter=myAsFilter
set default template=myAsTemplate
```

另一个重要方面是 **instance** 实例, 将 **router-id** 和 **group peers** (组对等体) 设置在一个 **instance** 实例中。RouterOS 添加了一个默认 **instance** 实例, 选举 **instance-id** 是根据所有网络接口中 IP 地址的最高位决定。

默认情况下, BGP 模板使用“**default**” **instance**。如果你需要调整或添加新的实例, 可以在 **/routing instance** 下完成。

CPU 亲和参数

input.affinity 和 **output.affinity** 是比较特别的参数, 它们允许控制在哪个进程 **input** 和 **output** 的活动会话将被处理:

- **alone** - 每个会话的 **input** 和 **output** 都在自己的进程来处理, 当有多个 CPU 核心和很多个 **Peer**(对等体) 时, 这种方式是最佳选择。
- **afi, instance, vrf, remote-as** - 尝试使用类似的参数在进程中运行新会话的 **input/output** to run input/output of new session in process with similar parameters
- **main** - 在 Main 进程中运行 **input/output**(可提高单核设备的性能, 也适用于少量 CPU 核心的设备)
- **input** - 在与 **input** 相同的进程中运行 **output**(只能设置 **output affinity**)

BGP 建立

在创建 BGP 连接时，至少需要包括以下参数：**remote.address**、**template**、**connect**、**listen** 和 **local.role**。默认情况下 **connect** 和 **listen** 参数是开启状态。

Connect 和 **listen** 参数指定 **Peer**（对等体）是否尝试连接和监听 **remote-address**，还是仅连接或只是监听。如果 **Listen** 设置为 **no**，BGP 将关闭端口监听，无法相应对端 BGP 设备的请求。

在 **Peer**（对等体）使用多跳连接的设置中，必须配置 **local.address**（类似于 ROSv6 中的 **update-source**）。

注意：RouterOS v7 可以从公开消息中获取远端 **ASN**，可以不用强制指定远端 **AS** 号，当然你也可以要求指定一个连接的 **AS** 号。

Peer role 对等的角色现在是一个强制参数，基本设置可以使用 **ibgp** 和 **ebgp**，也就是需要指定 **local.role** 参数指明网络建立在 **ibgp** 还是 **ebgp**。

基本的 iBGP 设置，监听整个本地网络的连接：

```
/routing/bgp/connection
add remote.address=10.155.101.0/24 listen=yes template=default local.role=ibgp
```

下面是一个非常基本的 eBGP 配置，两台路由器 **Route1** 的 IP 是 192.168.1.1, AS 65531，另外一台 **Router2** IP 是 182.168.1.2, AS 65532：

```
#Router1
/routing/bgp/connection
add name=toR2 remote.address=192.168.1.2 as=65531 local.role=ebgp

#Router2
/routing/bgp/connection
add name=toR1 remote.address=192.168.1.1 as=65532 local.role=ebgp
```

大多配置与 ROSv6 相似，除了一些 **output** 和 **input** 部分分组，如果熟悉 **capsman**，那么配置语法参数类似的。例如指定 **output** 选项可以设置为 **output.filter=myBgpChain**。

现在可以从 **/routing bgp session** 路径下查看对等体的连接状态是连接，还是断开，另外一个路由处理的调试信息，可以从 **/routing stats** 路径下查看

```
[admin@v7_ccr_bgp] /routing/stats/process> print interval=1
Columns: TASKS, PRIVATE-MEM-BLOCKS, SHARED-MEM-BLOCKS, PSS, RSS, VMS, RETIRED, ID, PID, RPID, PROCESS-TIME, KERNEL-TIME, CUR-B>
# TASKS PRIVATE-M SHARED-ME PSS RSS VMS RET ID PID R PROCESS-TI KERN>
0 routing tables 12.2MiB 20.0MiB 18.7MiB 42.2MiB 83.4MiB 8 main 319 0 19s750ms 8s50>
rib >
connected networks >
1 fib 512.0KiB 0 7.4MiB 30.9MiB 83.4MiB fib 384 1 5s160ms 22s5>
2 ospf 1024.0KiB 1024.0KiB 5.9MiB 25.9MiB 83.4MiB 382 ospf 388 1 1m42s170ms 1m31>
connected networks >
3 fantasy 512.0KiB 0 2061.0KiB 5.9MiB 83.4MiB fantasy 389 1 1s410ms 870m>
4 configuration and reporting 40.0MiB 512.0KiB 45.0MiB 64.8MiB 83.4MiB static 390 1 12s550ms 1s17>
5 rip 768.0KiB 0 5.3MiB 24.7MiB 83.4MiB rip 387 1 1s380ms 1s20>
connected networks >
6 routing policy configuration 512.0KiB 256.0KiB 2189.0KiB 6.0MiB 83.4MiB policy 385 1 1s540ms 1s20>
7 BGP service 768.0KiB 0 2445.0KiB 6.2MiB 83.4MiB bgp 386 1 6s170ms 9s38>
8 BGP Input 10.155.101.217 8.8MiB 6.0MiB 15.6MiB 38.5MiB 83.4MiB 20 21338 1 25s170ms 3s23>
BGP Output 10.155.101.217 >
9 Global memory 256.0KiB global 0 0 >
-- [Q quit|D dump|C-z pause|right]
```

BGP 路由过滤

Route filtering 路由过滤与 v6 版本略有不同。在 BGP 模板中，可以指定 `output.filter-chain`、`output.filter-select`、`input.filter` 以及多个 `input.accept-*` 选项。

现在 `input.accept-*` 允许在消息传入被解析和存储到内存之前直接过滤它们，这样可显著减少内存使用。常规情况下 `input` 过滤只能拒绝前缀，意味着仍然会消耗内存，并将在 `/routing route` 中显示为“not active, filtered”，

下面的实例，BGP 配置 `input` 过滤接受来自 `192.168.0.0/16` 子网前缀，而不修改任何属性。对于其他前缀，从收到的本地前缀值中减去 1，并设置 IGP metric 值为 OSPF ext 的值。另外，我们只接受地址列表中特定的前缀，以减少内存的使用。

如果没有指定 `routing filter` 链表，BGP 将尝试发布它在路由表中找到的每一条活跃路由

```
/ip/firewall/address-list
add list=bgp_list dst-address=192.168.1.0/24
add list=bgp_list dst-address=192.168.0.0/24
add list=bgp_list dst-address=172.16.0.0/24

/routing/bgp/template
set default input.filter=bgp_in .accept-nlri=bgp_list

/routing/filter/rule
add chain=bgp_in rule="if (dst in 192.168.0.0/16) {accept}"
add chain=bgp_in rule="set bgp-local-pref -1; set bgp-igp-metric ospf-ext-metric; accept"
```

在 `routing filter` 链表中，默认 `action` 动作为“drop”

在 v7 注意到 `network` 菜单消失了，如何宣告自己的网络？在 v7 网络宣告通过 `ip firewall address-list` 创建地址列表，并在 BGP 配置中被引用。

下面是 RouterOS v6 宣告 network 网络地址的一个实例配置：

```
/routing bgp network add network=192.168.0.0/24 synchronize=yes
/ip route add dst-address=192.168.0.0/24 type=blackhole
```

转换为 v7 的配置，通过 address-list 操作：

```
/ip/firewall/address-list/
add list=bgp-networks address=192.168.0.0/24

/ip/route
add dst-address=192.168.0.0/24 blackhole

/routing/bgp/connection
set peer_name output.network=bgp-networks
```

当只添加一个网络时，需要进行更多的配置，但当必须处理大量网络时，这就很简单了。v7 甚至允许为每个 BGP 连接指定它自己的一组网络。

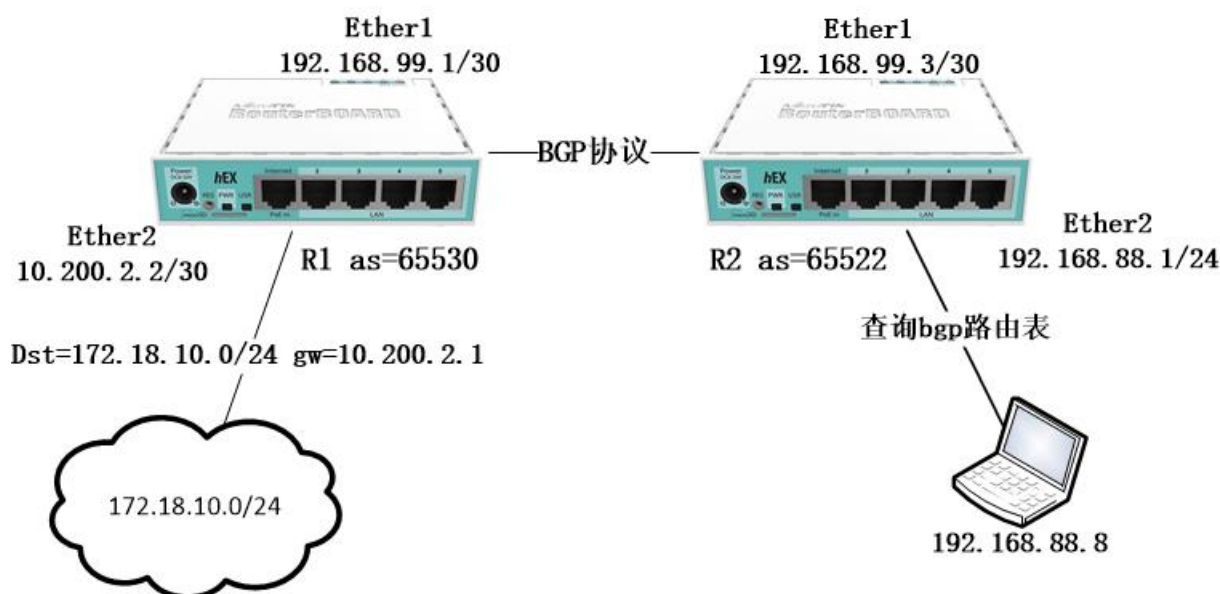
在 v7 中，不能关闭与 IGP 路由的同步（synchronization），只有在路由表中存在相应的 IGP 路由时，network 才会宣告发布

BGP 路由通过 address-list 发布路由到 routing table

RouterOS 的路由策略通过创建新的路由表（routing table），实现各种策略路由配置，OSPF 和 BGP 都支持路由发布和接收指定到一个新的路由表中，提供策略路由查询。在 RouterOS v7 的支持 network 地址宣告可以调用 /ip firewall addresss-list，这样发布路由配置更加灵活！

警告：确保 address-list 的 IP 子网在路由表中是活动状态，不管是华为、思科和锐捷的动态路由，发布的路由条目，必须在路由表是活动状态。

通过下面的实例将新建路由表和 address-list 发布 BGP 路由结合起来，如下网络拓扑



网络基本参数如上图，R1 和 R2 建立 BGP 路由关系，R2 路由器从 R1 获取的 BGP 路由放到新建的 bgp 路由表，并指定 R2 的内网主机 192.168.88.8，不查询 main 路由表，而是查询 bgp 路由表。

查看 R1 的 IP 地址配置

```
[admin@R1] /ip/address> print
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS          NETWORK          INTERFACE
0 192.168.99.1/30   192.168.99.0    ether1
1 10.200.2.2/30     10.10.10.0      ether2
```

查看路由表配置

```
[admin@R1] /ip/route> print
Flags: D - DYNAMIC; A - ACTIVE; c, s, b, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
#   DST-ADDRESS      GATEWAY      DISTANCE
DAc 192.168.99.0/30    ether1       0
DAc 10.200.2.0/29     ether2       0
0   As 172.18.10.0/24 10.200.2.1   1
```

创建地址列表，需要注意如果 172.18.10.0/24 的路由在 ip route 中变为不可用，bgp 也不会发布

```
[admin@R1]/ip firewall address-list
[admin@R1]/ip/firewall/address-list>add address=172.18.10.0/24 list=bgp
```

创建与 R2 路由器的 BGP 关系，并设置 output.network=bgp

```
[admin@R1]/routing bgp connection
```

```
[admin@R1]/routing/bgp/connection> add name=bgp1 as=65530 disabled=no local.role=ebgp
output.network=bgp remote.address=192.168.99.3/32 .as=65522 \
routing-table=main templates=default
```

查看 R2 的 IP 地址配置

```
[admin@R2] /ip/address> print
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS          NETWORK          INTERFACE
0 192.168.99.3/30    192.168.99.0     ether1
1 192.168.88.1/30    192.168.88.0     ether2
```

新建路由表取名 bgp

```
[admin@R2] /ip/address>/routing table
[admin@R2] /routing/table>add disabled=no name=bgp fib
```

修改模板的 as 号为 65522

```
[admin@R2] /routing/table>/routing bgp template
[admin@R2] /routing/bgp/template>set default as=65522 disabled=no
```

配置 bgp 连接，并设置 routing-table 为 bgp 路由表

```
[admin@R2] /routing/bgp/template>/routing bgp connection
[admin@R2] /routing/bgp/connection>add as=65522 disabled=no local.role=ebgp name=bgp1
remote.address=192.168.99.1/32 .as=65530 \
routing-table=bgp templates=default
```

命令行在 session 查看 BGP 建立情况，winbox 为 Peer cache

```
[admin@R2] /routing/bgp/session> print
Flags: E - established
0 E remote.address=192.168.99.1 .as=65530 .id=192.168.99.1 .refused-cap-opt=no .
capabilities=mp,rr,gr,as4 .messages=110 .bytes=2195 .eor=""

local.address=192.168.99.3 .as=65522 .id=192.168.99.3 .capabilities=mp,rr,gr,as4 .messages=105 .bytes=19
95
.eor=""
output.procid=56
input.procid=56 ebgp
routing-table=bgp hold-time=3m keepalive-time=1m uptime=1h44m43s50ms
```

“E”代表连接建立，进入 ip route 查看 bgp 路由表的路由条目：

```
[admin@R2] /ip/route> print where routing-table=bgp
Flags: D - DYNAMIC; A - ACTIVE; b, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
  DST-ADDRESS      GATEWAY      DISTANCE
DAb 172.18.10.0/24  192.168.99.1  20
[admin@R2] /ip/route>
```

可以看到 bgp 路由表里面标记为 DAb 的 172.18.10.0/24

进入 firewall mangle 标记源地址 192.168.88.8 的路由，设置 new-routing-mark 为 bgp

```
[admin@R2] /ip/firewall/mangle> add chain=prerouting src-address=192.168.88.8 action=mark-routing
new-routing-mark=bgp
```

如果需要，可以把 R2 到 R1 的线路配置上 nat

```
[admin@R2] /ip/route>/ip firewall nat
[admin@R2] /ip/firewall/nat> add chain=srcnat out-interface=ether1 action=masquerade
```

在多线情况下，可以在路由表里面添加一个默认出口地址，在获取 BGP 路由，例如你的 R2 路由器建立了一个 PPPoE 拨号接口

```
[admin@R2] /ip/address> print
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS      NETWORK      INTERFACE
0 192.168.99.3/30  192.168.99.0  ether1
1 192.168.88.1/30  192.168.88.0  ether2
2 D 100.64.11.2/32  100.64.11.1   pppoe-out1
```

为 bgp 路由表添加一条 pppoe 拨号的默认网关

```
[admin@R2] /ip/route>add gateway=pppoe-out1 routing-table=bgp
[admin@R2] /ip/route> print where routing-table=bgp
Flags: D - DYNAMIC; A - ACTIVE; b, y - COPY
Columns: DST-ADDRESS, GATEWAY, DISTANCE
  DST-ADDRESS      GATEWAY      DISTANCE
As  0.0.0.0/0       pppoe-out1    1
DAb 172.18.10.0/27  192.168.99.1  20
```

创建 pppoe-out1 的 nat 规则

```
[admin@R2] /ip/route>/ip firewall nat
[admin@R2] /ip/firewall/nat> add chain=srcnat out-interface=pppoe-out1 action=masquerade
```

R1 控制发给 R2 的所有路由条目, R2 访问外部可通过 nat 因此内网 IP 地址, 这个思路在扩展下, R1 的 BGP 路由还可发布如运营商地址列表或 CN 地址列表, 也可以结合之前 OSPF 的隧道方式建立 BGP 关系。

1.10 Routing Filters

RouterOS v7 在路由方面除了路由表 (routing/table) 改动最大外, 就是动态路由, 其实 RIP、OSPF 和 BGP 在配置时设置参数和模板选项变动下还好, 大多参数通过选项的方式可以设置, 但 Routing Filter 的改动几乎是逆天, 从 v7.1beta4 开始, routing filter rule 配置修改为类脚本语法。rule 规则采用 "if .. then" 配置方式, 或根据 "if" 语句中的条件应用操作。

这样的修改实在无法理解 MikroTik 的思路, 又一次增加了 RouterOS 的操作难度, 还需要去查找相关的字段, 写成代码来过滤, 不能好好的把属性参数写成选项来填写吗? 对于动态路由这部分, 已经上升到了运营商和相对较大的企业网络应用, 因此把这部分的门坎抬高是为了培训收费?

操作路径: routing/filter/rule

在 rule 中, 允许多个条件被叠加在一个规则中, 并像防火墙一样按顺序执行, 下面的实例, 匹配静态且是默认路由, 并执行接受, 可以写入在一条规则中 (在 routing filter 链表中, 默认 action 为 "drop"):

```
/routing/filter/rule
add chain=ospf_in rule="if (dst==0.0.0.0/0 && protocol static) { accept }"
```

例如, 在 v6 的规则 "/routing filter add chain=ospf_in prefix=172.16.0.0/16 prefix-length=24 protocol=static action=accept", 转换为 v7 的配置如下:

```
/routing/filter/rule
add chain=ospf_in rule="if (dst in 172.16.0.0/16 && dst-len==24 && protocol static) { accept }"
```

另一个实例, 前缀的匹配, 从 172.16.0.0/16 中匹配子网长度为 24 的范围, 并设置 BGP 的 mde 和 prepend 参数:

```
/routing/filter/rule
add chain=BGP_OUT rule="if (dst-len==24 && dst in 172.16.0.0/16) { set bgp-med 20; set
bgp-path-prepend 2; accept }"
```

可以通过以下配置匹配前缀长度范围

```
/routing/filter/rule
add chain=BGP_OUT rule="if (dst-len>13 && dst-len<31 && dst in 172.16.0.0/16) { accept }"
```

Filter 规则可以匹配或设置 communities 参数, 如能使用 large communities 和 extended communities

```
/routing/filter/rule
add chain=bgp_in rule=" bgp-communities-large append 200001:200001:10 "
```

如果有多个 community，需要在多个规则中应用，可以在 **large-community-set** 下定义 community 集合，并使用它们来匹配或设置：

```
/routing/filter/large-community-set
add set=myLargeComSet communities=200001:200001:10
```

```
/routing/filter/rule
add chain=bgp_in rule=" bgp-communities-large-set append myLargeComSet "
```

由于 route-target 编码在 extended community 属性中以更改或匹配 RT，因此需要对 extended community 属性进行操作：

```
/routing/filter/rule
add chain=bgp_in rule=" bgp-communities-ext append rt:327824:20 "
```

V7 版本的动态路由协议 route filter（路由过滤）规则采用了类似脚本的语法。

下面的实例是匹配子网 192.168.1.0/24，并且子网长度大于 24，就执行 distance 距离增加 1，如果没有匹配，就减去 distance 值 1

```
/routing filter rule
add chain=myChain rule="if (dst==192.168.1.0/24 && dst-len>24) {set distance +1; accept} else {set
distance -1; accept}"
```

也可以配置 “||” 或的方式匹配，如下：

```
/routing filter rule
add chain=ospf_out disabled=no rule="if(dst==192.168.81.0/24 || dst==192.168.80.0/24 ) {set distance
+1; accept}"
```

Routing Filter 语法

Filter 规则支持多条件匹配，语法构成如下：

```
if ( [matchers] ) { [actions] } else { [actions] }
```

属性的两种类型：

- **only readable** – 这类属性参数仅只读且不能修改，这类属性只能被匹配使用

- **readable/writable** –这类属性参数只读和可修改，既能执行过滤配置，也可匹配使用

只读属性可以与其他只读属性参数匹配(仅适用于数值型)或使用布尔运算符的常量值。

```
[matchers]:  
[prop readable] [bool operator] [prop readable]  
  
[actions]:  
[action] [prop writeable] [value]
```

仅只有一个条件匹配的操作，则不需使用布尔运算符，如下面没有布尔运算符的实例：

```
if ( protocol connected ) { accept }
```

下面是使用了布尔运算符：

```
if ( bgp-med < 30 ) { accept }
```

如果匹配的属性是只读，**matcher** 无需使用运算符

```
if ( ospf-dn ) { reject }
```

注意，路由过滤器链的默认动作是“拒绝”。

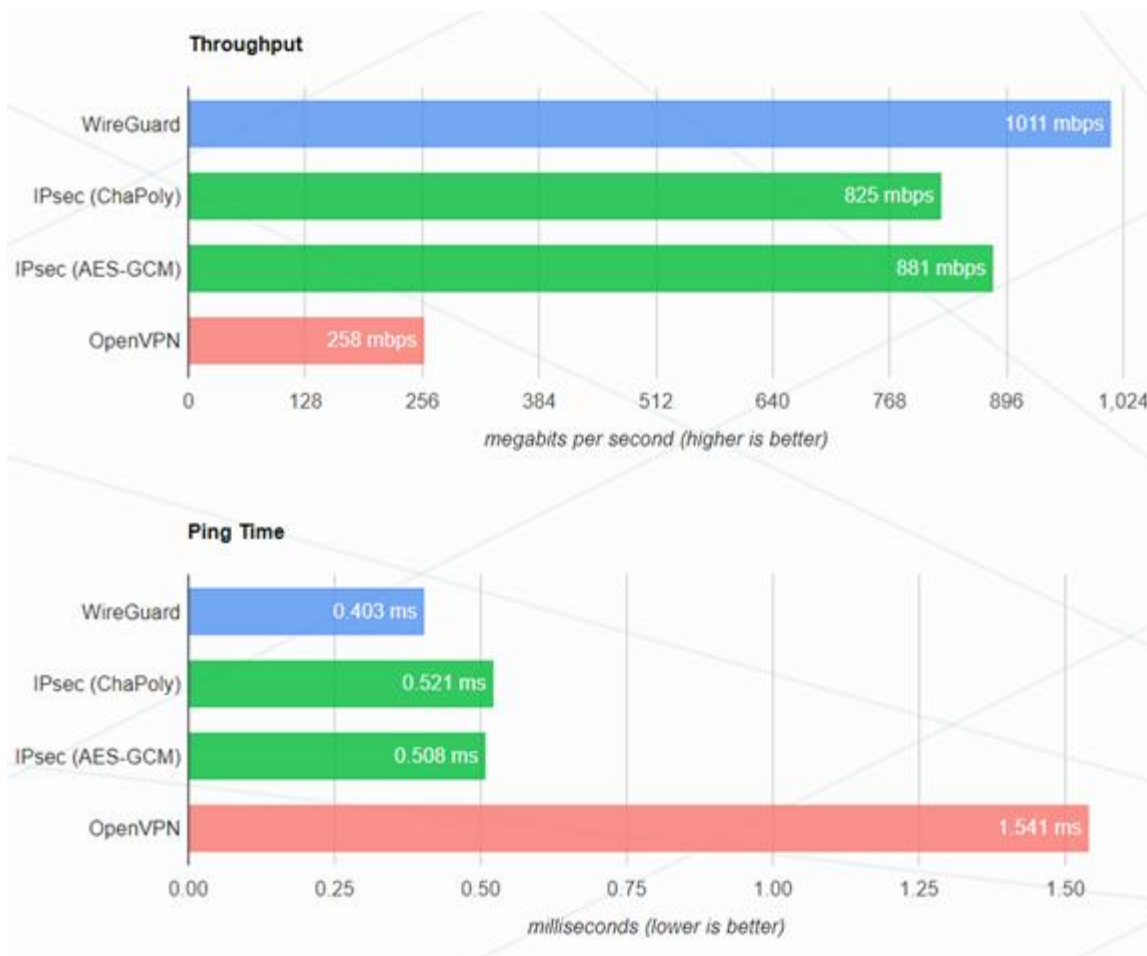
第二章 WireGuard

RouterOS v7 加入了 WireGuard, WireGuard 是一个极简而快速的加密 VPN 协议。其设计目标是比 IPsec 更快、更精简和高效,同时性能要比 OpenVPN 提升很多。WireGuard 被设计成一种通用 VPN,可以在多个平台上运行,适合许多不同的环境。最初是为 Linux 内核发布的,后支持跨平台 (Windows, macOS, BSD, iOS, Android), 能进行广泛的部署。

这个协议之前就看过,但没具体测试,后来大连小方提到,自己就动手测试了下,由于 Wireguard 极简的代码,加上高效性能,被加入 Linux 5.5 的内核中, RouterOS v7 采用的是 5.6 的内核,所以 v7 加入该功能是顺理成章的事情。WireGuard 到底有啥特点,我们先从性能表现来看,从官网看到的测试数据:

测试平台与配置

- Intel Core i7-3820QM 和 Intel Core i7-5200U
- Intel 82579LM 和 Intel I218LM gigabit 以太网卡
- Linux 4.6.1
- WireGuard 配置: 256-bit ChaCha20 with Poly1305 for MAC
- IPsec 配置 1: 256-bit ChaCha20 with Poly1305 for MAC
- IPsec 配置 2: AES-256-GCM-128 (with AES-NI)
- OpenVPN 配置: equivalently secure cipher suite of 256-bit AES with HMAC-SHA2-256, UDP mode
- 使用 iperf3 测试, 取 30 分钟平均值



从这个测试数据看使用的是 1Gbit 的网卡,跑出 1011mbps 的结果,说明带宽测试采用的是 UDP 协议,而非 TCP。从官方测试性能看, WireGuard 对于 IPsec 优势非常明显,而对于 OpenVPN 几乎是碾压。

在部署方面, WireGuard 各个节点是 Peer 对等体的概念,而非是 Server 与 Client 关系,相互之间是对等的,一个节点既可以是发起者,也可以是接收者,比如网上提到 VPN 的拓扑方案中,可以是点对点,点对多点的中心覆盖,而 WireGuard 不一样了,不仅可以点对点,也可以点对多点,每个节点同时可连接多个 Peer,看到过有人通过 WireGuard 建立全互联模式,三层的 Mesh 网状网络。

WireGuard 只支持 UDP 协议,不支持传统的 TCP-over-TCP 隧道,因为 TCP 网络性能并不理想,如果对 TCP 连接有需求,将 wireguard 的 UDP 包转换为 TCP,并可以使用如 udptunnel 和 udp2raw 来协助完成。

关于 NAT 穿透,通常情况下, WireGuard 网络在未被使用的情况下,会尽量保持“安静”,并非一个话痨协议。在大多数情况下,它只在 Peer 对等体,希望发送数据包时传输数据,当它没有被要求发送数据包时,它就停止发送,直到再次被请求。在点对点公网 IP 连接的网络环境是这样的,然而,当一个 Peer 端位于 NAT 或防火墙之后时,即使它不发送任何数据包的情况下,也希望能接收传入的数据包。因为 NAT 和防火墙会跟踪“会话连接”,他必须通过定期发送数据包来维持 NAT/防火墙连接会话映射的有效,即 UDP 的 timeout 时间,即被称为 keepalive。WireGuard 可以开启此选项,每隔一秒向对端点发送一个 keepalive 报文,当然适用于各种 NAT/防火墙的合理间隔在 25 秒。默认设置 keepalive 是关闭状态,因为公网 IP 连接的用户不需要这个功能。

WireGuard 使用的是 ChaCha20Poly1305,它几乎在所有通用 CPU 上的运行速度都非常快。虽然未被专用硬件支持(IPsec 支持硬件加速),但在 CPU 上的矢量指令(vector instructions)与 AES-NI 指令处于相同的优先级(有时甚至更快)。

总结下 WireGuard 的特点:

1. 是基于 UDP 协议连接,具备网络性能优势
2. 运行在 Linux 内核,在性能上优于 IPsec 和 OpenVPN
3. 每个节点通过公钥识别进行验证,加密采用的是 ChaCha20Poly1305。
4. 每个节点是平等的,既可以作为 server,又可以作为 client
5. 两端仅需要一个配置公网 IP,另一端在 nat 后也能连接,nat 后的节点,仅需要对端节点公钥,以及公网 IP (endpoint)和端口(endpoint port),而公网节点只需要填写连接端的公钥即可

属性描述

属性	描述
comment (字符; 默认:)	注释说明.
disabled (yes / no; 默认: no)	是否禁用隧道
listen-port (整型; 默认: 13231)	WireGuard 服务监听端口
mtu (整型[0..65536]; 默认: 1420)	三层 MTU 值
name (字符; 默认:)	隧道名称
private-key (字符; 默认:)	Bases64 私钥,如果未指定,会在接口创建时自动生成

属性	描述
public-key (字符)	base64 公钥，从私钥计算生成
running (yes / no)	接口是否运行

Peers 属性

属性	描述
allowed-address (IP/IPv6 prefix; 默认:)	允许对端 peer 通过的 IPv4/v6 地址列表，如需要允许所有 IP 通过可以指定 0.0.0.0/0，ipv6 可以写作 ::/0
comment (string; Default:)	注释说明
disabled (yes / no; 默认: no)	是否禁用 peer.
endpoint-address (IP/Hostname; 默认:)	对端 IP 或主机域名，也可设置为空，以允许任何地址进行远程连接。
endpoint-port (整型:0..65535; 默认:)	对端 UDP 端口，也可设置为空，以允许任何端口进行远程连接。
interface (字符; 默认:)	用于与对端主机连接的 WireGuard 接口
persistent-keepalive (整型:0..65535; 默认: 0)	一秒为间隔，从 1 到 65535(含 1)，是为了保持有状态防火墙或 NAT 映射持续有效而向对端发送空数据包的频率。例如，当节点在 NAT 之后，隧道无流量产生的情况下，为保持 nat 会话连接状态，可设置 25 秒的持续存活时间间隔。
preshared-key (字符; 默认:)	一个 base64 的预共享密钥。该参数可选的。该选项增加了一个额外的对称密钥层，将其混合到现有的公钥中，用于抗量子计算。
public-key (字符; 默认:)	对端 peer 的公钥

2.1 WireGuard 配置实例

下面通过 RouterOS 配置 WireGuard 隧道，并搭建一个测试环境，对比测试多个隧道协议的传输带宽，这个环境是在自己家里搭建，采用 RB 设备测试，其中一台是当前在售性能最低端的 hAP mini，拓扑结构如下：



在这个环境中，一共使用了 4 台 RB 设备，分别是：

1. R1: RB951Ui-2HnD 和 R2: hAP mini 作为 wireguard 的互联
2. 用户端 RB951Ui-2HnD，作为模拟终端设备进行 bandwidth-test 带宽测试发起端
3. cAP ac: 作为 bandwidth-test 带宽测试节点

这样网络把两台 WireGuard 隧道设备放在带宽测试设备中间，更能反映实际的使用环境和吞吐量，如果只是两台 R1 和 R2 两台设备测试，本地带宽测试会消耗 CPU，影响转发性能。在这个环境中，R1 和 R2 在 ether1 接口上都做 nat，非静态路由模式，也未开启 fasttrack，以下的配置仅提供隧道配置过程，其他基本网络配置不在此文描述。

首先在 R1 创建接口

```
[admin@R1] /interface/wireguard> add disabled=no listen-port=13231 mtu=1420 name=wireguard1
```

查看 public key

```
[admin@R1] /interface/wireguard> print
Flags: X - disabled; R - running

0  name="wireguard1" mtu=1420 listen-port=13231 private-key="qAgK90ek...X+kt/GU="
    public-key="JwTMcEc...TRXg0="
```

在 R2 创建接口

```
[admin@R2] /interface/wireguard> add disabled=no listen-port=13231 mtu=1420 name=wireguard1
```

查看 public key

```
[admin@R2] /interface/wireguard> print
```

```
Flags: X - disabled; R - running
```

```
0 X name="wireguard1" mtu=1420 listen-port=13231 private-key="GJb2dFT...FttkDLEc="
public-key="f0FjnuH...288Wc="
```

在 R1 创建 peer, 将 R1 作为发起端, 配置对端的 IP 和端口, 添加 R2 的 public key 和 allowed-address 允许通过的 IP 段,

```
[admin@R1] /interface/wireguard/peers> add
allowed-address=192.168.88.0/24,10.20.30.0/24,192.168.99.0/24 \
endpoint-address=192.168.88.10 endpoint-port=13231 interface=wireguard1 \
public-key="f0FjnuH...0Y288Wc="
```

在 WireGuard 中, 对等体严格通过其公钥(32 字节的 Curve25519)来识别。即在公钥和允许的 IP 地址段之间存在关联映射, 要通过 WireGuard 隧道的所有 IP 都需要加入 allowed-address 中与对端的 public key 建立对应关系。

我们将 R2 作为接收端, 在 R2 创建 Peer, 添加 R1 的 public key 和允许通过的 IP 段

```
[admin@R2] /interface/wireguard/peers> add
allowed-address=192.168.88.0/24,10.20.30.0/24,192.168.99.0/24 interface=wireguard1 public-key=\
"JwTMcE...cTRXg0="
```

在 R1 上添加 wireguard 接口 ip 地址

```
[admin@R1] /interface/wireguard/peers> /ip address/
[admin@R1] /ip/address> add address=10.20.30.2/24 interface= wireguard1
```

在 R2 上添加 wireguard 接口 ip 地址

```
[admin@R2] /interface/wireguard/p eers> /ip address/
[admin@R2] /ip/address> add address=10.20.30.1/24 interface= wireguard1
```

由于当前 v7 还不支持 wireguard 的 log 日志支持和状态现实, 所以无法判断是否连接成功, 只能通过 ping 测试, 对端 IP 判断是否连接, 在 R2 上测试 R1 的 WireGuard 接口 IP 是否连接。

```
[admin@R2] > ping 10.20.30.2
```

```

SEQ HOST                SIZE TTL TIME
0 10.20.30.2            56  64 1ms106us
1 10.20.30.2            56  64 517us
2 10.20.30.2            56  64 520us
3 10.20.30.2            56  64 511us

sent=4 received=4 packet-loss=0% min-rtt=474us  avg-rtt=605us max-rtt=1ms106us

```

连接成功后，下面是我对几种 VPN 隧道协议的 bandwidth-test 工具，使用 tcp 协议，单向测试结果如下图：

连接成功后，在 R2 路由器上将到 192.168.99.2 的静态路由指向 wireguard1，并配置从 wireguard1 接口出方向的 nat 规则

```

[admin@R2] /ip/route>add dst-address=192.168.99.2/32 gateway= wireguard1
[admin@R2] /ip/route>/ip/firewall/nat
[admin@R2] /ip/firewall/nat>add chain=srcnat out-interface= wireguard1 action=masquerade

```

防火墙配置

防火墙配置，仅适用于使用了 RouterOS 的默认防火墙，因为默认的防火墙在 input 链表会阻止隧道建立连接，所以需要在两端的防火墙 filter input 链表中，允许 IP 和端口通过，并且需要将规则移动到顶部。

R1

```

/ip/firewall/filter
add action=accept chain=input dst-port=13231 protocol=udp src-address=192.168.88.20

```

R2

```

/ip/firewall/filter
add action=accept chain=input dst-port=13231 protocol=udp src-address=192.168.88.10

```

规则建立后需要使用 move 命令移动到 input 链表最顶部，例如：通过 print 命令查看所有规则序号，如果当前规则序号为 10，移动到第一条：

```

[admin@R2] /ip/firewall/filter>move 10 0

```

在 winbox 中可以直接使用鼠标进行拖动到顶部，这里不再过多赘述。

RouterOS 默认的防火墙规则是不阻止转发，如果你有特殊的 ACL 访问控制需求，可以参考《RouterOS 入门到精通 v6e》的防火墙章节。

关于 WireGuard 的带宽测试，可以参考 irouters.com 网站的内容，大概从测试结果看未使用 IPsec 加密的 L2TP 的传输带宽性能最好，超过了 WireGuard，但在使用 IPsec 的 L2TP 加密后，性能远不如 WireGuard，我平时喜欢使用 SSTP 在 RouterOS 设备之间建立连接，因为 SSTP 端口灵活，在 RouterOS

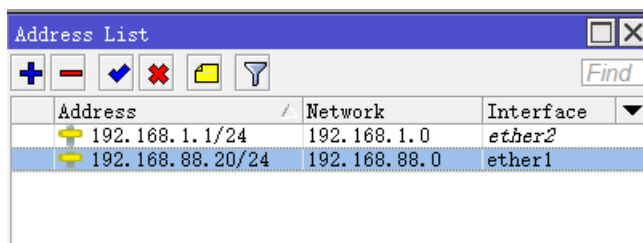
建立隧道无需证书，但 SSTP 的传输性能在 RouterOS 支持的所有隧道协议中性能是最差的，现在 v7 有了 WireGuard 就不一样了。

2.2 Windows 与 RouterOS 的 Wireguard 连接

WireGuard 有多个平台应用程序，包括 Windows、MAC、Android 等版本，可从官网下载 <https://www.wireguard.com/install/>，

基本配置

假设 RouterOS 外网 IP 是 192.168.88.20 配置在 ether1 上，Winbox 配置如下：

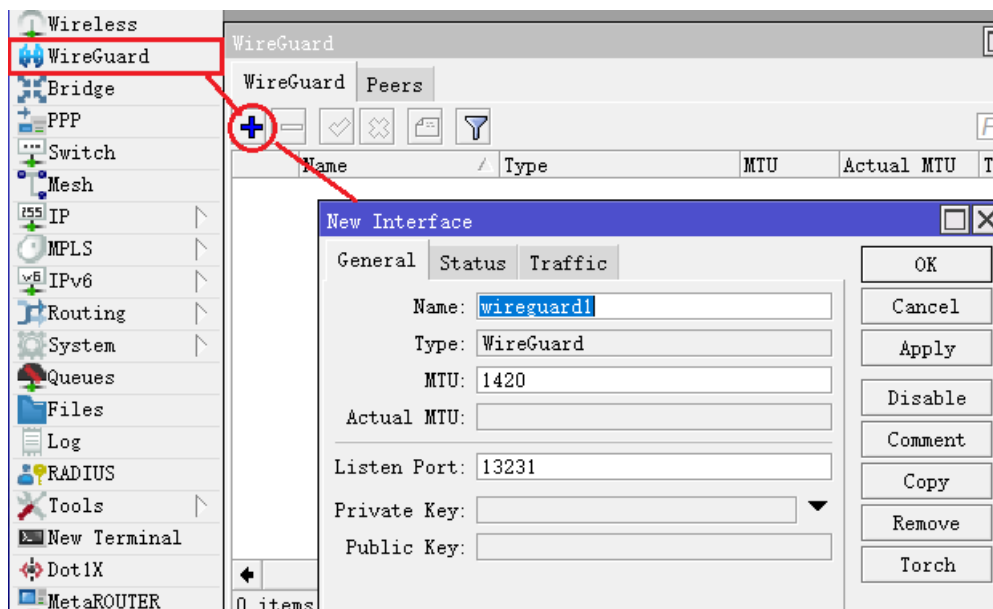


Address	Network	Interface
192.168.1.1/24	192.168.1.0	ether2
192.168.88.20/24	192.168.88.0	ether1

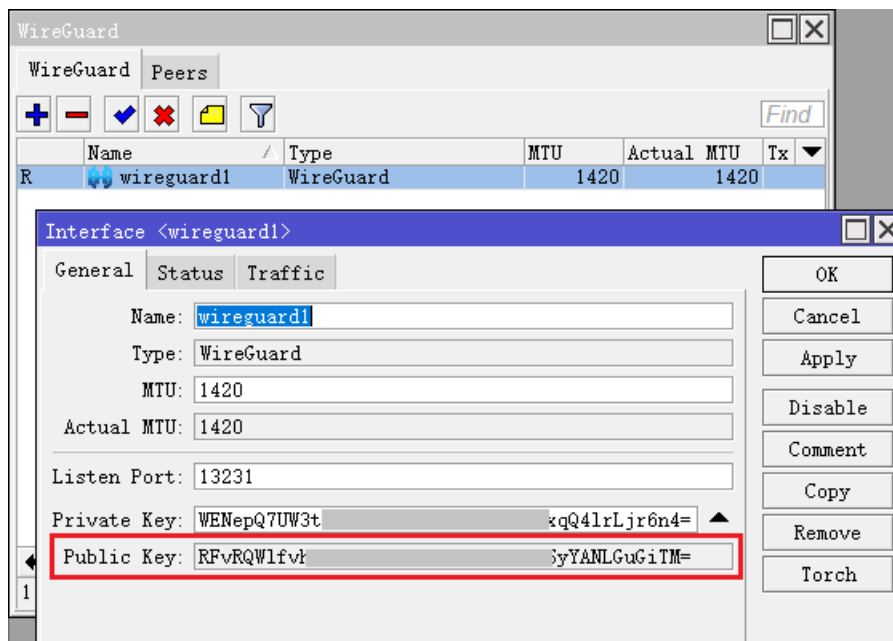
windows 电脑作为请求发起端，RouterOS 作为接收端，分配给两端的隧道互联 IP 如下：

- RouterOS wireguard 接口 IP 是 10.20.30.1/24
- windows 的 wireguard 接口配置 10.20.30.2/24

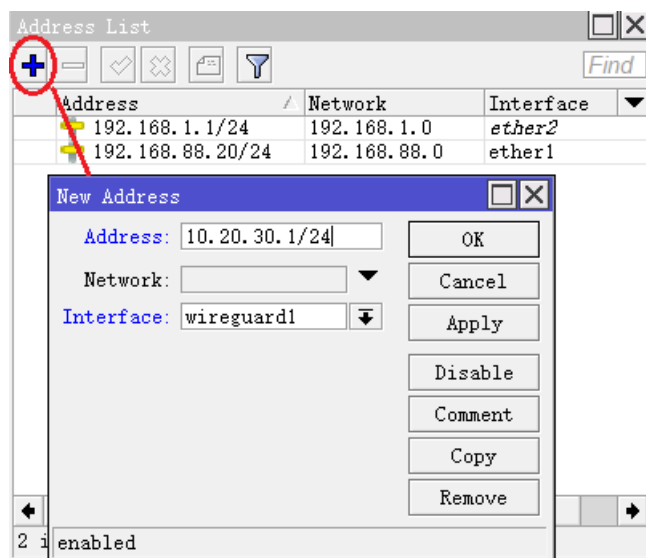
首先，在 RouterOS 创建 WireGuard 接口，默认端口是 13231，操作如下：



创建接口完成后，会自动生成公钥，可查看 Public Key

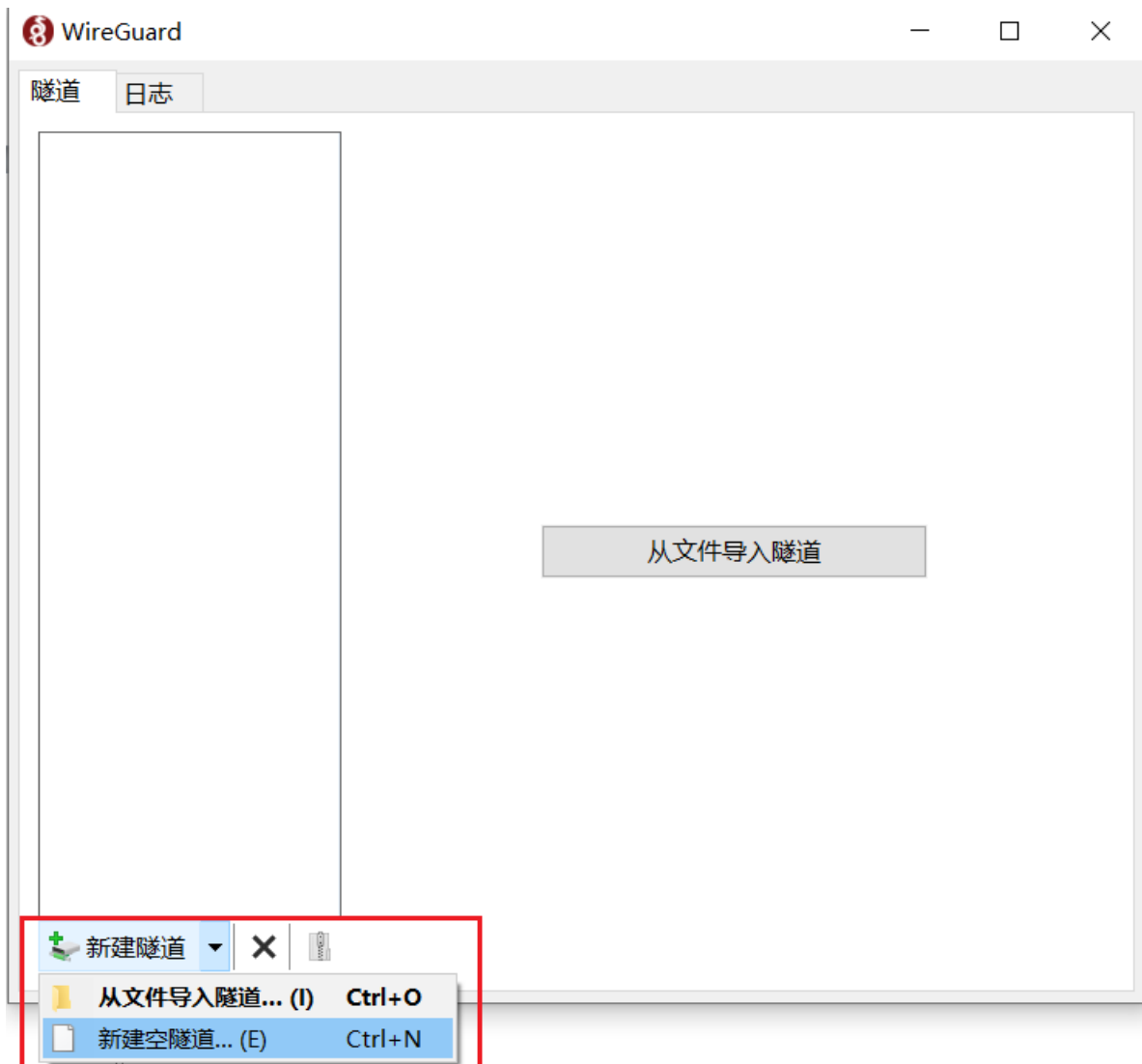


在 ip address 中添加接口 wireGuard 的接口 ip 地址 10.20.30.1/24

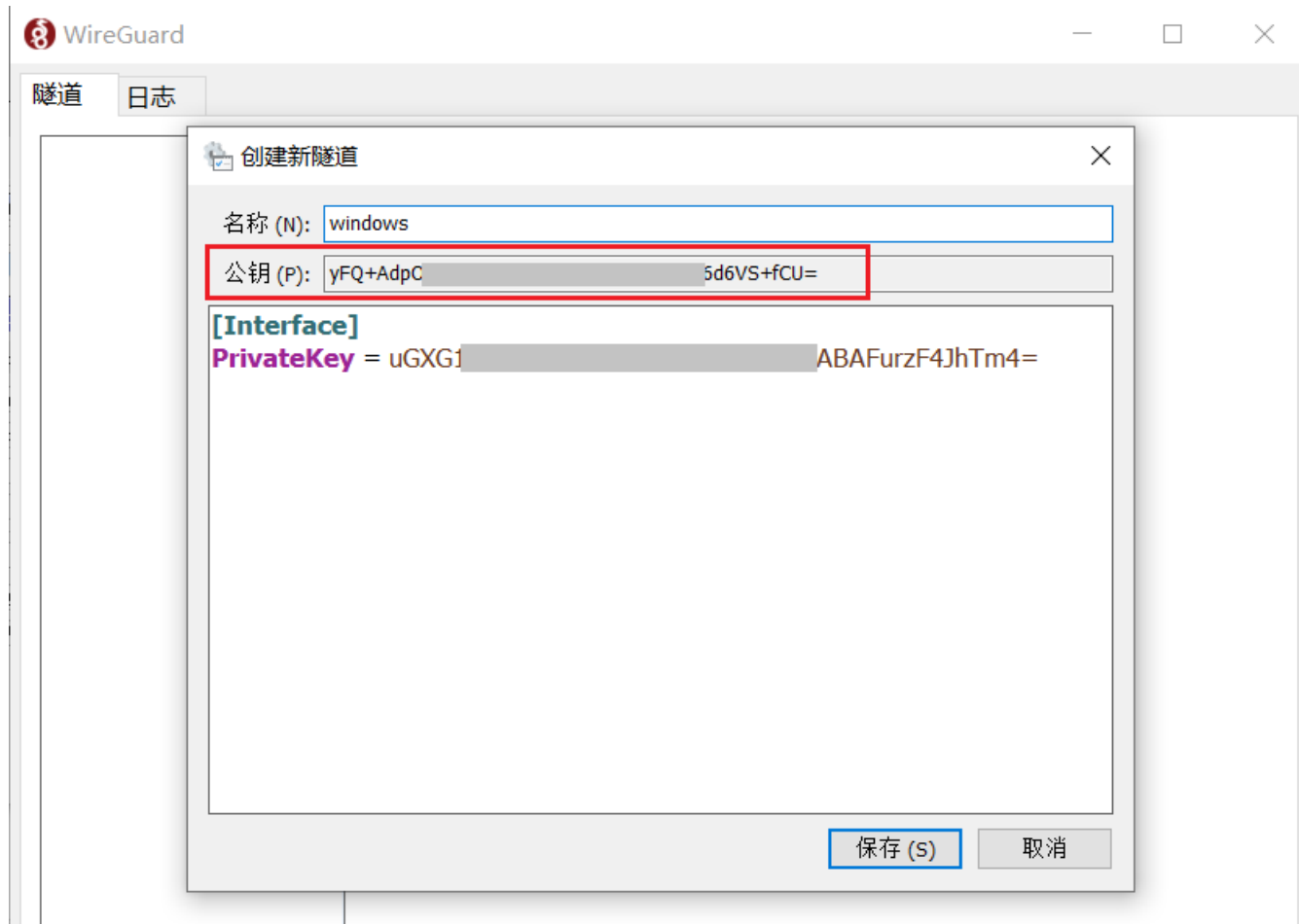


Windows 配置

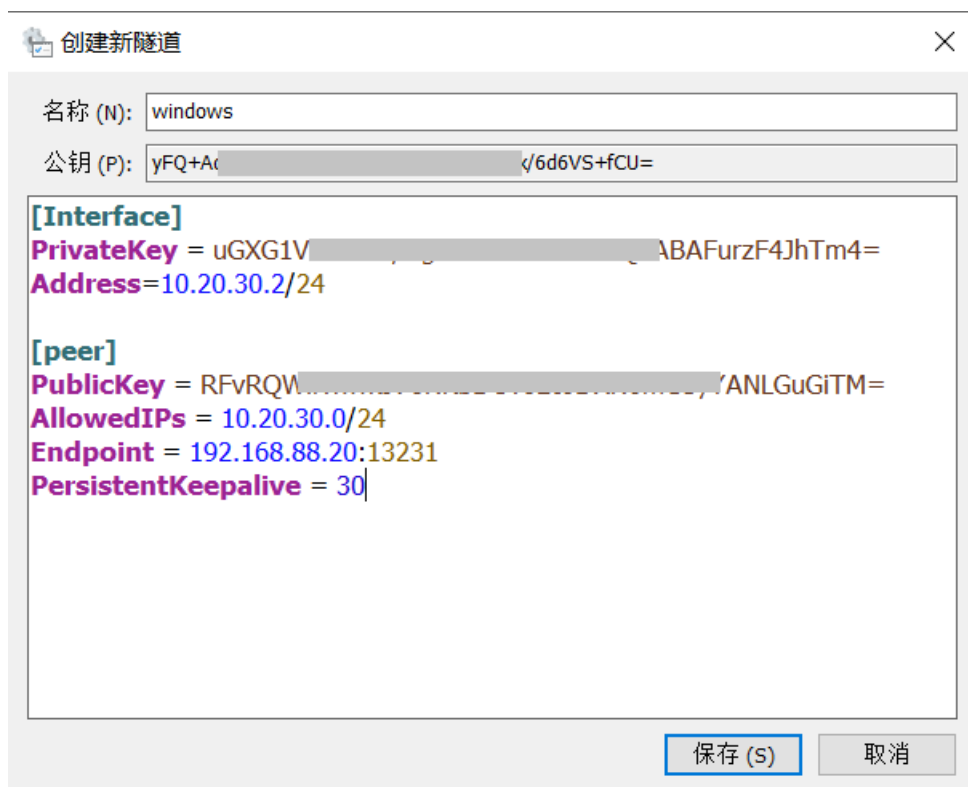
将 Windows 如何安装 WireGuard 程序就省略了，下面是安装好后，打开程序，并选择“新建空隧道”



新建后，会自动生成 windows 的公钥



然后在创建新隧道对话框中，写入以下 windows 配置



文本配置如下

[Interface]**PrivateKey** = uGXG1V...JhTm4=**Address**=10.20.30.2/24**[peer]****PublicKey** = RFvRQW...yYANLGuGiTM= (RouterOS public key)**AllowedIPs** = 10.20.30.0/24**Endpoint** = 192.168.88.20:13231**PersistentKeepalive** = 30

参数配置说明：

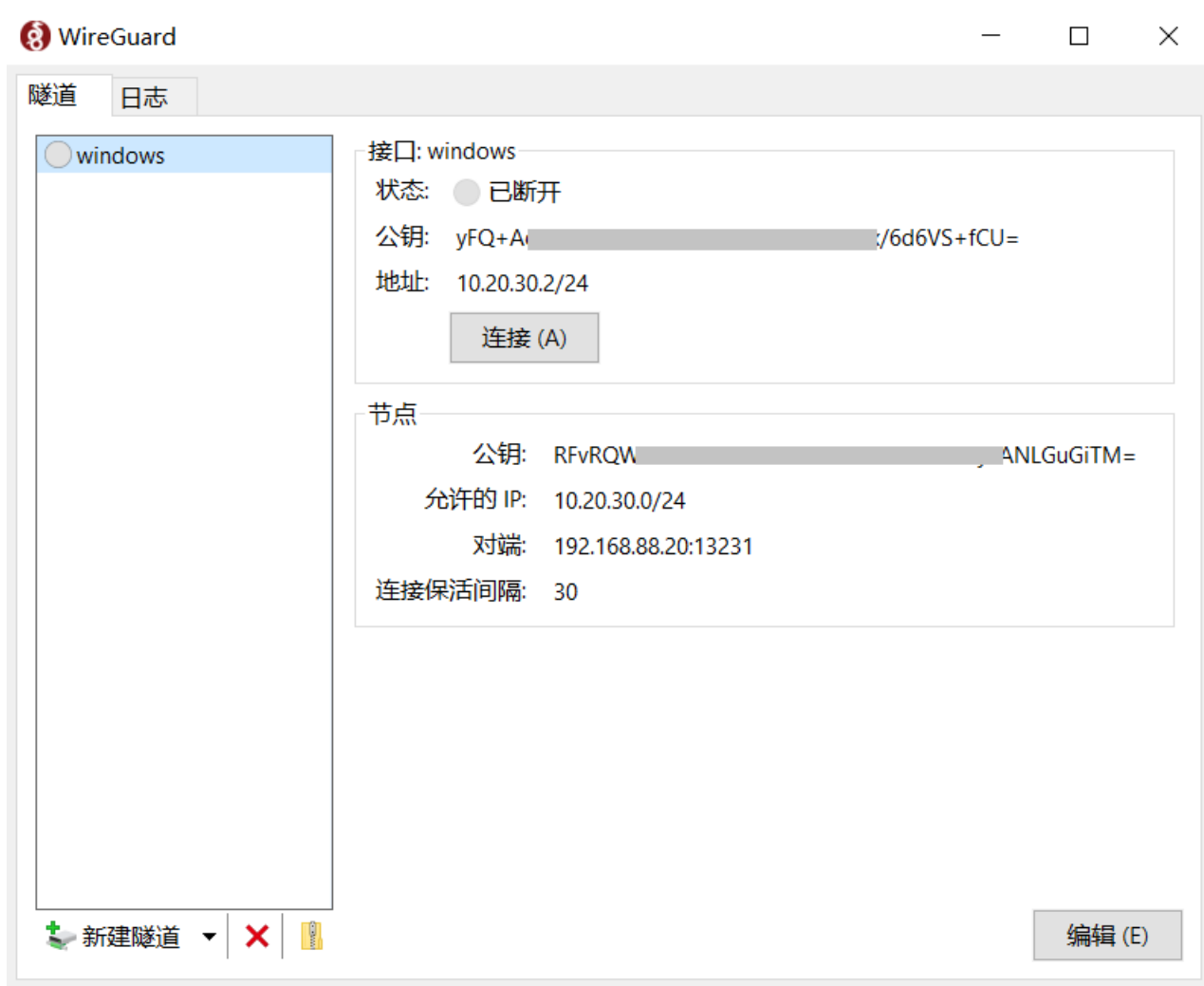
[Interface]

为 Windows 接口配置，这除了自动生成的私钥，需要添加 WireGuard 在 windows 创建的接口 IP 地址 10.20.30.2/24

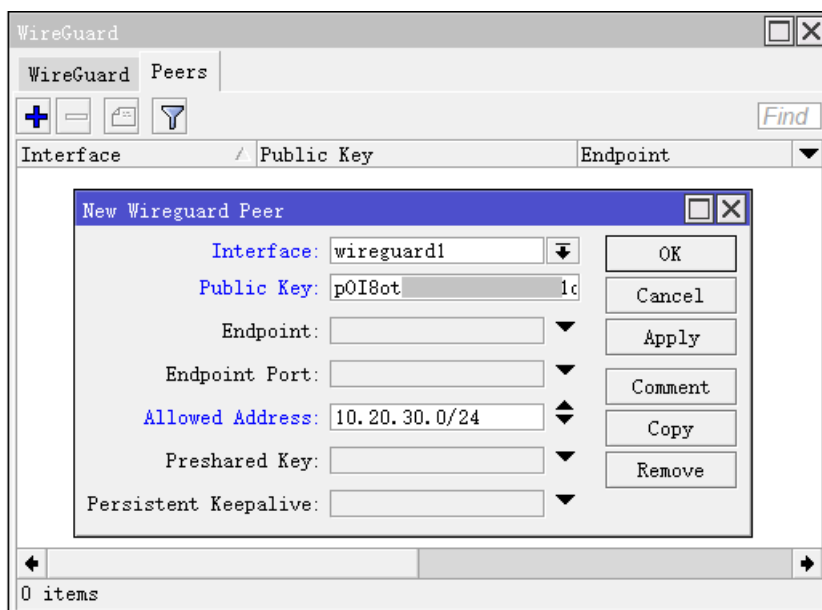
[peer]

- public key，需要指定 RouterOS 的 public key。
- endpoint (RouterOS 的外网 IP 地址)，实例中是 192.168.88.20 端口是 13231，
- allowedIPs 为允许通过的 IP 地址段，实例中为 10.20.30.0/24，
- keepalive 作用是每隔 30 秒发送一个询问数据请求，用于 nat 会话连接保持。

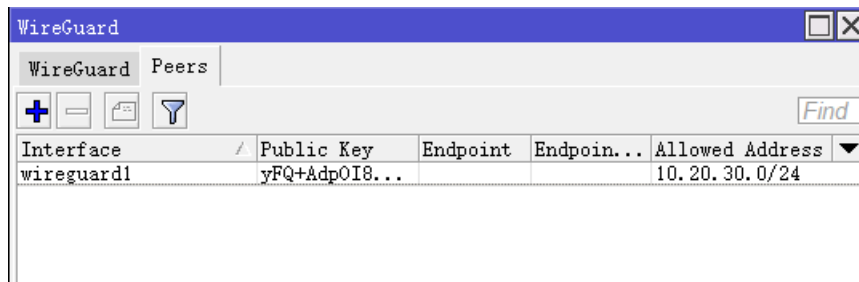
创建完成后如下：



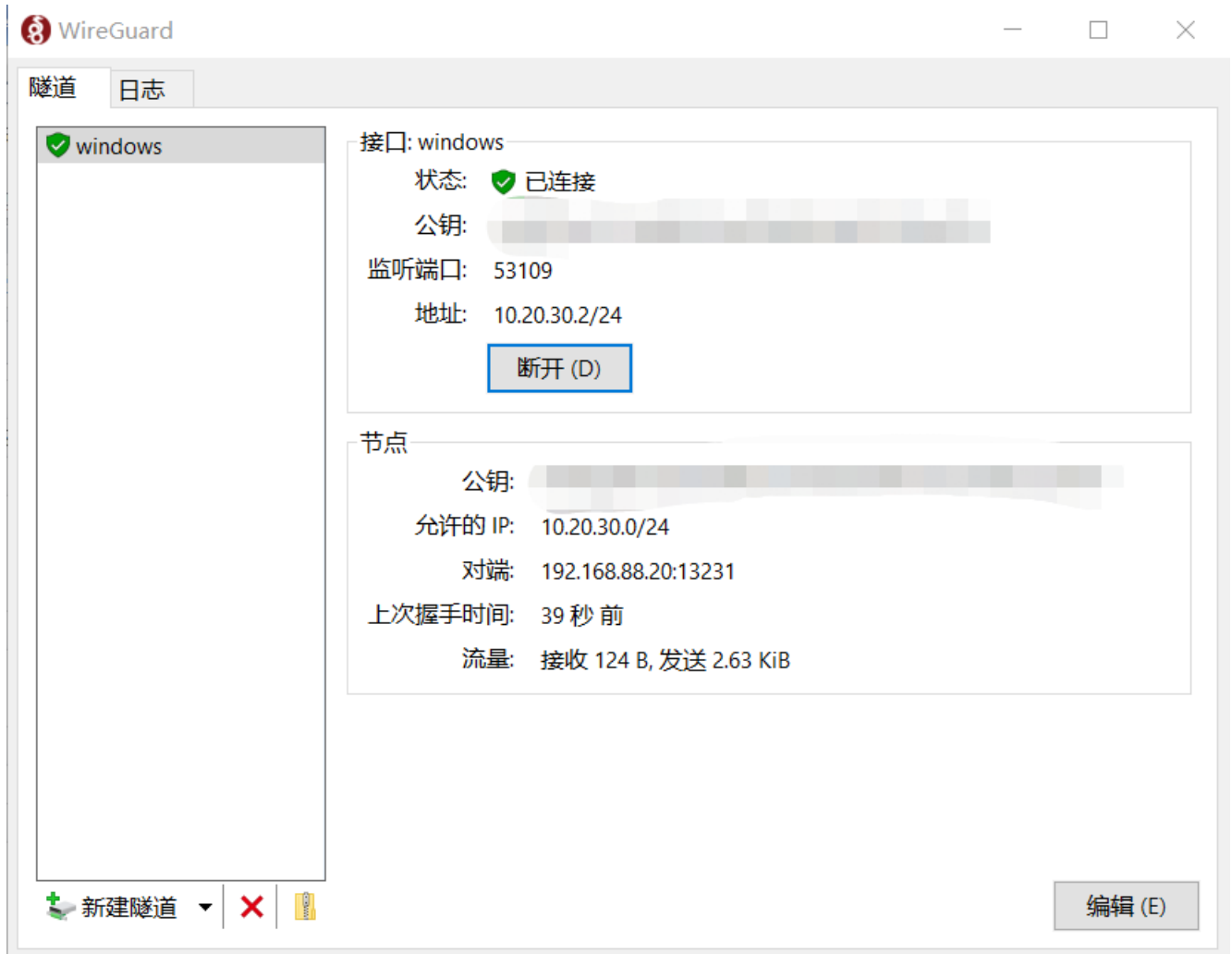
RouterOS 也在 Peer 中做相应的配置，仅需要指定 public key 为 windows 端的公钥和 allowed address，无需指定 endpoint 参数



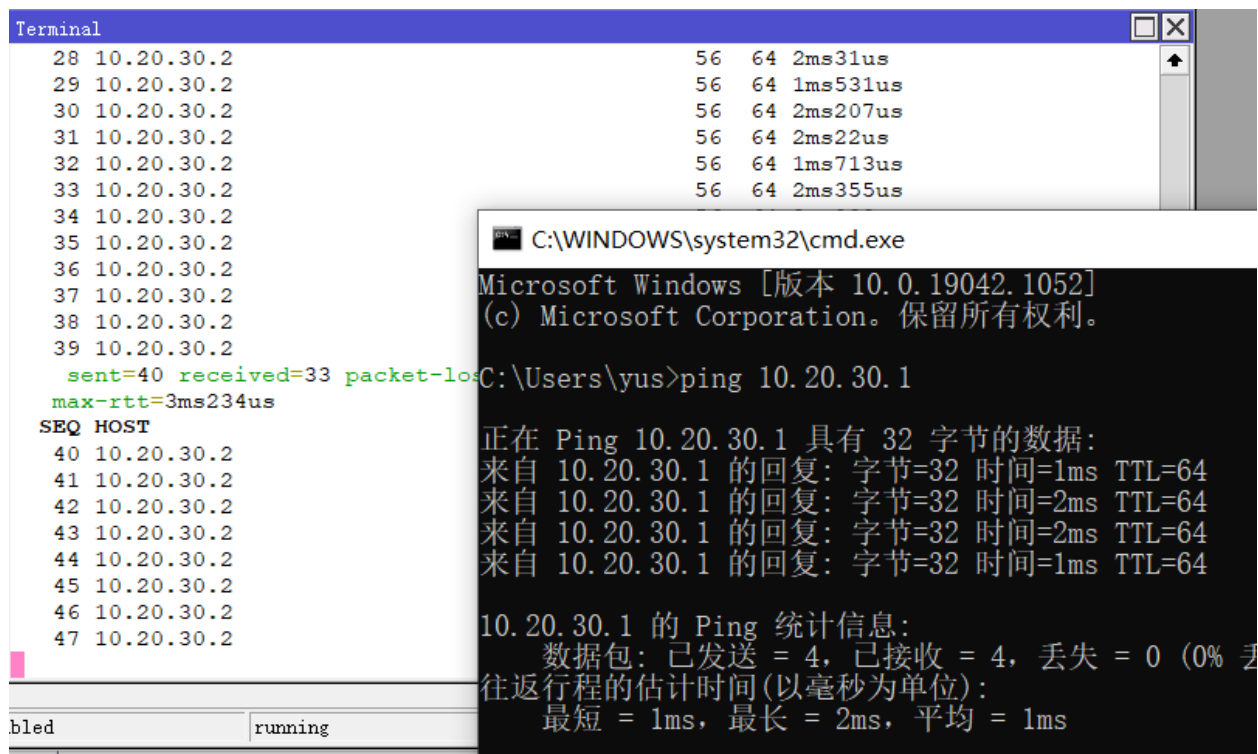
添加完成后，RouterOS 只需要将对端的公钥绑定，如果公钥匹配，会把请求的源地址和端口作为 endpoint



配置完成后，在 windows 电脑点击连接，如果连接成功后，状态会显示已连接



最后在 windows 和 RouterOS 上都进行 ping 测试



两端隧道的 IP 地址是 10.20.30.1 和 10.20.30.2，因此需要在配置中都允许 10.20.30.0/24 的地址段通过。

以上为一个基本的 Wireguard 隧道建立，如果还有其他 IP 地址段，就添加多个 IP 地址段，如果是想让任意 IP 通过，可以设置为 0.0.0.0/0 的 IP 通过。具体的访问根据你的路由配置决定。

RouterOS 代理上网配置

通常希望使用 Wireguard 隧道代理上网，根据之前的实例，RouterOS 已经配置好了网络（RouterOS 必须是公网 IP 地址），我们需要修改 RouterOS 和 windows 配置，

Windows 配置如下：

[Interface]

PrivateKey = uGXG1V...JhTm4=

Address=10.20.30.2/24

DNS = 61.139.2.69

[peer]

PublicKey = RFvRQW...yYANLGuGiTM= （RouterOS public key）

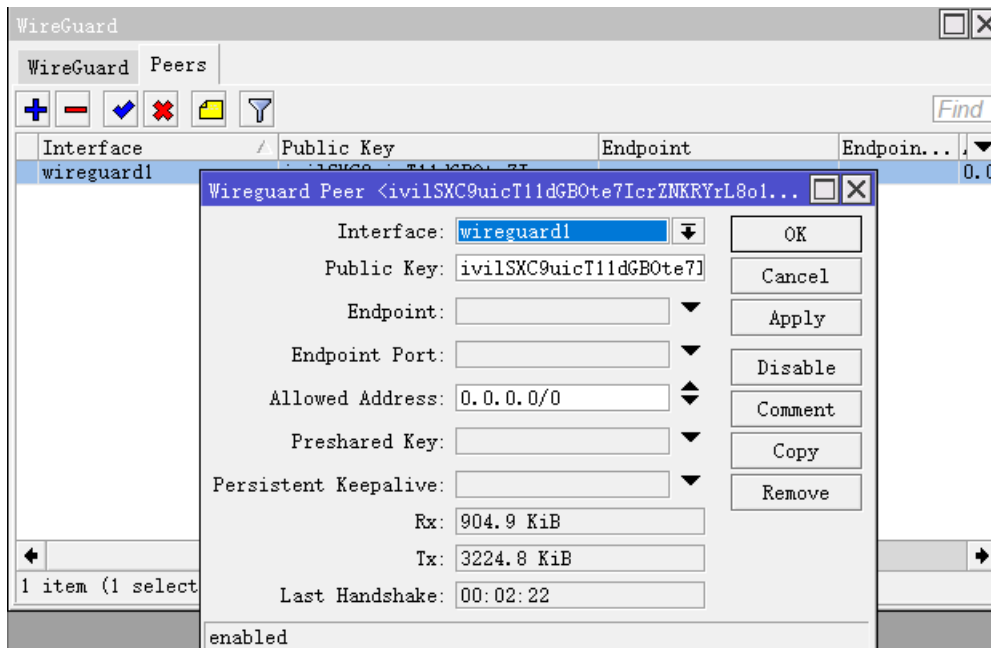
AllowedIPs = 0.0.0.0/0

Endpoint = 192.168.88.20:13231

PersistentKeepalive = 30

[interface]参数部分添加 DNS 服务器地址(如果不设置 DNS 将导致无法解析域名)，在[peer]部分设置 AllowedIPs=0.0.0.0/0 允许所有地址通过。

RouterOS 配置如下：

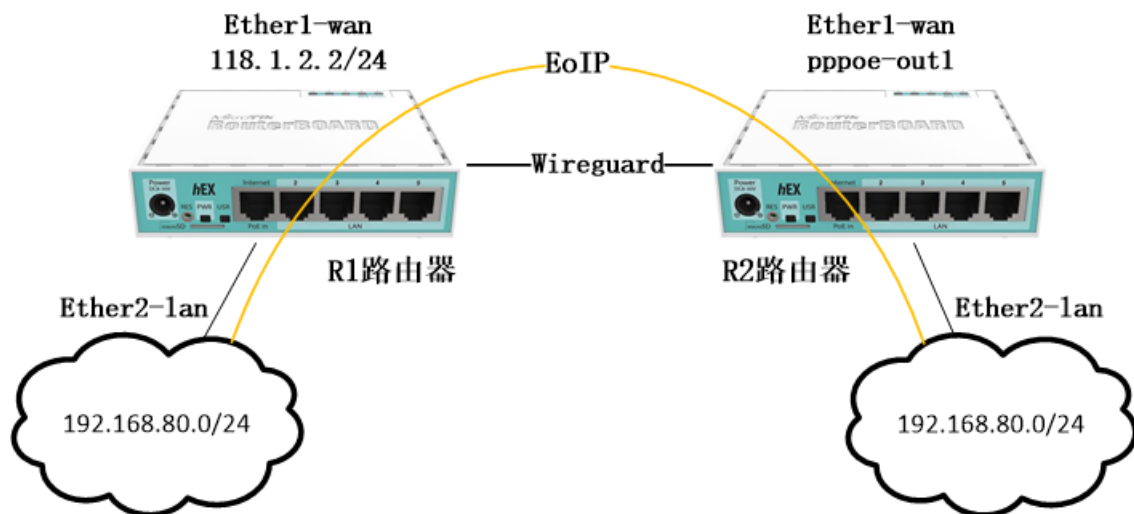


RouterOS 配置修改 allowed address 为 0.0.0.0/0，允许所有 IP 地址通过。

2.3 基于 WirGuard 的 EoIP 隧道

对于 RouterOS 的 EoIP 隧道，能提供远程网络的二层透传，不过通常要求两端是对等的，即两端 IP 地址能直接互访，建立在非 nat 网络，因此对网络要求很高！如果有一端路由器是公网 IP，而另一端是私网 IP，则无法建立 EoIP 隧道。但可以通过建立如 PPTP、L2TP、SSTP 和 WirGuard 等不要求两端使用公网 IP 地址的隧道，在这类隧道上建立 EoIP，即隧道中运行隧道！

网络拓扑如下图：



上图 R1 路由器有一个公网 IP 地址，而 R2 路由器是 PPPoE 的拨号，获取的是私网 IP 地址。

配置 R1 路由器的外网 IP 地址

添加 ether1 的 wan 口 IP 地址

```
[admin@R1] >/ip address
[admin@R1] /ip/address>add address=118.1.2.2/24 interface=ether1
```

配置 R2 路由器创建 PPPoE 拨号

```
[admin@R2] >/interface pppoe-client
[admin@R2] /interface/pppoe-client >add user=yus password=yus interface=ether1 add-default-route=yes
```

为 R1 路由器创建 wireguard 接口，端口为 13231，并查看 public-key

```
[admin@R1] /ip/address>/interface wireguard
[admin@R1] /interface/wireguard>add listen-port=13231 name=wireguard1
[admin@R1] /interface/wireguard> pr
Flags: X - disabled; R - running
0 R name="wireguard1" mtu=1420 listen-port=13231
   public-key=" a1EW/i18PQJqyH67y.....k7YgQ5feodjfk "
```

创建 R2 的 wireguard 接口，端口为 13231，并查看 public-key

```
[admin@R2] /ip/address>/interface wireguard
[admin@R2] /interface/wireguard>add listen-port=13231 name=wireguard1
[admin@R2] /interface/wireguard> print
Flags: X - disabled; R - running
0 R name="wireguard1" mtu=1420 listen-port=13231
   public-key=" i1QW/i18PQJqyH62j.....m7YgQ5feoHa2hU= "
```

创建 R1 路由的 Peer，添加 R2 路由器 wireguard 的 public-key，并允许所有地址通过

```
[admin@R1] /interface/wireguard>/interface wireguard peers
add interface=wireguard1 allowed-address=0.0.0.0/0 public-key=\
"i1QW/i18PQJqyH62j.....m7YgQ5feoHa2hU="
```

创建 R2 路由的 Peer，添加 R1 的 public-key 和允许通过所有地址外，还需要设置 endpoint 也就是 R1 路由器的公网 IP 地址

```
[admin@R1] /interface/wireguard>/interface wireguard peers
add interface=wireguard1 endpoint=118.1.2.2 allowed-address=0.0.0.0/0 public-key=\
"a1EW/i18PQJqyH67y.....k7YgQ5feodjfk"
```

为 wireguard 接口分配 IP 地址

R1 路由器

```
[admin@R1] /ip/address>add address=1.2.3.1/24 interface=wireguard1
```

R2 路由器

```
[admin@R2] /ip/address>add address=1.2.3.2/24 interface=wireguard1
```

配置 EoIP 隧道

为 R1 路由器创建 EoIP 隧道，remote-address 为 R2 路由器的 wireguard1 接口 IP 地址，隧道 id 为 10，mtu 设置为 1500

```
[admin@R1] /ip/address>/interface eoip
[admin@R1] /interface/eoip >add name=eoip-tunnel1 remote-address=1.2.3.2 tunnel-id=10 mtu=1500
```

为 R2 路由器创建 EoIP 隧道，remote-address 为 R1 路由器的 wireguard1 接口 IP 地址，隧道 id 为 10，mtu 设置为 1500

```
[admin@R2] /interface/eoip >add name=eoip-tunnel1 remote-address=1.2.3.1 tunnel-id=10 mtu=1500
```

创建 bridge

为 R1 路由器创建 bridge1 接口

```
[admin@R1] >/interface bridge
[admin@R1] /interface/bridge>add name=bridge1
```

将 eoip-tunnel1 和内网口 ether2 加入 bridge1

```
[admin@R1] /interface/bridge>/interface bridge port
[admin@R1] /interface/bridge/port >add bridge=bridge1 interface=eoip-tunnel1
[admin@R1] /interface/bridge/port >add bridge=bridge1 interface=ether2
```

R2 路由器创建 bridge1 接口

```
[admin@R2] >/interface bridge
[admin@R2] /interface/bridge>add name=bridge1
```

将 eoip-tunnel1 和内网口 ether2 加入 bridge1

```
[admin@R2] /interface/bridge>/interface bridge port
[admin@R2] /interface/bridge/port >add bridge=bridge1 interface=eoip-tunnel1
[admin@R2] /interface/bridge/port >add bridge=bridge1 interface=ether2
```

R1 路由器为 bridge1 分配 IP 地址 192.168.80.1

```
[admin@R1] /ip/address>add address=192.168.80.1/24 interface=bridge1
```

R2 路由器为 bridge1 分配 IP 地址 192.168.80.2

```
[admin@R2] /ip/address>add address=192.168.80.2/24 interface=bridge1
```

从 R1 测试 R2 的 192.168.80.2:

```
[admin@R1] > ping 192.168.80.2
```

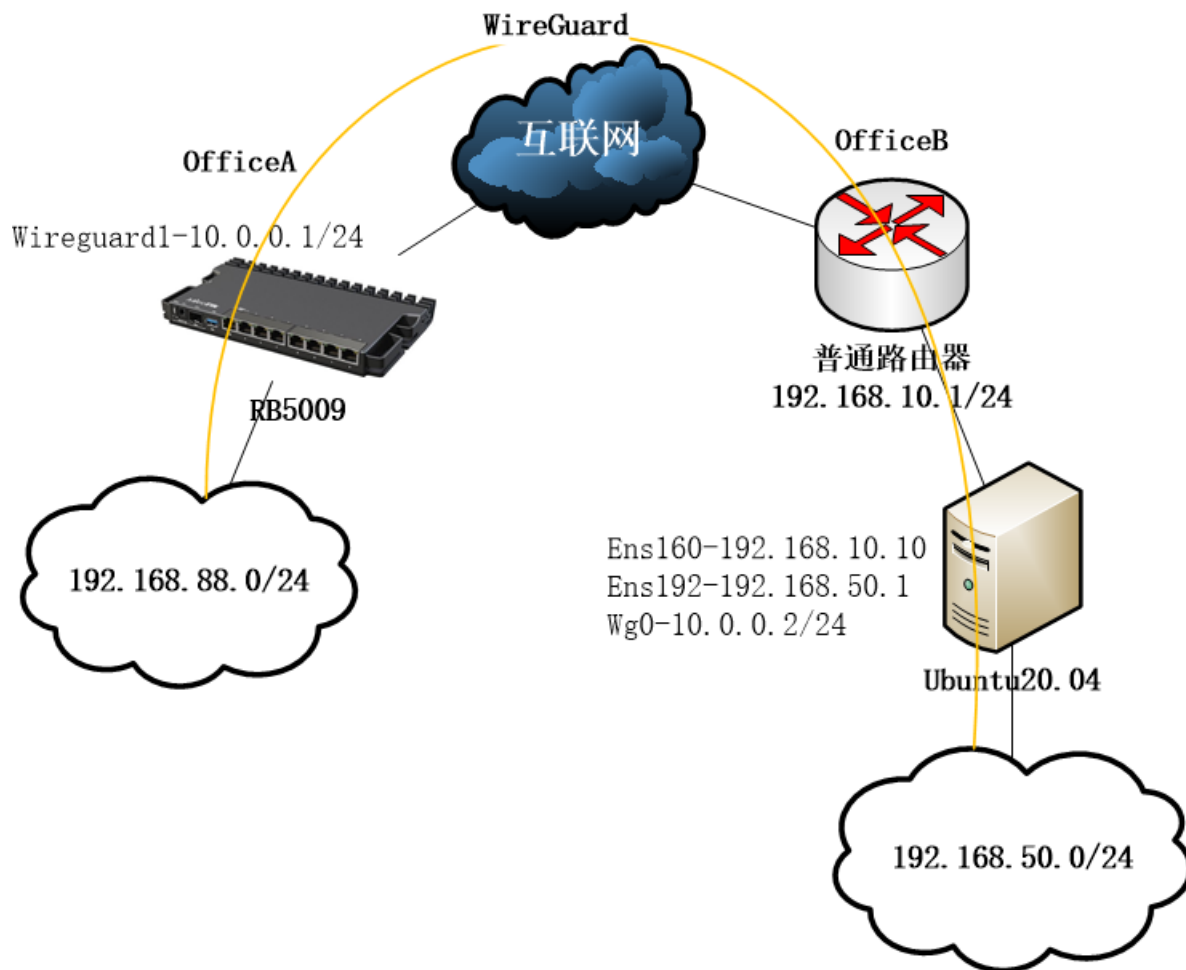
SEQ	HOST	SIZE	TTL	TIME	STATUS
0	192.168.80.2	56	64	997us	
1	192.168.80.2	56	64	1ms271us	
2	192.168.80.2	56	64	1ms125us	
3	192.168.80.2	56	64	1ms104us	
4	192.168.80.2	56	64	1ms118us	
5	192.168.80.2	56	64	1ms110us	

sent=6 received=6 packet-loss=0% min-rtt=997us avg-rtt=1ms120us
max-rtt=1ms271us

这样 R1 和 R2 路由器两端的内网已经打通 EoIP 的二层隧道。

2.4 与 Ubuntu20.04 建立 WireGuard

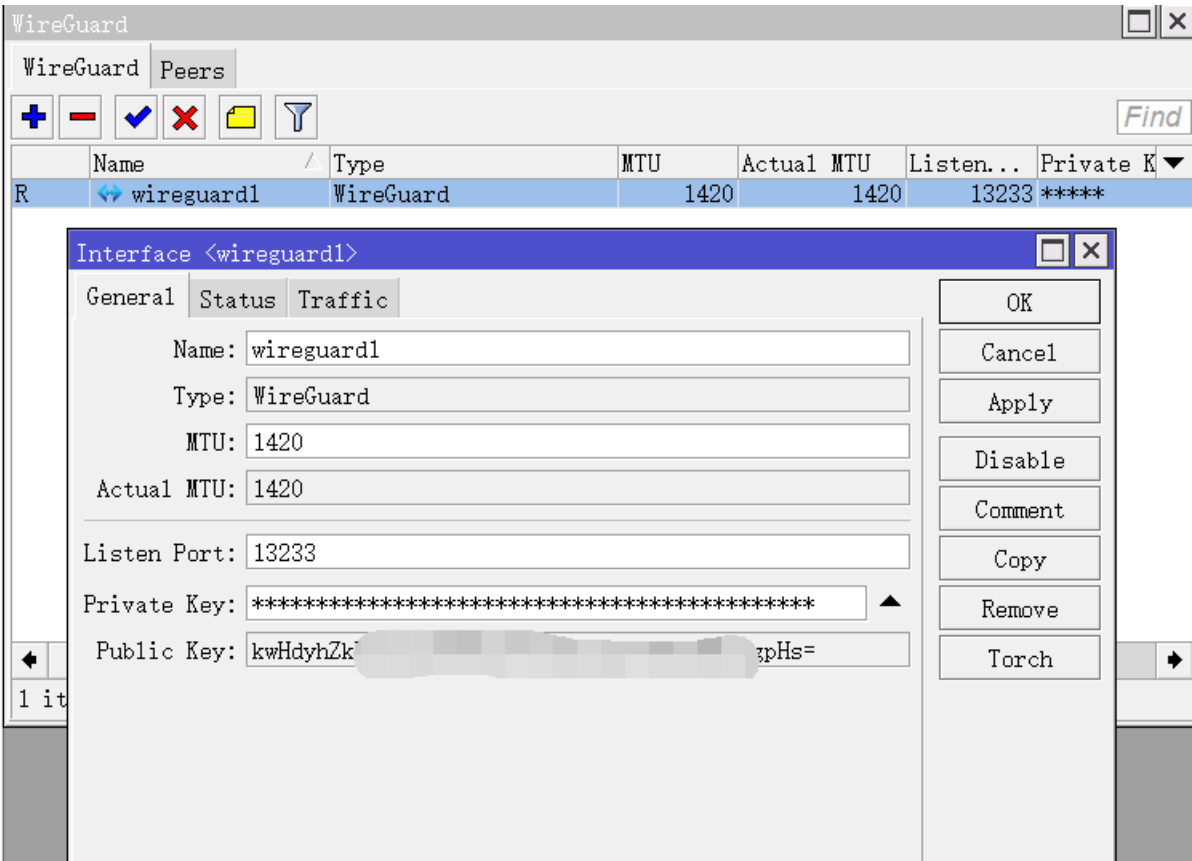
考虑这样一个网络情况，当两个异地办公区，officeA 和 officeB 需要建立 WireGuard 隧道，officeA 是 RouterOS，而 officeB 是普通路由器，受限于普通路由器不支持 WireGuard 协议，无法建立互联，但 officeB 可以搭建 VM 虚拟机搭建一台 ubuntu20.04 服务器，这样可以通过 ubuntu20.04 与 RouterOS 建立 WireGuard 的互联，如下拓扑：



在这个拓扑上可以看到，RB5009 和 Ubuntu20.04 建立 WireGuard 隧道，在 ubuntu 下有一个独立的网络 192.168.50.0/24 与 OfficeA 的 192.168.88.0/24 做路由互联访问。

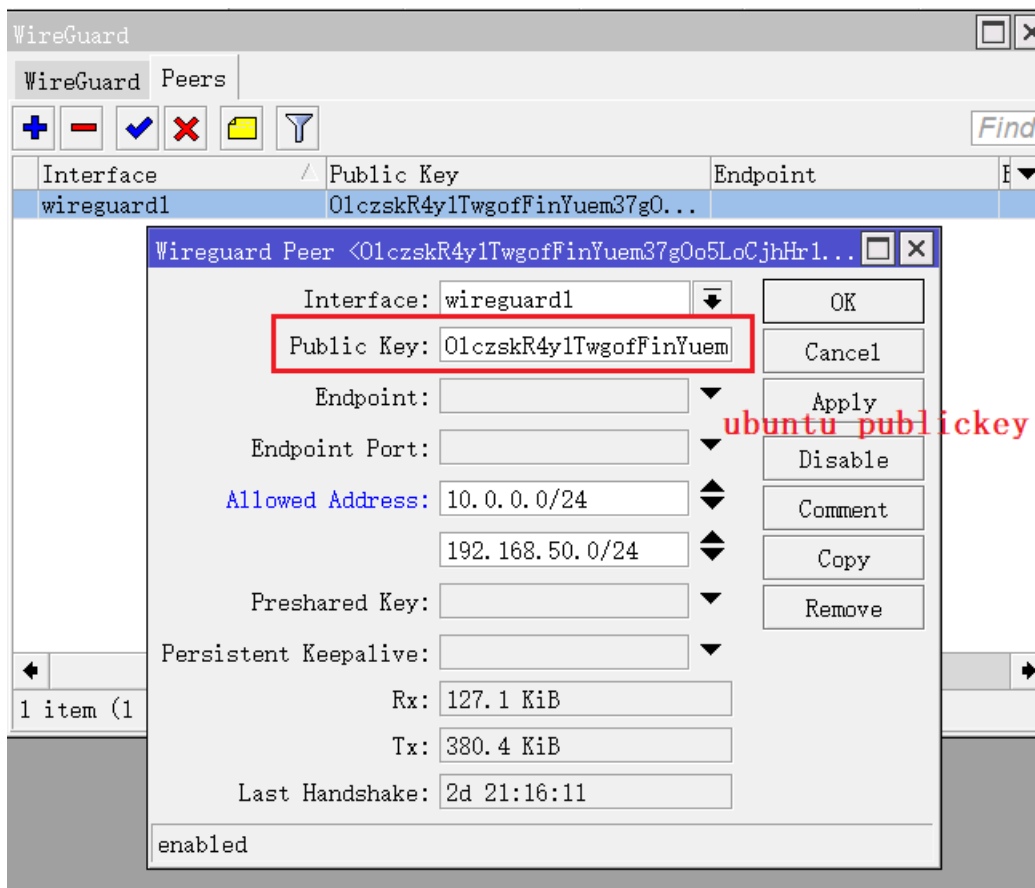
RB5009 配置

基于 RouterOS 的 RB5009，能获取公网 IP 地址上网，具体网络接入配置在此省略，直接介绍配置 wireguard，首先在 wireguard 下新建一个接口 wireguard1，如下：

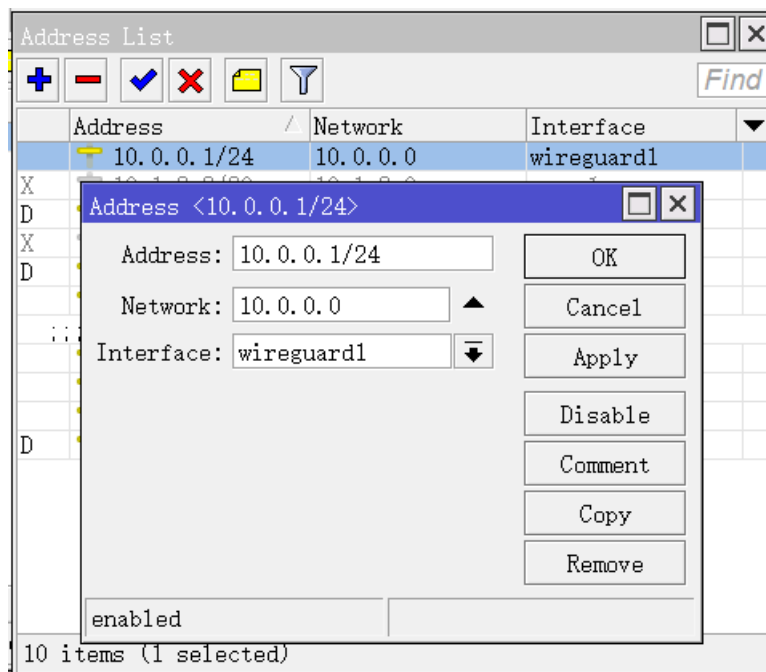


记录下 RouterOS 上 wireguard1 接口的 Public Key，需要在 ubuntu 配置时添加。

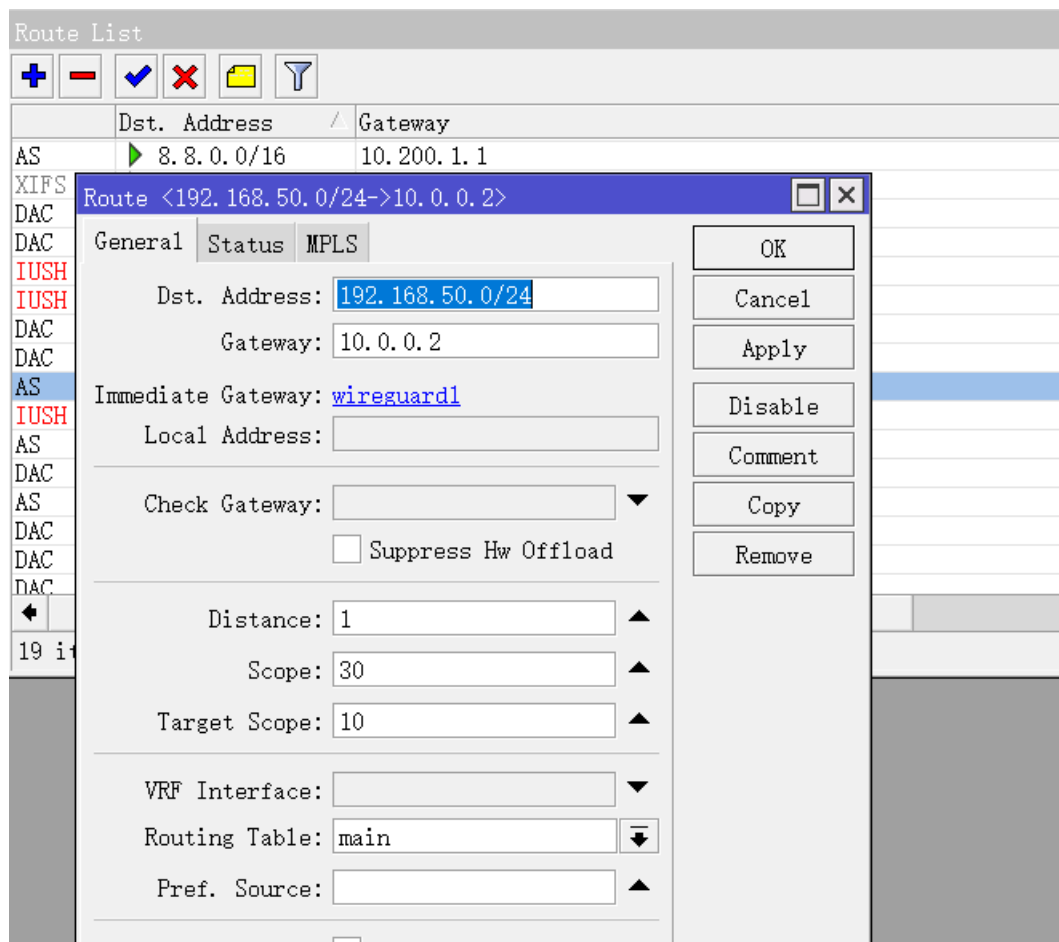
RouterOS 在 wireguard 的 peer 添加 ubuntu 的客户端连接参数（Public Key 是已经获取，在后面对 ubuntu 配置的介绍会有），由于 ubuntu 是在 officeB 的内网，所以无需指定 endpoint 的地址。



进入/ip address，添加 wireguard1 接口 IP 地址 10.0.0.1/24，winbox 配置如下：



在/ip route 下添加到目标 IP 地址段 192.168.50.0/24 的静态路由，winbox 配置如下：



Ubuntu 20.04 配置

首先通过 apt-get 更新镜像库，并安装 wireguard，操作如下：

```
apt-get update
apt-get install wireguard
```

安装完成后，ubuntu 生成 wireguard 的私钥和公钥，并存放在指定目录下

```
wg genkey | tee /etc/wireguard/privatekey | wg pubkey | tee /etc/wireguard/publickey
```

在 /etc/wireguard/publickey 下查看生成的 Public Key，用于添加到 RouterOS 的 wireguard1 的 Peer 配置，之前在 RouterOS 配置提到。

```
cat /etc/wireguard/publickey
```

同样查看 privatekey，用于参数配置

```
cat /etc/wireguard/ privatekey
```

在 /etc/wireguard/，通过 vi 创建配置文件 wg0.conf

```
vi /etc/wireguard/wg0.conf ,
```

添加以下内容，

- Interface 接口参数，服务端口是 13233,接口 IP 地址是 10.0.0.2/24，并设置 Ubuntu 自己的 privatekey
- Peer 为对端参数: PublicKey 填写 RouterOS publickey, AllowedIPs 填写需要通过所有 IP 地址，Endpoint 设置 RouterOS 的公网 IP 地址（假设为 88.88.88.88），端口为 13233，设置 PersistentKeepalive 为 25 秒（用于存活探测）

```
[Interface]
ListenPort = 13233
Address = 10.0.0.2/24
PrivateKey = Ubuntuprivatekey

[Peer]
PublicKey = 复制 RouterOSpublickey
AllowedIPs = 0.0.0.0/0
Endpoint = 88.88.88.88:13233
PersistentKeepalive = 25
```

然后修改配置文件和 key 的权限

```
chmod 600 /etc/wireguard/{privatekey,wg0.conf}
```

这个时候如果 ubuntu 作为路由接入其他网络，需开启 Linux 的 ip forward 转发功能，通 vi 编辑 **/etc/sysctl.conf**，找到这一行 **#net.ipv4.ip_forward = 1** 去掉注释符 “#” 修改为

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

退出 vi 编辑，立即生效指令如下，

```
sysctl -p
```

通过命令启用 wg0 接口

```
root@yus:~# wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.0.0.2 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] wg set wg0 fwmark 51820
[#] ip -4 route add 0.0.0.0/0 dev wg0 table 51820
```



```
[#] ip -4 rule add not fwmark 51820 table 51820
[#] ip -4 rule add table main suppress_prefixlength 0
[#] sysctl -q net.ipv4.conf.all.src_valid_mark=1
[#] iptables-restore -n
```

注意当 AllowedIPs 是 0.0.0.0/0 时，Linux 会自动创建一个路由表，生成策略路由，如果你配置主机 IP 和 ubuntu 不在同一网段，会造成路由中断，切记考虑你连接 ubuntu 的 IP 配置

通过 wg 命令查看连接状态

```
root@yus:~# wg
interface: wg0
public key: OlczskR4y1Txxxxxxxxm37gOo5LoCjhHr1u0VUDzY=
private key: (hidden)
listening port: 13233
fwmark: 0xca6c
peer: kWHdyhZkxxxxPHDvMRsgpHs=
endpoint: 88.88.88.88:13233
allowed ips: 0.0.0.0/0
latest handshake: 1 minute, 47 seconds ago
transfer: 10.95 KiB received, 11.41 KiB sent
persistent keepalive: every 25 seconds
```

如果不希望允许所有 IP 通过，可以这样修改配置文件

```
[Interface]
ListenPort = 13233
Address = 10.0.0.2/24
PrivateKey = privatekey

[Peer]
PublicKey = publickey
AllowedIPs = 10.0.0.0/24,192.168.88.0/24
Endpoint = 88.88.88.88:13233
PersistentKeepalive = 25
```

修改 AllowedIPs 后，使用以下命令关闭 wg0 接口

```
wg-quick down wg0
```

再次启用网卡时，可以看到通过 ip -4 route add 添加了静态路由

```
root@yus:~# wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.0.0.2/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
[#] ip -4 route add 192.168.88.0/24 dev wg0
```

查看路由，可以看到 **wireguard** 自动添加了静态路由：

```
root@yus:~# ip route
default via 192.168.10.1 dev ens160 proto static
10.0.0.0/24 dev wg0 proto kernel scope link src 10.0.0.2
192.168.10.0/24 dev ens160 proto kernel scope link src 192.168.10.10
192.168.50.0/24 dev ens192 proto kernel scope link src 192.168.50.11
192.168.88.0/24 dev wg0 scope link
```

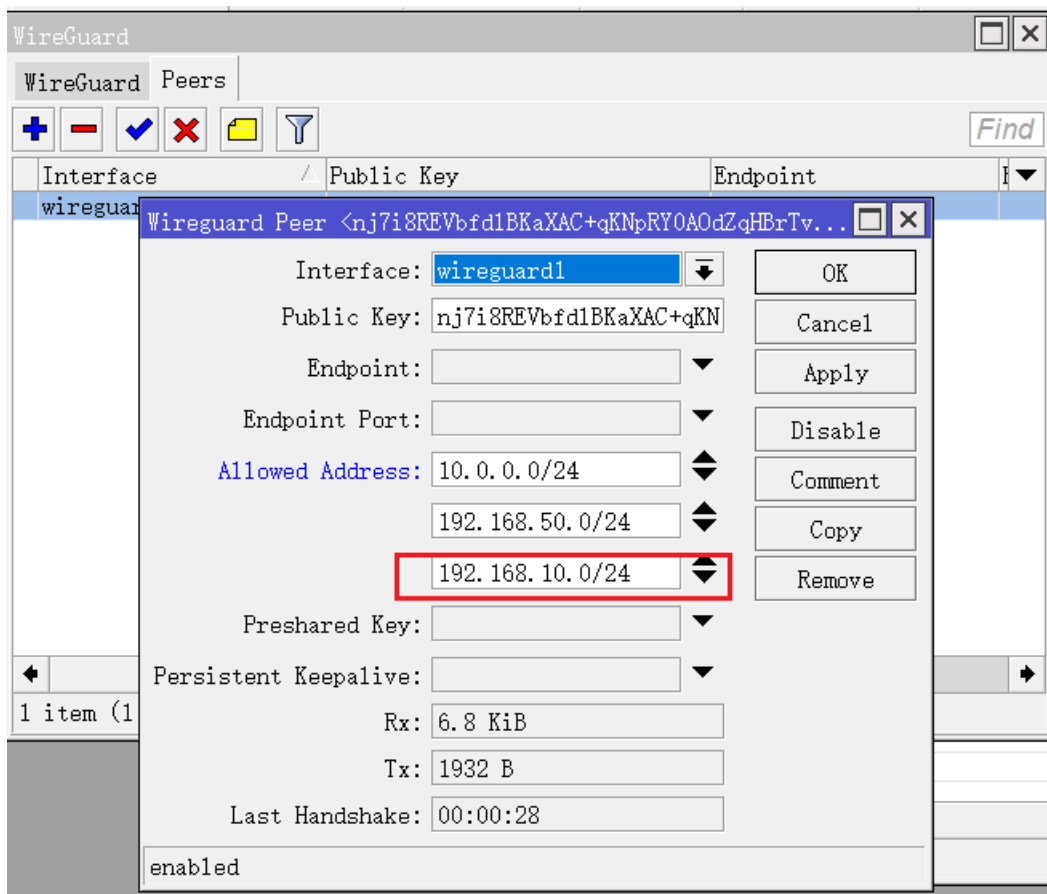
这样 RB5009 和 ubuntu 的 wireguard 隧道建立，可以测试 192.168.88.0/24 到 192.168.50.0/24 路由是否连通。

提醒：192.168.50.0/24 网络是 ubuntu 的 ens192 网卡下，ens160 接入的是实际的普通路由器 192.168.10.0/24 局域网，如果希望 192.168.10.0/24 能访问到 RB 的 192.168.88.0/24 网络，需要在普通路由器配置静态路由指向 ubuntu 的 ens160 的接口 IP，如果是华为路由器配置如下：

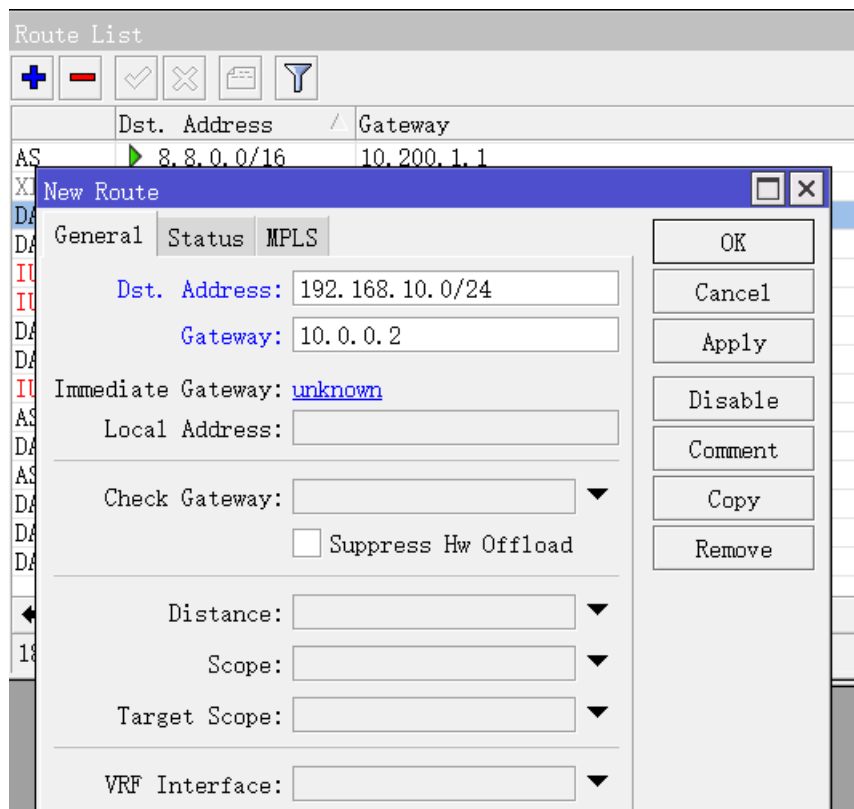
```
[HuaWei]ip route-static 192.168.88.0 24 192.168.10.10
```

RouterOS 补充配置

在 peer 上增加允许 IP 地址段 192.168.10.0/24，如果你是 0.0.0.0/0 则无需做配置



需要在 ip route 下添加到 192.168.10.0/24 经过 10.0.0.2 的静态路由：

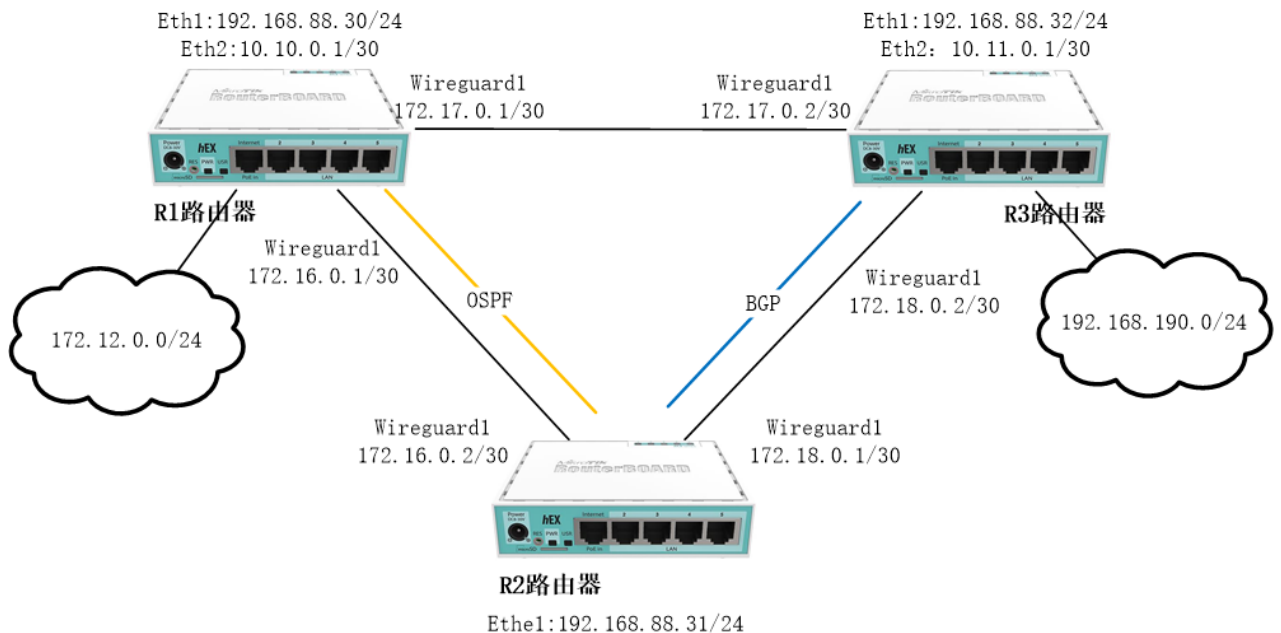


注意：以上配置采用静态路由方式，不考虑 nat 规则，如果你在 ubuntu 或者 routeros 配置了 nat 规则，需要注意接口出方向被转为接口 IP 地址的情况，这里不做讨论。

2.5 Wireguard 的多节点组网

在多个异地网络节点，需使用 Wireguard 隧道协议进行组网，搭建一个相互对等的互访网络结构，非中心点方式，在下面的实例中每台 RouterOS 使用一个 Wireguard 实例接口，创建多个 Peer 使用 OSPF 和 BGP 方式建立路由关系。

网络拓扑如下，R1 和 R2 之间建立 OSPF，R2 与 R3 之间使用 BGP，如下拓扑图，为了方便实验，3 台路由器都在一个 192.168.88.0/24 的局域网，三台路由器都创建一个 Wireguard1 的隧道接口，通过 Peer 连接对端。



特别说明：此事例在每台 RouterOS 配置的单 Wireguard 实例接口（wireguard1），因此在一台设备上的多个 Peer 配置中不允许相同 allowed-address 出现，否则会出现连接故障。

特别注意：此方式由于是单接口实例，并不支持对于多个点连接 Peer 有重复的 IP 或 IP 段出现在 allowed-address，解决的方法则使用多 Wireguard 实例接口（监听的 UDP 端口也需不同），即一个 Wireguard 接口对应一个远端设备，当设备数量增加时，同时也会带来复杂逻辑对应配置关系，而对于 Mesh 组网应该采用多实例的 Wireguard 接口对应多个远端节点，使得不同实例下的 Peer 允许相同 IP 地址段通过，来实现路由线路的 Mesh 冗余组网。

三台路由器的网络配置：

R1 路由器：

- Ether1 接口 IP: 192.168.88.30/24,
- Ether2 接口 IP: 10.10.0.1/30
- To R2 Wireguard 互联 IP: 172.16.0.1/30
- To R3 Wireguard 互联 IP: 172.17.0.1/30

R2 路由器：

- Ether1 接口 IP: 192.168.88.31/24,

- To R1 Wireguard 互联 IP: 172.16.0.2/30
- To R3 Wireguard 互联 IP: 172.18.0.1/30

R3 路由器:

- Ether1 接口 IP: 192.168.88.32/24,
- Ether2 接口 IP: 10.11.0.1/30
- To R2 Wireguard 互联 IP: 172.17.0.2/30
- To R3 Wireguard 互联 IP: 172.18.0.2/30

注意: 三台设备的 Wireguard 两两相连的互联 IP 地址子网需要区分开, 甚至不能在一个 Wireguard 实例下的不同 Peer 出现相同子网情况, Peer 中的 `allowed-address` 会无法正确判断子网归属于那个 Peer 导致连接失效。对于单 Wireguard 实例同时与多点的连接时需注意这个问题, 例如在使用 OSPF 组网时, 涉及到向对端设备发送组播 Hello 报文时会受到此影响, 只能通过多 Wireguard 接口来解决。

在此实例中使用 OSPF 与 BGP 组网, 当然 BGP 连接不存在 OSPF 的组播 Hello 报文 IP 相同问题, 导致一个 Wireguard 实例下多个 Peer 使用相同 IP 地址/子网连接失效问题。

基础网络配置

R1 路由器

配置 IP 地址:

```
[admin@R1] > /ip address
[admin@R1] /ip/address>add address=192.168.88.30/24 interface=ether1
[admin@R1] /ip/address>add address=10.10.0.1/30 interface=ether2
```

创建 Wireguard 接口

```
[admin@R1] /ip/address>/interface wireguard
[admin@R1] /interface/wireguard >add listen-port=13231 name=wireguard1
```

创建完成后自行查看 public key, 用于对端连接, 这里不再赘述

R2 路由器

配置 IP 地址:

```
[admin@R2] > /ip address
[admin@R2] /ip/address>add address=192.168.88.31/24 interface=ether1
```

创建 Wireguard 接口

```
[admin@R2] /ip/address>/interface wireguard
```

```
[admin@R2] /interface/wireguard >add listen-port=13231 name=wireguard1
```

创建完成后自行查看 public key，用于对端连接，这里不再赘述

R3 路由器

配置 IP 地址：

```
[admin@R3] > /ip address
[admin@R3] /ip/address>add address=192.168.88.32/24 interface=ether1
```

创建 Wireguard 接口

```
[admin@R3] /ip/address>/interface wireguard
[admin@R3] /interface/wireguard >add listen-port=13231 name=wireguard1
```

创建完成后自行查看 public key，用于对端连接，这里不再赘述

Wireguard 连接

分别给 3 台路由器的 Wireguard1 接口分配对应 的 IP 地址，这里每台路由器的 Wireguard1 接口配置两个 IP 地址

R1 路由器

```
[admin@R1] /interface/wireguard >/ip address
[admin@R1] /ip/address> add address=172.16.0.1/30 interface=wireguard1
[admin@R1] /ip/address> add address=172.17.0.1/30 interface=wireguard1
```

R2 路由器

```
[admin@R2] /interface/wireguard >/ip address
[admin@R2] /ip/address> add address=172.16.0.2/30 interface=wireguard1
[admin@R2] /ip/address> add address=172.18.0.1/30 interface=wireguard1
```

R3 路由器

```
[admin@R3] /interface/wireguard >/ip address
[admin@R3] /ip/address> add address=172.17.0.2/30 interface=wireguard1
[admin@R3] /ip/address> add address=172.18.0.2/30 interface=wireguard1
```

3 台路由器的 IP 地址和 Wireguard 创建完成后，测试网络连接正常，即可以开始配置 Wireguard 的 Peer

R1 路由器

连接 R2 路由器的 192.168.88.31，指定端口 13231 和对端的 Public key，设置 allowed-address=172.16.0.0/30，由于要和 R2 建立 OSPF 关系，需允许组播地址 224.0.0.5 通过，发送 Hello 报文，否则 OSPF 无法建立

```
[admin@R1] /interface/wireguard >peer
[admin@R1] /interface/wireguard/peers> add allowed-address=172.16.0.0/30,224.0.0.5/32
endpoint-address=192.168.88.31 endpoint-port=\
13231 interface=wireguard1 persistent-keepalive=30s public-key=\
"EZIREKCgf4bwS+kEwzKXsVoayai9LfEVwG+tTghLhTA="
```

连接 R3 路由器的 192.168.88.32，指定端口 13231 和对端的 Public key，设置 allowed-address=172.17.0.0/30

```
[admin@R1] /interface/wireguard/peers>add allowed-address=172.17.0.0/30
endpoint-address=192.168.88.32 endpoint-port=13231 \
interface=wireguard1 persistent-keepalive=10s public-key=\
"WB/NYqr4y/9lyedhOb9/UYLwBet+kG6B7ROHD56h+FE="
```

R2 路由器

连接 R1 路由器的 192.168.88.30，指定端口 13231 和对端的 Public key，设置 allowed-address=172.16.0.0/30，需允许组播地址 224.0.0.5 通过，发送 Hello 报文，否则 OSPF 无法建立

```
[admin@R2] /interface/wireguard >peer
[admin@R2] /interface/wireguard/peers> add allowed-address=172.16.0.0/30,224.0.0.5/32
endpoint-address=192.168.88.30 \
endpoint-port=13231 interface=wireguard1 persistent-keepalive=10s \
public-key="znDZo7Jotu2Vlgk3lu8ZNfQoelG5bTnUvssHpOiJPTQ="
```

连接 R3 路由器的 192.168.88.32，指定端口 13231 和对端的 Public key，设置 allowed-address=172.18.0.0/30

```
[admin@R2] /interface/wireguard/peers> add allowed-address=172.18.0.0/30
endpoint-address=192.168.99.32 endpoint-port=\
13231 interface=wireguard1 persistent-keepalive=10s public-key=\
"WB/NYqr4y/9lyedhOb9/UYLwBet+kG6B7ROHD56h+FE="
```

R3 路由器

连接 R1 路由器的 192.168.88.30，指定端口 13231 和对端的 Public key，设置 allowed-address=172.17.0.0/30

```
[admin@R3] /interface/wireguard >peer
[admin@R3] /interface/wireguard/peers> add allowed-address=172.17.0.0/30
endpoint-address=192.168.88.30 endpoint-port=\
13231 interface=wireguard1 persistent-keepalive=10s public-key=\
"znDZo7Jotu2Vlgk3lu8ZNfQoelG5bTnUvssHpOiJPTQ="
```

连接 R2 路由器的 192.168.88.31，指定端口 13231 和对端的 Public key，设置 allowed-address=172.18.0.0/30

```
[admin@R3] /interface/wireguard/peers>add allowed-address=172.18.0.0/30
endpoint-address=192.168.88.31 endpoint-port=\
13231 interface=wireguard1 persistent-keepalive=10s public-key=\
"EZIREKCgf4bwS+kEwzKXsVoayai9LfEVwG+tTghLhTA="
```

3 台路由器的 Wireguard 连接建立完成，注意所有 Peer 都开启了 persistent-keepalive 为 30 秒，目的是能保持状态连接。

OSPF 配置

R1 路由器

R1 与 R2 建立 OSPF 关系，用于发布 R1 的路由让 R2 能接收，在 R1 路由器下有一个网段是 172.12.0.0/24 经过 10.10.0.2，配置一条静态路由

```
[admin@R1]>/ip route
[admin@R1] /ip/route >add disabled=no dst-address=172.12.0.0/24 gateway=10.10.0.2
```

配置 OSPF，在 instance 添加 ospf-instance-1 并重分布静态路由 static

```
[admin@R1] /ip/route >/routing/ospf instance
[admin@R1] /routing/ospf/instance> add disabled=no name=ospf-instance-1 redistribute=static
```

创建骨干区域 area 0

```
[admin@R1] /routing/ospf/instance>/routing ospf area
[admin@R1] /routing/ospf/area > add disabled=no instance=ospf-instance-1 name=ospf-area-0
```

创建 OSPF 模板，设置 networks 为 Wireguard 的互联 IP 子网段，设置 type 为 ptp

```
[admin@R1] /routing/ospf/area >/routing ospf interface-template
[admin@R1] /routing/ospf/interface-template > add area=ospf-area-0 disabled=no
networks=172.16.0.0/30 type=ptp
```


R2 路由器

配置与 R1 路由器类似，只是不用重分布路由

```
[admin@R2] >/routing/ospf/instance
[admin@R2] /routing/ospf/instance> add disabled=no name=ospf-instance-1
[admin@R2] /routing/ospf/instance>/routing ospf area
[admin@R2] /routing/ospf/area > add disabled=no instance=ospf-instance-1 name=ospf-area-0
[admin@R2] /routing/ospf/area >/routing ospf interface-template
[admin@R2] /routing/ospf/interface-template > add area=ospf-area-0 disabled=no
networks=172.16.0.0/30 type=ptp
```

BGP 配置

由于 allowed-address 限制了 224.0.0.5 组播不能在多个 Peer 允许，因此 OSPF 无法在点对多节点组网的 Wireguard 使用，所以另外节点采用了 BGP 连接

R2 与 R3 配置 BGP 采用相同的私有 AS65530，建立 ibgp

R2 路由器

配置 BGP，设置 as 号为 65530，并连接 R2 路由器 172.18.0.2 远端 as 号是 65530，并设置重分布静态路由

```
[admin@R2] >/routing bgp connection
[admin@R2] /routing/bgp/connection> add as=65530 disabled=no local.role=ibgp name=bgp1
remote.address=172.18.0.2/32 \
.as=65530 routing-table=main templates=default
```

R3 路由器

R3 路由器通过 ether2 接口连接了另外一个网络，并配置了一条静态路由到 192.168.190.0/24，经过网关 10.11.0.2

```
[admin@R3] >/ip route
[admin@R3] /ip/route>add disabled=no dst-address=192.168.190.0/24 gateway=10.11.0.2
```

配置 BGP，设置 as 号为 65530，并连接 R2 路由器 172.18.0.1 远端 as 号是 65530，并设置重分布静态路由

```
[admin@R3] /ip/route>/routing bgp connection
[admin@R3] /routing/bgp/connection> add as=65530 disabled=no local.role=ibgp name=bgp1
output.redistribute=static remote.address=\
172.18.0.1/32 .as=65530 routing-table=main
```

在 R3 路由器查看 BGP 会话情况，前缀 E 代表关系已经建立：

```
[admin@R3] /routing/bgp/session> print
```

Flags: E - established

0 E name="bgp1-1"

remote.address=172.18.0.1 .as=65530 .id=172.18.0.1 .capabilities=mp,rr,gr,as4 .messages=665 .bytes=12635 .eor=""

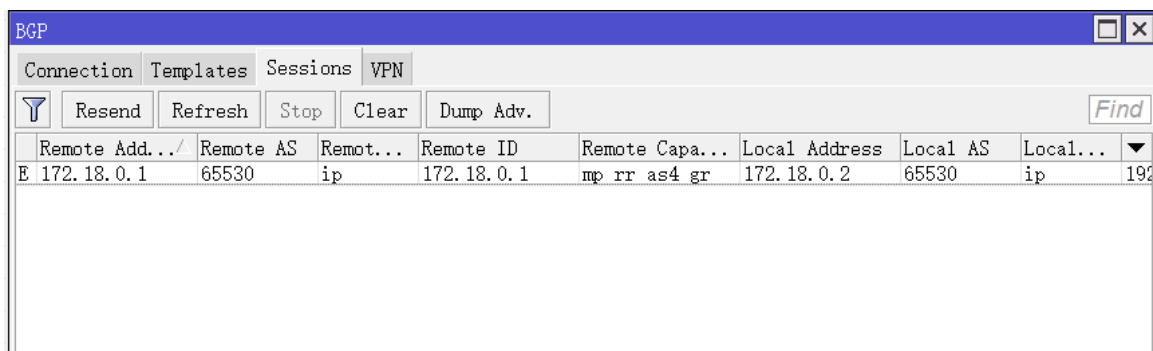
local.role=ibgp .address=172.18.0.2 .as=65530 .id=192.168.88.1 .capabilities=mp,rr,gr,as4 .messages=666 .bytes=12684 .eor=""

output.procid=20

input.procid=20 ibgp

multihop=yes hold-time=3m keepalive-time=1m uptime=11h4m39s520ms last-started=mar/04/2023 08:03:57 last-stopped=mar/04/2023 08:03:57

winbox 查看



Remote Address	Remote AS	Remote ID	Remote Capabilities	Local Address	Local AS	Local ID	Status
E 172.18.0.1	65530	ip 172.18.0.1	mp rr as4 gr	172.18.0.2	65530	ip 192.168.88.1	Established

查看 R2 的路由表

```
[admin@R2] /ip/route> print
```

Flags: D - DYNAMIC; A - ACTIVE; c, b, o, d, y - COPY

Columns: DST-ADDRESS, GATEWAY, DISTANCE

DST-ADDRESS	GATEWAY	DISTANCE
DAd 0.0.0.0/0	192.168.88.1	1
DAo 172.12.0.0/24	172.16.0.1%wireguard1	110
DAd 172.16.0.0/30	wireguard1	0
DAd 172.18.0.0/30	wireguard1	0
DAd 192.168.88.0/24	ether1	0
DAb 192.168.190.0/24	172.18.0.1	200

从路由表可以看到从 R1 获取的 OSPF 路由和 R3 获取的 BGP 路由

第三章 Zerotier

RouterOS 从 v7.1rc2 开始支持 ZeroTier 虚拟网络隧道, ZeroTier 作为一款商业化平台, 对于 MikroTik 产品发展更加多元化。当前 RouterOS 仅 ARM 和 ARM64 构架的设备支持 ZeroTier 功能。

ZeroTier 是一个自带网络虚拟化引擎的应用, 它实现了以太网虚拟化传输, 类似于 VXLAN 构建在加密安全的全局点对点网络。它提供网络虚拟化和管理能力, 与企业 SDN 交换机类似, 能穿透本地网络连接到外网, 并支持几乎任何类型的应用程序或设备的网络连接。

例如, 在家庭局域网的 NAS 访问, 可以通过 ZeroTier 分享给另一个朋友的家庭网络, 而且能穿透 NAT 网络, 访问内网的设备不需要配置端口映射等

3.1 本地网络与网关配置建议

ZeroTier 对于 RouterOS 的推荐配置和绝大多数的普通网关设备基本一致:

- 不要限制出方向的 UDP 传输 (通常使用 UDP 高位端口)
- 网络中支持 UPnP 或 NAT-PMP 能大大提高 ZeroTier 末端映射外部端口的性能, 并完全避免 NAT 穿透
- 推荐使用 IPv6 连接, 如果两端网络都支持 IPv6, 则可大大提高直连的可靠性, 这样的网络连接无需 NAT (IPv6 完全不需要 NAT, 只会增加网络的安全和复杂性), 对于两端网络的防火墙需允许双向 UDP 会话通过。
- 请不要使用 Symmetric (对称) NAT, 使用 "full cone" (全锥型) 或 "port restricted cone" (端口限制圆锥型) NAT。对于 Symmetric NAT 非常影响点对点传输, 会降低 VoIP、video 聊天、游戏、WebRTC 等, 包括 ZeroTier。
- 在 ZeroTier 端和互联网之间不要存在超过一层的 NAT 转换。由于多层 NAT 的状态和行为之间的混乱交错, 多层 NAT 会引起连接不稳定。
- nat 的端口映射或连接会话超时时间不应小于 60 秒。
- 对于 NAT 设备, 单个公网 IP 地址转换的内网设备数量不超过 16000 个, 以保证每个公网 IP 映射的端口数量足够。
- 交换机和无线 AP 接入点应该允许本地设备之间的局域网内部通信。关闭任何“二层隔离”功能。一些交换机为网络稳定和安全会撤脂更细粒的端口控制, 通常可能需让这些交换机上配置允许本地 UDP/9993 流量的出入

Zerotier 一款可以随时随地轻松连接云平台，提供移动、桌面和数据中心的网络穿透。不管你的网络设备是公网 IP，还是私网 IP 地址，都能接入平台，连接到 **Zerotier** 平台后就像在同一个交换机接入的二层网络设备一样。简单来说，它就是一个局域网组建工具。

Zerotier 会监听 3 个 udp 端口

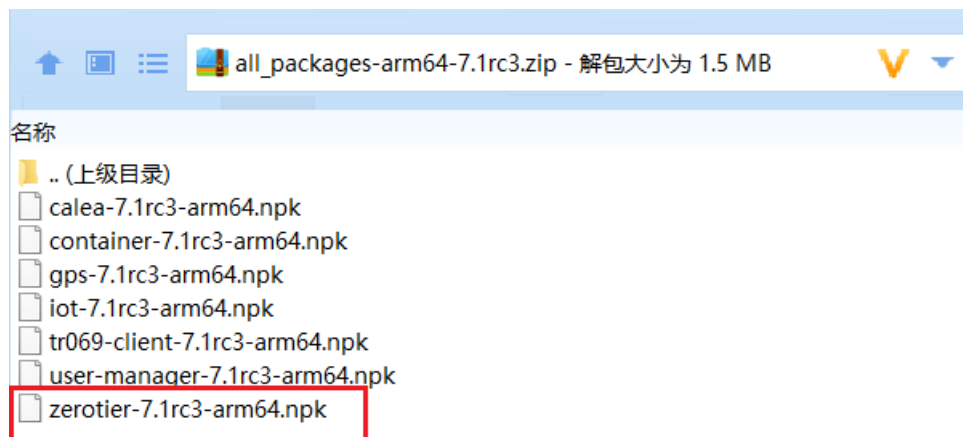
- 默认为 UDP/9993
- 一个随机高位端口，运行在 **ZeroTier** 服务的 IP 地址上
- 一个随机高位端口，用于 UPNP/NAT-PMP 映射

使用 **ZeroTier** 步骤：

1. 注册 **ZeroTier**
2. 创建私有局域网，并获得 **Network ID**
3. 部署或安装其他平台客户端，并配置 **Network ID**，连接到 **ZeroTier** 官网平台
4. 在 **ZeroTier** 官网控制台，允许已经注册客户端加入私有局域网

因需要 **ZeroTier** 官方服务器的支持，所以对于网络连接是有限制的，当前所知免费网络是 25 台设备，超过了就要付费。

ZeroTier 功能包当前需要单独安装，没集成到 **system** 功能包中，因此需要单独安装，在官网下载对应型号的 **Extra packages** 扩展包，下载解压可以看到：



选择 **zerotier** 上传到 RouterOS 的 **File** 目录下，然后使用 **reboot** 命令重启安装，安装成功后在 **/system packages** 查看是否安装成功，操作路径为 **/zerotier**。

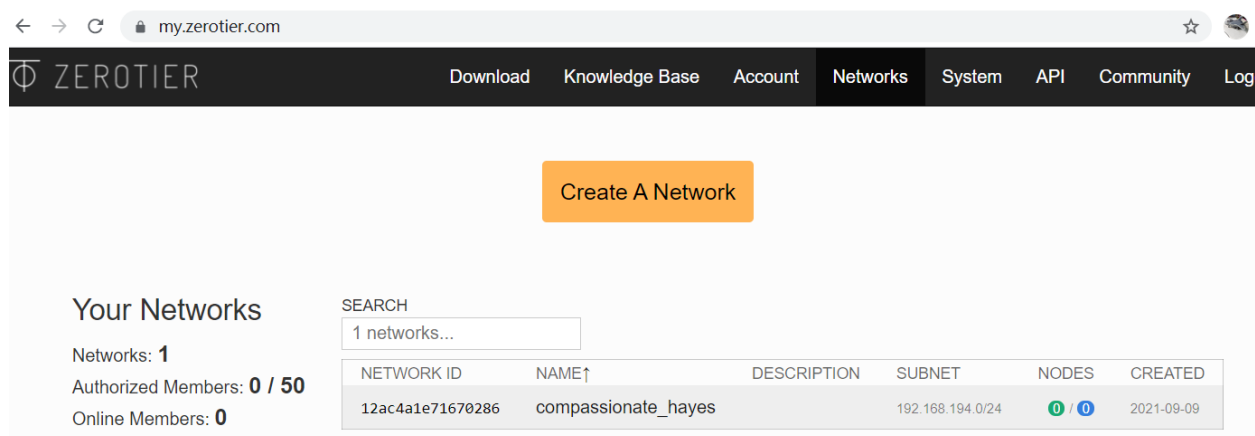
Package List					
<div> <input type="button" value="Check For Updates"/> <input type="button" value="Enable"/> <input type="button" value="Disable"/> <input type="button" value="Uninstall"/> <input type="button" value="Unschedule"/> <input type="button" value="Downgrade"/> <input type="button" value="Check Installation"/> </div>					
Name	Version	Build Time	Scheduled		
container	7.1rc3	Sep/08/2021 10:29:39			
routeros	7.1rc3	Sep/08/2021 10:29:39			
zerotier	7.1rc3	Sep/08/2021 10:29:39			

3.2 RouterOS 配置实例

下面的 Zerotier 通过命令行配置（RouterOS v7.2rc2 开始支持 winbox 配置），需要进入 zerotier 目录，通过 print 命令查看当前配置，可以看到默认就有一条连接 ZeroTier 中心控制台的规则，处于禁用状态。

```
[admin@MikroTik] > zerotier/
[admin@MikroTik] /zerotier> print
Flags: X, R - ONLINE
Columns: NAME, PORT
#    NAME  PORT
;;; ZeroTier Central controller - https://my.zerotier.com/
0 X   zt1   9993
```

在官网平台注册后，会自动生成 network id，也可以自己新建 network id



进入 interface 创建需要连接的 network id，并指定 instance 实例为 zt1

```
[admin@MikroTik] /zerotier> interface/
[admin@MikroTik] /zerotier/interface> add network=12ac4a1e71670286 instance=zt1
```

返回到 zerotier 目录，启用 zt1 实例

```
[admin@MikroTik] /zerotier/interface> ..
[admin@MikroTik] /zerotier> enable zt1
```

查看 zt1 实例状态，如果连接后会显示 R

```
[admin@MikroTik] /zerotier> print
Flags: R - ONLINE
Columns: NAME, PORT
#    NAME  PORT
```

```

::: ZeroTier Central controller - https://my.zerotier.com/
0 R   zt1   9993
[admin@MikroTik] /zerotier>

```

查看官网控制台，online members 处已经显示有 1 台设备

在 access control 默认定义为 private 的私有网络，要求验证才能加入 zerotier 的网络，在 advance 可以配置静态路由

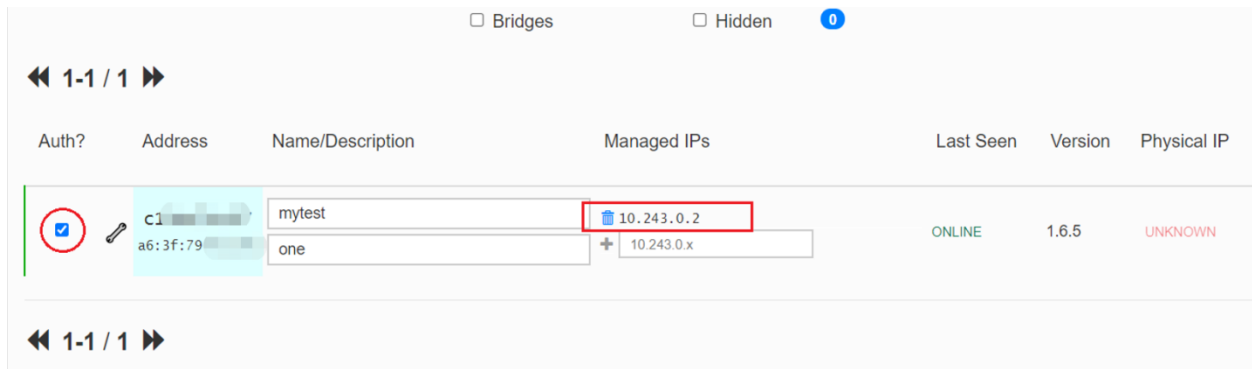
由于是 private 模式，查看 interface 的状态时，可以看到是 access_denied

```

[admin@MikroTik] /zerotier/interface> print
Columns: NAME, MAC-ADDRESS, NETWORK, STATUS
# NAME      MAC-ADDRESS      NETWORK      STATUS
0 zerotier2 A6:3F:79:11:28:28 f78b647fea7 ACCESS_DENIED

```

进入后台配置连接主机参数，勾选 **Auth**，并在 **managed IP** 分配地址



再次查看状态变为 OK

```
[admin@MikroTik] /zerotier/interface> print
Flags: R - RUNNING
Columns: NAME, MAC-ADDRESS, NETWORK, NETWORK-NAME, STATUS
#  NAME      MAC-ADDRESS      NETWORK      NETWORK-NAME  STATUS
0 R zerotier2  A6:3F:79:11:28:28  f78b647fea7  tiny_davies   OK
```

进入/ip address 查看获取的 IP 地址

```
[admin@MikroTik] /zerotier/interface> /ip add print
Flags: D - DYNAMIC
Columns: ADDRESS, NETWORK, INTERFACE
#  ADDRESS      NETWORK      INTERFACE
;;; defconf
0  192.168.88.1/24  192.168.88.0  ether15
1 D 192.168.99.34/24  192.168.99.0  ether16
2 D 10.243.104.144/16  10.243.0.0    zerotier2
3 D 10.243.0.2/16     10.243.0.0    zerotier2
```

一台 RouterOS 连接完成，剩下的就是其他 RouterOS 设备，或者 win、Android 等平台的客户端，连接到 ZeroTier 控制台，并授权允许加入到私有网络，然后通过 ZeroTier 服务器组建一个二层网络。

第四章 Container (Docker)

Container 是 MikroTik 添加 Docker(TM)的功能，允许用户在 RouterOS 中运行容器环境。此功能是在 RouterOS v7.1rc3 中新增，在 RouterOS v7rc5 暂时取消，Container 又在 v7.4beta4 重新恢复。由于 RouterOS 的 Container 在 v7 前期几个版本调整较多，可能存在内容配置差异，具体请参考 help.mikrotik.com

操作路径: /container

相关资料: <https://www.docker.com>

注意: RouterOS 系统版本 v7.4beta4 以上，要求安装独立的 Container package 功能包 (container 安装可以参考 [RB5009 开机测试视频](#))。

当前 v7.4beta4 启用 Container 条件: :

1. RouterOS 硬件要求 ARM、ARM64 和 x86，版本在 v7.4beta4 以上，并安装 Container 功能包
2. 开启 container 模式，实现对设备的物理访问
3. 扩展的硬盘或 USB 存储需格式化为 ext3/ext4

Containers 占用大量磁盘空间，强烈推荐使用 USB/SATA、SD 卡和 NVMe 外部存储。对于部分 RB 设备仅支持 USB，推荐通过 USB 扩展 SSD，这样具有更快的文件操作性能。部分 CCR2xxx 设备支持 NVMe 能在文件操作性能上提供更大的优势。

4.1 启用 Container 模式

与之前的 container 部署不同，7.4beta4 后要求设置 container 模式，由于新增了 Device-mode，默认情况下是限制 container 使用。Device-mode 是一种对设备功能限制属性，即限制对特定功能配置的访问。有两种模式: enterprise 和 home (企业模式和家庭模式)。

所有设备默认都使用 enterprise，该模式允许除 container 外的所有功能。home 模式禁用以下功能: scheduler, socks, fetch, bandwidth-test, traffic-gen, sniffer, romon, proxy, hotspot, email, zerotier, container。

首先确认你的设备工作在 enterprise，如果在 home 需要修改为 enterprise

```
[admin@MikroTik] > system/device-mode/print
mode: enterprise
```

当在 enterprise 模式下，更新 container 模式，启用 container 模式命令如下:

```
[admin@MikroTik] > /system/device-mode/update container=yes
update: please activate by turning power off or pressing reset or mode button
      in 5m00s
-- [Q quit|D dump|C-z pause]
```


在 5 分钟内，拔插电源重启设备（远程重启无效）或者按设备的 **reset** 按钮确认执行 **container** 模式，设备会重启生效进入 **container** 模式。如果在规定的时间内没有关机或按下 **reset** 按钮，则取消模式切换。如果同时运行另一个更新命令，则两个更新命令都将被取消。

重启之后，查看 **device-mode**

```
[admin@MikroTik] /system> device-mode/print
mode: enterprise
container: yes
[admin@MikroTik] /system>
```

4.2 创建网络

从这里开始 4.2-4.5 章节，将根据 MikroTik 官方的安装 Pi-hole 的流程进行介绍，

首先需要给 **container** 创建 **veth** 网络接口，用于 **docker** 的访问，可以给 **docker** 分配一个独立的玩了个，如创建一个 **Bridge**，将各个 **docker** 的设备网卡加入到这个 **bridge** 分组中，当然也可以把分配给 **docker** 的 **veth** 加入到你当前的 **Bridge** 网络，例如 hAP ac2 的 **ether2-ether5** 是一个 **bridge** 分组，直接将 **veth** 接口加入到默认的 **bridge**，IP 地址是 192.168.88.1/24，**veth** 的 IP 地址也分配中这个段中。

我这里分为路由模式和桥接模式，路由模式是 **docker** 网络独立于实际物理网络的一个 **bridge**，而桥接则是把 **docker** 加入到当前物理网络的 **bridge** 网段内。

路由模式

首先为 **docker** 网络创建一个 **bridge** 用于桥接网络，取名 **docker**

```
/interface/bridge/add name=docker
```

为 **dockers** 接口分配一个新的网段 172.17.0.0/16

```
/ip/address/add address=172.17.0.1/16 interface=docker
```

创建 **veth** 网卡，并分配一个新的 IP 段
















```
/interface/veth/add name=veth1 address=172.17.0.2/16 gateway=172.17.0.1
```

并将 **veth1** 加入到 **bridge** 中

```
/interface/bridge/port add bridge=dockers interface=veth1
```

在 **bridge port** 下 winbox 显示的配置如下，**veth1** 属于 **docker**：

Bridge

Bridge	Ports	Port Extensions	VLANs	MSTIs	Port MST Overrides	Filters	NAT	Hosts	MDB
									
#		Interface	Bridge	Horizon	Trusted	Priorit...	Path Cost	PVID	
::: defconf									
0	IH	 ether2	bridge		no	80	10	1	
::: defconf									
1	IH	 ether3	bridge		no	80	10	1	
::: defconf									
2	IH	 ether4	bridge		no	80	10	1	
::: defconf									
3	H	 ether5	bridge		no	80	10	1	
::: defconf									
4	IH	 ether6	bridge		no	80	10	1	
::: defconf									
5	IH	 ether7	bridge		no	80	10	1	
::: defconf									
6	H	 sfp-sfpplus1	bridge		no	80	10	1	
7		 veth1	docker		no	80	10	1	
8	IH	 ether1	bridge		no	80	10	1	

9 items (1 selected)

设置 src-nat 允许 172.17.0.0/16 的 IP 地址访问互联网

```
/ip/firewall/nat/add chain=srcnat action=masquerade src-address=172.17.0.0/16
```

桥接模式

创建网络环境还可以直接将 veth 网络接口与你当前 LAN 网络桥接，例如，RB5009 设备，默认配置下 ethre2-ether8 有一个 bridge 接口的 LAN 网络中，IP 地址是 192.168.88.1/24，

特别提示：如果是基于 CHR 的虚拟化平台，bridge 到物理网络需要考虑虚拟化平台的网卡配置，如混杂模式等，请自行查阅资料










可以直接给 veth1 接口配置一个 192.168.88.254（bridge 的 DHCP 需要定义地址池范围排除 192.168.88.254）

```
/interface/veth/add name=veth1 address=192.168.88.254/24 gateway=192.168.88.1
```

将 veth1 添加到 bridge 中

```
/interface/bridge/port add bridge=bridge interface=veth1
```

在 bridge port 下 winbox 显示的配置如下，veth1 属于 bridge:

Bridge										
Bridge		Ports	Port Extensions	VLANs	MSTIs	Port MST Overrides	Filters	NAT	Hosts	MDB
<div><div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div><div><div><div></div><div></div></div><div><div></div><div></div></div></div></div></div>										
#		Interface	Bridge	Horizon	Trusted	Priorit...	Path Cost	PVID	F	
::: defconf										
0	IH	 ether2	bridge		no	80	10	1	d	
::: defconf										
1	IH	 ether3	bridge		no	80	10	1	d	
::: defconf										
2	IH	 ether4	bridge		no	80	10	1	d	
::: defconf										
3	H	 ether5	bridge		no	80	10	1	d	
::: defconf										
4	IH	 ether6	bridge		no	80	10	1	d	
::: defconf										
5	IH	 ether7	bridge		no	80	10	1	d	
::: defconf										
6	H	 sfp-sfpplus1	bridge		no	80	10	1	d	
7		 veth1	bridge		no	80	10	1	d	
8	IH	 ether1	bridge		no	80	10	1	d	
9 items (1 selected)										

4.3 添加环境变量和 mount

为 container 添加环境变量（可选）

```
[admin@MikroTik] >/container/envs/add list=pihole_envs name=TZ value="Europe/Riga"
[admin@MikroTik] >/container/envs/add list=pihole_envs name=WEBPASSWORD value="mypassword"
[admin@MikroTik] >/container/envs/add list=pihole_envs name=DNSMASQ_USER value="root"
```

定义 mount 参数（可选）:

```
[admin@MikroTik] >/container/mounts/add name=etc_pihole src=disk1/etc dst=/etc/pihole
[admin@MikroTik] >/container/mounts/add name=dnsmaq_pihole src=disk1/etc-dnsmaq.d
dst=/etc/dnsmaq.d
```

src=指向配置文件存储在 RouterOS 的路径(也可以是 **src=disk1/etc_pihole**, 该设置为将配置文件放在外部 USB 存储上), **dst**=指向 container 配置的路径(关于 **dst** 指向配置, 根据 container 镜像不同有区别, 具体请参考 container 手册)。如果 **src** 路径没配指定, 那么将默认使用 **dst** 下的路径配置 container。

4.4 添加 container 镜像

获取 docker 有多种方式, 官方给出了三种, 如果希望在创建 container 时记录日志, 可以设置 **logging=yes**, 镜像的 **root-dir** 根路径存储设备格式为 **ext3** 或 **ext4**, 不建议将 container 安装到设备的内部存储。

1、外部库获取镜像

设置 registry-url(用于从 Docker 登录下载镜像)

```
[admin@MikroTik] >/container/config/set registry-url=https://registry-1.docker.io
```

国内的镜像源可以参考 <https://registry.docker-cn.com>, <https://docker.mirrors.ustc.edu.cn> 或者 <https://6kx4zyno.mirror.aliyuncs.com>

pull 镜像配置参考:

```
[admin@MikroTik] >/container/add remote-image=pihole/pihole:latest interface=veth1
root-dir=disk1/pihole mounts=dnsmaq_pihole,etc_pihole envlist=pihole_envs
```

镜像将自动下载, 并存放到 root 目录, 状态通过 print 命令查看

```
[admin@MikroTik] /container> print
0 name="daa13504-5b4a-43df-bd17-581ecd298483" tag="pihole/pihole:latest" os="" arch=""
  interface=veth1 envlist="pihole_envs" root-dir=disk1/pihole mounts=dnsmaq_pihole,etc_pihole
  dns="" status=extracting
```

2、从 PC 导入镜像

从 docker 官方获取 PiHole 的 container 文件, 请确保下载正确的版本, 匹配 RouterOS 硬件架构, 区分 ARM、ARM64 和 x86。

例如获取 arm64 构架的 docker 镜像:

```
docker pull --platform=arm64 pihole/pihole:last
```

然后保存为 tar

```
docker save pihole/pihole > pihole.tar
```

下面是 MikroTik 官方从 docker hub 使用 sha256 sum 以获取映像文件(以下链接的操作日期为 2022 年 6 月 16 日, 仅供参考)

arm64:

```
docker pull
pihole/pihole:latest@sha256:4cef8a7b32d318ba218c080a3673b56f396d2e2c74d375bef537ff5e41fc4638

docker save pihole/pihole > pihole.tar
```

arm:

```
docker pull
pihole/pihole:latest@sha256:684c59c7c057b2829d19d08179265c79a9ddabf03145c1e2fad2fae3d9c36a94

docker save pihole/pihole > pihole.tar
```

amd64:

```
docker pull
pihole/pihole:latest@sha256:f56885979dcffeb902d2ca51828c92118199222ffb8f6644505e7881e11eeb85

docker save pihole/pihole > pihole.tar
```

下载并解压文件后，将其上传到 RouterOS。从 tar 映像创建一个 container

```
/container/add file=pihole.tar interface=veth1 envlist=pihole_envs root-dir=disk1/pihole
mounts=dnsmaq_pihole,etc_pihole hostname=PiHole
```

3、在 PC 创建镜像

基于 Linux 系统

要使用 Dockerfile 并制作自己的 docker 包，需要安装 docker 以及 buildx 或其他 builder 工具包。简单的方法是下载和安装 Docker 引擎：

<https://docs.docker.com/engine/install/>

安装后检查是否有额外的架构可用：

```
docker buildx ls
```

会返回如下信息：

```
NAME/NODE DRIVER/ENDPOINT STATUS PLATFORMS
default * docker
      default default  running linux/amd64, linux/arm64, linux/riscv64, linux/ppc64le, linux/s390x, linux/386,
linux/arm/v7, linux/arm/v6
```

如果不是-安装额外的架构：

```
docker run --privileged --rm tonistiigi/binfmt --install all
```

pull 或 create 你的项目包含 Dockerfile 和 build，提取镜像

```
git clone https://github.com/pi-hole/docker-pi-hole.git
```

```
cd docker-pi-hole
docker buildx build --no-cache --platform arm64 -t pihole .
docker save pihole > pihole.tar
```

上传 pihole .tar 到 RouterOS 设备。

可以对 Linux 系统上的镜像和目标对象进行裁剪，可以[参考 docker 官方文档](#)

RouterOS 从镜像创建 container

```
/container/add file=pihole.tar interface=veth1 envlist=pihole_envs mounts=dnsmaq_pihole,etc_pihole
hostname=PiHole
```

4.5 container 启动与设置

使用 /container/print 命令确保 container 已经添加，并且 status=stopped 状态，

```
[admin@MikroTik] /container> print
0 name="daa13504-5b4a-43df-bd17-581ecd298483" tag="pihole/pihole:latest" os="linux" arch="arm64"
  interface=veth1 envlist="pihole_envs" root-dir=disk1/pihole mounts=dnsmaq_pihole,etc_pihole
  dns="" status=stopped
```

然后执行如下命令启用：

```
[admin@MikroTik] /container> start 0
```

查看 status 是否变为 running

```
[admin@MikroTik] /container> print
0 name="daa13504-5b4a-43df-bd17-581ecd298483" tag="pihole/pihole:latest" os="linux" arch="arm64"
  interface=veth1 envlist="pihole_envs" root-dir=disk1/pihole mounts=dnsmaq_pihole,etc_pihole
  dns="" status=running
```

启用完成后，应该可以通过浏览器访问 PiHole 的 web 面板，路径是 <http://172.17.0.2>

Docker 的端口映射

端口可以使用 dst-nat 映射到 Docker 到你的外网接口，如下面的操作，将 Pi-hole 的 80 端口映射到出口 pppoe-out1 的 8088 端口

```
/ip firewall nat
add action=dst-nat chain=dstnat in-interface=pppoe-out1 dst-port=8088 protocol=tcp
to-addresses=172.17.0.2 to-ports=80
```

当配置完成 Pi-hole 后，可以设置 RouterOS 的 DNS，如：

```
/ip dns set servers=172.17.0.2
```

Shell

RouterOS 支持进入 docker 配置，使用 shell 命令可以进入内部终端控制台，shell 命令使用只能在容器 running 后，如下面操作

```
[admin@MikroTik] /container> print
0 name="c93ef326-56cf-4c43-baca-707173451564" tag="library/nginx:latest"
  os="linux" arch="arm64" interface=veth1 root-dir=disk1/nginx2
  mounts=nginxlog dns="" logging=yes status=running
[admin@MikroTik] /container> shell 0
root@MikroTik:/# ls
access.log  dev                error.log  lib    opt    run    sys  var
bin         docker-entrypoint.d  etc        media  proc  sbin  tmp
boot        docker-entrypoint.sh home        mnt    root  srv   usr
root@MikroTik:/#
```

如上有一个 nginx 的容器，查看编号为 0，使用 shell 0 进入，执行 apt-get update 命令

```
root@MikroTik:/# apt-get update
Get:1 http://deb.debian.org/debian bullseye InRelease [116 kB]
Get:2 http://deb.debian.org/debian-security bullseye-security InRelease [48.4 kB]
Get:3 http://deb.debian.org/debian bullseye-updates InRelease [44.1 kB]
Get:4 http://deb.debian.org/debian bullseye/main arm64 Packages [8069 kB]
21% [4 Packages 564 kB/8069 kB 7%]
```

RAM 限制

由于部分 RB 或 CCR 设备的 RAM 有限，可以才有软件限制 RAM 大小保证路由器的正常运行，如果 RAM 的使用超过了临界值，cgroup 的进程将被限制，且处于难以回收的状态之下，甚至导致路由器重启情况。

```
/container/config/set ram-high=200M
```

开机启动

从 v7.6beta6 支持在路由器重启后，自动启动 Container，使用 start-on-boot 选项

```
/container/set 0 start-on-boot=yes
```

```
/container/print
```

```
0 name="2e679415-2edd-4300-8fab-a779ec267058" tag="test_arm64:latest" os="linux" arch="arm"
interface=veth2 root-dir=disk1/alpine mounts="" dns="" logging=yes start-on-boot=yes status=running
```

4.6 安装 nginx (v7.4beta4)

创建 veth2 网卡，设置与 RB 路由器内网 IP 地址相同段，配置地址为 192.168.88.100，网关 192.168.88.1，直接使用桥接模式。

```
/interface/veth/add name=veth2 address=192.168.88.100 gateway=192.168.88.1
```

将 veth2 添加到 bridge 中

```
/interface/bridge/port add bridge=bridge interface=veth2
```

配置 mount 参数，设置 nginx 的 log 配置路径，如果使用默认参数，会提示 error.log 错误，需要单独指定 nginx 的 log 路径

```
/container mounts add dst=/var/log/nginx/ name=nginxlog src="/disk1/nginx"
```

通过官方镜像站点获取 nginx 最新的镜像，并设置 logging=yes 有助于排查错误

```
/container/add remote-image=nginx:latest interface=veth2 root-dir=disk1/nginx mounts=nginxlog
logging=yes
```

等待 nginx 从官方镜像站点下载，等 status=stopped 后，可以启动 nginx

```
[admin@MikroTik] /container> print
```

```
0 name="daa13504-5b4a-43df-bd17-581ecd298483" tag="pihole/pihole:latest" os="linux"
arch="arm64" interface=veth1 envlist="pihole_envs" root-dir=disk1/pihole
mounts=dnsmasq_pihole,etc_pihole dns="" status=running
```

```
1 name="a7b0174c-934b-46fe-a352-90be295c0f60" tag="library/nginx:latest" os="linux"
arch="arm64" interface=veth2 root-dir=disk1/nginx mounts=nginxlog dns="" logging=yes
status=stopped
```

当前 nginx 的镜像是序列是 1，使用 start 1 命令启动

```
[admin@MikroTik] /container> start 1
```

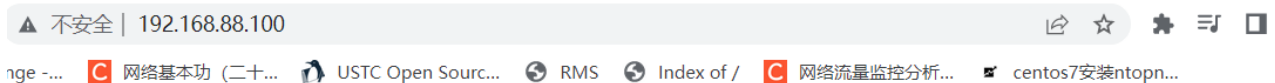
确认 nginx 的 status=running


```
[admin@MikroTik] /container> print
```

```
0 name="daa13504-5b4a-43df-bd17-581ecd298483" tag="pihole/pihole:latest" os="linux"
  arch="arm64" interface=veth1 envlist="pihole_envs" root-dir=disk1/pihole
  mounts=dnsmasq_pihole,etc_pihole dns="" status=running

1 name="a7b0174c-934b-46fe-a352-90be295c0f60" tag="library/nginx:latest" os="linux"
  arch="arm64" interface=veth2 root-dir=disk1/nginx mounts=nginxlog dns="" logging=yes
  status=running
```

可以打开浏览器输入 192.168.88.100，能正常访问到 nginx 的 index 页面



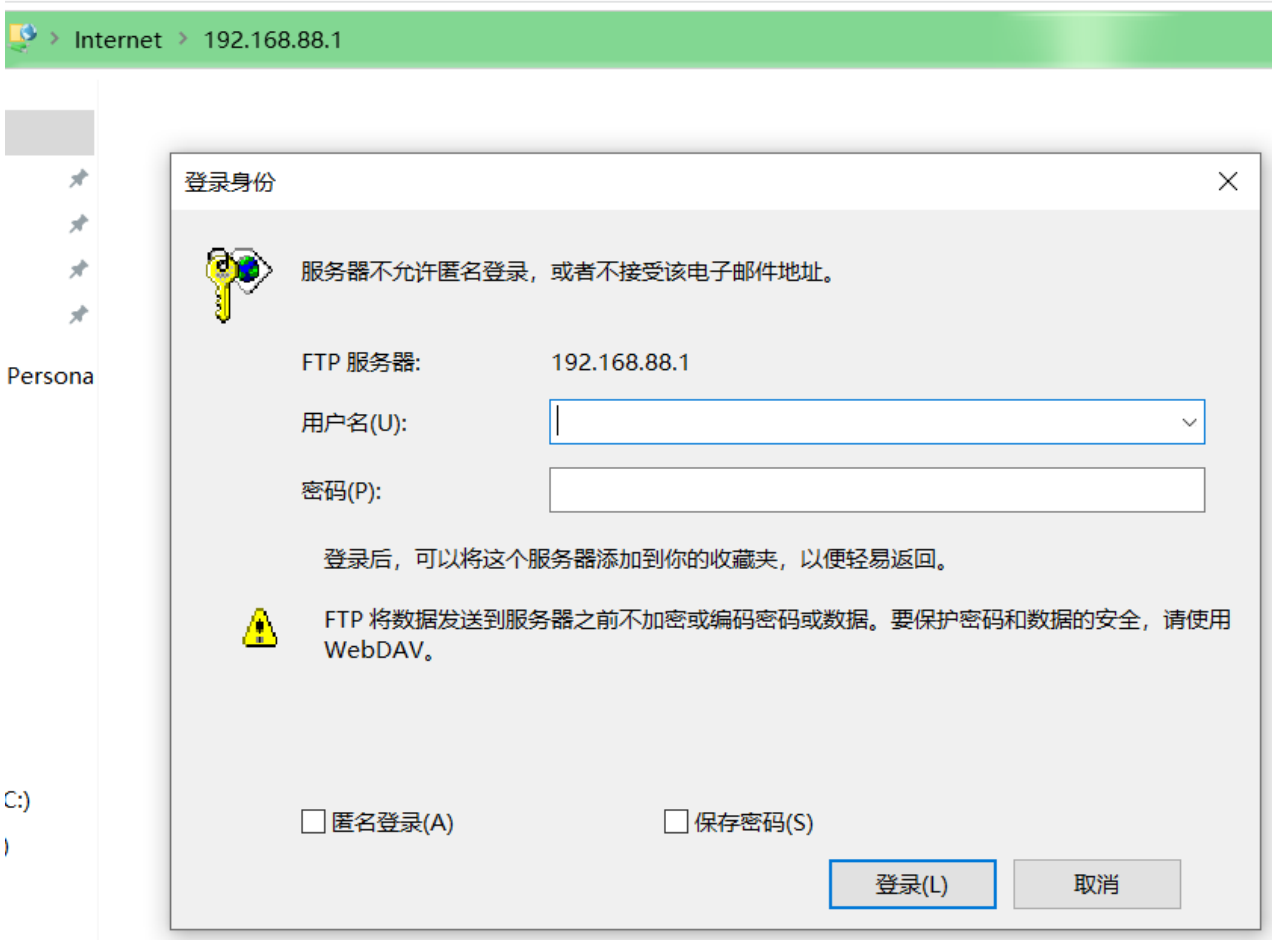
Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

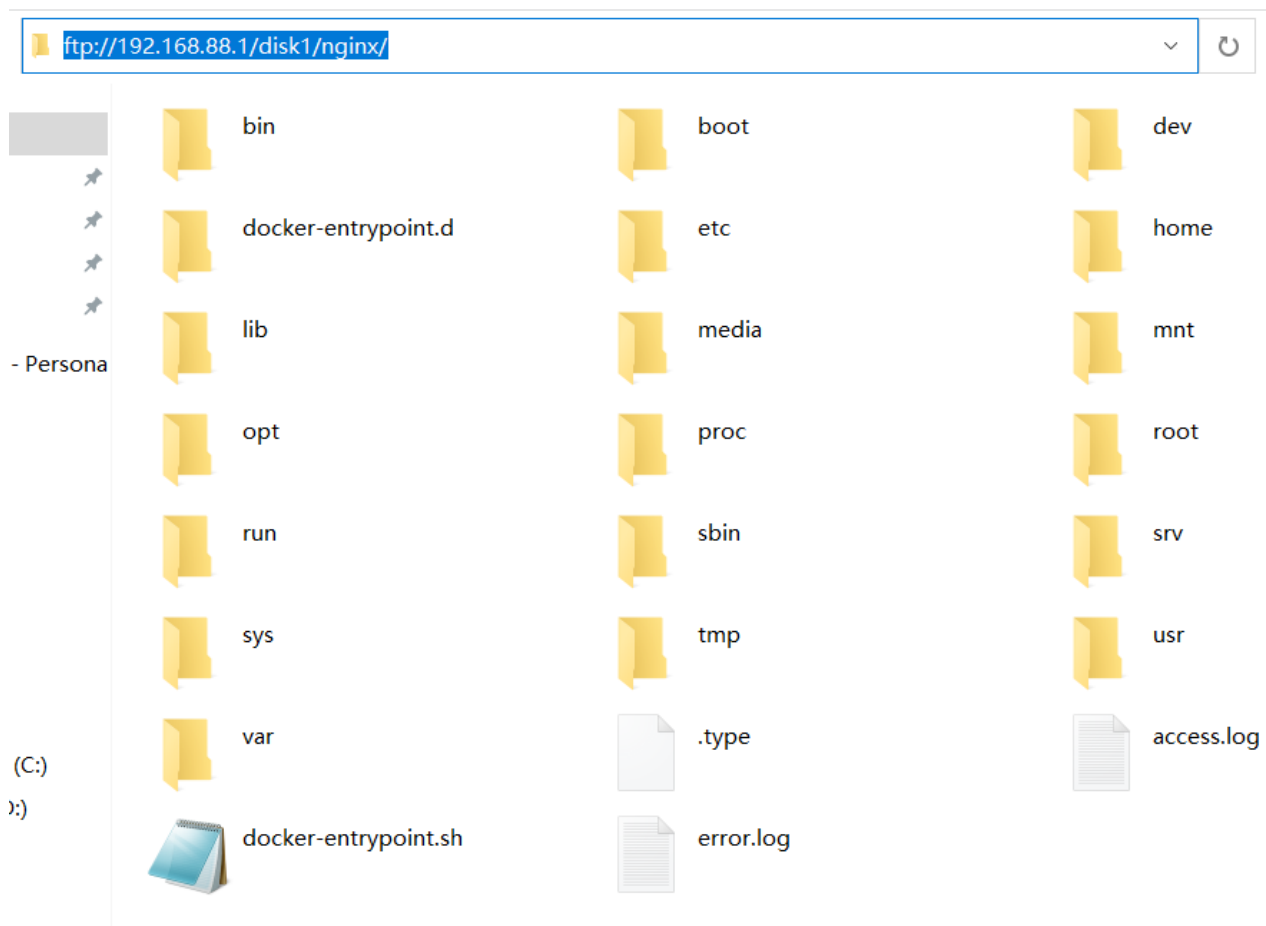
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

可以通过 ftp 登录，可以查看到 docker 配置文件



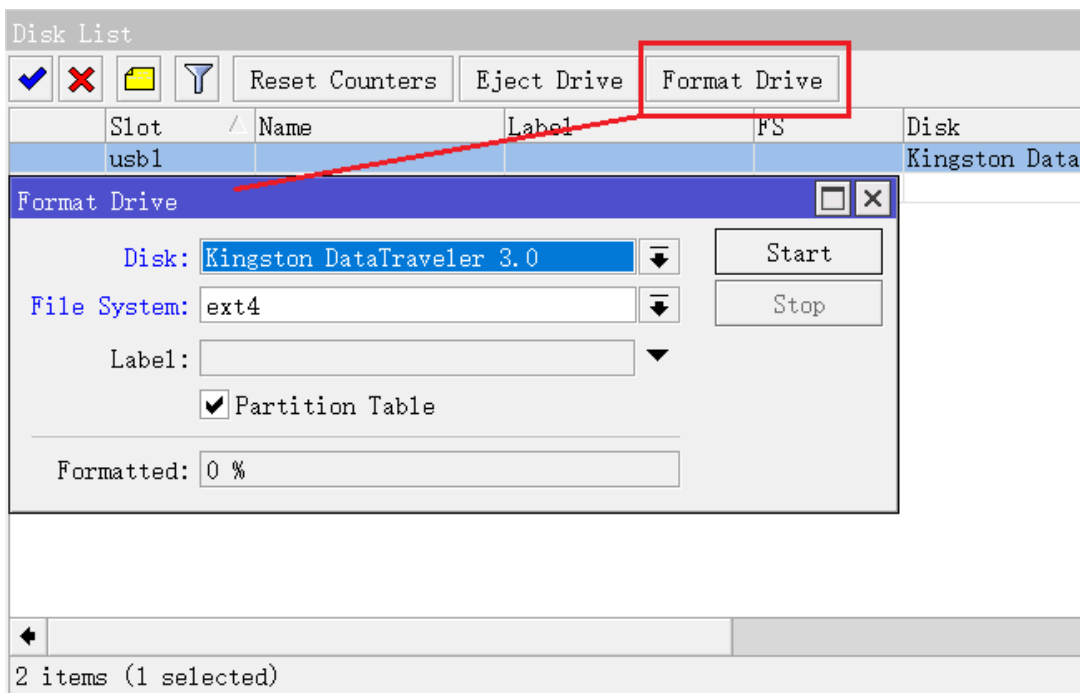
进入/disk1/nginx 目录，能看到你想要的配置文件



4.7 hAP ac3 安装 DNSMasq

hAP ac3 是 IPQ4018 采用 ARM 平台，支持 USB 扩展，用容器安装 DNSMasq 解决，我的思路是使用 ubuntu 的镜像，然后在通过 apt-get 安装 DNSMasq

需要一个 U 盘，作为 Docker 的镜像存储，hAP ac3 的存储空间只要有 128M，首先插上 U 盘，并格式化，Winbox 在 /system/disk 进行格式化，选择 ext4 格式：

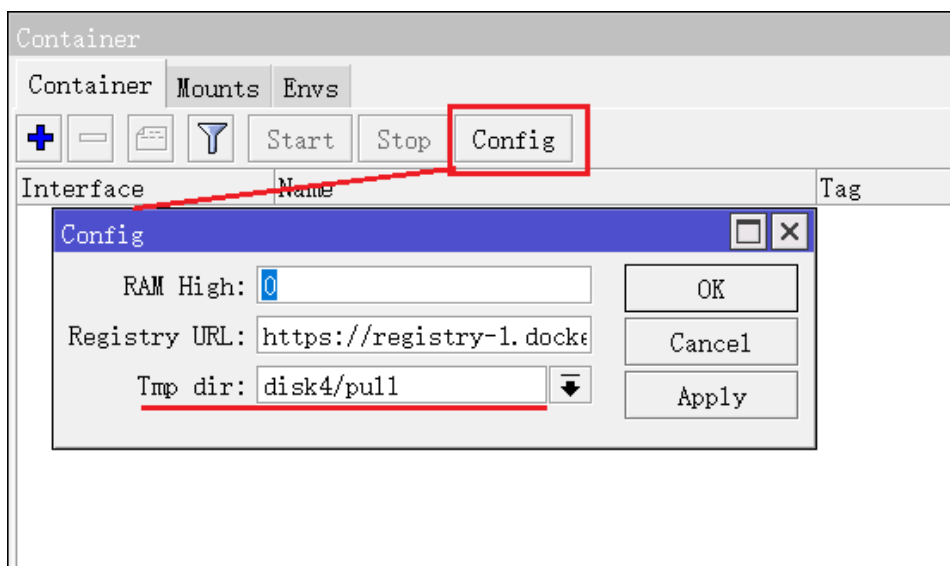


命令行/disk/print 查看情况，格式化完成后，已经将 U 盘挂载到 disk4

```
[admin@ac3] /disk> print
Flags: M, r - RAID-MEMBER; p - PARTITION
Columns: SLOT, MODEL, SERIAL, INTERFACE, NAME, FS, FREE, SIZE
#  SLOT  MODEL    SERIAL  INTERFACE  NAME  FS  FREE  SIZE
0  usb1  Kingston  B06EBF60xx  USB2.10 480Mbps      30943995904
1  Mp usb1-part1          disk4 ext4 30 299017216 30943995392

[admin@ac3] /disk>
```

初始化 container 配置，需修改 Tmp dir，镜像源地址我使用默认，可以根据自己需要修改，设置 disk4/pull 为拉取镜像下载的缓存目录



命令行查看配置

```
[admin@ac3] /container/config> print
ram-high: 0
registry-url: https://registry-1.docker.io
tmpdir: disk4/pull

[admin@ac3] /container/config>
```

进入/interface/veth 创建 veth 网口，并设置网关为 172.17.0.1，后面会分配给网桥的 docker 接口

```
[admin@ac3]/interface/veth> add address=172.17.0.2/24 gateway=172.17.0.1 name=veth1
```

在 RouterOS 创建一个 docker 的桥接

```
[admin@ac3]/interface/bridge>add name=docker
```

将 veth1 加入 docker 的 bridge 中

```
[admin@ac3]/interface/bridge/port>add bridge=docker interface=veth1
```

分配 docker 桥接的 IP 地址

```
[admin@ac3]/ip/address>add address=172.17.0.1/24 interface=docker
```

添加配置，remote-image=ubuntu:latest，拉取镜像名称，设置 interface=veth1，设置文件目录为 disk4/ubuntudns，命令如下：

```
add remote-image=ubuntu:latest interface=veth1 root-dir=disk4/ubuntudns logging=yes cmd="tail -f /dev/null"
```

容器启动时常会遇到问题导致启动失败，可以利用 tail -f /dev/null 让容器一直处于 runing 状态。

下面通过命令执行添加

```
[admin@ac3] /container> add remote-image=ubuntu:latest interface=veth1
root-dir=disk4/ubuntudns logging=yes cmd="tail -f /dev/null"
```

添加完成，等待从源拉取文件，status 为 extracting

```
[admin@ac3] /container> print
0 name="aadeaf01-572e-4672-b82d-16b08a0a1dc5" tag="ubuntu:latest" os="" arch="" interface=veth1
cmd="tail -f /dev/null" root-dir=disk4/ubuntudns mounts="" dns="" logging=yes status=extracting
```

当前 ubuntu 的镜像拉取完成后，status 会变为 stop

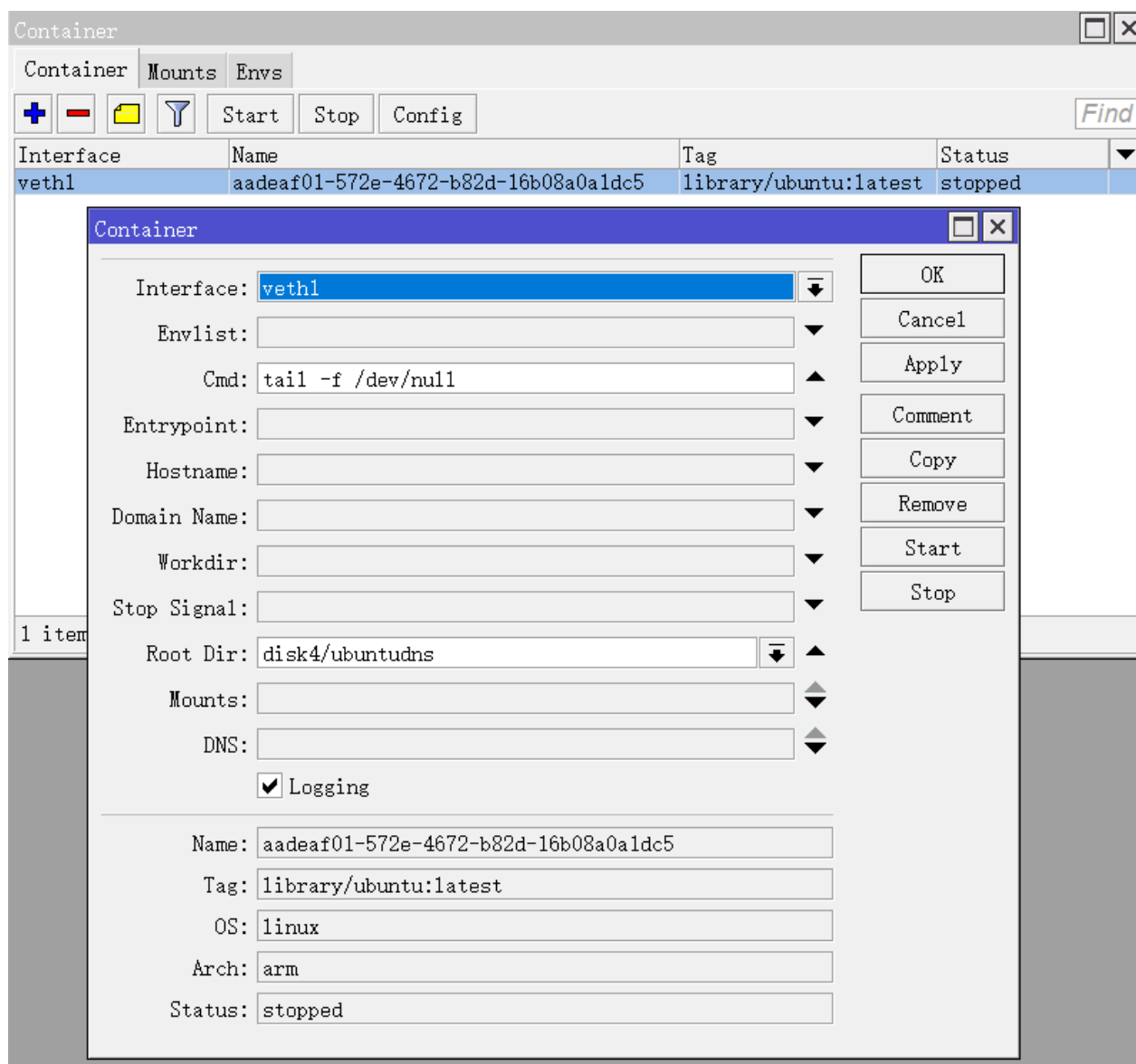
```
[admin@ac3] /container> print
```

```
0 name="aadeaf01-572e-4672-b82d-16b08a0a1dc5" tag="library/ubuntu:latest" os="linux" arch="arm"
interface=veth1
```

```
cmd="tail -f /dev/null" root-dir=disk4/ubuntudns mounts="" dns="" logging=yes status=stopped
```

```
[admin@ac3] /container>
```

Winbox 查看配置



Status 变为 stopped 后，可以启动，点击 start，或者命令行配置：

```
[admin@ac3] /container> start 0
```

然后通过 print 命令查看状态，一直 running 表示运行成功

```
[admin@ac3] /container> print
0 name="aadeaf01-572e-4672-b82d-16b08a0a1dc5" tag="library/ubuntu:latest" os="linux" arch="arm"
interface=veth1 cmd="tail -f /dev/null" root-dir=disk4/ubuntudns mounts="" dns="" logging=yes
status=running
[admin@ac3] /container>
```

使用 shell 命令进入 ubuntu 系统

```
[admin@ac3] /container> shell 0
0;root@ac3: /root@ac3:/#
```

剩下的就和 ubuntu 操作一样，首先 update 下

```
0;root@ac3: /root@ac3:/# apt-get update
Get:1 http://ports.ubuntu.com/ubuntu-ports jammy InRelease [270 kB]
0% [1 InRelease 270 kB/270 kB 100%]
```

安装需要的工具

```
0;root@ac3: /root@ac3:/# apt-get install net-tools
0;root@ac3: /root@ac3:/# apt-get install vim
```

安装 dnsmasq

```
0;root@ac3: /root@ac3:/# apt-get install dnsmasq
```

在/etc/dnsmasq.d/下创建一个新的域名配置文件

```
0;root@ac3: ~root@ac3:~# vi /etc/dnsmasq.d/domain.conf
```

大致写入这些内容转发到 223.5.5.5

```
server=/taobao.com/223.5.5.5
server=/aliyun.com/223.5.5.5
...
```

启动 dnsmasq 服务

```
0;root@ac3: ~root@ac3:~#service dnsmasq start
```

使用 netstat 查看 dnsmasq 服务是否启动

```
0;root@ac3: ~root@ac3:~# netstat -ntupl
```

(Not all processes could be identified, non-owned process info

will not be shown, you would have to be root to see it all.)

Active Internet connections (only servers)

Proto	Recv-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0.0.0.0:53	0.0.0.0:*	LISTEN	797/dnsmasq
tcp6	0	:::53	:::*	LISTEN	797/dnsmasq
udp	0	0.0.0.0:53	0.0.0.0:*		797/dnsmasq
udp6	0	:::53	:::*		797/dnsmasq

需要注意 RouterOS 的/ip/firewall/nat 规则要允许 172.17.0.2 的主机能后被 nat 转换，

在命令行设置开机启动

```
[admin@ac3] /container> set 0 start-on-boot=yes
```

```
[admin@ac3] /container> print
```

```
0 name="aadeaf01-572e-4672-b82d-1608a1dc5" tag="library/ubuntu:latest" os="linux" arch="arm"
interface=veth1 cmd="tail -f /dev/null" root-dir=disk4/ubuntudns mounts="" dns="" logging=yes
start-on-boot=yes status=running
```

最后使用内网主机测试 172.17.0.2 的 DNS 解析，注意此配置，不支持重启后自动启动 dnsmasq 服务，需要进入 shell 启动 dnsmasq 的服务。

特别说明下 hAP ac3 的内存只有 256M，运行的 docker 只能是哪些内存需求小的应用，如果内存耗尽，早期版本 hAP ac3 会自动重启，Container 支持通过 RAM 大小限制避免重启的发生。

第五章 IoT

iOT 作为 RouterOS 的新功能吧，包含了对 GPIO、蓝牙、lora 的支持，还有 modbus 的 RS485 接口，但官方还没有提供配置接口，v7 还在不断完善这些功能。

5.1 GPIO

GPIO（General Purpose I/O Ports）为通用输入/输出端口，即通过引脚输出高低电平或者通过它们读入引脚的状态-是高电平，还是低电平。当前支持 GPIO 的 RB 设备只有 RBM33G 和 KNOT。

访问 GPIO 设置，需要安装 iot 功能包，关于更多的 GPIO 资料可以访问 [link](#)，对于 RouterOS 的 GPIO 是基于 RouterBOARD 读取和接收信号，能用于以下的场景：

1. 通过 ADC 输入测量电压
2. 从另一个设备接收到 0 和 1 信号- "dry contact"
3. 通过发送逻辑 0 或 1 信号到引脚来控制连接的继电器

操作路径：/iot gpio

GPIO 配置分为以下：

- analog (/iot gpio analog)
- digital (/iot gpio digital)

注意：在以下的配置实例中，采用的是 KNOT 产品作为参考，其他产品可能不同的引脚方案，但原理相同。对于 RBM33G 的 /iot gpio 从 v7 版本开始支持。当前 GPIO 设置只能通过 CLI 命令行操作。

下图为 KNOT 的 GPIO 针脚位置说明

analog

```
[admin@device] /iot gpio analog> print
```

#	NAME	VALUE	OFFSET
0	pin2	0mV	0mV
1	pin3	32mV	0mV

```
value = adc_input + offset
```

```
[admin@device] /iot gpio analog> set pin2 offset
```

Offset ::= [-]Num[mV]

Num ::= -2147483648..2147483647 (integer number)

```
[admin@device] /iot gpio analog> set pin2 offset 2
```

```
[admin@device] /iot gpio analog> print
```

#	NAME	VALUE	OFFSET
0	pin2	2mV	2mV
1	pin3	0mV	0mV

digital

在“digital”部分,你可以发送/接收一个逻辑 0 或 1 信号使用数字输出/输入引脚(输出引脚是“open drain 开路漏极”):

```
[admin@device] /iot gpio digital> print
```

```
Flags: X - disabled
```

#	NAME	DIRECTION	OUTPUT	INPUT	SCRIPT
0	pin5	input	0	0	
1	pin4	output	0		
2	pin6	output	0		

引脚的“DIRECTION”,即信号方向,可以是“input”(接收信号的引脚)或“output”(发送信号的引脚)。当引脚的方向设置为 output 时,可以配置 output 值。改变“OUTPUT”值将信号发送到引脚。

```
[admin@device] /iot gpio digital> set pin4 output=
```

```
Output ::= 0 | 1
```

```
[admin@device] /iot gpio digital> set pin4 output=1
```

```
[admin@device] /iot gpio digital> print
```

```
Flags: X - disabled
```

#	NAME	DIRECTION	OUTPUT	INPUT	SCRIPT
0	pin5	input	0	0	
1	pin4	output	1		
2	pin6	output	0		

“SCRIPT” 字段允许配置 RouterOS 脚本,当“INPUT”或“OUTPUT”值改变(从 0 到 1 或从 1 到 0)时,脚本将被触发。

```
[admin@device] /iot gpio digital> set pin4 script=script1
```

```
[admin@device] /iot gpio digital> set pin5 script="/system .."
```

```
[admin@device] /iot gpio digital> print
```

```
Flags: X - disabled
```

#	NAME	DIRECTION	OUTPUT	INPUT	SCRIPT
0	pin5	input	0	0	/system ..
1	pin4	output	1		script1
2	pin6	output	0		

控制继电器

GPIO 实现的场景之一是使用数字输出引脚“控制其他继电器”。基本上，发送“0”或“1”信号到引脚。整个操作自动化，可以通过 RouterOS 计划任务 scheduler，来执行脚本

例如，在 /system script 下创建两个脚本，第一个脚本，执行 pin4 引脚信号为 0，取名 "output=0"：

```
/iot gpio digital set pin4 output=0
```

然后，配置第二个脚本，执行 pin4 引脚信号为 1，取名 "output=1"：

```
/iot gpio digital set pin4 output=1
```

现在两条执行脚本准备好了，并配置到 schedule 下，第一个计划任务调用 output=0 的脚本，每 30 秒执行一次：

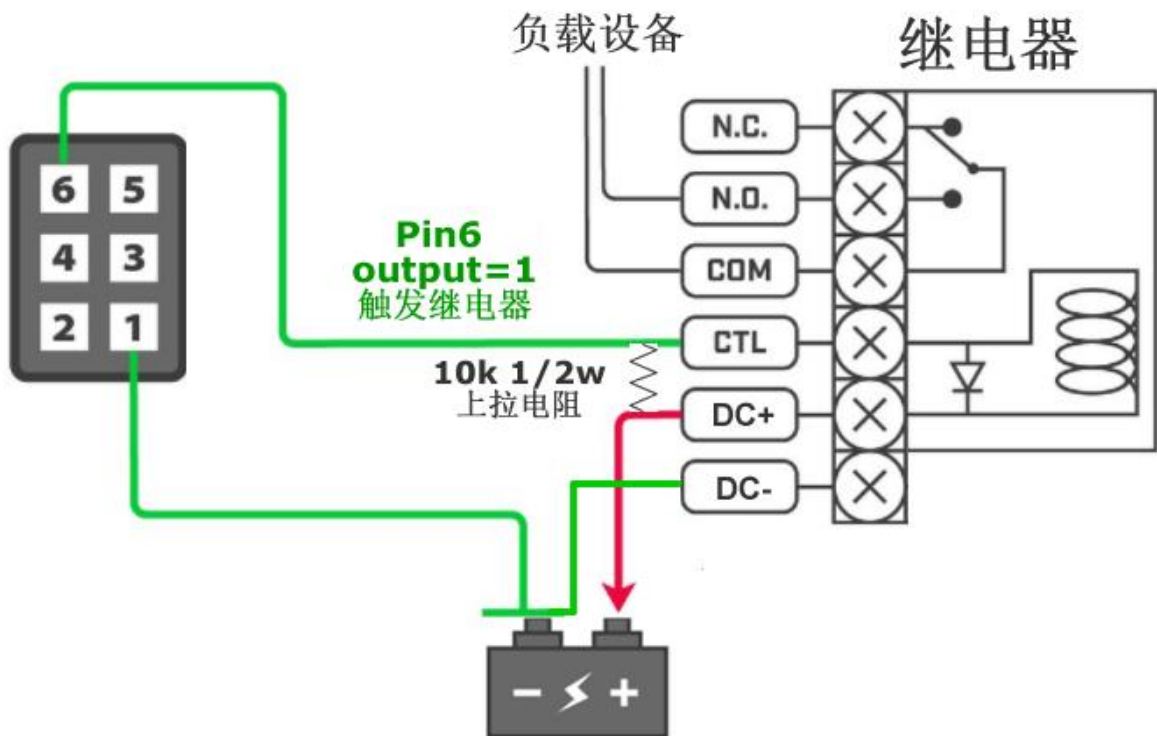
```
[admin@device] /system scheduler> add name=run-30s interval=30s on-event="output=0"
```

配置第二个计划任务调用 output=1，每 45 秒执行一次

```
[admin@device] /system scheduler> add name=run-45s interval=45s on-event="output=1"
```

因此，设备将每 30 秒自动发送一个 output=0 的信号到 pin4 引脚(数字输出引脚)，每 45 秒发送一个 output=1 的信号。计划任务执行的时间长短，可以根据实际情况调整。

下面图是通过继电器控制负载设备，如光敏电阻和人体红外感应传感器。



在触发 Pin6 针脚和供电 DC+ 之间需要接入一个上拉电阻。

监测输入信号

监测输入信号，通过 digital input pins（数字输出引脚），例如，当 pin 的“INPUT”值发生变化，方向为“INPUT”时(当 RouterOS 设备从另一台社保接收到“0 或 1”信号时)，该脚本将启动 email 通知或 MQTT/HTTPS (fetch)发送方式

E-mail 通知脚本：

```
/tool e-mail send to=config@mydomain.com subject=[/system identity get name] body="$[/iot gpio digital get pin5 input]"
```

以上脚本添加到/system script 取名 script1，并定义到 pin5 引脚，方向为 input

```
[admin@device] /iot gpio digital> set pin5 script=script1
[admin@device] /iot gpio digital> print
```

Flags: X - disabled

#	NAME	DIRECTION	OUTPUT	INPUT	SCRIPT
0	pin5	input	0	0	script1
1	pin4	output	0		script1
2	pin6	output	0		

因此，每当输入值发生变化(从 0 到 1 或从 1 到 0)，脚本就会自动启动一个 email 通知，

第六章 User Manager v5

RouterOS v7 版本发布了一个全新的 User Manager。从 v7.0beta4 开始，可以从官网下载 Extra packages 的 zip 压缩文件中获取。能应用于除 SMIPS 平台外的其他所有平台。主要特点是支持 EAP 认证和自定义 RADIUS 字段属性。管理配置不再提供 WEB 接口，直接通过 winbox 或者 CLI 进行操作，保留 User Manager 的客户访问的 WEB 界面。

User Manager 是在 RouterOS 中 RADIUS 服务器，User Manager 按照 RFC2865 和 RFC3579 定义的 RADIUS 标准工作，为指定的服务提供集中的用户认证和授权。拥有用户数据库可以更好的管理用户账号和各种参数。它支持多种不同的认证方式，包括 PAP、CHAP、MS-CHAP、MS-CHAPv2、EAP-TLS、EAP-TTLS 和 EAP-PEAP。在 RouterOS 中，DHCP、Dot1x、Hotspot、IPsec、PPP、Wireless 是从 User Manager 中受益最多的功能。每个用户都可以使用 WEB 界面查看自己的帐户信息，并管理可用的配置文件。此外，提供了支付 PayPal 网关配置接口。可以生成定制的报表，提供完善的账户管理。

User Manager 的 CLI 路径从 v7 开始，从“/tool user-manager”变动到“/user-manager”下操作。并提供 Winbox 配置支持，不会像以前会有一个单独的管理员 WEB 界面。客户 WEB 访问路径为“http://IP/um”

RADIUS 的基本工作原理：用户接入 NAS（认证服务器），NAS（认证服务器）向 RADIUS 服务器发送 Access-Require 数据包提交用户信息，基本的用户名和密码等相关信息；RADIUS 服务器对用户名和密码进行验证，必要时可以发出一个 Challenge，要求进一步对用户认证，如果匹配，给 NAS 返回 Access-Accept 数据包，允许用户进行下一步工作，否则返回 Access-Reject 数据包，拒绝用户访问；如果允许访问，NAS 向 RADIUS 服务器提出计费请求 Account-Require，RADIUS 服务器响应 Account-Accept，对用户的计费开始。

6.1 Database（数据库）

所有 RADIUS 相关信息存储在单独的数据库中，可在“database”子路径下配置。特别注意，两个参数“Enabled”和“db-path”是唯一不存储在 User Manager 的数据库中，存储在 RouterOS 系统配置中，这意味着这些参数会受到 RouterOS 配置复位的影响。其余的配置，包括会话和支付数据都存储在设备 FLASH 的单独 SQLite 数据库中。

属性	描述
db-path (<i>string</i> ; 默认:)	存放数据库文件的路径。
db-size(只读)	当前数据库使用大小
free-disk-space (只读)	存储数据库的磁盘上剩余的可用空间。
load (<i>name</i>)	恢复以前创建的备份文件，格式为.umb
migrate-legacy-db (<i>database-path</i> ; <i>overwrite</i>)	迁移老版本的 user manager (RouterOS v6 或之前的版本)转换为新标准。并覆盖当前数据库。
optimize-db ()	

属性	描述
save (name; overwrite)	保存 User Manager 当前数据库。

默认数据库存放在 `/file/user-manager5` 路径下，可以通过 `save` 命令定期进行数据库的备份。

```
[admin@MikroTik] /user-manager/database> save name=usermandb
Saving user-manager database
Database backup saved
```

6.2 Attributes（字段属性）

RADIUS 字段属性是定义的验证、信息和配置参数，在 RADIUS 服务器和客户端之间传递。User Manager 允许在 Attributes 目录中自定义的字段属性。RouterOS 已经提供预定义的字段属性，这些属性包括标准的字段属性和 RouterOS 私有字段属性。

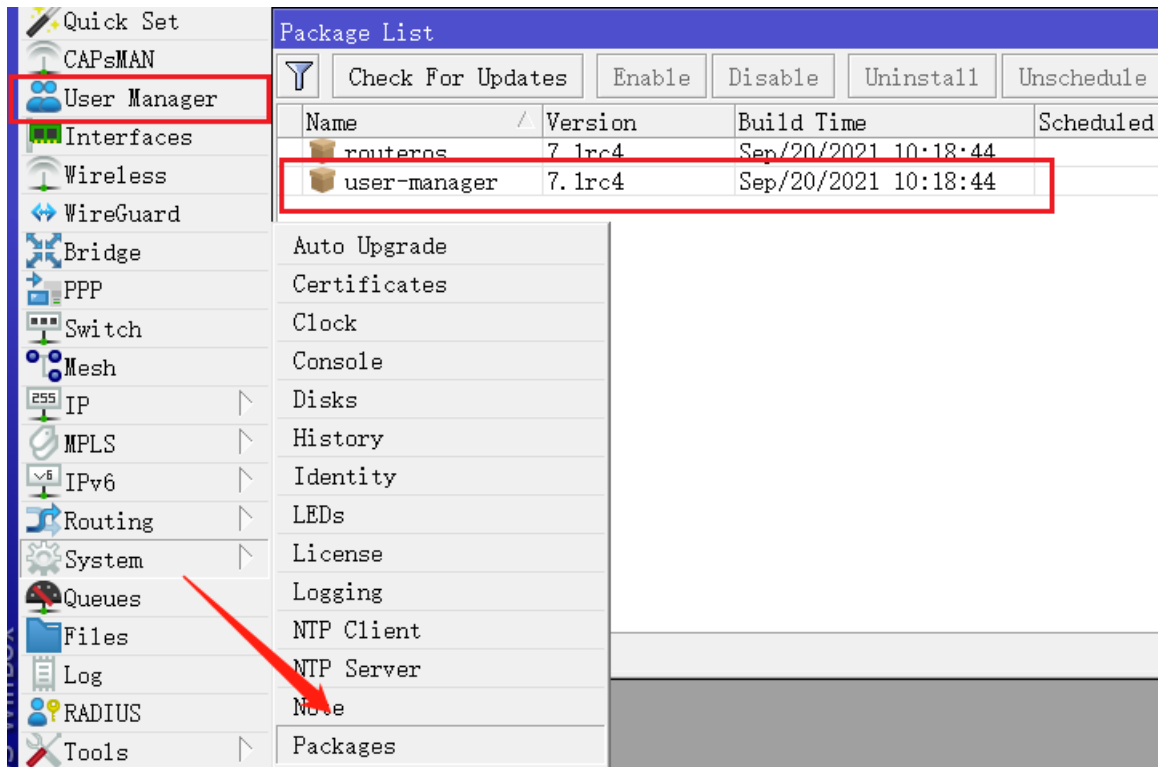
V7 版本的 User Manager 最大的特点在于可以自定义字段属性，不同厂商有自己的 RADIUS 字段，例如，在 Linux 下使用广泛的 FreeRADIUS，默认有一个路径存放 `dictionary`（字典），用于存放了不同厂商的字段属性，在配置文件中需要通过 `INCLUDE` 来调用，同时需要在数据库中创建这些字段属性，用于验证和数据请求。如果字典中没有厂商字段将无法匹配对应的参数

属性	描述
name (string; Default:)	属性名称
packet-types (string; Default: access-accept)	access-accept -在 RADIUS Access-Accept 报文中使用此属性 access-challenge -在 RADIUS Access-Challenge 报文中使用此属性
type-id (integer:1..255; Default:)	指定 Attribute 属性标识号。
value-type (string; Default:)	值类型，包括以下： hex ip-address - IPv4 或 IPv6 IP address ip6-prefix - IPv6 prefix macro string uint32
vendor-id (integer; Default: 0)	IANA 分配指定的企业识别号码

具体的属性描述请参阅[官方文档](#)

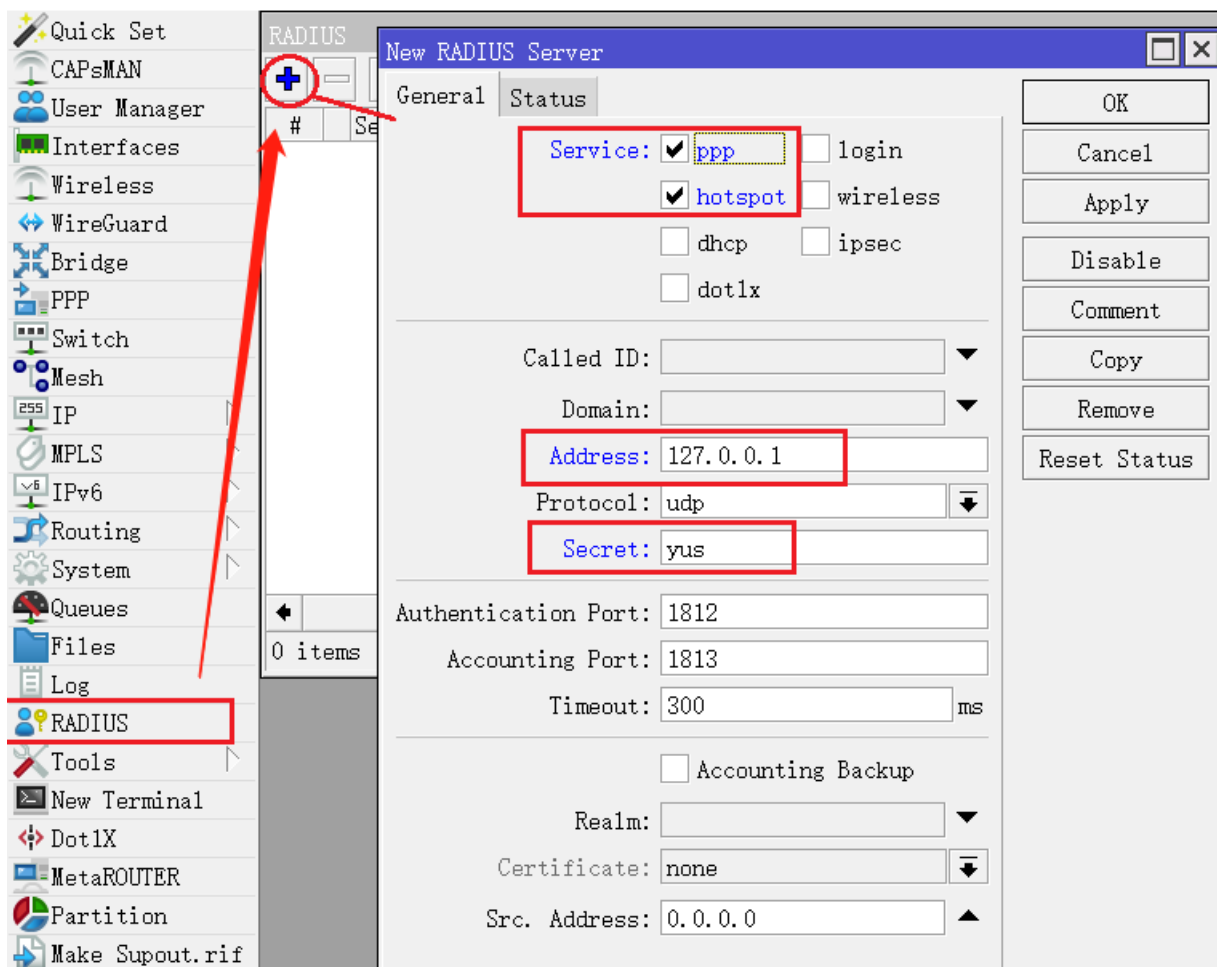
6.3 User Manager v5 配置实例

首先需要单独安装 user-manager 功能包，具体安装步骤请查阅 RouterOS 入门到精通，安装成功后，可以在 /system packages 查看，是否有 user-manager 功能包，同时可以在 Winbox 上看到 User Manager 选项：



启用 RADIUS

不管是本地 RouterOS，还是远端的 RouterOS 连接 User Manager，首先需进入 RADIUS 选项，创建新的规则，配置对接参数。

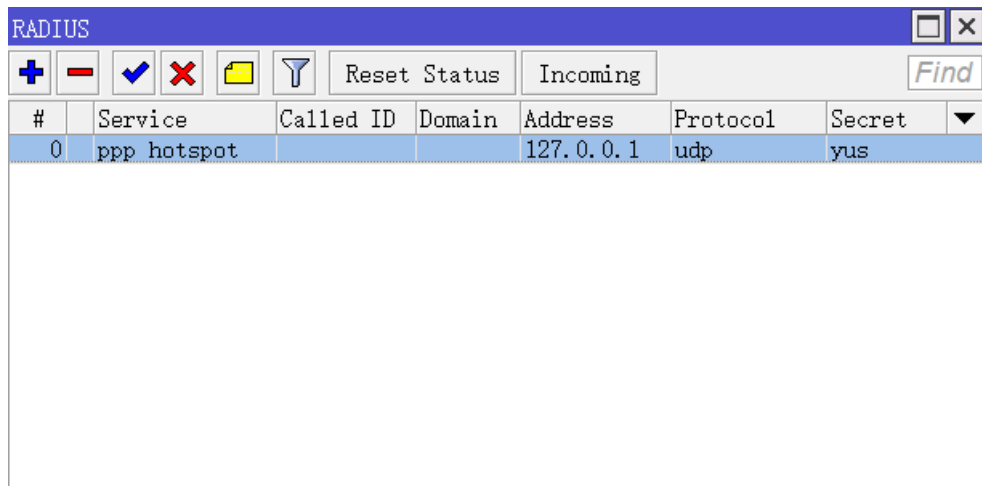


配置参数介绍：

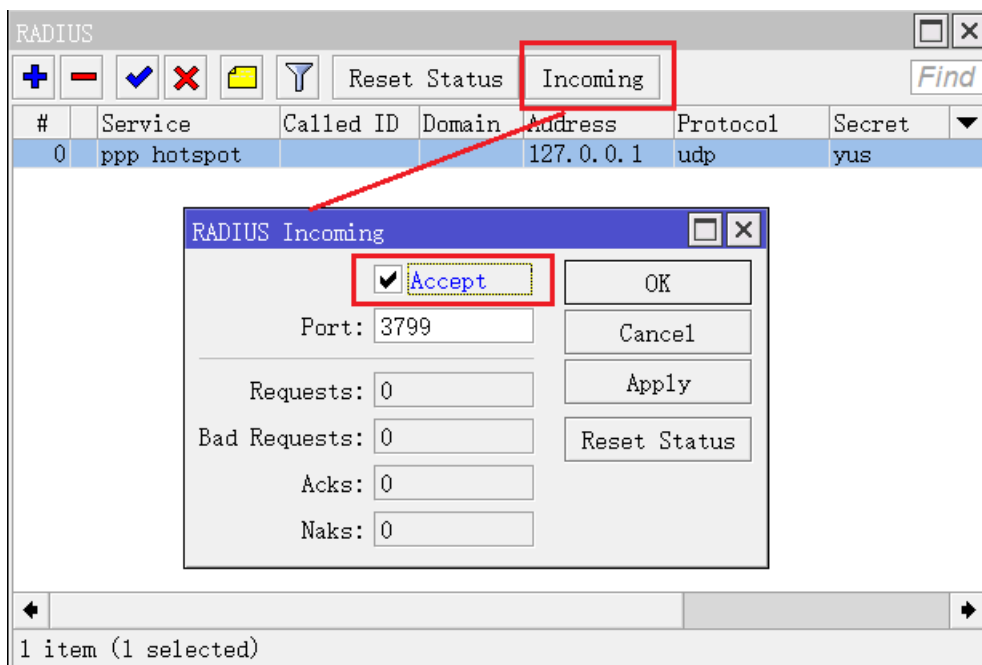
- **Service:** 是需要对接 RADIUS 用户认证的服务, 例如勾选 PPP 和 hotspot. PPP 对接的是如 PPPoE、PPTP、L2TP、OVPN 和 SSTP 等点对点隧道协议的账号认证服务。
- **Address:** 是 RADIUS 服务器的 IP, 这里在本地路由器启用 User Manager, 因此可以配置本地环回 IP 127.0.0.1, 也可以使用路由器接口 IP, 例如当在 ether2 接口配置了 192.168.88.1/24, 也可以用 192.168.88.1 连接。远端 RouterOS 连接, 需填写接口 IP 地址, 确保防火墙未阻断 udp/1812 和 1813 端口。
- **Secret:** 是路由器和 RADIUS 通信密钥

以上是连接 RADIUS 最基本的连接参数, 如果要指定访问 user-manager 的源地址 IP, 可以通过 src-address 参数。

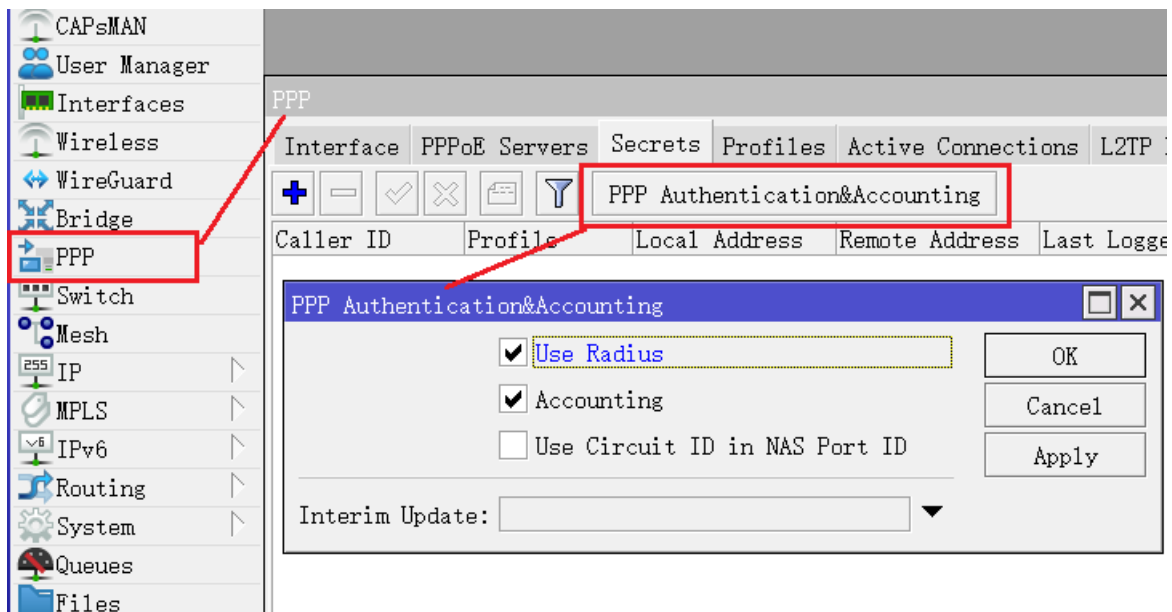
添加完成后显示如下：



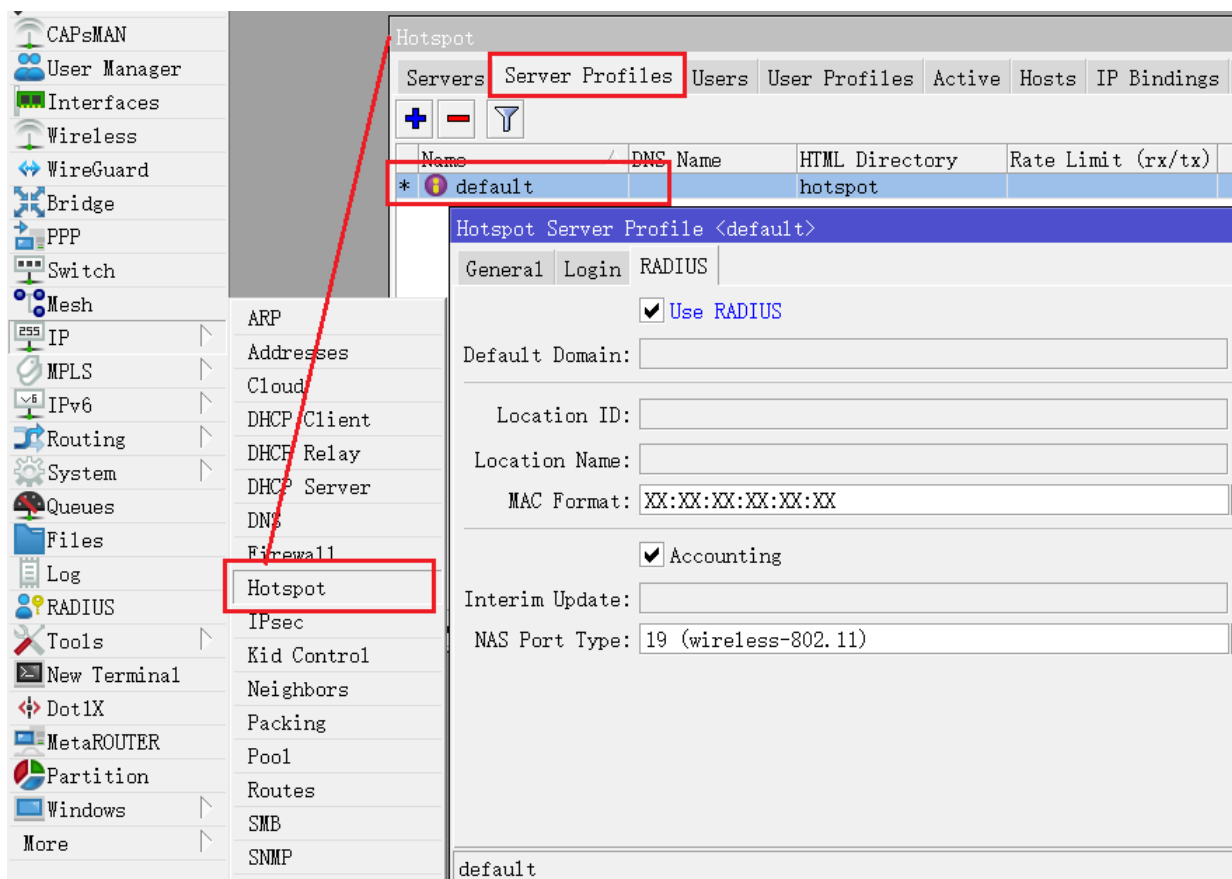
Incoming 功能，通过启用指定端口，允许终止 RADIUS 服务器上已经认证在线账号的会话，即发出 DM (Disconnect-Messages)，强制用户下线。



RouterOS 的每个功能还需要启用账号认证，配置 PPP (PPPoE、PPTP、L2TP、OVPN、SSTP)，连接 Radius 配置



Hotspot 认证，进入/ip hotspot profile 下的 default 规则，启用 RADIUS，如果新建了其他 profile 规则也需要单独启用。

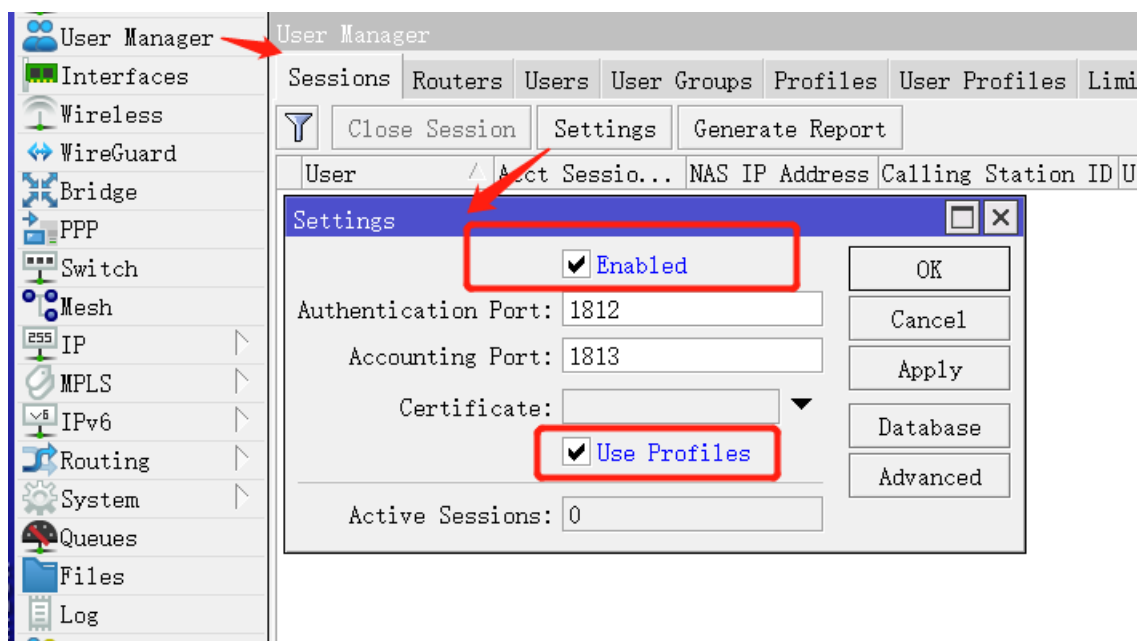


对应的功能启用完成后，下一步就配置 User Manager

配置 User Manager

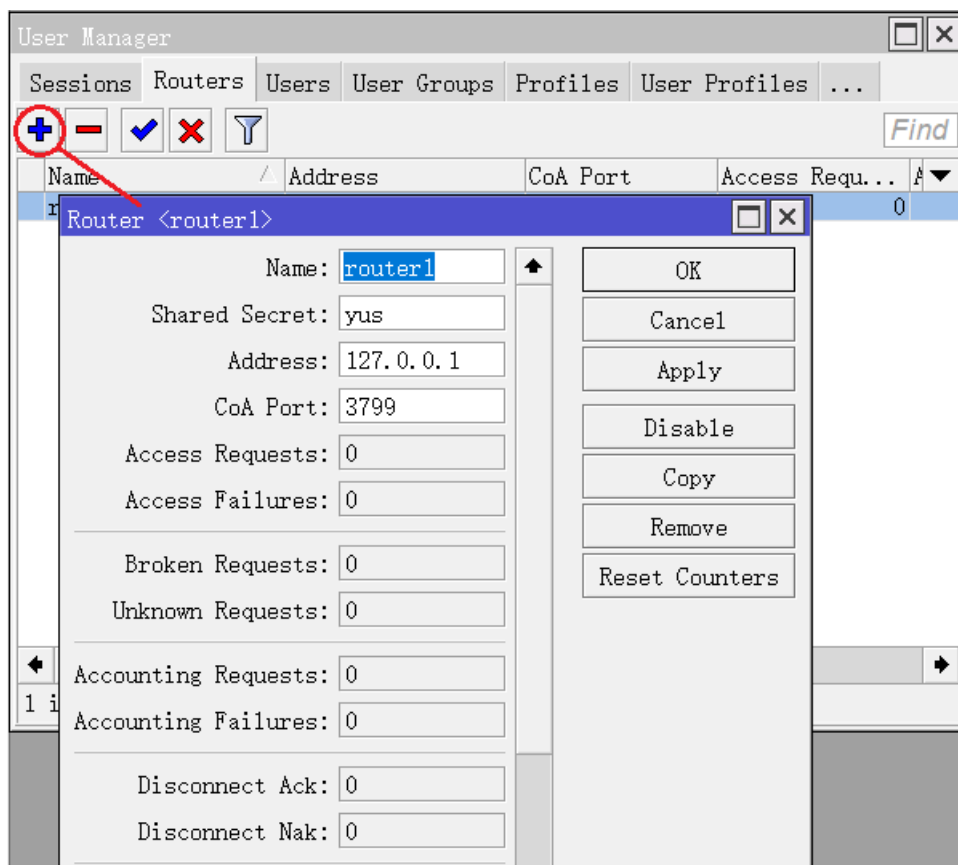
前面部分介绍了 PPP 和 Hotspot 开启 Radius 认证功能，User Manager 默认情况下是没有开启 Radius Server 服务的，因此需要先启动服务。

在 Sessions 的 settings 下，勾选 enabled，启用 User Manager 服务，验证和计费端口分别是 1812 和 1813，并启用 user profiles



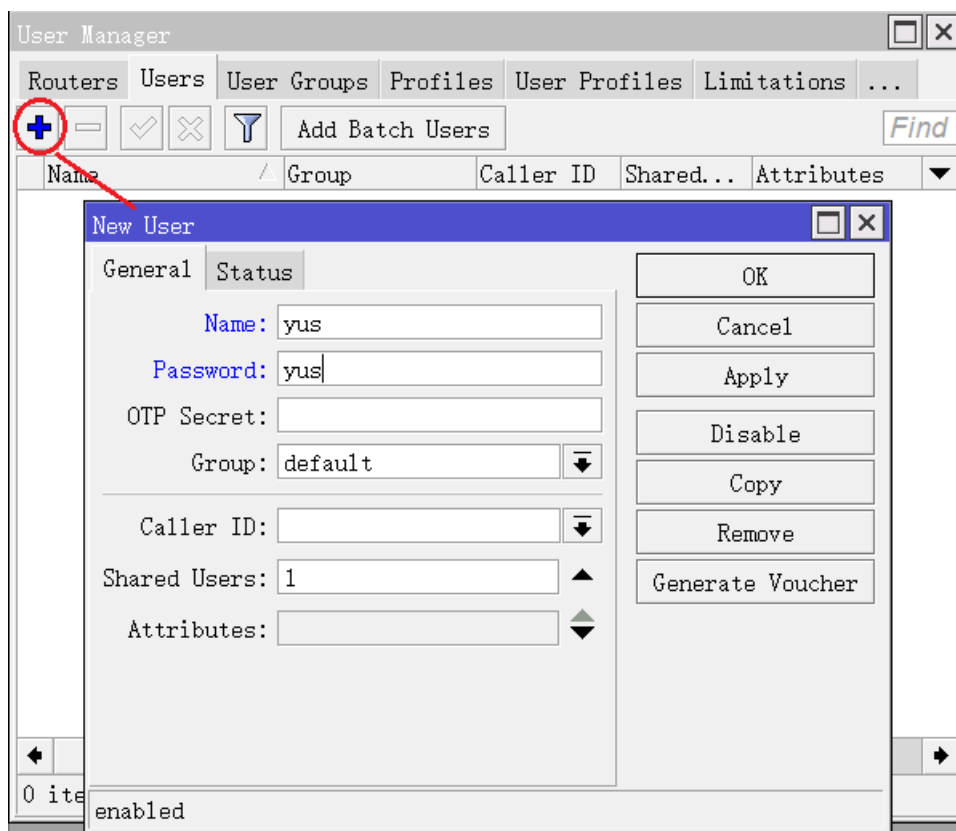
注意: User Profile 参数如果启用，对应的 User profiles 策略会被匹配（如果仅提供简单的账号认证，可以不用开启）

本实例是 RouterOS 为自身的 PPP 或 Hotspot 开启 User Manager 服务，仅需在 Routers 下，添加对接的路由器为本地环回地址 127.0.0.1，并配置密钥，CoA Port 就是路由器对应的 incoming 端口。



注意：如果其他 RouterOS 要对接 User Manager，在 Router 下配置对接的路由器，需要填写对端 IP 地址。

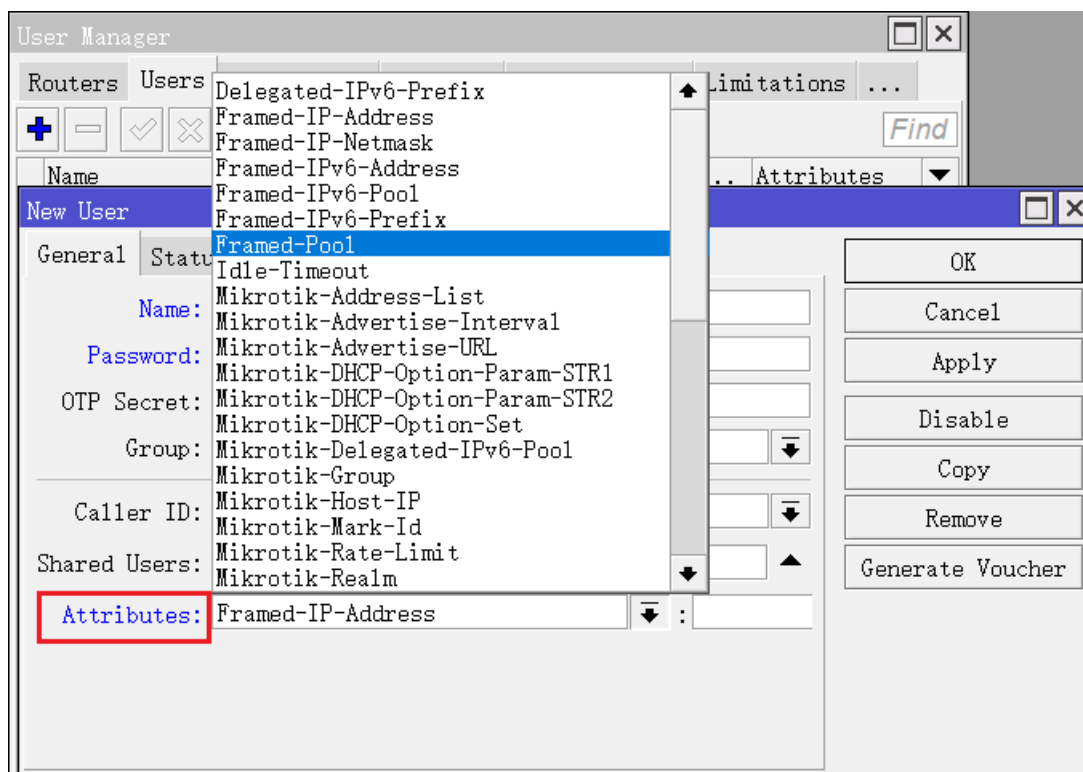
User Manager 以上两步配置完成后，路由器 RADIUS 和 User Manager 就完成对接，剩下的就是创建用户账号：



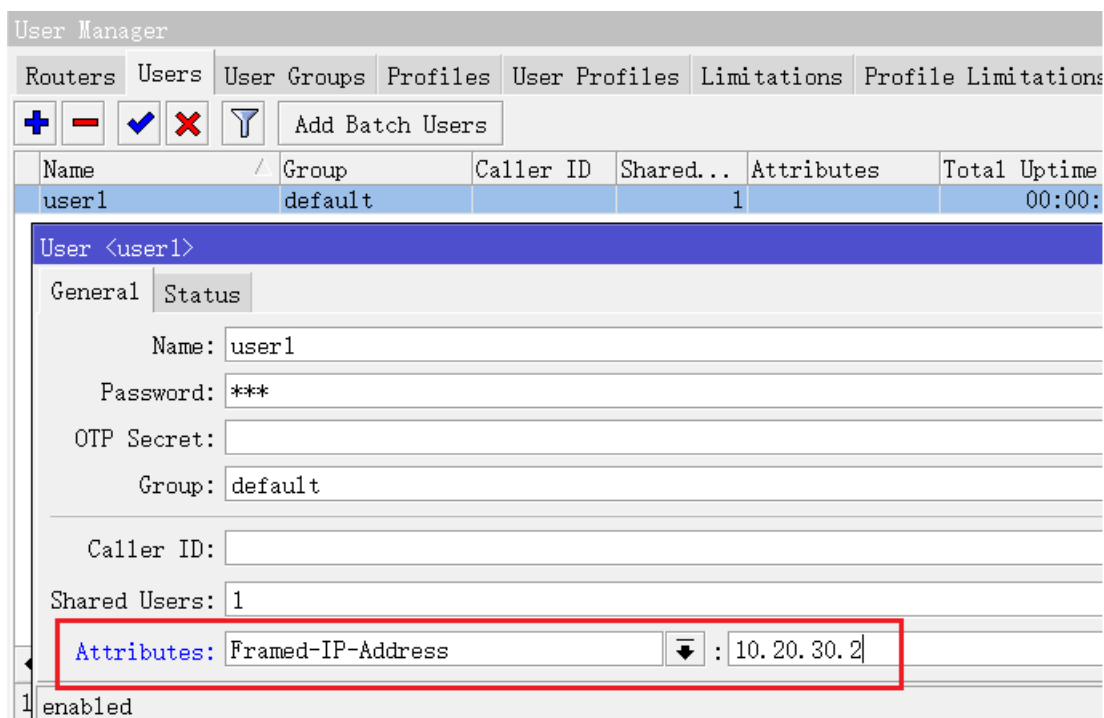
以上是最基本的用户账号和密码配置，Shared users 账号共享使用设置为 1，即账号只能一人在线。

Caller ID 可以手动设置，也可以选择 bind，bind 代表初次登录将绑定用户的 caller ID（PPPoE 为用户请求拨号 MAC 地址，PPTP、L2TP、SSTP、OVPN 等为用户请求连接的 IP 地址）

下面一个 Attributes 参数（字段属性），就是 user-mangerv5 功能改进最大的亮点，在该账号下点开 Attributes 后，可以看到很多参数可以选择，包含了 Radius 默认字段，以及 MikroTik 私有字段属性：



可以选择 Framed-IP-Address 字段，给 user1 账号分配一个指定的 IP 地址



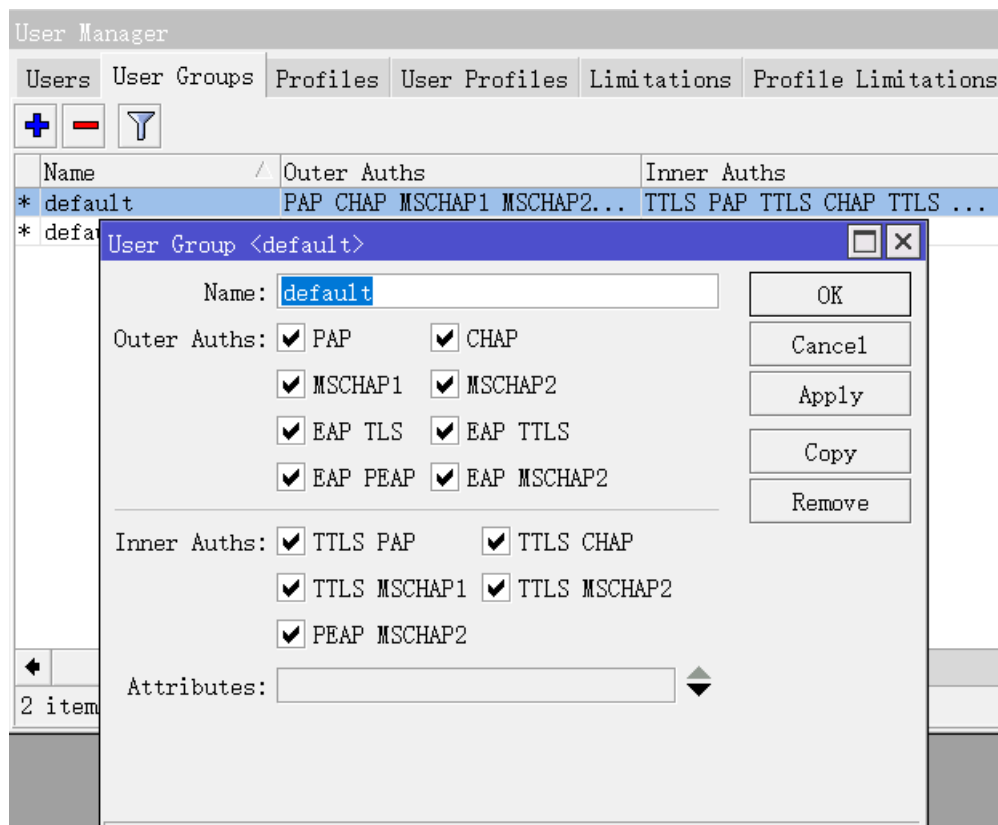
也可以指定 Framed-Pool 调用 RouterOS 在/ip pool 中创建的地址池 pool1

User <user1>	
General	Status
Name:	user1
Password:	***
OTP Secret:	
Group:	default
Caller ID:	
Shared Users:	1
Attributes:	Framed-Pool : pool1
enabled	

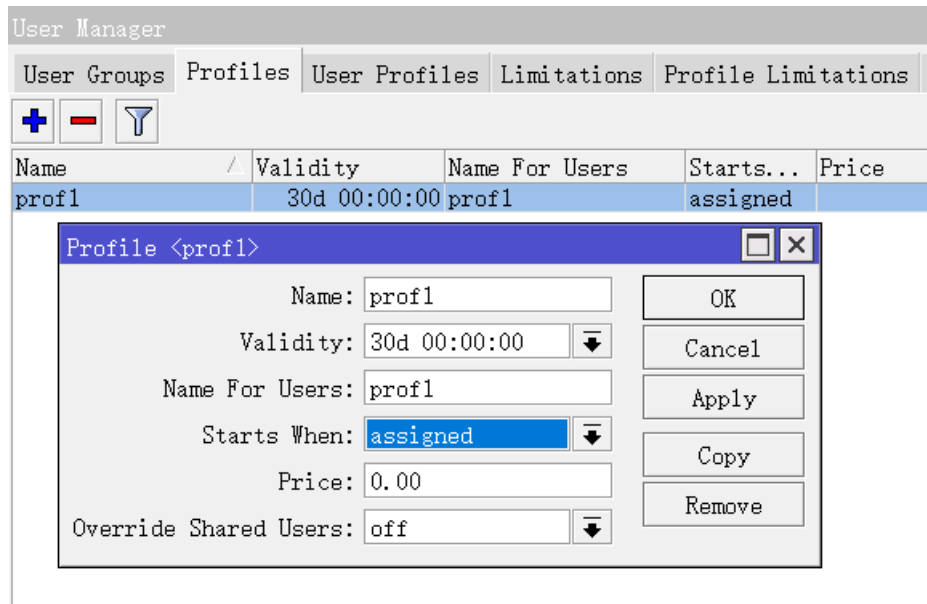
Attributes 可定义多个字段属性给 user1 账号，例如 user1 在认证成功登录后，将 user1 账号的 IP 添加到 RouterOS 的/ip firewall address-list 中，并定义地址列表名称为 user-address-list

User <user1>	
General	Status
Name:	user1
Password:	***
OTP Secret:	
Group:	default
Caller ID:	
Shared Users:	1
Attributes:	Framed-Pool : pool1
	Mikrotik-Address-List : user-address-list
enabled	

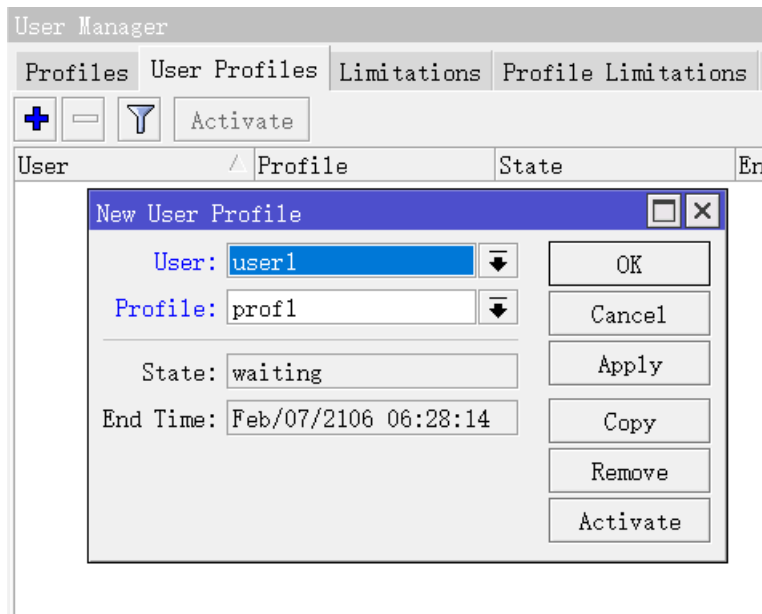
在 user group 用户分组下的验证选择，分组功能下同样可以选择 Attributes 参数



在 profile 定义有效时间和其他参数,这里 **Validity** 有效时间为 30 天,格式是“天 时:分:秒”,start When 选择 **assigned**,即分配即计费开始,还可以选择 **first auth**,第一次认证后开始计费。



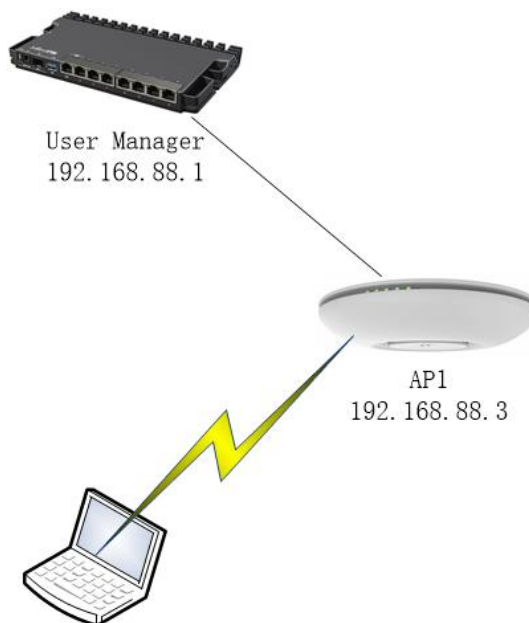
进入 user profiles, 就可以将用户账号和策略绑定在一起, user1 和 prof1 匹配, 并会自动计算出 end time 到期时间。



以上 User Manager 的账号认证配置基本完成。

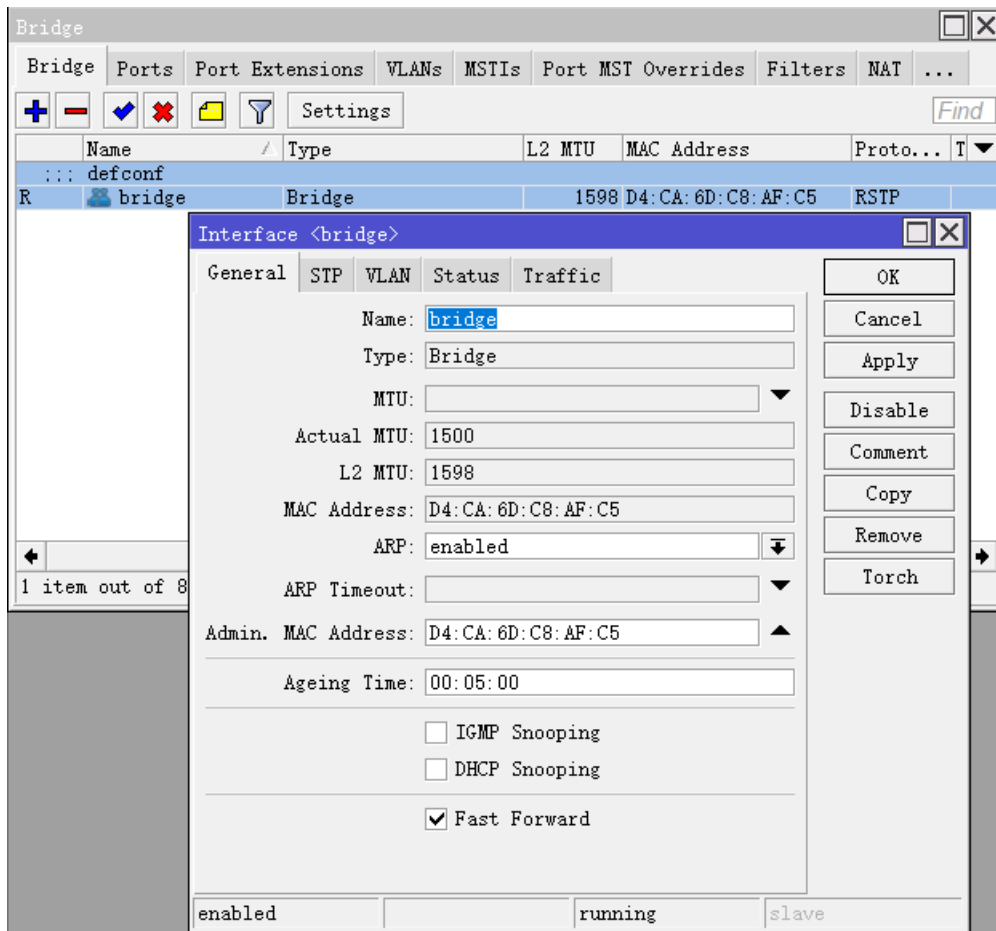
6.4 WiFi 创建 802.1x EAP 认证

User Manager v5 支持用户验证基于 802.1x EAP (Extensible Authentication Protocol) 协议，客户通过账号和密码登录 WiFi，使用 PEAP 方式无需客户端添加证书。在这个实例中，一台 cAP ac 设备作为 AP 网桥，一台 RB5009 作为网关和 User Manager 认证设备，拓扑如下



cAP ac 端配置

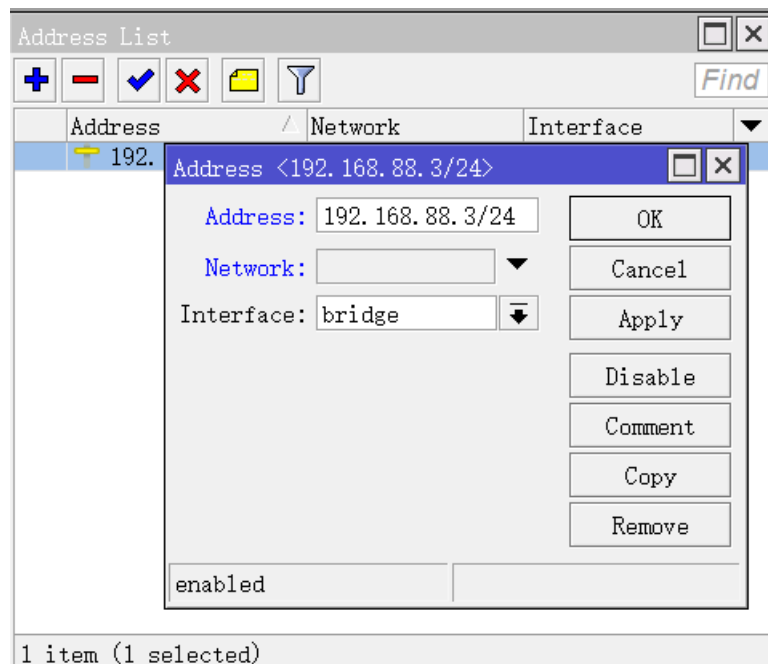
cAP ac 端创建一个 Bridge 接口，cAP ac 默认配置有一个 bridge



将 wlan1、wlan2 和 ether1 加入 bridge

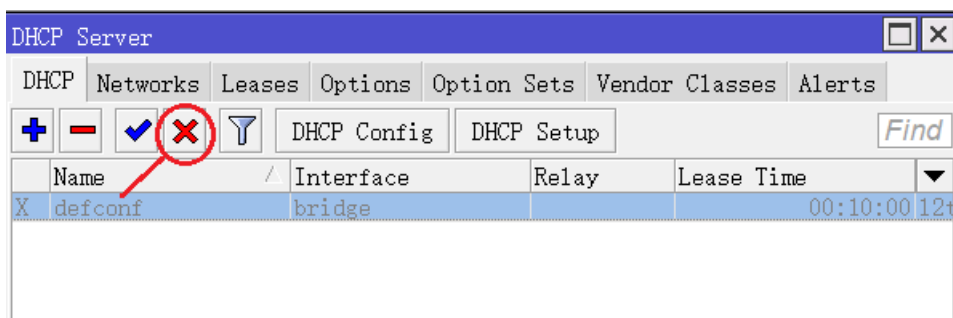
Bridge								
Bridge Ports								
#		Interface	Bridge	Horizon	Trusted	Priorit...	Path Cost	RoI
0	I	wlan1	bridge		no	80	10	dis
1		wlan2	bridge		no	80	10	des
2	H	ether1	bridge		no	80	10	des

进入/ip address, 配置 Bridge 接口 IP 地址, 用于 RB5009 建立 IP 通信, 访问 User Manager:

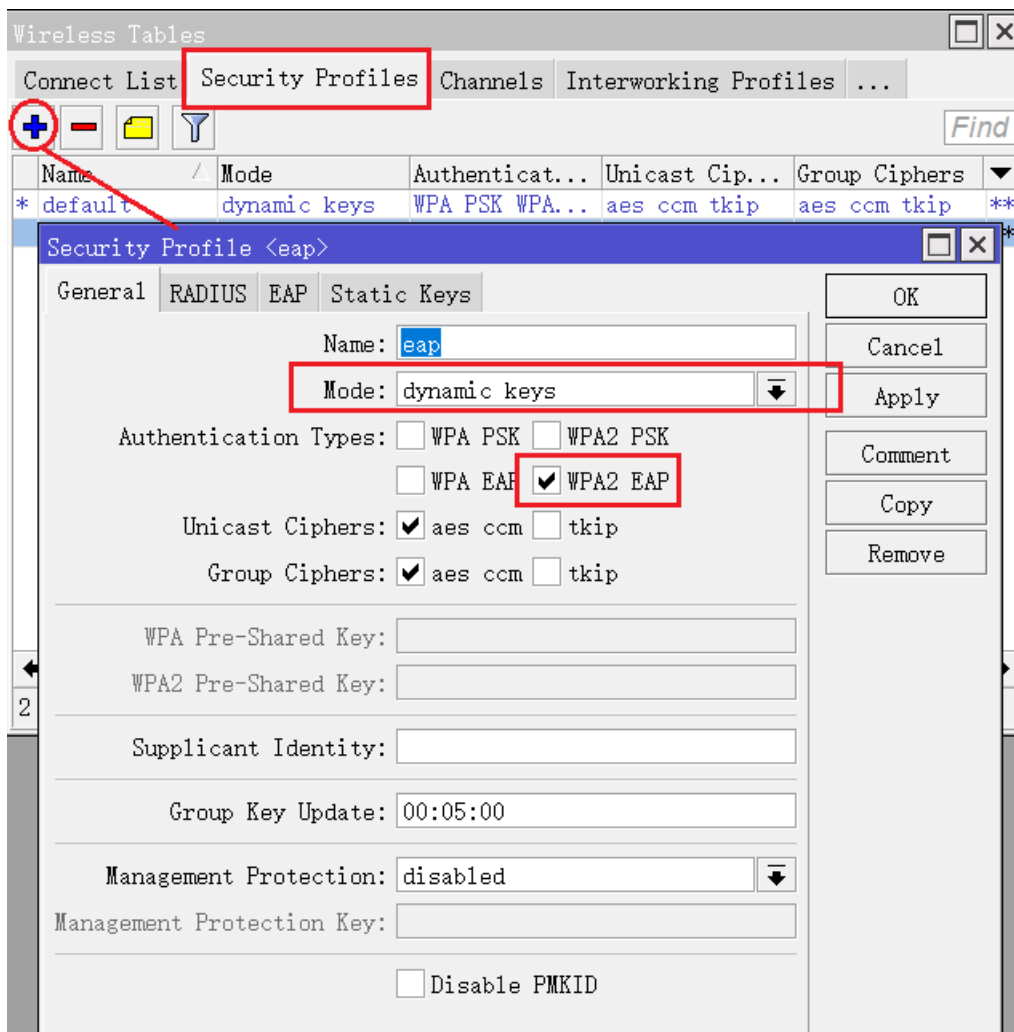


如果 CAP ac 需要被其他网段访问，或者访问互联网，需要在/ip route 下添加 gateway 网关。

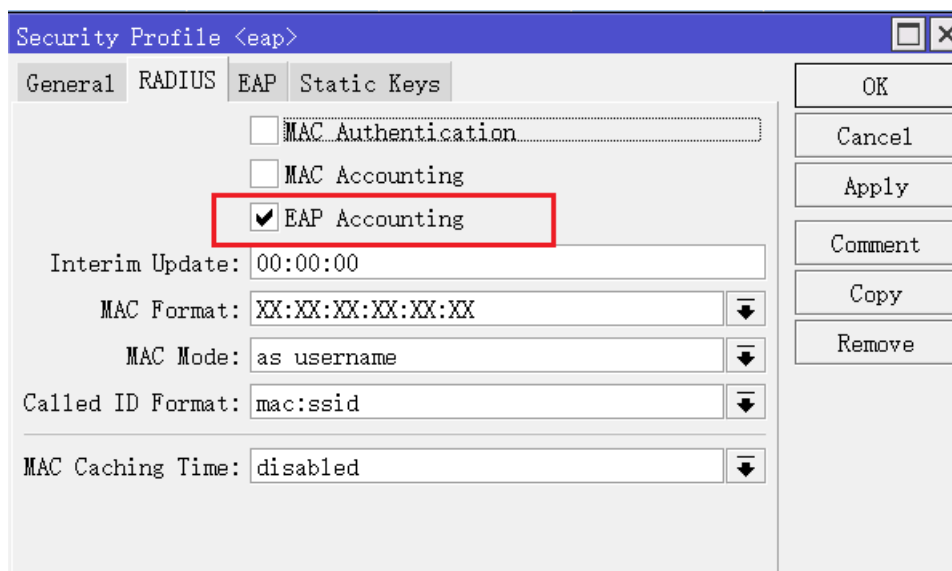
在/ip dhcp-Server 下查看 DHCP-Server 服务，这里选择禁用，由 RB5009 提供 DHCP-Server 服务



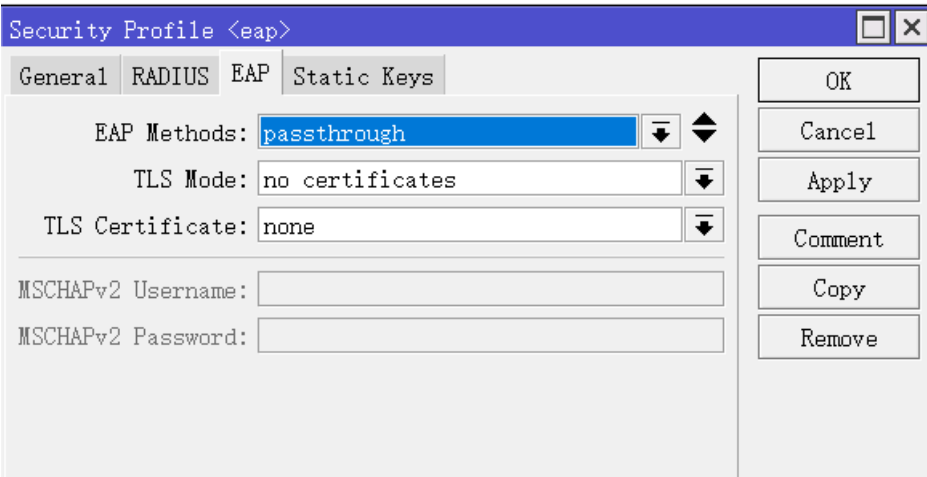
进入 CAP ac 的 wireless security，创建一条 eap 的安全策略：



选择 RADIUS 启用 EAP Accounting（发送计费信息到 User Manager 的 1813 端口）



在 EAP 栏下，默认设置，EAP methods 选择 passthrough，直通到 RADIUS 服务器



规则创建完成如下：

Wireless Tables						
WiFi InterfacesW60G StationNstreme DualAccess ListRegistrationConnect ListSecur						
+ - [icon] [icon]						
Name /	Mode	Authenticat...	Unicast Cip...	Group Ciphers	WPA Pre-Shared	
* default	dynamic keys	WPA PSK WPA...	aes ccm tkip	aes ccm tkip	*****	
eap	dynamic keys	WPA2 EAP	aes ccm	aes ccm	*****	

cAP ac有 wlan1 和 wlan2 分别是 2.4GHz 和 5GHz 频率的网卡，都配置为 ap-bridge 模式，下面以 Wlan1 网卡为例，在 wireless 下设置无线的 SSID 为 myEAP，security profile 选择 eap

Interface <wlan1>

GeneralWirelessHTHT MCSWDSNstremeStatusTraffic

Mode:ap bridge

Band:2GHz-G/N

Channel Width:20/40MHz Ce

Frequency:2417MHz

SSID:myEAP

Security Profile:eap

WPS Mode:disabled

Frequency Mode>manual-txpower

Country:no_country_set

Installation:any

Default AP Tx Limit:bps

Default Client Tx Limit:bps

☒ Default Authenticate

☒ Default Forward

☐ Hide SSID

OK

Cancel

Apply

Disable

Comment

Advanced Mode

Torch

WPS Accept

WPS Client

Setup Repeater

Scan...

Freq. Usage...

Align...

Sniff...

Snooper...

Reset Configuration

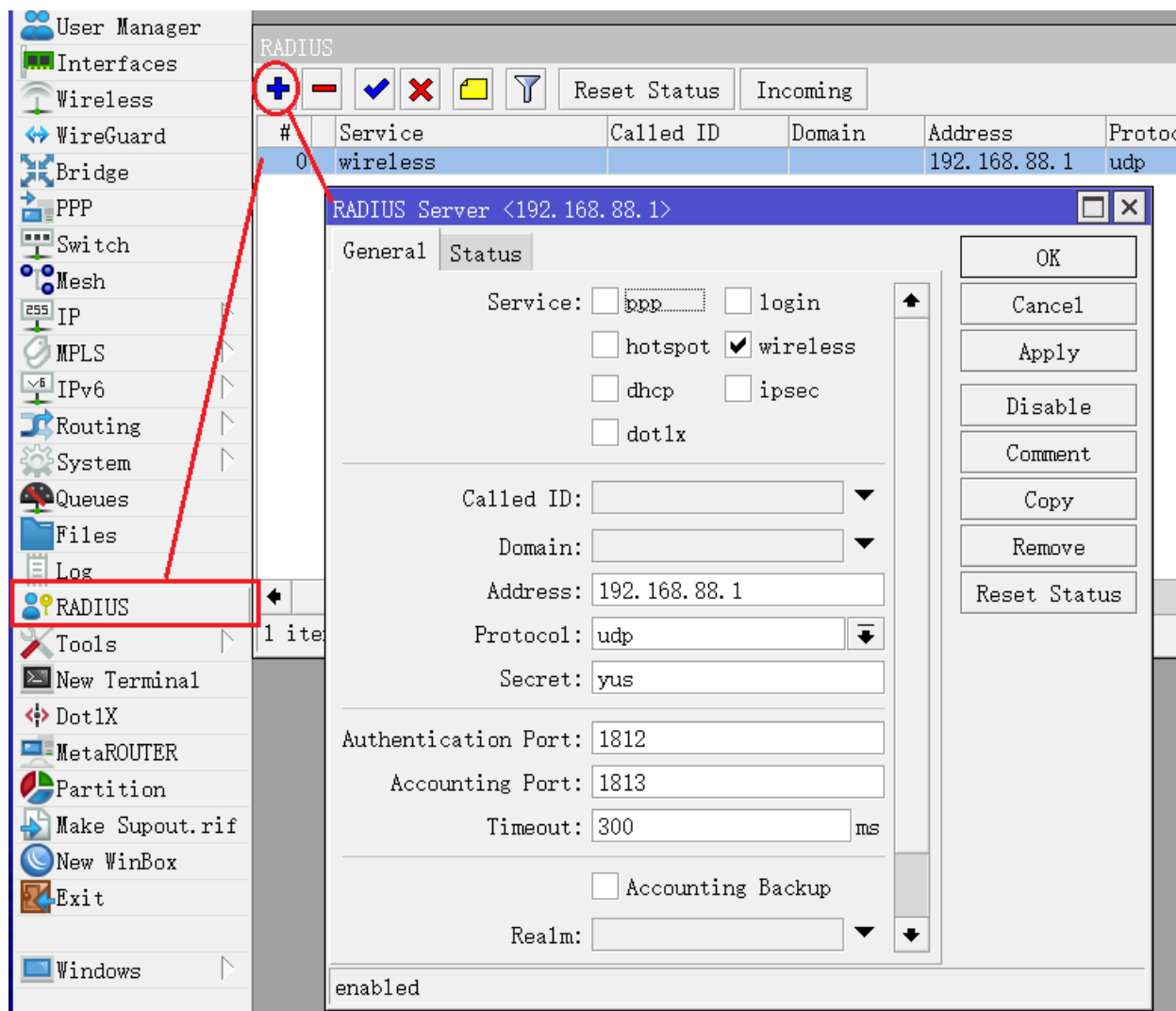
enabled

running

slave

running ap

进入 RADIUS，添加一条连接 User Manager 的规则，配置如下：



勾选 wireless 项，设置连接的 IP 地址 192.168.88.1，secret 为 yus

RB5009 配置

可使用 MikroTik RB5009 出厂的默认配置，Bridge 接口 IP 地址为 192.168.88.1/24，DHCP-Server 服务开启，由 RB5009 分配 IP 地址，通过网线连接 cAP ac，cAP ac 通过无线桥接将 IP 分发给无线客户端。

RB5009 需要安装 User Manager 功能包，在启用 User Manager 服务前，需为 User Manager 创建 CA 证书，可以在 certificate 路径下创建证书

```
[admin@MikroTik] >/certificate
[admin@MikroTik] /certificate>
add name=eap-ca key-size=secp384r1 digest-algorithm=sha384 days-valid=800 key-usage=key-cert-sign,
crl-sign
```

签发 eap-ca 证书

```
[admin@MikroTik] /certificate>sign eap-ca ca-crl-host=192.168.88.1
```

为 User Manager 创建服务器证书

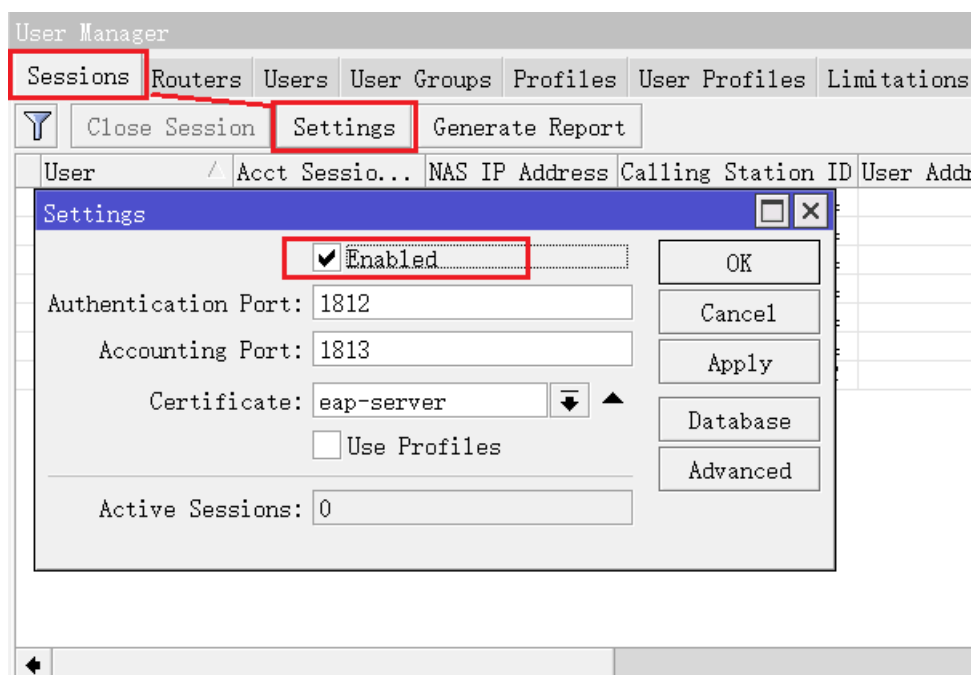
```
[admin@MikroTik] /certificate>
```

```
add name=eap-server common-name=192.168.88.1 key-size=secp384r1 digest-algorithm=sha384 days-val  
id=800 key-usage=tls-server
```

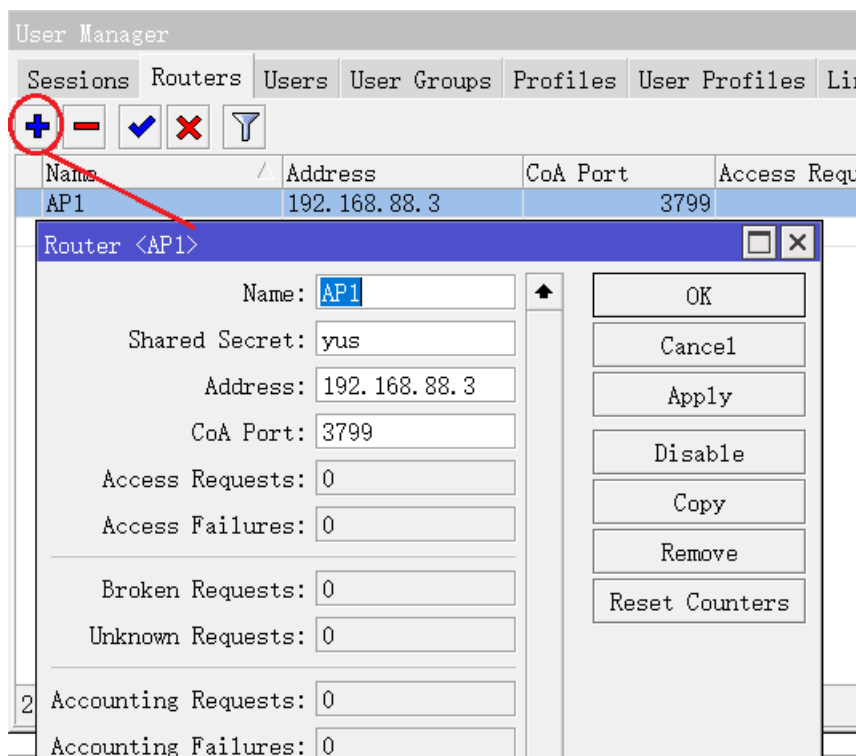
使用 eap-ca 证书签发 eap-server 证书

```
[admin@MikroTik] /certificate>sign eap-server ca=eap-ca
```

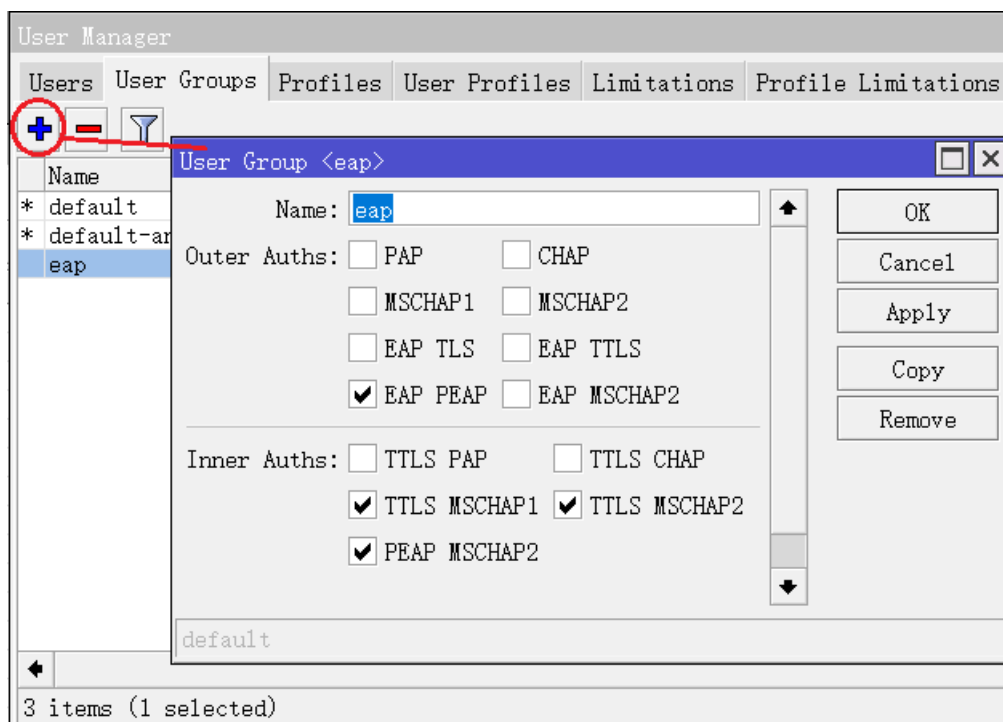
证书创建完成后，启用 User Manager 服务，并选择 Certificate 为 eap-server 证书



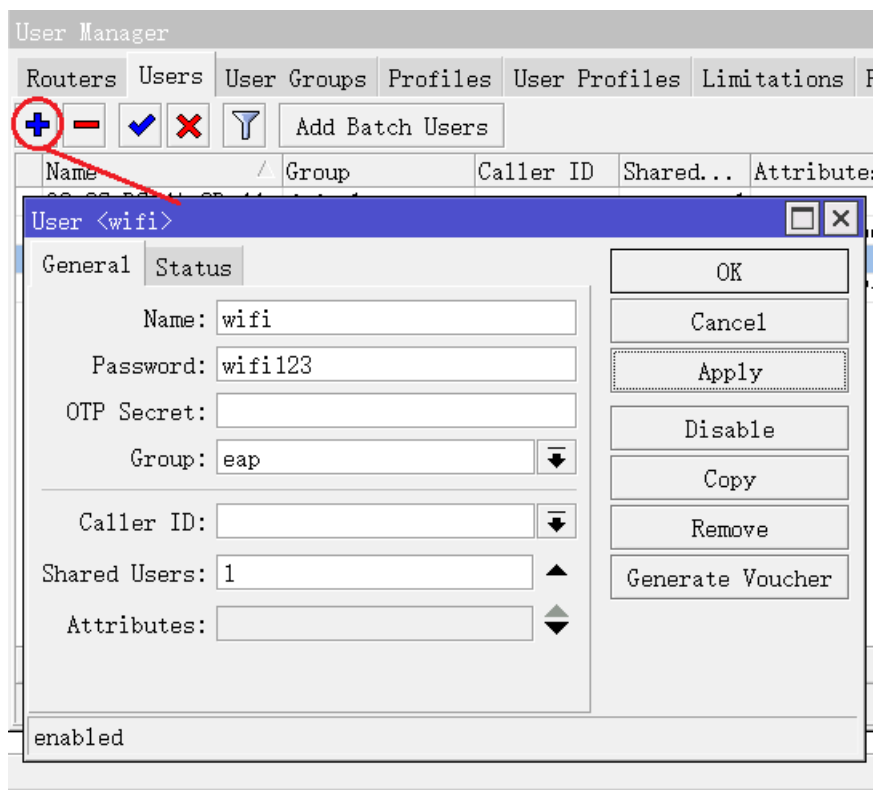
在 Router 栏下添加 cAP 的 IP 地址，和相同的 secret 参数，允许连接 User Manager



在 User Groups 下创建一个新的组 eap，选择响应的验证类型：



在 Users 栏下创建账号，账号 wifi，密码 wifi123



Windows 连接 WiFi

以上配置完成后，windows 连接 SSID 名为 myEAP 的 WiFi，输入账号密码



提示如下信息，仍然连接



等待验证完成后，WiFi 会显示连接成功。

Android 手机连接时选择不验证证书，苹果的手机和笔记本连接会提示“信任”证书。

第七章 Certificate 证书

RouterOS 支持管理和创建自签 CA 证书，基于 RFC 5280 实现，所有证书是 X.509 v3，且所有证书指纹均为 SHA1。从 v6.18 开始，sha256 用于证书指纹和哈希。所有 Key 和 CA 导出密码都与硬件 ID 加密存储。CA CRL 更新发生在每次证书撤销和 24 小时后。

特别注意：在 RouterOS 创建证书时，系统日期和时间必须正确设置，可以使用 RouterOS 的 Cloud 同步时间或者用 NTP。

RouterOS 证书的管理使用主要包括以下：

- 用于路由器所需连接的证书管理 collecting all certificates inside the router;
- 管理并创建自签证书;
- 控制和设置 SCEP 相关配置。

从 RouterOS v6 开始，证书的有效性使用本地时区偏移来确认，在以前的版本中，它是 UTF。

操作路径：/certificate

在/certificate 路径下用于管理证书、添加模板、颁发证书和管理 SCEP 客户端。

7.1 证书简单操作

证书模板

当执行证书签发或证书请求命令后，模板会被删除并生成 CA 证书，默认情况下签发时间为一年

```
/certificate
add name=CA-Template common-name=CAtemp key-usage=key-cert-sign,crl-sign
add name=Server common-name=server
add name=Client common-name=client
```

使用 print detail 命令显示详细内容：

```
[admin@4k11] /certificate> print detail
Flags: K - private-key; L - crl; C - smart-card-key; A - authority; I - issued, R - revoked; E - expired; T - trusted
0  name="CA-Template" key-type=rsa common-name="CAtemp" key-size=2048 subject-alt-name=""
days-valid=365 key-usage=key-cert-sign,crl-sign

1  name="Server" key-type=rsa common-name="server" key-size=2048 subject-alt-name=""
days-valid=365
key-usage=digital-signature,key-encipherment,data-encipherment,key-cert-sign,crl-sign,tls-server,tls-client
```

```
2 name="Client" key-type=rsa common-name="client" key-size=2048 subject-alt-name=""
days-valid=365
key-usage=digital-signature,key-encipherment,data-encipherment,key-cert-sign,crl-sign,tls-server,tls-client
```

注意，如果 CA 证书被删除，那么链中所有已颁发的证书也将被删除。

签发证书

证书需要通过 **sign** 签发。在以下示例中，将对证书进行签名，并为添加服务器证书的 CRL-URL，例如签发服务器证书时指定 **ca-crl-host:192.168.88.1**，即服务器 IP 地址：

```
/certificate
sign CA-Template
sign Client
sign Server ca-crl-host=192.168.88.1 name=ServerCA
```

通过 **print** 命令检查证书签发情况：

```
[admin@MikroTik] /certificate> print
Flags: K - private-key; L - crl; A - authority; T - trusted
Columns: NAME, COMMON-name, FINGERPRINT
#      NAME      COMMON  FINGERPRINT
0  K AT  CA-Template  CAtemp  0c7aaa7607a4dde1bbf33deaae6be7bac9fe64ba47d64e8a73dcad6cf8
1  K AT  Client      client  b3ff25ecb166ea41e15733a7493003f3ea66310c103903e98fe324c3659f
2  K LAT ServerCA  server  152b88c9d81f4b765a59e2302e01efd1fbf11ceed6e59f4977787a5bb980
```

需要注意：密钥签名过程的时间取决于指定证书的密钥大小。如果值为 **4k** 或更高，那么在功能较弱的 CPU 设备上签名证书可能需要更多的时间。

导出证书

可以使用密钥和 CA 证书导出客户端证书 It is possible to export client certificates with keys and CA certificate:

```
/certificate
export-certificate CA-Template
export-certificate ServerCA export-passphrase=yourpassphrase
export-certificate Client export-passphrase=yourpassphrase
```

导出证书后，可以在 **/file** 下找到：

```
[admin@MikroTik] > file print
```

Columns: NAME, TYPE, SIZE, CREATION-TIME

#	NAME	TYPE	SIZE	CREATION-TIME
0	skins	directory		jan/19/2019 00:00:04
1	flash	directory		jan/19/2019 01:00:00
2	flash/rw	directory		jan/19/2019 01:00:00
3	flash/rw/disk	directory		jan/19/2019 01:00:00
4	pub	directory		jan/19/2019 02:42:16
5	cert_export_CA-Template.crt	.crt file	1119	jan/19/2019 04:15:21
6	cert_export_ServerCA.crt	.crt file	1229	jan/19/2019 04:15:42
7	cert_export_ServerCA.key	.key file	1858	jan/19/2019 04:15:42
8	cert_export_Client.crt	.crt file	1164	jan/19/2019 04:15:55
9	cert_export_Client.key	.key file	1858	jan/19/2019 04:15:55

7.2 Let's Encrypt 证书

RouterOS v7 支持“www-ssl”服务的 Let's Encrypt 证书，如下使用 'enable-ssl-certificate' 命令启用 Let's Encrypt 证书，并自动更新服务：

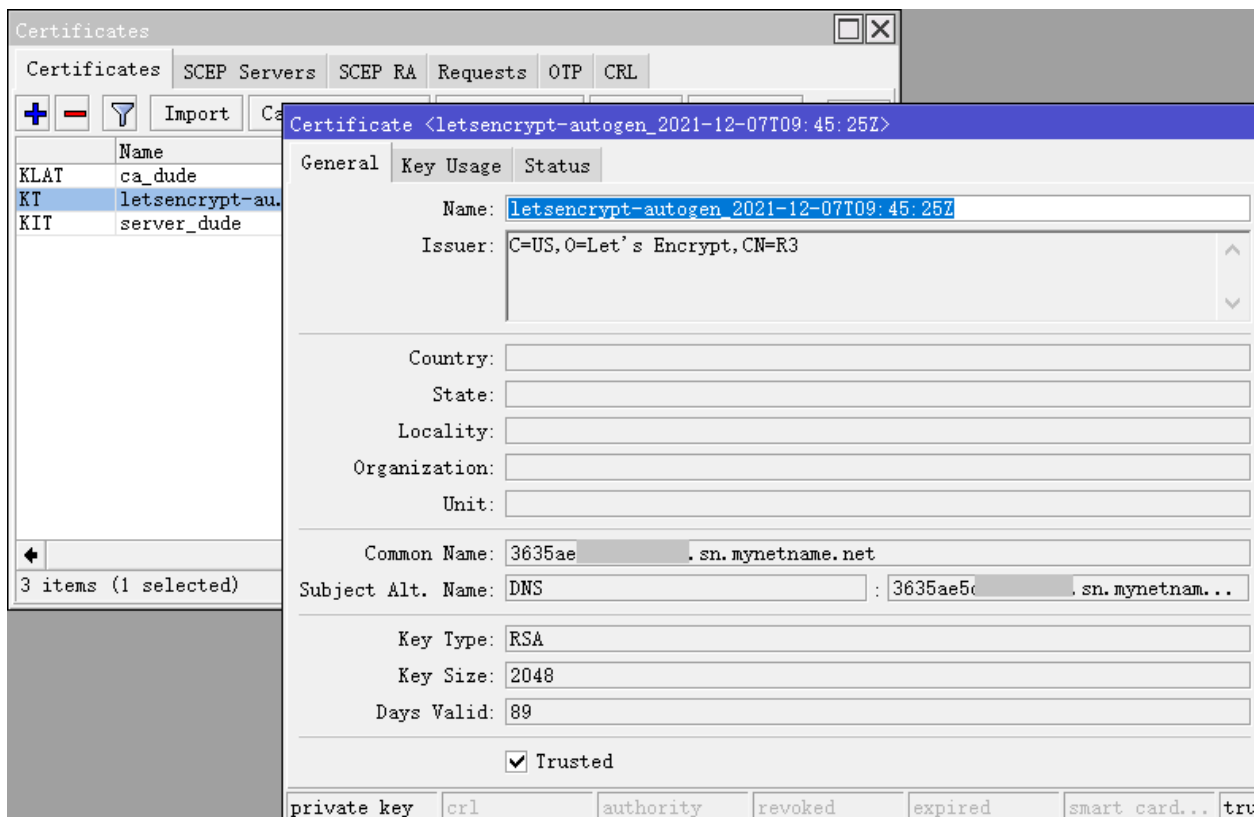
```
/certificate
enable-ssl-certificate
```

命令包含一个参数 'dns-name' 指定域名，用于证书生成，如开启本地 RouterOS 的 www-ssl 访问，可以使用 ip cloud 的 ddns 域名，在生成证书时，需将域名解析指向主机的 IP，并开启本地的 TCP/80 端口（在国内使用，请确保运营商开放 IP 地址的 TCP/80 端口）

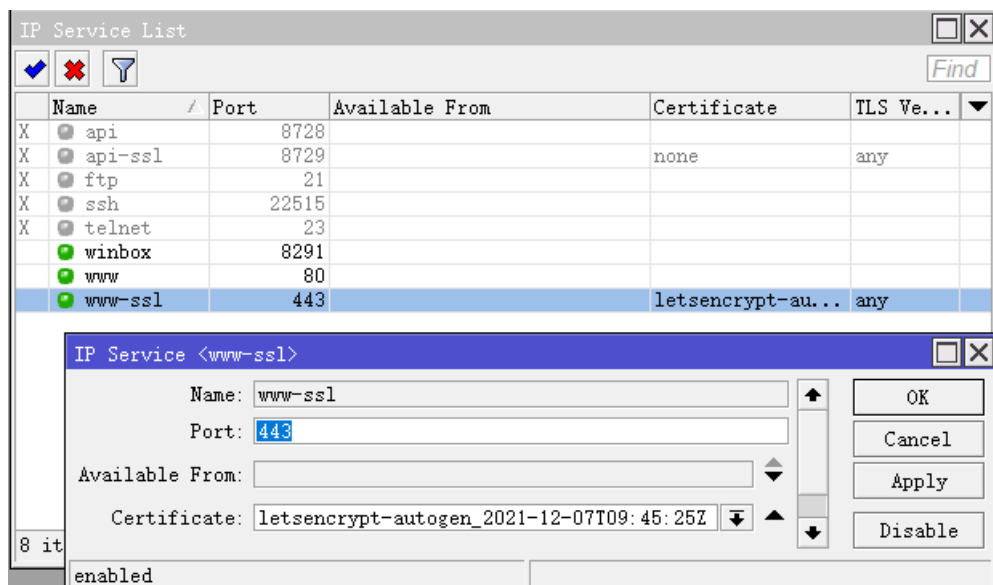
例如，使用/ip cloud 的 ddns 域名，并通过命令生成证书。特别提醒：需确保路由器互联网连接正常，并在/ip service 开启路由器的 TCP/80 和 443 端口（公网 IP 的 80 端口也需通畅，家宽和专线都是禁止了 80 和 443 端口，专线可以申请开通）

```
[yus@MikroTik] /certificate> enable-ssl-certificate dns-name=3635aexxxx.sn.mynetname.net
progress: [success] ssl certificate updated
```

提示：progress success，表示创建成功，在 certificates 下可以看到生成的证书



RouterOS 会把生成的 lets encrypt 证书自动添加到 www-ssl 服务下:



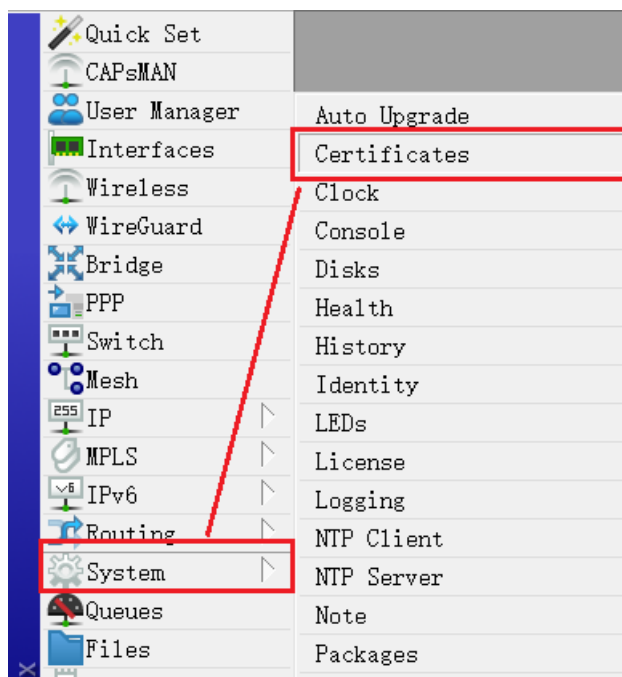
这样在浏览器输入 <https://3635aexxxx.sn.mynetname.net>, 访问 RouterOS 路由器的 https 服务。

7.3 创建 SSTP 证书

两台 RouterOS 之间建立 SSTP 隧道, 可以不要求证书认证, 但对于 Windows 系统或者其他平台的客户端的 SSTP 隧道是需要证书, 这里演示在 RouterOS 的 certificate 下如何创建 SSTP 证书。

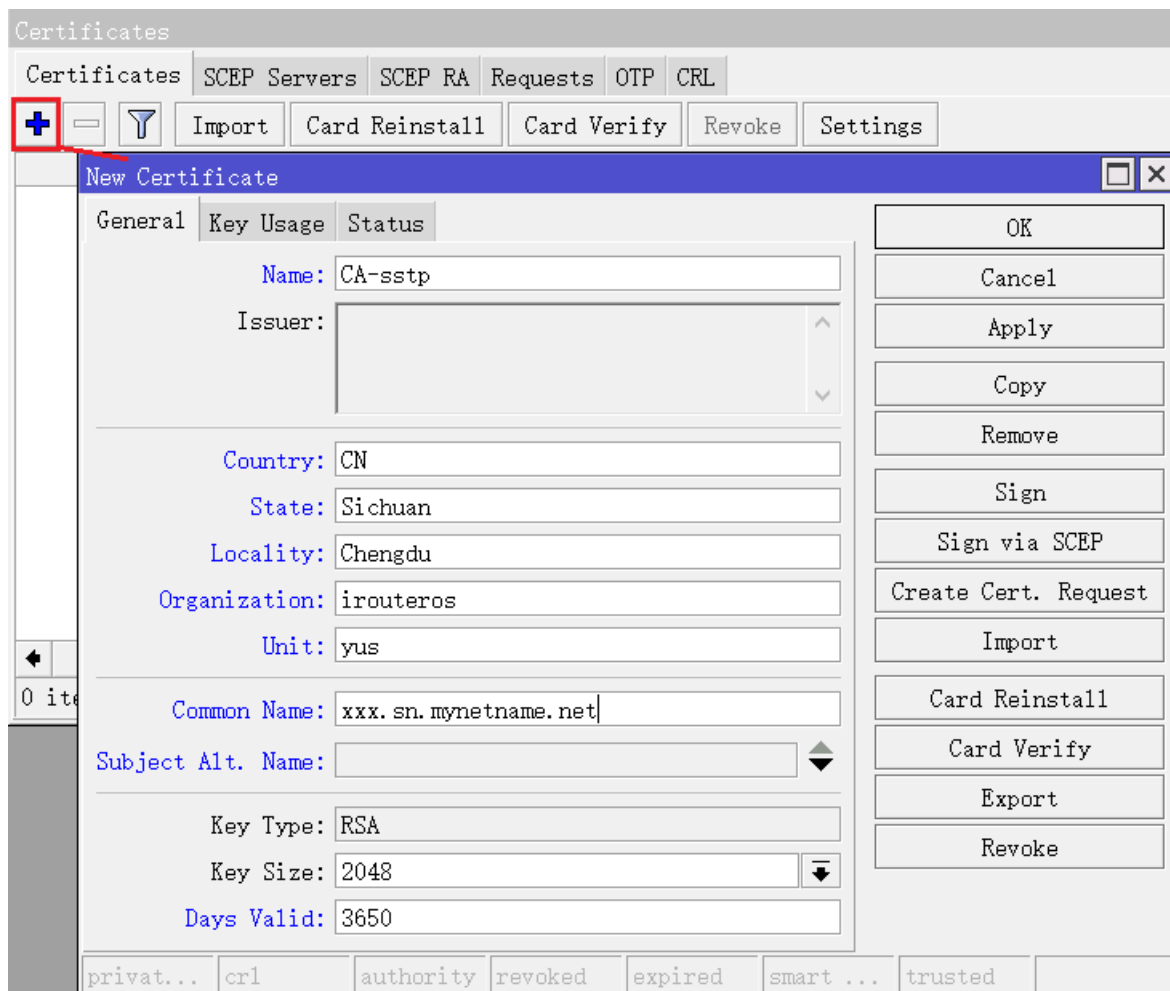
生成证书

在 winbox 中配置，首先进入 certificates 目录

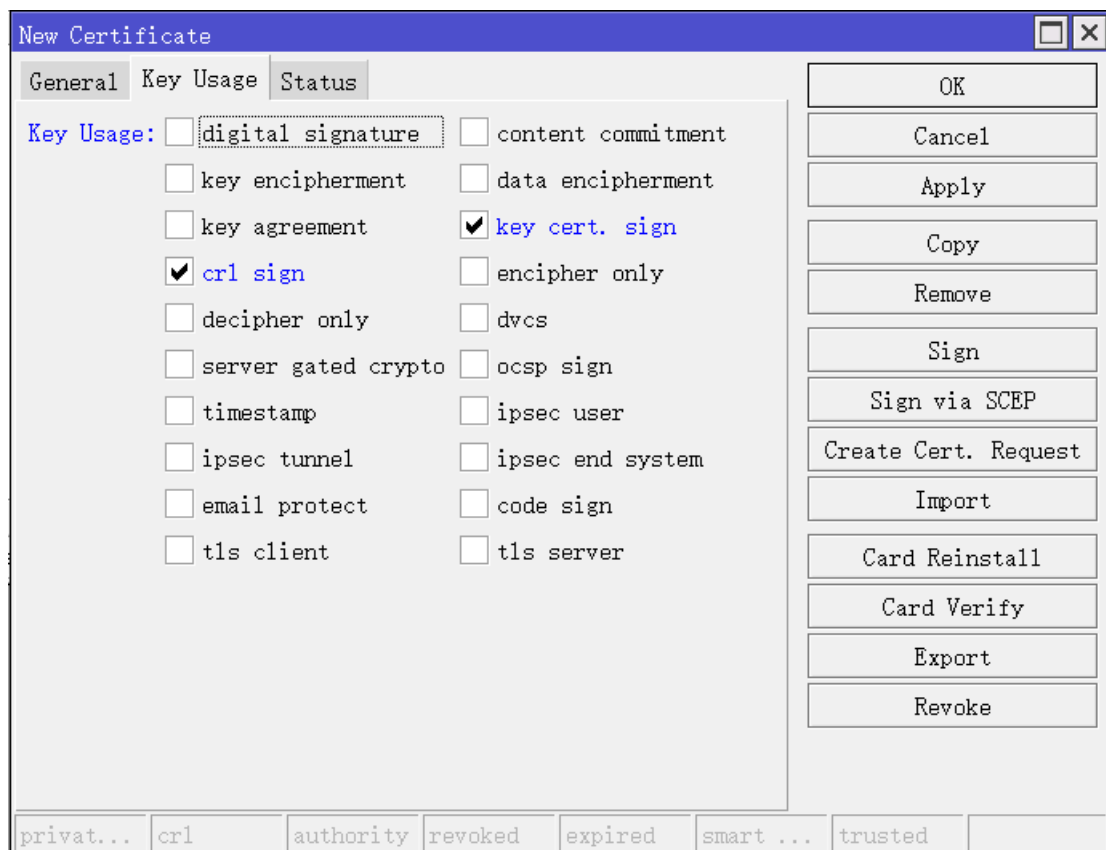


新建证书，取名 CA-sstp，填写以下相关信息，Common Name 此处填写服务器地址，例如 RouterOS 设备支持 cloud 服务，从 Mikrotik 官方分配到一个 xxx.sn.mynetname.net 的域名，可以用该域名作为 SSTP 的服务器地址，当然如果你的 IP 地址是固定的，也使用 IP 地址作为 Common Name。但我建议是使用域名更加谨慎，因为 IP 可能会变，但域名一般不会，并能修改解析 IP！

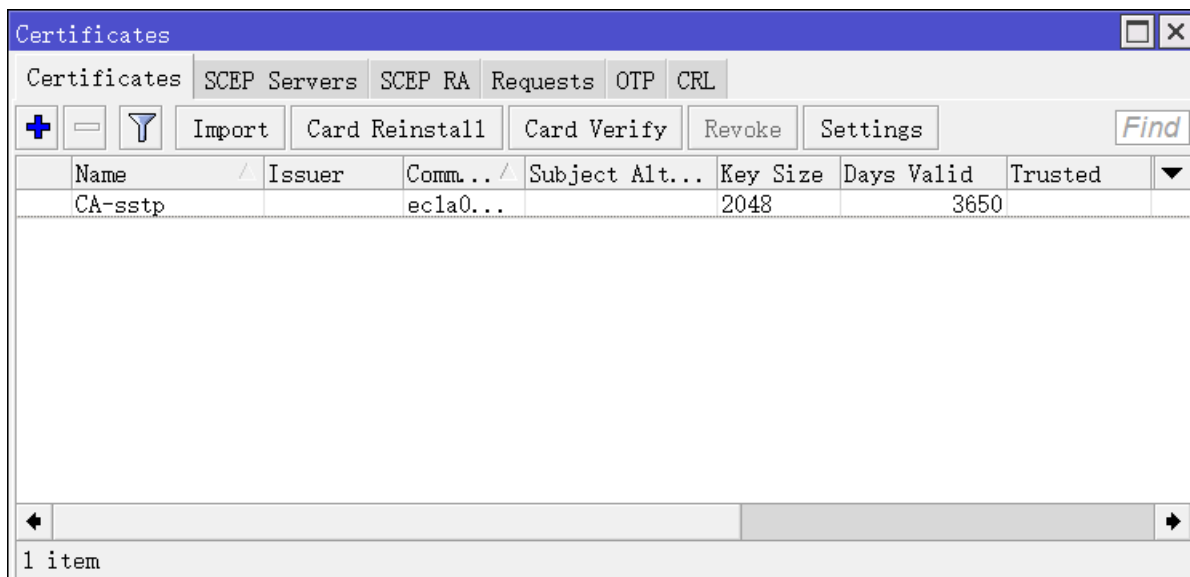
在 Days Valid 参数是证书的有效期，这里选择 3650 天，也就是 10 年有效。



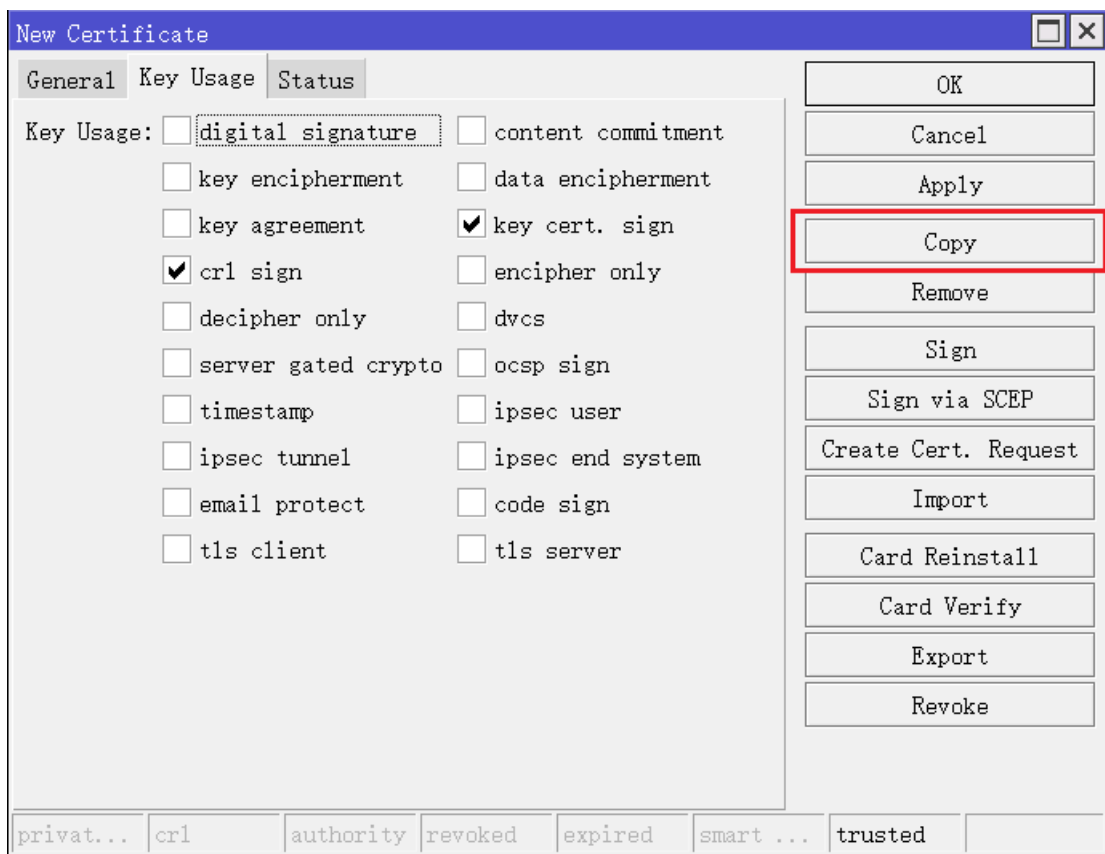
在 Key usage 下选择 key cert. sign 和 crl sign



点击 ok 生成 CA 证书



下一步生成 server 服务器证书，我们可以在原理的 CA 证书上，点 copy，



Copy 后生成新的规则，修改 Key Usage 参数，如下：

New Certificate

General Key Usage Status

Key Usage: ☒ digital signature ☐ content commitment
☒ key encipherment ☐ data encipherment
☐ key agreement ☐ key cert. sign
☐ crl sign ☐ encipher only
☐ decipher only ☐ dvcs
☐ server gated crypto ☐ ocsf sign
☐ timestamp ☐ ipsec user
☐ ipsec tunnel ☐ ipsec end system
☐ email protect ☐ code sign
☐ tls client ☒ tls server

OK
Cancel
Apply
Copy
Remove
Sign
Sign via SCEP
Create Cert. Request
Import
Card Reinstall
Card Verify
Export
Revoke

privat... crl authority revoked expired smart ... trusted

并修改名称为 **server-sstp**:

Certificate <cert1>

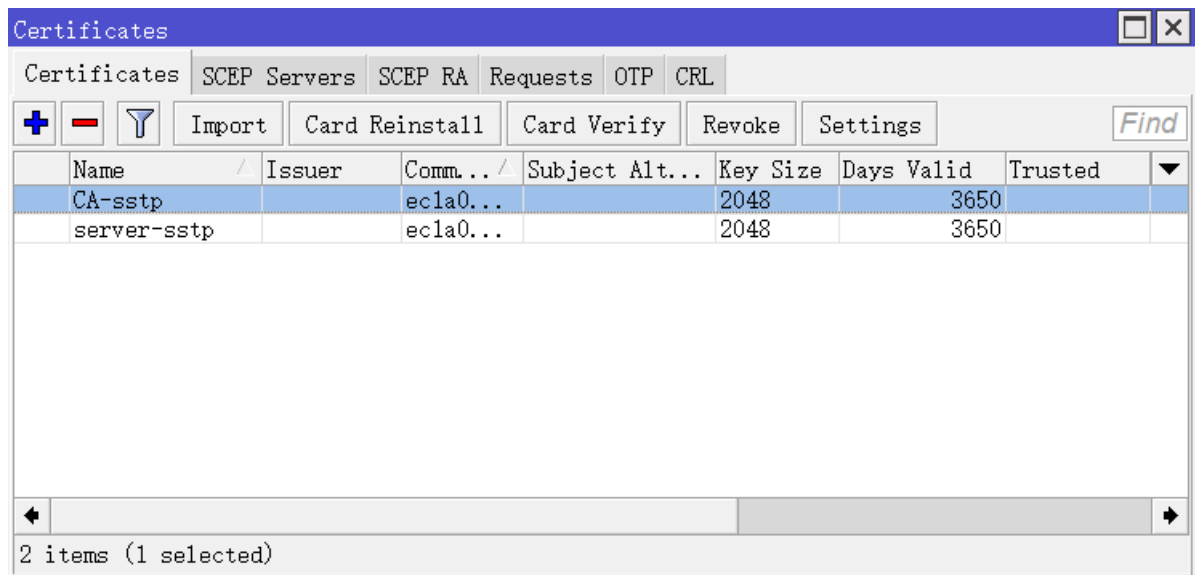
General Key Usage Status

Name: server-sstp
Issuer:
Country: CN
State: Sichuan
Locality: Chengdu
Organization: irouteros
Unit: yus
Common Name: xxx.sn.mynetname.net
Subject Alt. Name:
Key Type: RSA
Key Size: 2048
Days Valid: 3650

OK
Cancel
Apply
Copy
Remove
Sign
Sign via SCEP
Create Cert. Request
Import
Card Reinstall
Card Verify
Export
Revoke

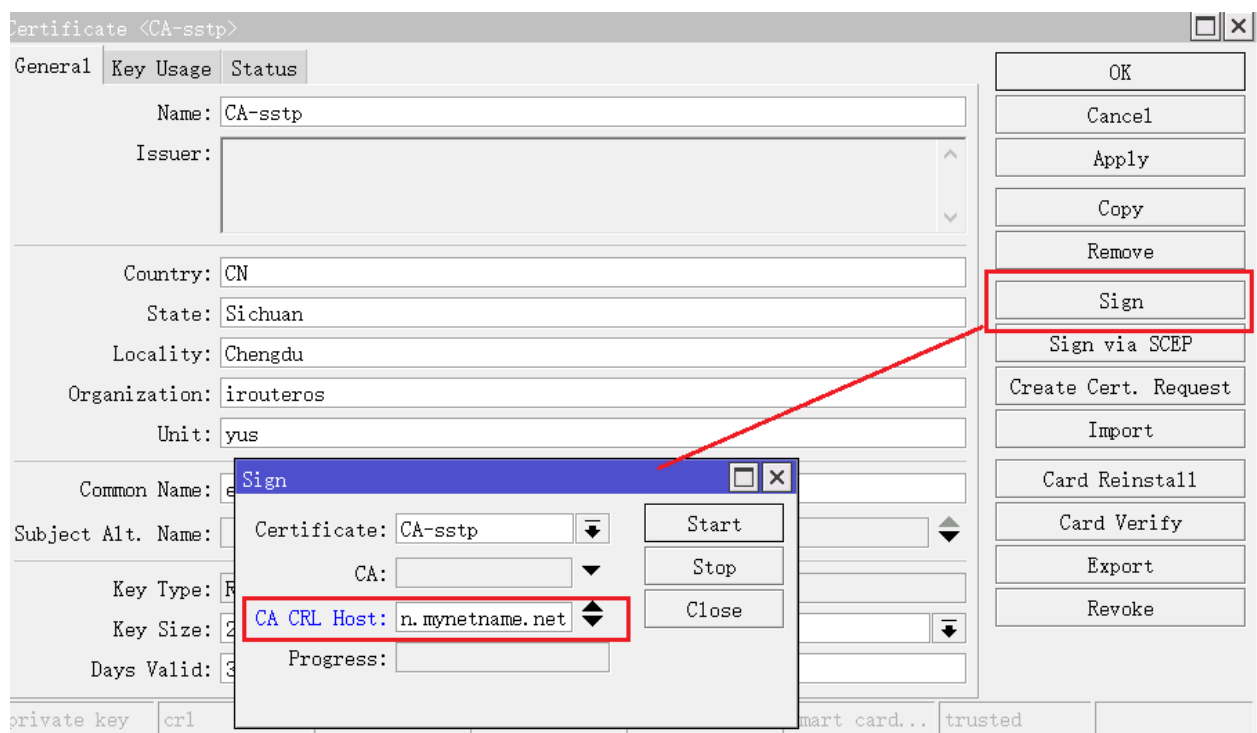
privat... crl authority revoked expired smart ... trusted

点击 **ok** 后，会生成。

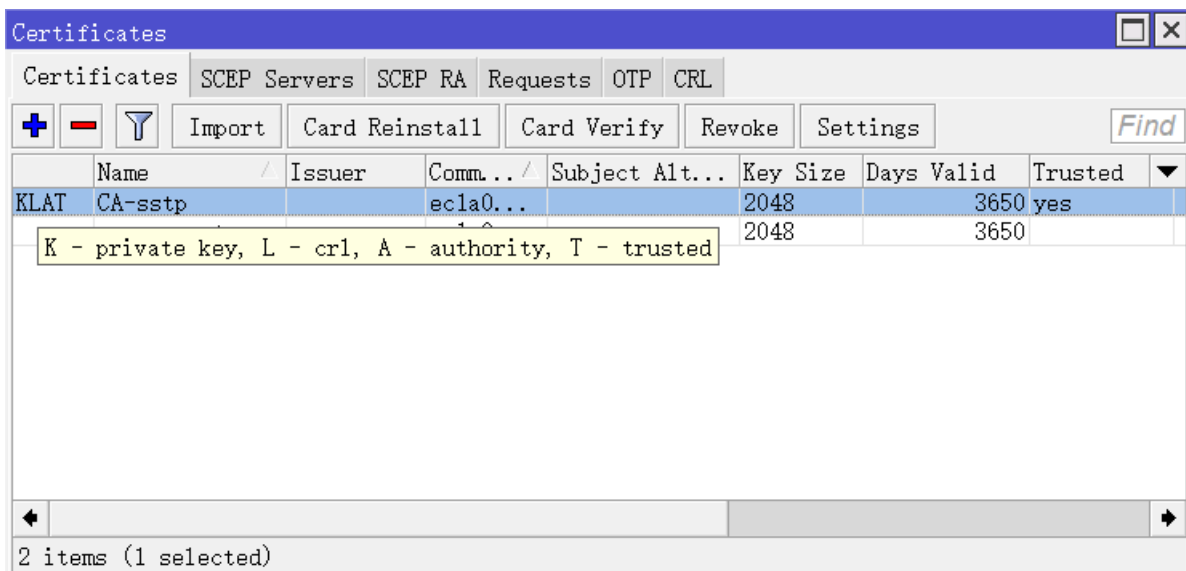


另外一份客户端证书，可以不用生成，windows 电脑只需要一份 CA 证书即可，Server 证书用于 RouterOS。

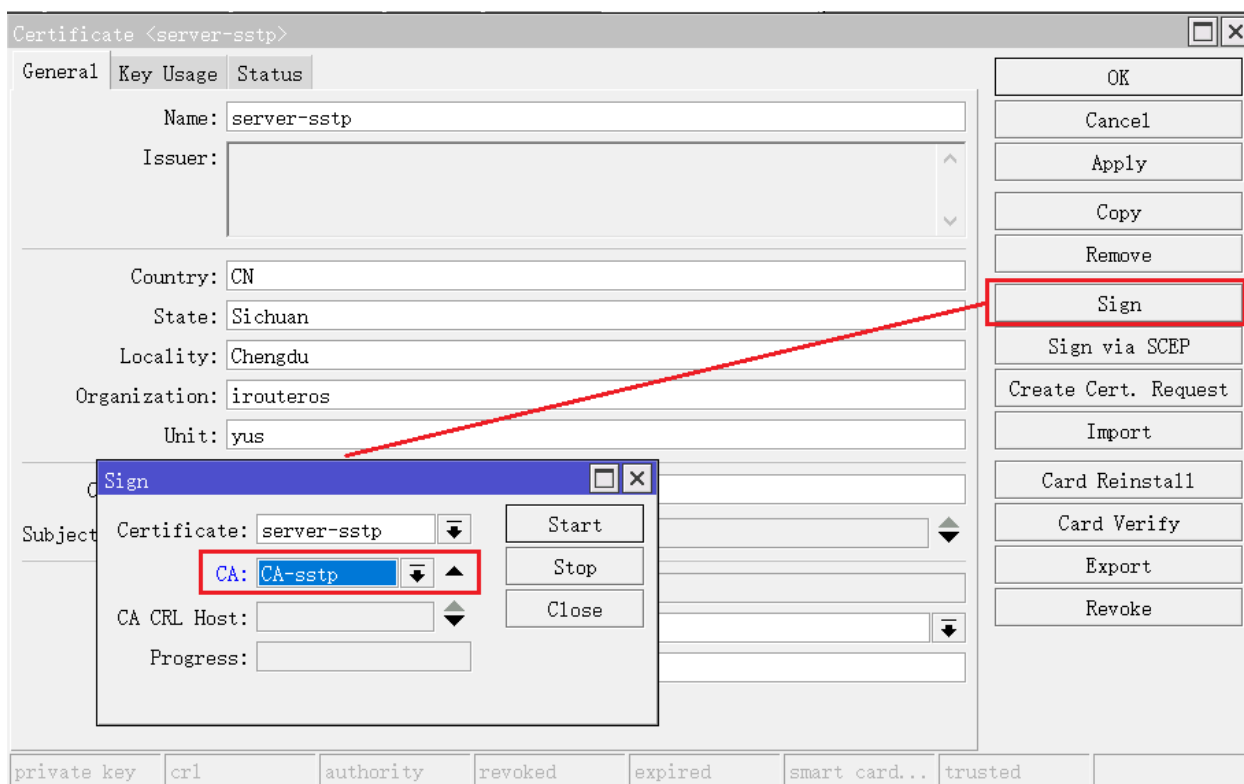
下面一步就是签发证书，打开 CA-sstp 证书，选择 Sign，CA_CRL_Host 输入 xxx.sn.mynetname.net 或 IP 地址，填写好后，点击 Start 开始签发。



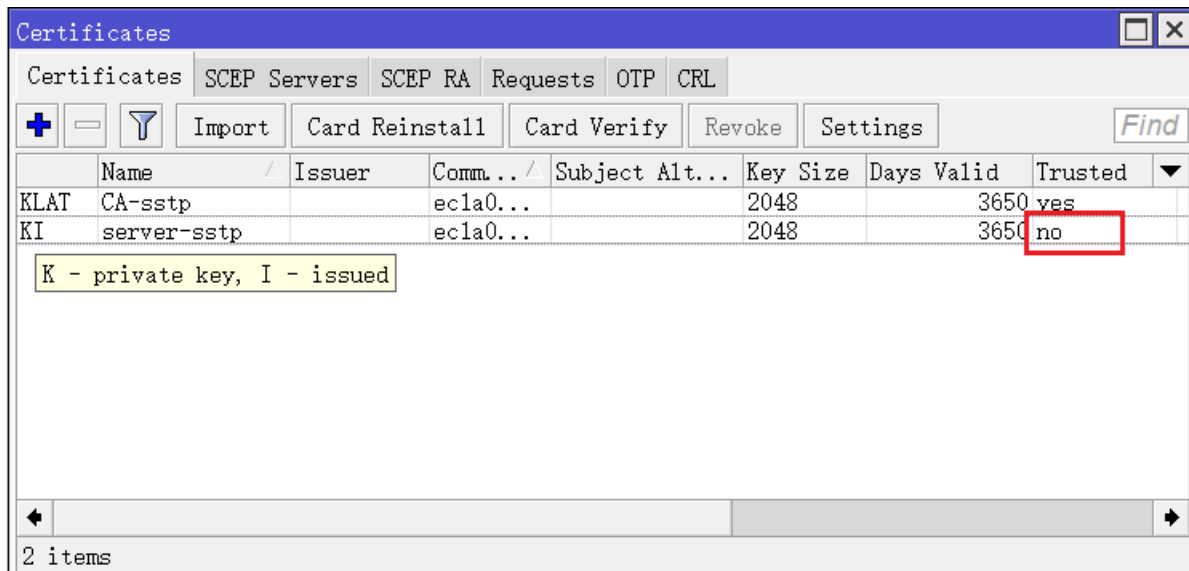
生成结果如下，在 CA-sstp 证书的前缀储小 KLAT，代表签发成功：



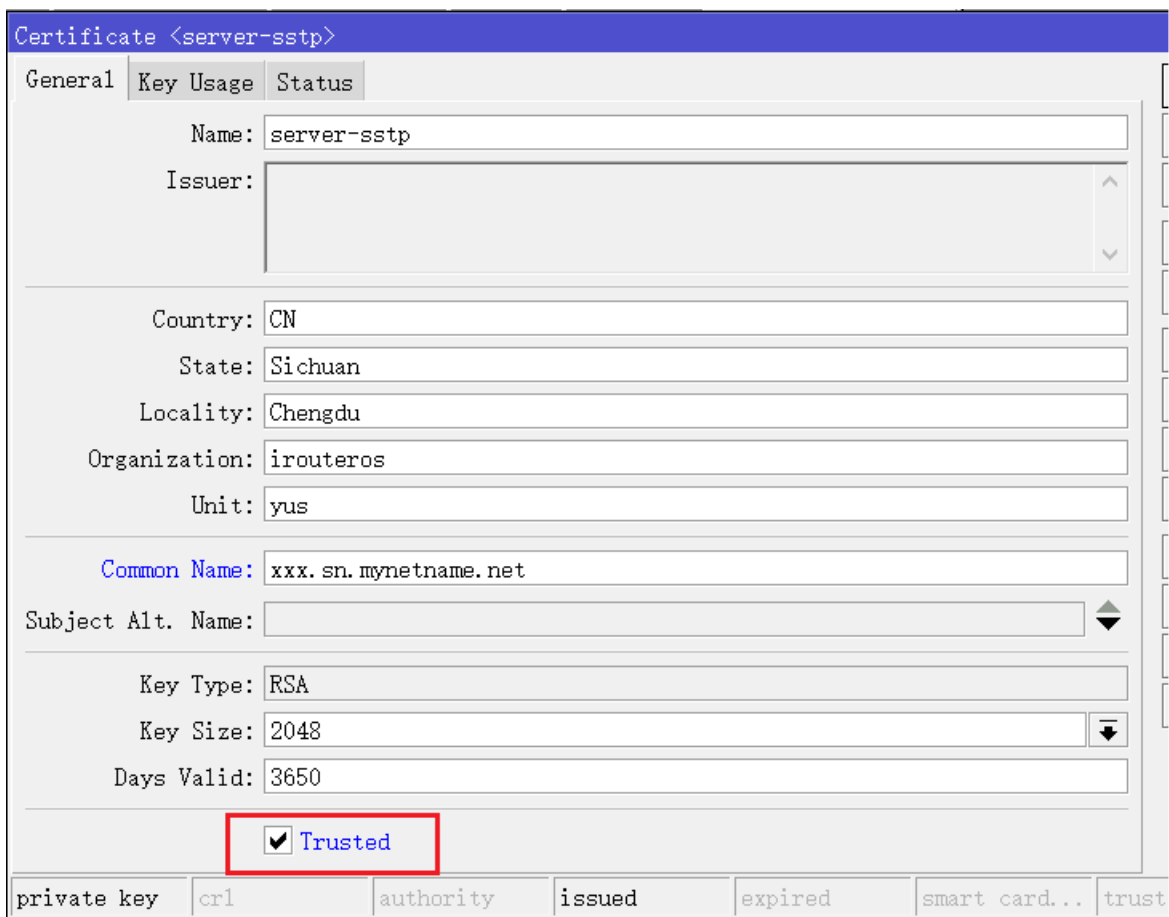
然后签发 Server-sstp 证书，在 CA 项选择 CA-sstp 证书签发



证书签发完成后，前缀储小 KI，但 trusted 是 no

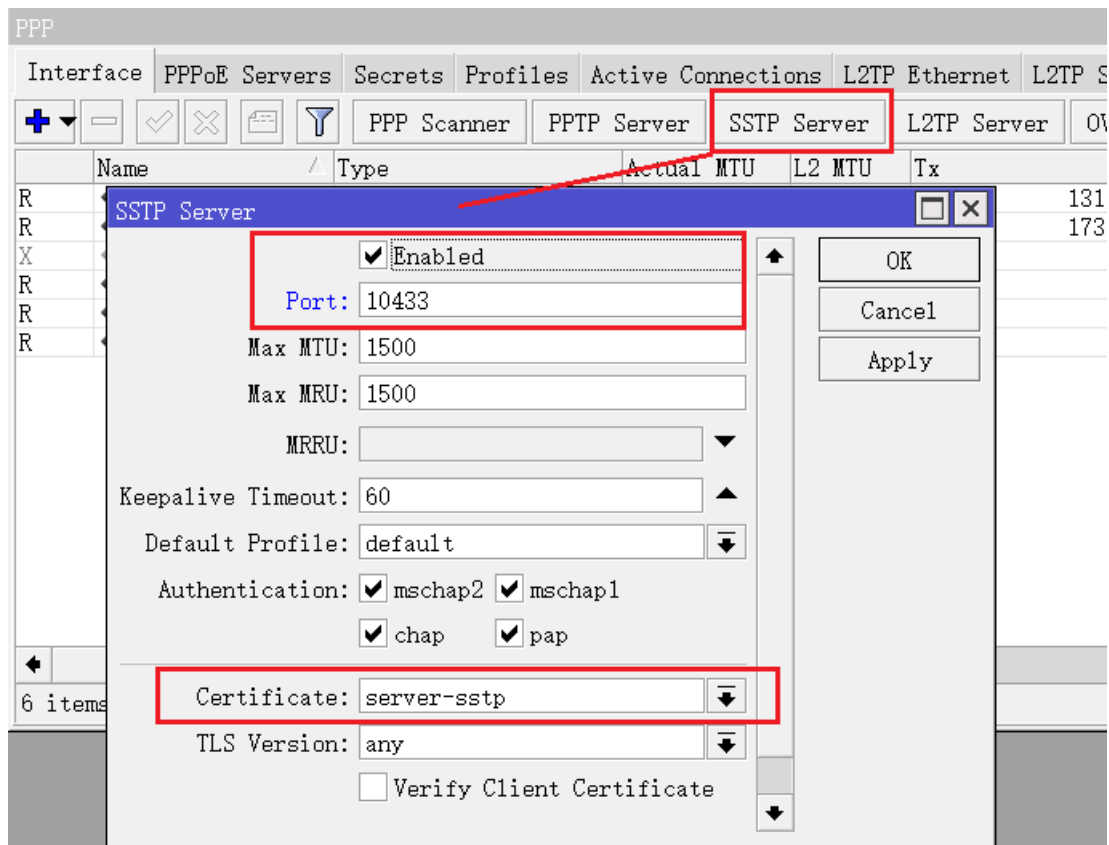


打开 server-sstp 证书，选择 trusted



配置 SSTP-server 证书

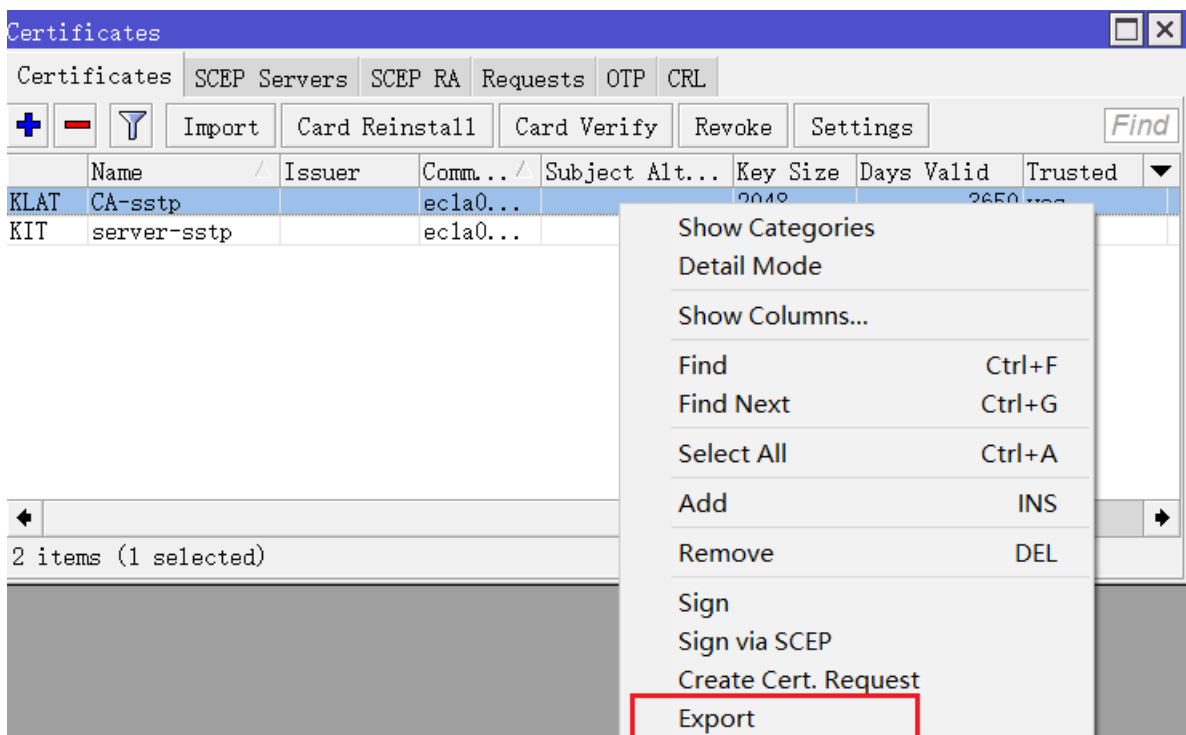
以上证书生成完成，在本地 RouterOS 进入 PPP 目录，设置 SSTP-Server，将 server-sstp 证书配置到 SSTP server，因为运营商限制了 443 端口，将 Port 修改为 10443，并勾选 Enabled 启用服务。



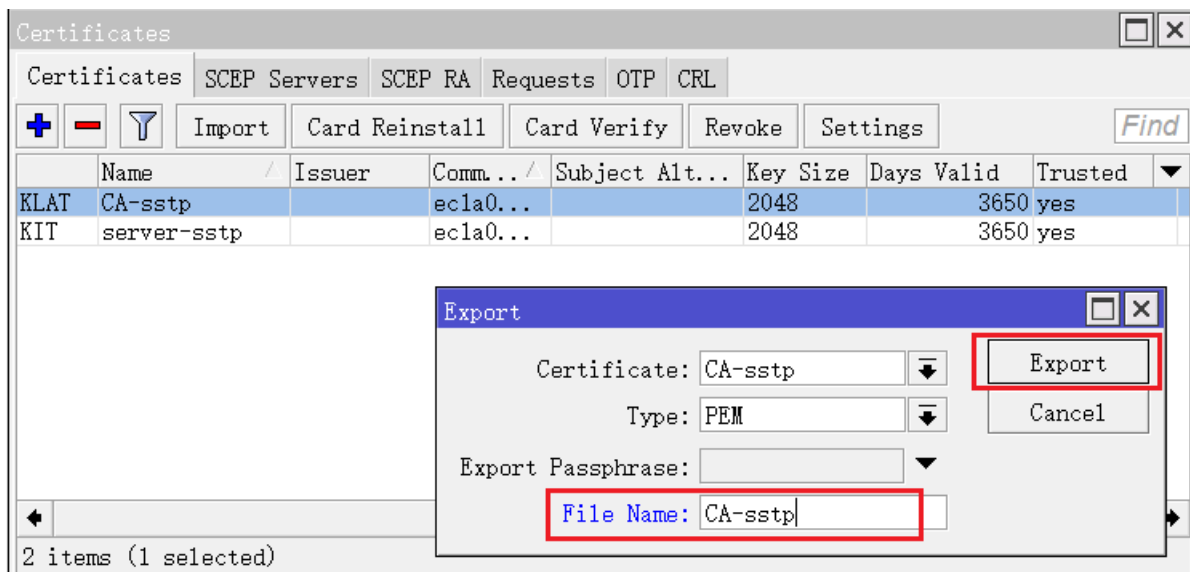
SSTP 服务器如何创建账号和 IP 地址分配，这里就省略，这方面的配置请参考 RouterOS 入门到精通 v6 的 SSTP 章节。

导出证书

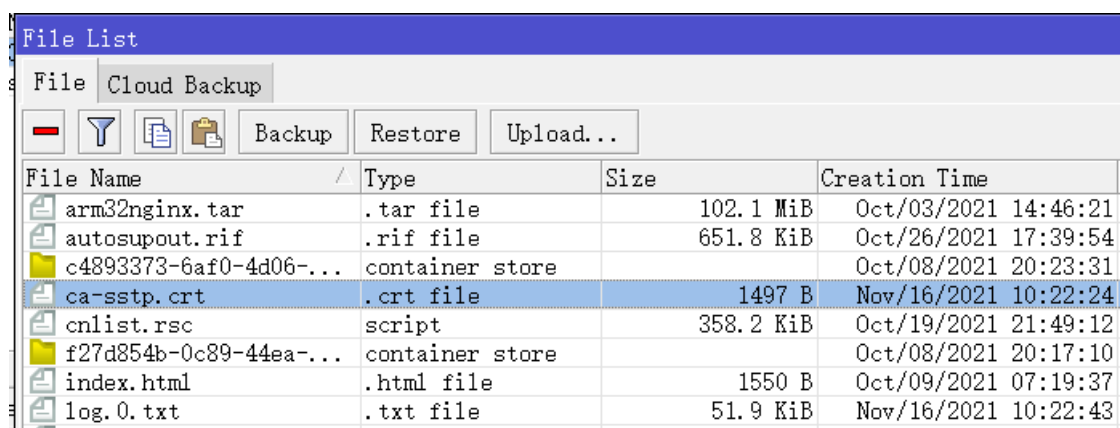
在 Certificates 下，选择 CA-sstp 证书，右键后会出现 Export



取名 CA-sstp, 然后点 Export

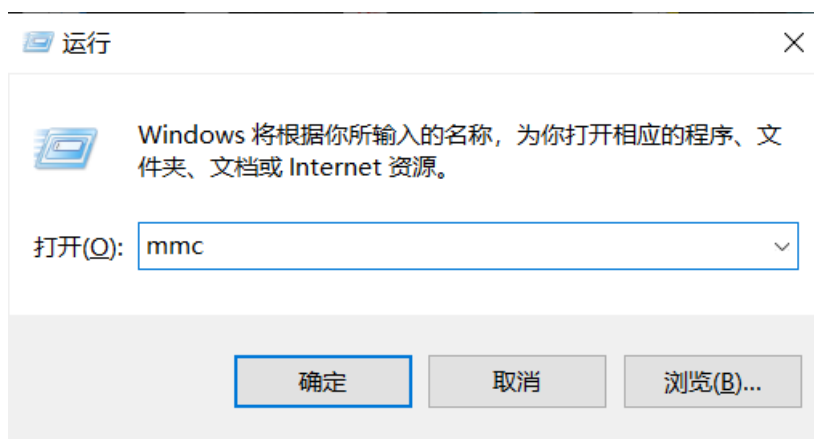


进入/file 目录下可以找到

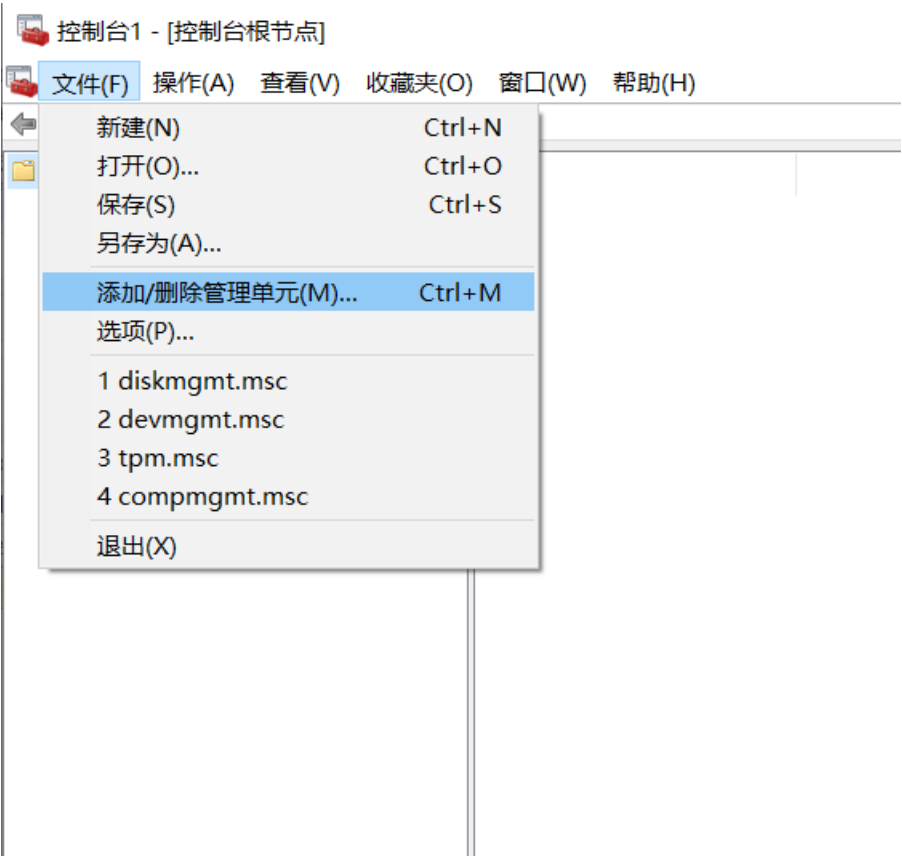


导入 windows

接下来是将 CA-sstp.crt 从 File 上下载到 windows10 电脑, 在 windows10 电脑上输入 win+R, 并在运行里输入 mmc 回车



打开后，点击文件选择“添加/删除管理单元”



选择“证书”，选择计算机账户号，选完成后，再点确定

添加或删除管理单元

你可以在计算机上为此控制台选择可用的管理单元并配置所选的一组管理单元。对于可扩展的管理单元，你可以配置要

可用的管理单元(S):

管理单元	供应商
磁盘管理	Microsoft 和 VE...
打印管理	Microsoft Corp...
服务	Microsoft Corp...
高级安全 Windows Defe...	Microsoft Corp...
共享文件夹	Microsoft Corp...
计算机管理	Microsoft Corp...
任务计划程序	Microsoft Corp...
设备管理器	Microsoft Corp...
事件查看器	Microsoft Corp...
授权管理器	Microsoft Corp...
文件夹	Microsoft Corp...
性能监视器	Microsoft Corp...
证书	Microsoft Corp...
组策略对象编辑器	Microsoft Corp...
组件服务	Microsoft Corp...

所选管理单元(E):

控制台根节点

证书管理单元

该管理单元将始终为下列帐户管理证书:

☐ 我的用户帐户(M)

☐ 服务帐户(S)

☒ 计算机帐户(C)

描述:

证书管理单元允许你浏览自己的、一个服务的或一台计算

出现了证书-当前用户，进入受信任的根证书签发机构

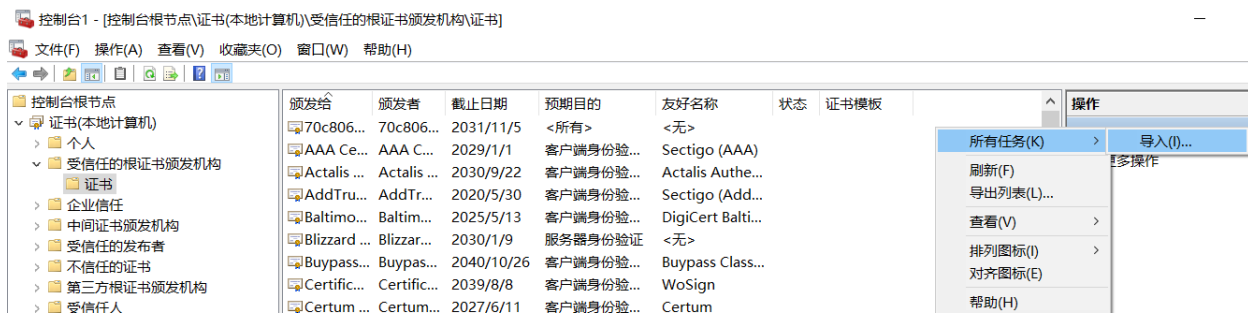
控制台1 - [控制台根节点\证书 - 当前用户]

文件(F) 操作(A) 查看(V) 收藏夹(O) 窗口(W) 帮助(H)

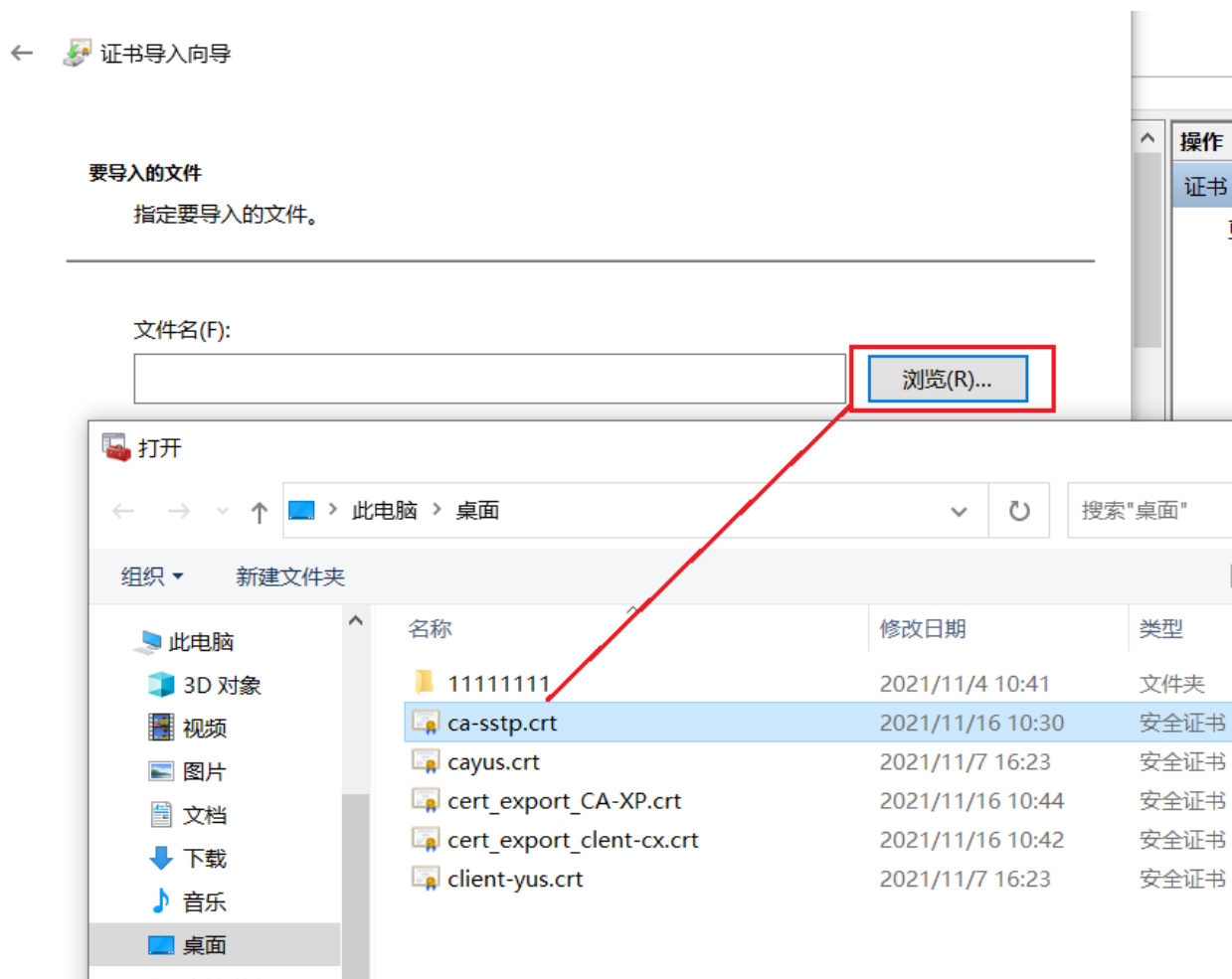
控制台根节点
> 证书 - 当前用户

逻辑存储名
个人
受信任的根证书颁发机构
企业信任
中间证书颁发机构
Active Directory 用户对象
受信任的发布者
不信任的证书
第三方根证书颁发机构
受信任人
客户端身份验证颁发者
其他人
Local NonRemovable Certificates
MSIEHistoryJournal
证书注册申请
智能卡受信任的根


进入受信任的根证书签发机构下的证书，在空白处右键，选择所有任务->导入



按照引导下一步操作后，选择浏览找到 CA-sstp.crt 证书导入



跟着下一步，继续

←  证书导入向导

证书存储

证书存储是保存证书的系统区域。

Windows 可以自动选择证书存储，你也可以为证书指定一个位置。

☐ 根据证书类型，自动选择证书存储(U)

☒ 将所有的证书都放入下列存储(P)

证书存储:





















受信任的根证书颁发机构

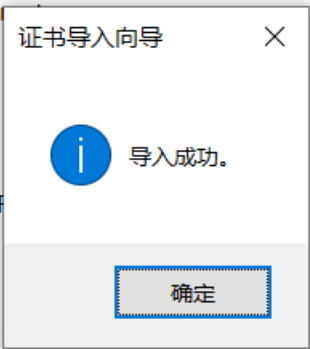
浏览(R)...

下一页(N)

取消

下一步后，提示导入成功：

颁发给	颁发者	截止日期	预期目的	友好名称	状态	证书模板
 70c806...	70c806...	2031/11/5	<所有>	<无>		
 AAA Ce...	AAA C...	2029/1/1	客户端身份验...	Sectigo (AAA)		
 Actalis ...	Actalis ...	2030/9/22	客户端身份验...	Actalis Authe...		
 AddTru...	AddTr...	2020/5/30	客户端身份验...	Sectigo (Add...		
 Baltimo...	Baltim...	2025/5/13	客户端身份验...	DigiCert Balti...		
 Blizzard ...	Blizzar...	2030/1/9	服务器身份验证	<无>		
 Buypass...	Buypas...	2040/10/26	客户端身份验...	Buypass Class...		
 Certific...	Certific...	2039/8/8	客户端身份验...	WoSign		
 Certum ...	Certum...	2027/6/11	客户端身份验...	Certum		
 Certum ...	Certum...	2029/12/31	客户端身份验...	Certum T...		
 CFCA A...	CFCA A...	2035/9/28	<所有>	<无>		
 CFCA A...	CFCA A...	2043/3/8	<所有>	<无>		
 CFCA A...	CFCA A...	2043/3/8	<所有>	<无>		
 CFCA C...	CFCA C...	2041/5/12	<所有>	<无>		
 CFCA E...	CFCA E...	2029/12/31	客户端身份验...	CFCA EV F...		
 CFCA E...	CFCA E...	2029/12/31	<所有>	<无>		
 CFCA G...	CFCA G...	2042/8/21	<所有>	<无>		
 CFCA Id...	CFCA I...	2040/6/30	<所有>	<无>		
 CFCA T...	CFCA T...	2032/8/29	<所有>	<无>		
 CFCA T...	CFCA T...	2032/8/29	<所有>	<无>		



在网络里面创建 VPN，SSTP 端口修改为 10443，服务器连接为 xxx.sn.mynetname.net:10443，然后输入账号和密码

编辑 VPN 连接

下次连接时，将应用这些更改。

连接名称

sstp

服务器名称或地址

xxx.sn.mynetname.net:10443

VPN 类型

安全套接字隧道协议(SSTP) ▾

登录信息的类型

用户名和密码 ▾

用户名(可选)

yus

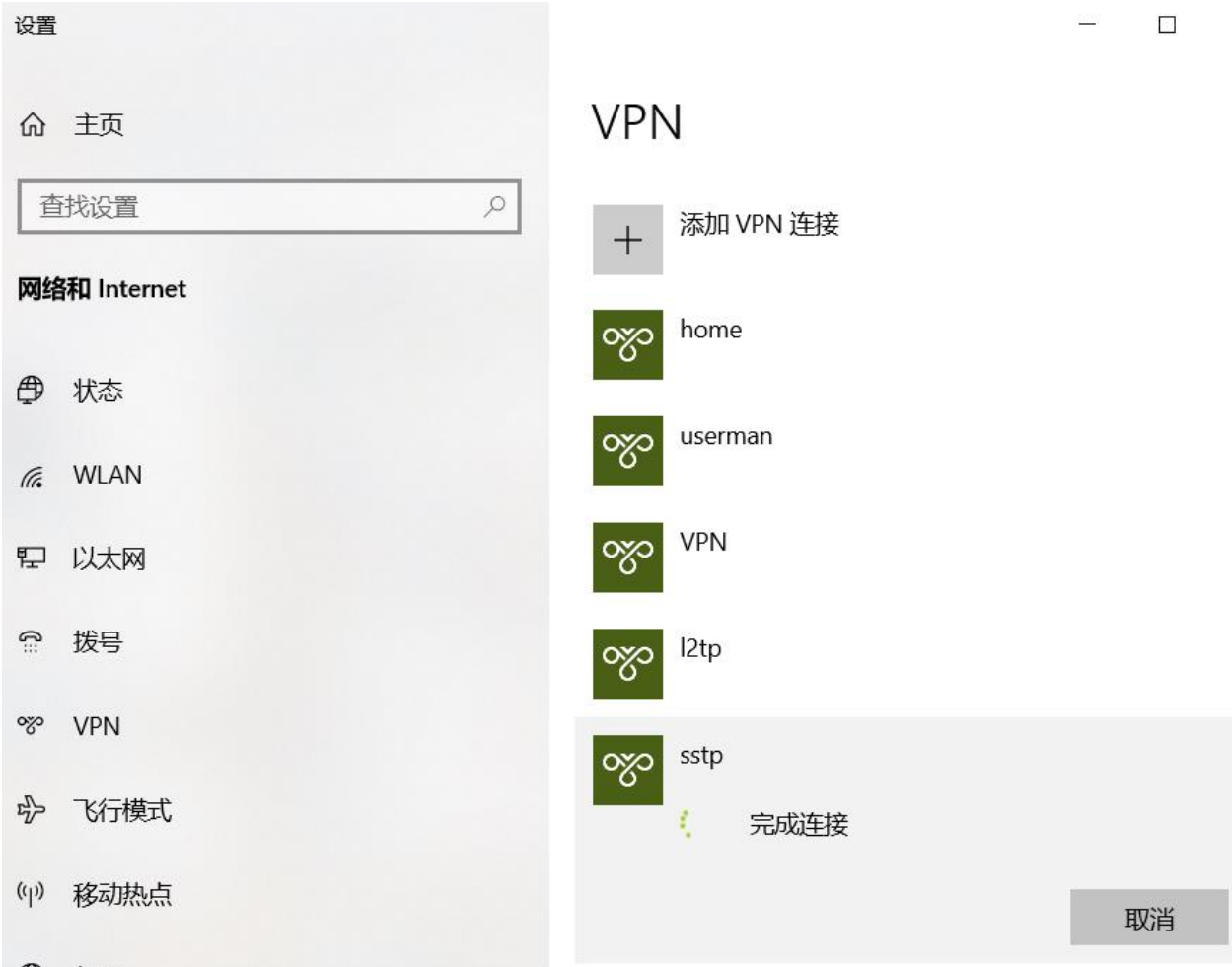
密码(可选)

●●● ▮

保存

取消

然后进行连接，测试成功



如果 SSTP 登录出现：“此句柄相关联的操作锁定现在与其他句柄相关联”，导致这样的情况大概率是 SSTP 服务器设置中勾选了“**verify Client Certificate**”，需要取消该选项。

第八章 Dot1x（基于以太网 802.1x）

Dot1x 是在 RouterOS 中实现 IEEE 802.1x 标准。802.1x 协议是基于 Client/Server 的访问控制和认证协议。它可以限制未经授权的用户/设备通过接入端口(access port)访问 LAN/WLAN。802.1x 只允许 EAPOL（EAP over LAN 基于局域网的扩展认证协议）数据通过设备连接的交换机端口；认证通过以后，正常的可以顺利地通过以太网端口。

IEEE802.1x 协议为二层协议，不需要到达三层，对设备的整体性能要求不高，可以有效降低建网成本，802.1x 网络组成包含：一个请求客户端、一个认证服务端和一个 RADIUS 服务器，当前 RouterOS 支持客户端和服务端，支持 EAP 模式，包括 EAP-TLS, EAP-TTLS, EAP-MSCHAPv2 和 PEAPv0/EAP-MSCHAPv2，同时配合基于 RouterOS v7 的 User Manager v5（RADIUS 服务器），实现一整套组网方案。

需要提醒的是，Dot1x 不像 PPPoE、PPTP 和 hotspot 等应用可以在 RouterOS 本地创建账号，Dot1x 的账号管理需要连接 RADIUS 服务器，因此在用 Dot1x 服务器时，用户账号和密码需要向 RADIUS 服务器请求（RouterOS v7 版本的 User Manager v5 版本支持 Dot1x 认证）。

操作路径：/interface dot1x

提示：此功能不支持 SMIPS 的 RB 设备 (hAP lite, hAP lite TC 和 hAP mini)。

8.1 客户端

属性

属性	描述
anon-identity (字符串, 默认:)	匿名的外层 EAP 认证的身份。仅与 eap-ttls 和 eap-peap 方法一起使用。如果没有设置，将使用 identity 值用于外层 EAP 认证。
client-certificate (string; Default:)	System/Certificates 中列出的证书名称。当使用 eap-tls 方法时要求调用证书
eap-methods (eap-tls eap-ttls eap-peap eap-mschapv2; Default:)	用于身份验证的 EAP 方法列表
identity (string; Default:)	客户端请求者的 EAP 身份验证，即账号
interface (string; 默认:)	选择运行客户端的网络接口
password (string; 默认:)	设置密码

只读属性

属性	描述
status (authenticated authenticating disabled)	状态说明： <ul style="list-style-type: none"> authenticated – 客户端认证成功；

- `authenticated without server` – 允许访问端口但无法连接服务器；
- `authenticating` – 连接到达服务器，正在进行身份验证过程；
- `connecting` – 认证过程初始化阶段；
- `disabled` – 客户端被禁用；
- `error` – 发生内部错误；
- `interface is down` – 父级端口未连接运行；
- `rejected` – 服务器拒绝认证

8.2 服务器

首先 `dot1x` 服务器启用在指定网络端口后，默认将阻断除用于认证的 `EAPOL` 报文外的所有数据通信。客户端认证成功后，接口将接收该端口上所有的数据流量。如果接口连接到一台交换机，交换机下连接多个主机，当至少有一个客户端通过身份验证时，其他所有主机都会允许通过。在认证失败的情况下，可以分配一个指定 `VLAN ID`，分配认证失败的主机加入的这个 `VLAN` 网络中，例如：`Guest` 来宾终端加入到一个 `Guest VLAN`。

操作路径：/interface dot1x server

当在 `Bridge` 端口上创建 `dot1x` 服务器时，`Bridge` 应该运行(R/M)STP，否则客户端发送的 `EAP` 报文将不能被正确接收。缺省情况下，使用 `protocol-mode=rstp` 创建网桥接口。如果 `Bridge` 端口不发送 `bpdu` 或忽略接收到的 `bpdu`，则可以设置该 `Bridge` 端口的 `edge=yes`。

属性	描述
<code>accounting</code> (<i>yes / no</i> ; 默认: <code>yes</code>)	是否发送 <code>RADIUS accounting</code> 数据到认证服务器
<code>auth-timeout</code> (<i>time</i> ; 默认: <code>1m</code>)	用于 <code>EAP</code> 身份验证的总时间
<code>auth-types</code> (<i>dot1x / mac-auth</i> ; 默认: <code>dot1x</code>)	指定接口的身份验证类型。当两种方式同时被选择，服务器将优先选择 <code>dot1x</code> 认证，只有经过 3 个 <code>retrans-timeout</code> 周期后，认证类型才会回到 <code>mac-auth</code> 。为了让 <code>MAC-auth</code> 认证类型工作，服务器接口应该至少接收一个包含客户端设备源 <code>MAC</code> 地址的帧。
<code>guest-vlan-id</code> (<i>integer: 1..4094</i> ; 默认: <code>!guest-vlan-id</code>)	终端设备不支持 <code>Dot1x</code> 认证，且未配置 <code>Mac-auth</code> 认证，并在 3 个 <code>retrans-timeout</code> 周期后，被分配到 <code>Guest VLAN</code> 。当 <code>dot1x</code> 的客户端正常启用后，并成功进行了 <code>dot1x</code> 的认证，该端口将从 <code>Guest VLAN</code> 中删除。该设置仅在 <code>RouterOS 7.2</code> 版本后生效，在启用桥 <code>vlan</code> 过滤时生效。默认情况下 <code>Guest-VLAN</code> 功能是关闭。
<code>interface</code> (<i>string</i> ; 默认:)	选择运行客户端的网络接口

属性	描述
interim-update (<i>time</i> ; 默认: 0s)	RADIUS 定时更新消息间隔
mac-auth-mode (<i>mac-as-username</i> / <i>mac-as-username-and-password</i> ; 默认: mac-as-username)	当启用 MAC 认证, 定义 RADIUS 属性选择认证账号的方式 MAC 作为用户名, 还是同时作为用户和密码
radius-mac-format (<i>XX-XX-XX-XX-XX-XX</i> / <i>XX:XX:XX:XX:XX:XX</i> / <i>XXXXXXXXXXXX</i> / <i>XX-XX-XX-XX-XX-XX</i> / <i>XX:XX:XX:XX:XX:XX</i> / <i>XXXXXXXXXXXX</i> ; 默认: XX:XX:XX:XX:XX:XX)	定义使用 MAC 身份验证时客户端的 MAC 地址在格式方式
reauth-timeout (<i>time</i> ; 默认: !reauth-timeout)	启用服务器端口重新认证功能。当启用 Dot1x 身份验证类型时, 服务器将尝试通过向客户端发送 EAP-Request Identity 来重新验证客户端。当启用 MAC -auth 认证类型时, 服务器将尝试使用 RADIUS 服务器的最后一次 MAC 地址重新对客户端进行认证。该设置仅在 RouterOS 7.2 版本后可用。默认情况下, re-authentication 功能处于关闭状态。
reject-vlan-id (<i>integer: 1..4094</i> ; 默认: !reject-vlan-id)	当 RADIUS 服务器响应认证类型为 Access-Reject 信息, 即认证失败时分配的 VLAN。如果 RADIUS 服务器根本没有响应, 此属性将不适用, 同时客户端身份验证超时, 该服务也可用。在启用桥 vlan-filtering 时生效。默认情况下, reject VLAN 功能处于关闭状态。
retrans-timeout (<i>time</i> ; 默认: 30s)	如果没有收到客户端请求的响应, 将重新发生验证信息的时间间隔。
server-fail-vlan-id (<i>integer: 1..4094</i> ; 默认: !server-fail-vlan-id)	当 RADIUS 服务器无响应, 并出现请求超时后, 分配的 VLAN。该设置仅在 RouterOS 7.2 版本后生效, 在启用桥 vlan-filtering 时生效。默认情况下, server-fail VLAN 功能处于关闭状态。

操作路径: `/interface dot1x server active`

相关 Dot1x 服务接口的状态, 列在以下列表中

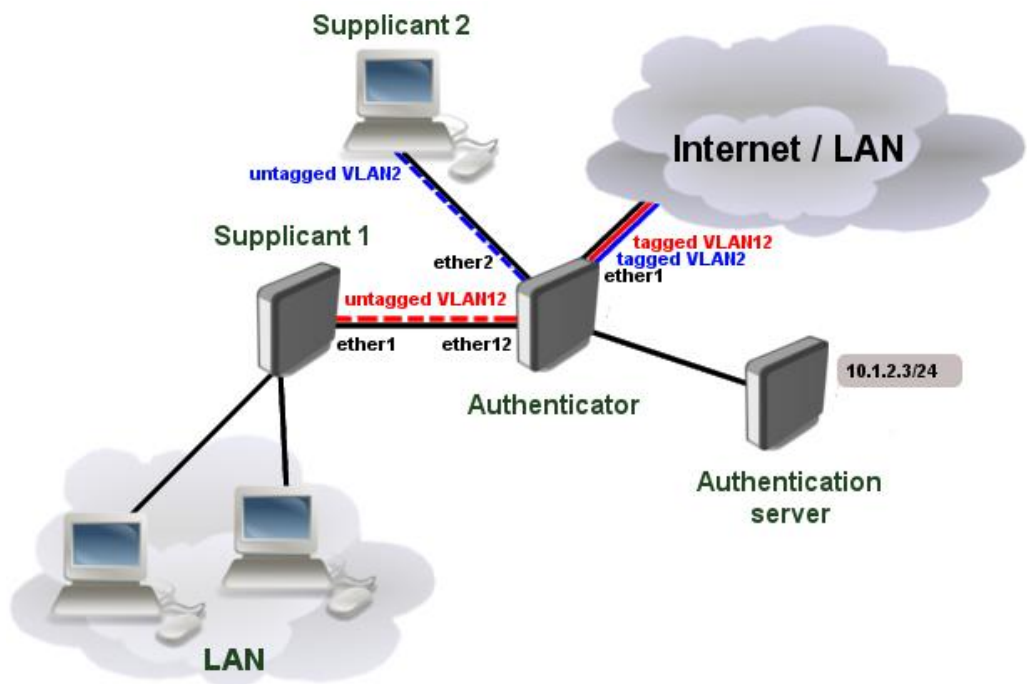
属性	描述
auth-info (<i>string</i>)	认证信息 Authentication information: dot1x dot1x (guest vlan) dot1x (reject vlan) dot1x (server fail vlan)

属性	描述
	mac-auth mac-auth (reject vlan) mac-auth (server fail vlan)
client-mac (mac-address)	客户端的 MAC 地址
interface (string)	网络接口名称
session-id (string)	唯一的 session 身份 ID
username (string)	客户端请求认证的用户名
vlan-id (string)	指定网络接口的 Untagged VLAN ID。Bridge 需启用 VLAN Filtering

状态属性

属性	描述
status (string)	网络接口的状态： <ul style="list-style-type: none">authorized – 访问网络接口被允许；iface-down – 网络接口未连接运行；rejected-holding – 访问被 RADIUS 服务器拒绝；un-authorized -访问网络接口被拒绝；

8.3 功能与实例



在该实例中，基于二层建立 802.1x 网络，Authenticator 作为 Dot1x 认证服务器，Authentication server 为 RADIUS 服务器，supplicant1 为 Dot1x 的 RouterOS 请求者，即客户端路由器。

注意：在该官方实例中，给出的 dot1x 环境的 RADIUS 服务器已经搭建完成，在该事例中没用引入 RADIUS 服务器的搭建操作，如果需要完成整个操作事例，需要另外了解 RADIUS 服务器配置。

RADIUS 配置

首先，要为 802.1x 服务器创建一个 RADIUS 客户端，对于 RADIUS 服务器，可以不需要在同一子网端内，只要路由可达即可，因此需考虑到在防火墙策略允许访问 RADIUS 服务器 IP 地址。

```
/radius
add address=10.1.2.3 secret=radiussecret service=dot1x
```

注意：如果 RADIUS 通信是基于公网连接，建议采用 RadSec 与 RADIUS 服务器通信

配置完成 RADIUS 后，创建 dot1x 认证，ether2 和 ether12 两个接口实例。

```
/interface dot1x server
add interface=ether2 interim-update=30s comment=accounted
add interface=ether12 accounting=no comment=notaccounted
```

基于 VLAN 的端口分配

RouterOS 中支持使用在 bridge VLAN filtering 下的 VLAN 接口作为验证端口。这需要使用到 RADIUS 的 Tunnel-Type、Tunnel-Medium-Type 和 Tunnel-Private-Group-ID 字段属性。关于 Bridge VLAN filtering 的 VLAN 具体参数和配置，请参考 Bridge 章节内容。

创建 VLAN 端口，确保相应接口已经添加到了 bridge 下，而且 VLAN filtering 已经启用，配置脚本如下：

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether12
```

ether1 作为 trunk 接口，vlan2 和 vlan12 在该接口打上 tag 标签

```
/interface bridge vlan
add bridge=bridge1 tagged=ether1 vlan-ids=2
add bridge=bridge1 tagged=ether1 vlan-ids=12
```

启用 RADIUS debug logs，便于查看 RADIUS 信息日志，在该实例中，在 RADIUS 服务器的“Access-Accept message”会接收到 Tunnel 属性：

```
09:51:45 radius,debug,packet received Access-Accept with id 64 from 10.1.2.3:1812
09:51:45 radius,debug,packet      Tunnel-Type = 13
09:51:45 radius,debug,packet      Tunnel-Medium-Type = 6
09:51:45 radius,debug,packet      Tunnel-Private-Group-ID = "12"
(..)
09:51:45 radius,debug,packet      User-Name = "dot1x-user"
```

这时在 active 菜单下，在 ether12 接口上创建的 VLAN 12 下的用户认证登录认证成功

```
/interface dot1x server active print
0 interface=ether12 username="dot1x-user" user-mac=00:0C:42:EB:71:F6 session-id="86b00006" vlan=12

/interface bridge vlan print detail
Flags: X - disabled, D - dynamic
0 D bridge=bridge1 vlan-ids=1 tagged="" untagged="" current-tagged=""
current-untagged=bridge1,ether3

1 bridge=bridge1 vlan-ids=2 tagged=ether1 untagged="" current-tagged=ether1
current-untagged=ether2

2 bridge=bridge1 vlan-ids=12 tagged=ether1 untagged="" current-tagged=ether1
current-untagged=ether12
```

RouterOS 客户端配置

CA 证书要求 eap-tls, eap-ttls 和 eap-peap 验证模式，另外一个客户端证书要求 eap-tls 模式，例如，已经导入一个 P12 正式，并签发成功，关于证书相关信息参考 System/Certificates。

```
/certificate print
Flags: K - private-key, L - crt, C - smart-card-key, A - authority, I - issued, R - revoked, E - expired, T - trusted
#      NAME                COMMON-NAME    SUBJECT-ALT-NAME    FINGERPRINT
0 KAT dot1x-client      ez_dot1x-client    IP:10.1.2.34
1 LAT dot1x             CA                                ca
```

然后添加一个 dot1x 客户端，用于发起认证请求

```
/interface dot1x client
add anon-identity=anonymous client-certificate=dot1x-client eap-methods=eap-tls identity=dot1x-user
interface=ether1 password=dot1xtest
```

如果验证成功，接口状态 **status** 将显示 “**authenticated**”

```
/interface dot1x client print
Flags: I - inactive, X - disabled
0 interface=ether1 eap-methods=eap-peap identity="dot1x-user" password="dot1xtest"
anon-identity="anonymous" client-certificate=dot1x-client status="authenticated"
```

端口 VLAN ID 分配

当 **bridge** 启用 **vlan-filtering**，可为加入 **Dot1x** 的接口重新分配 **VLAN ID**。需使用 **RADIUS Tunnel-Type**、**Tunnel-Medium-Type** 和 **Tunnel-Private-Group-ID** 三个属性来实现。请注意，支持 **VLAN ID** 下发需要开启 **Bridge** 的 **vlan-filtering**，并支持硬件加速（**hardware offloaded**），这样设备才能支持该属性

首先，确保交换机的端口加入到 **bridge**，并开启 **vlan-filtering**

```
/interface bridge
add name=bridge1 vlan-filtering=yes
/interface bridge port
add bridge=bridge1 interface=ether1
add bridge=bridge1 interface=ether2
add bridge=bridge1 interface=ether12
```

进入 **bridge vlan** 需为 **ether1** 接口出方向发送数据带 **VLAN tagged** 标签，即配置为 **trunk**

```
/interface bridge vlan
add bridge=bridge1 tagged=ether1 vlan-ids=2
add bridge=bridge1 tagged=ether1 vlan-ids=12
```

可以在 **system logging** 启用 **RADIUS** 调试日志，在 **log** 中看到完整的 **RADIUS** 消息，此事例中，从 **RADIUS** 服务器的 **Access-Accept** 消息中接收到 **Tunnel** 属性：

```
09:51:45 radius,debug,packet received Access-Accept with id 64 from 10.1.2.3:1812
09:51:45 radius,debug,packet Tunnel-Type = 13
09:51:45 radius,debug,packet Tunnel-Medium-Type = 6
09:51:45 radius,debug,packet Tunnel-Private-Group-ID = "12"
(..)
09:51:45 radius,debug,packet User-Name = "dot1x-user"
```

VLAN ID 现在出现在 **active** 活动列表中，**untagged** 端口被添加到之前创建的静态 **VLAN** 配置中。

```
/interface dot1x server active print
0 interface=ether12 username="dot1x-user" user-mac=00:0C:42:EB:71:F6 session-id="86b00006" vlan=12
```

查看 bridge vlan 配置

```

/interface bridge vlan print detail
Flags: X - disabled, D - dynamic
 0 D bridge=bridge1 vlan-ids=1 tagged="" untagged="" current-tagged=""
current-untagged=bridge1,ether3

 1 bridge=bridge1 vlan-ids=2 tagged=ether1 untagged="" current-tagged=ether1
current-untagged=ether2

 2 bridge=bridge1 vlan-ids=12 tagged=ether1 untagged="" current-tagged=ether1
current-untagged=ether12

```

动态 switch rule 配置

当前 RouterOS 支持通过配置 RADIUS 服务器 Reply 下发 switch rule 的属性，可以使用 mikrotik - switches - filter 字段属性来完成，请参阅 RADIUS 供应商字典（RADIUS vendor dictionary.）。工作方式：当客户端通过 RADIUS 认证服务器的身份验证成功后，RADIUS 服务器会返回 mikrotik - switches - filter 属性到 RouterOS 交换机。交换机根据接收到的信息，在 switch rule 上创建动态访问规则。只要客户端会话处于活动状态并且接口正在运行，创建规则就会处于活动状态。交换机规则的创建有一定的顺序和限制：

- mac-protocol, src-mac-address (从 RouterOS v7.2 版本支持 src-address (IPv4/mask, 从 RouterOS 7.2 支持), dst-address (IPv4/mask), protocol (IPv4) src-port (要求许可版本 L4, 从 RouterOS 7.2 支持), dst-port 要求许可版本 L4, 条件参数才被支持)
- 可以使用十六进制或十进制表示 mac-protocol 和 protocol 参数(例如 UPD 可以设置为 protocol 17 或 protocol 0x11)
- src-port 和 dst-port 支持单个端口号或者一个连续的范围值(例如, src-port 10 或 src-port 10-20)
- src-mac-address 支持格式"xx:xx:xx:xx:xx:xx" 或"xxxxxxxxxxxx", 不带源 MAC 地址的 switch rule 可用 none 关键字设置(例如 src-mac-address none)
- src-mac-address(如果未设置属性值), switch 和 ports 条件参数会自动为每个规则设置
- 每个规则应该以一个 action 属性结尾, 支持的值是 drop 或 allow。如果没有设置操作属性, 将使用默认的 allow 值。
- 为单个客户端的请求者分配多个 switch rule 规则, 必须用逗号分隔", "

下面是 Mikrotik-Switching-Filter 属性实例, 如何动态创建 Switch rule 规则:

```

# 丢弃 arp 帧(EtherType: 0x0806 或 2054)
Mikrotik-Switching-Filter = "mac-protocol 2054 action drop"

/interface ethernet switch rule print
Flags: X - disabled, I - invalid, D - dynamic
 0 D ;;; dot1x dynamic

```



```

switch=switch1 ports=ether1 src-mac-address=CC:2D:E0:11:22:33/FF:FF:FF:FF:FF:FF
mac-protocol=arp copy-to-cpu=no redirect-to-cpu=no mirror=no new-dst-ports=""

# 允许 UDP(IP protocol: 0x11 或 17)目标端口 100, 并丢弃其他数据
Mikrotik-Switching-Filter = "protocol 17 dst-port 100 action allow, action drop"

/interface ethernet switch rule print
Flags: X - disabled, I - invalid, D - dynamic
0 D;;; dot1x dynamic
switch=switch1 ports=ether1 src-mac-address=CC:2D:E0:11:22:33/FF:FF:FF:FF:FF:FF protocol=udp
dst-port=100 copy-to-cpu=no redirect-to-cpu=no mirror=no

1 D;;; dot1x dynamic
switch=switch1 ports=ether1 src-mac-address=CC:2D:E0:11:22:33/FF:FF:FF:FF:FF:FF copy-to-cpu=no
redirect-to-cpu=no mirror=no new-dst-ports=""

# 允许源 MAC 地址验证通过, 丢弃其他所有数据
Mikrotik-Switching-Filter = "action allow, src-mac-address none action drop"

/interface ethernet switch rule print
Flags: X - disabled, I - invalid; D - dynamic
0 D;;; dot1x dynamic
switch=switch1 ports=ether1 src-mac-address=CC:2D:E0:01:6D:EB/FF:FF:FF:FF:FF:FF copy-to-cpu=no
redirect-to-cpu=no mirror=no

1 D;;; dot1x dynamic
switch=switch1 ports=ether1 copy-to-cpu=no redirect-to-cpu=no mirror=no new-dst-ports=""

```

在这个实例中, **Supplicant2** 接入 **ether2**, 仅允许访问 **192.168.50.0/24** 网络的 **UDP/50** 端口, 其他所有流量将被丢弃。首先确保 **hardware offloading** 在所有 **bridge** 端口能工作, 否则 **switch rule** 将无法工作

```

/interface bridge port print
Flags: X - disabled, I - inactive, D - dynamic, H - hw-offload
# INTERFACE BRIDGE HW PVID PRIORITY PATH-COST INTERNAL-PATH-COST HORIZON
0 H ether1 bridge1 yes 1 0x80 10 10 none
1 H ether2 bridge1 yes 1 0x80 10 10 none
2 H ether12 bridge1 yes 1 0x80 10 10 none

```

通过启用 **RADIUS** 调试日志, 可以看到 **RADIUS** 信息的完整 **attributes** 属性。下面的实例中, 从 **RADIUS** 服务器的 **Access-Accept** 消息中接收到 **mikrotik - switches - filter** 属性:


```
02:35:38 radius,debug,packet received Access-Accept with id 121 from 10.1.2.3:1812
```

```
(..)
```

```
02:35:38 radius,debug,packet      MT-Switching-Filter = "mac-protocol 2048 dst-address 192.168.50.0/24
dst-port 50 protocol 17 action allow,action drop"
```

动态的 Switch rule 现在显示在切换菜单下：

```
/interface ethernet switch rule print
```

```
Flags: X - disabled, I - invalid, D - dynamic
```

```
0  D;;; dot1x dynamic
```

```
switch=switch1 ports=ether2 src-mac-address=CC:2D:E0:11:22:33/FF:FF:FF:FF:FF:FF mac-protocol=ip
dst-address=192.168.50.0/24 protocol=udp dst-port=50 copy-to-cpu=no redirect-to-cpu=no mirror=no
```

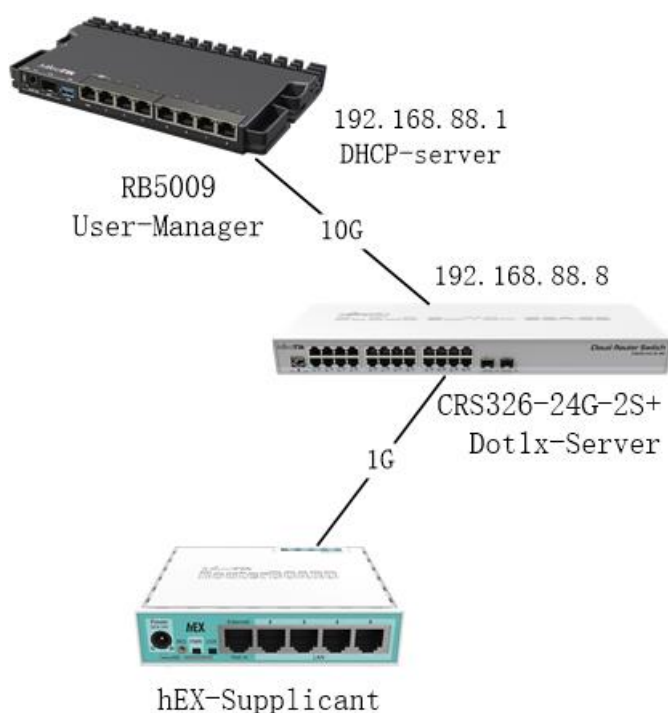
```
1  D;;; dot1x dynamic
```

```
switch=switch1 ports=ether2 src-mac-address=CC:2D:E0:11:22:33/FF:FF:FF:FF:FF:FF copy-to-cpu=no
redirect-to-cpu=no mirror=no new-dst-ports=""
```

动态 switch rule 创建仅支持带有 switch rule 功能的设备，包括：CRS3xx 系列交换机以及 CCR2116 和 CCR2216，以及采用 QCA8337, Atheros8327 和 Atheros8316 交换芯片的设备。而 CRS1xx/2xx 系列不支持该功能。需要注意不同设备的 ACL 最大规则数有所不同，请参阅官方资料 [CRS3xx](#), [CRS5xx](#), [CCR2116](#), [CCR2216 table](#) 和 [basic switch chip table](#)

8.4 Dot1x 组网配置实例

该实例网络拓扑组成包括网关路由器 RB5009（安装 User-Manager），CRS326-24G-2S+ 交换机和终端设备 hEX，网络拓扑如下：



RB5009 作为网关路由器，实现 DHCP 服务和 nat 转换等基本上网功能，同时安装 User-Manager 功能包，配合 CRS326-24G-2S+ 的 dot1x 服务器，完成 radius 认证功能。hEX 作为接入的客户端（windows 也可以作为客户端，具体设置请自行查阅）。

在开启 Dot1x 服务前，需要创建一份用于 Dot1x 加密验证的 CA 证书，并安装到 User-Manager

首先创建证书模板

```
/certificate
add name=CA-Template common-name=CAtemp key-usage=key-cert-sign,crl-sign
add name=Server common-name=server
```

使用 print detail 命令显示详细内容：

```
[admin@MikroTik] /certificate> print detail
Flags: K - private-key; L - crl; C - smart-card-key; A - authority; I - issued, R - revoked; E - expired; T - trusted
 0  name="CA-Template" key-type=rsa common-name="CAtemp" key-size=2048 subject-alt-name=""
    days-valid=365 key-usage=key-cert-sign,crl-sign

 1  name="Server" key-type=rsa common-name="server" key-size=2048 subject-alt-name=""
    days-valid=365
    key-usage=digital-signature,key-encipherment,data-encipherment,key-cert-sign,crl-sign,tls-server,tls-client
```

注意，如果 CA 证书被删除，那么链中所有已颁发的证书也将被删除。

签发证书

证书需要通过 sign 签发。在以下示例中，将对证书进行签名，并为添加服务器证书的 CRL-URL，例如签发服务器证书时指定 ca-crl-host:192.168.88.1，即服务器 IP 地址：

```
/certificate
sign CA-Template
sign Server ca-crl-host=192.168.88.1 name=ServerCA
```

通过 print 命令检查证书签发情况：

```
[admin@MikroTik] /certificate> print
Flags: K - private-key; L - crl; A - authority; T - trusted
Columns: NAME, COMMON-name, FINGERPRINT
#      NAME      COMMON  FINGERPRINT
0  K AT  CA-Template  CAtemp  0c7aaa7607a4dde1bbf33deaae6be7bac9fe64ba47d64e8ad6cf8
2  K LAT ServerCA  server  152b88c9d81f4b765a59e2302e0efd1fbf11ceed6e59f4977787a5bb90
```

RB5009 配置

设置 User-manager 证书，将之前创建号的 serverCA 证书设置到 User-Manager 的 certificate

```
/user-manager
set certificate=serverCA enabled=yes
```

添加指定接入的路由器（NAS）

```
/user-manager router
add address=192.168.88.100 name=dot1
```

创建一个 user-group 取名 eap

```
/user-manager user group
add name=eap inner-auths=tls-mschap1,tls-mschap2,peap-mschap2 outer-auths= eap-peap
```

在 user 下创建账号，并设置 group 为 eap

```
/user-manager user
add name=dotx1 password=dotx1 group=eap
```

提示：User-manager 配置部分可以参考 User-Manager 章节，具体配置类似 WiFi 的 802.1x 认证

CRS326 交换机配置

作为 Dot1x 的服务器，需要将所有端口加入 Bridge，这部分设置省略，请参考之前的配置

创建 bridge 的 IP 地址 192.168.88.100，与 Radius 服务器的 RB5009 在同一 IP 地址段

```
/ip address
add address=192.168.88.100/24 interface-bridge1
```

连接 Radius 服务器，RB5009 的 IP 地址为 192.168.88.1，并启用 user-manager 服务

```
/radius
add address=192.168.88.1 service=dot1x
```

配置交换机的 dot1x 的服务器，并添加 ether2 和 ether3 为认证端口

```
/interface dot1x server
add auth-timeout=30s interface=ether2 retrans-timeout=15s
add auth-timeout=30s interface=ether3 retrans-timeout=15s
```

hEX 客户端配置

在/interface dot1x 的 client 添加 ether4 作为接入端口，并填写账号密码

```
/interface dot1x client  
add eap-methods=eap-mschapv2 identity=dotx1 password= dotx1 interface=ether4
```

第九章 VXLAN

VXLAN (Virtual eXtensible Local Area Network)是为了解决 IEEE 802.1Q 中 VLAN id(4096)有限的问题而设计的一种隧道协议，在 IETF RFC 7348 中做了详细的描述。使用 VXLAN，该标识符的大小扩展到 24-bit(16777216)。它在第三层网络上创建了一个第二层传输的解决方案，VXLAN 协议运行在 UDP 协议上。

与大多数隧道不同，VXLAN 是 1 到 N 的网络，而不是点对点的网络。VXLAN 设备可以像网桥一样动态地学习另一个端点的 IP 地址。组播或单播用于泛洪广播、未知单播和组播流量。

只有在同一 VXLAN 网段内的设备才能相互通信，每个 VXLAN 网段用一个 24-bit 的网段标识，称为 VNI (VXLAN Network Identifier)。

VXLAN 隧道的终端节点称为 VTEP(VXLAN tunnel endpoint)，是 VXLAN 网络的边缘设备，是 VXLAN 隧道的起点和终点，源服务器发出的原始数据帧，在 VTEP 上被封装成 VXLAN 格式的报文，并在 IP 网络中传递到另外一个 VTEP 上，并经过解封转还原出原始的数据帧，最后转发给目的服务器。

配置包括 VXLAN 接口和 VTEP 两个参数项，Winbox 只开放了 VXLAN 项配置，而 VTEP 配置仅提供了命令行。

9.1 基本属性

操作路径: /interface vxlan

属性	描述
allow-fast-path (yes no; 默认: yes)	是否启用 Fast Path 处理。当启用 IPv6 VTEP 或 VRF 的 VXLAN 接口不能使用 Fast Path。该设置从 RouterOS 7.8 版本开始提供。
arp-timeout (auto integer; 默认: auto)	ARP 列表中的 ARP 记录在没有收到来自 IP 报文后保留的时间。当设置为 auto 时，等同于 IP/Settings 下的 <code>arp-timeout</code> ，默认值是 30 秒
dont-fragment (disabled enabled inherit; 默认: disabled)	DF (Don't Fragment)flag 控制一个数据包在通过网络发送之前是否可以被分解成更小的数据包，称为分段。在配置 VXLAN 时，该设置可以决定 IPv4 的外部头是否有 DF 标志，当封装的报文超过接口 MTU 时，可以控制报文的分段。该设置有三个选项: <code>disabled</code> -在 IPv4 的外部报头上没有设置 DF flag，这意味着如果报文太大而无法通过出接口发送，则会被分片。当 VXLAN 使用 IPv6 时，允许数据包分片。 <code>enabled</code> – DF flag 总是设置在 IPv4 的外部报头上，这意味着数据包不会被分片，如果它们超过了出接口的 MTU 就会被丢弃。这也避免了 VXLAN 使用 IPv6 时的数据包分片。 <code>inherit</code> -外部 IPv4 报头上的 DF flag 基于内部 IPv4 DF flag。如果内部 IPv4 报头设置了 DF 标志，外部 IPv4 报头也会设置 DF 标志。如果报文超过了出接口的 MTU，并且配置了 DF，该报文将被丢弃。如果内部报文为非 ip 报文，则外部 IPv4 头不设置 DF 标志，可以

属性	描述
	对报文进行分片。如果内部报文是 IPv6 报文，则外部 IPv4 报头将始终设置 DF 标志，报文不能被分片。注意:当 VXLAN 使用 IPv6 时，此设置不生效,如同被禁用。该设置从 RouterOS v7.8 开始支持
group (IPv4 IPv6; 默认:)	通过指定组播组地址，可以在 vtep 之间转发广播、未知单播和组播流量。此属性需要指定网络接口。网络接口将使 IGMP 加入指定的组播组，确保添加必要的 PIM 和 IGMP 配置。设置此属性后，vteps-ip-version 自动更新为使用的组播 IP。
interface (name; 默认:)	组播转发使用的网络接口。此属性需要指定 Group 参数。
local-address (IPv4 IPv6; 默认:)	用于设置 VXLAN 接口的本地源地址。如果不设置，将选择出接口的一个 IP 地址作为 VXLAN 的源地址。设置此属性后，vteps-ip-version 自动更新为使用的本地 IP 版本。该设置从 RouterOS 7.7 版本开始支持。
mac-address (read-only; 默认:)	自动分配的接口 MAC 地址。此设置不可更改
max-fdb-size (integer: 1..65535; 默认: 4096)	限制 VXLAN 在 FDB (forwarding database)中存储 MAC 地址的最大数量。
mtu (integer; 默认: 1500)	VXLAN 接口默认设置 MTU 为 1500。l2mtu 将根据关联的接口(减去 VXLAN 头部对应的 50 字节)自动设置。如果不指定接口，则使用 65535 的 l2mtu 值。l2mtu 是无法修改。
name (text; 默认: vxlan1)	Vxlan 接口名称
port (integer: 1..65535;默认: 8472)	设置 UDP 端口
vni (integer: 1..16777216; 默认:)	VXLAN 网络身份识别-VXLAN Network Identifier (VNI)
vrf (name; 默认: main)	配置 vtep 监听和连接的 VXLAN 接口的 VRF。当使用 interface 和 multicast group 设置时，不支持 VRF。同一 UDP 端口不能同时用于多个路由表中。该设置从 RouterOS 7.7 版本开始支持
vteps-ip-version (ipv4 ipv6; 默认: ipv4)	配置静态 vtep 使用的 IP 协议版本。RouterOS 的 VXLAN 接口不支持双栈，如果远端 vtep 配置的 IP 协议版本不同将无法连接。当设置 multicast group 或 local-address 属性时，vteps-ip-version 自动更新为使用的 IP 版本。

操作路径: /interface vxlan vteps

属性	描述
interface (name; 默认:)	VXLAN 网络名称
port (integer: 1..65535; 默认: 8472)	使用的 UDP 端口号

属性	描述
remote-ip (IPv4; 默认:)	远端 VTEP 的 IPv4 地址

9.2 Forwarding table

从 RouterOS 7.9 版本开始，vxlan 能从远端的 VTEP 上学习到 MAC 地址，并在 fdb 下显示。

操作路径：/interface vxlan fdb

属性	描述
interface (read-only: name)	Vxlan 接口名称 Name of the VXLAN interface.
mac-address (read-only: MAC address)	MAC 地址
remote-ip (read-only: IPv4 IPv6 address)	IPv4 或 IPv6 远端 VTEP 的目标地址

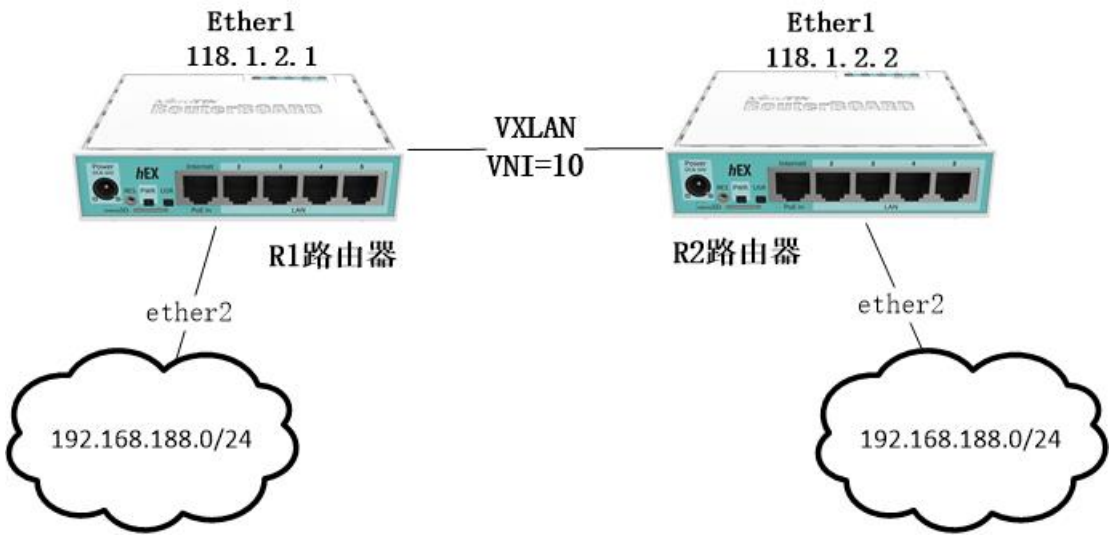
使用命令查看 fdb 转发表

```
[admin@MikroTik] > /interface vxlan fdb print
0 remote-ip=2001::2 mac-address=56:FF:AA:1A:72:33 interface=vxlan1
1 remote-ip=2002::2 mac-address=AE:EC:C4:12:8B:B9 interface=vxlan1
2 remote-ip=192.168.10.20 mac-address=FE:AF:58:31:A7:B6 interface=vxlan2
```

下面的实例通过在两端节点各自创建一个单 VXLAN 隧道连接配置

9.3 VXLAN 配置实例

下面的实例是两个静态 VTEP 端之间创建单个 VXLAN 隧道，连接两端局域网的 192.168.188.0/24 网络，打通二层互联。



在两台路由器同时创建 VXLAN 接口，使用默认 UDP 端口，设置 vni=10，下面是 R1 路由器 IP 地址配置

```
[admin@R1] /ip/address> print
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS          NETWORK          INTERFACE
0 118.1.2.1/24      118.1.2.0        ether1
```

R2 路由器的 IP 地址配置

```
[admin@R2] /ip/address> print
Columns: ADDRESS, NETWORK, INTERFACE
# ADDRESS          NETWORK          INTERFACE
0 118.1.2.2/24      118.1.2.0        ether1
```

R1 路由器和 R2 路由器同时配置 vxlan 接口，设置默认端口和 vni=10

```
[admin@R1] >/interface vxlan
[admin@R1] /interface/vxlan> add name=vxlan1 port=8472 vni=10

#R2
[admin@R2] /interface/vxlan> add name=vxlan1 port=8472 vni=10
```

注意，如果按照上面的配置，在 Winbox 创建时会报错，要求设置组播地址和接口，因此可以做以下配置

```
[admin@R2] /interface/vxlan>add group=239.0.0.1 interface=ether1 name=vxlan1 port=8472 vni= 10
```

接下来，配置两台路由器的 VTEP，在 Winbox 还未添加 VTEP 的配置界面，暂时只能在命令行操作，

路由器 1 配置

```
[admin@R1] /interface/vxlan> vteps
[admin@R1] /interface/vxlan/vteps>add interface=vxlan1 remote-ip=118.1.2.2
```

R2 路由器配置

```
[admin@R2] /interface/vxlan> vteps
[admin@R2] /interface/vxlan/vteps>add interface=vxlan1 remote-ip=118.1.2.1
```

隧道配置完成，之后可以将 VXLAN 接口和 ether2 以太网口加入到 Bridge 中，R2 路由器配置

```
[admin@R1] >/interface bridge
```



```
[admin@R1] /interface/bridge>add name=bridge1
[admin@R1] /interface/bridge> port
[admin@R1] /interface/bridge/port>add interface=vxlan1 bridge=bridge1
[admin@R1] /interface/bridge/port>add interface=ether2 bridge=bridge1
```

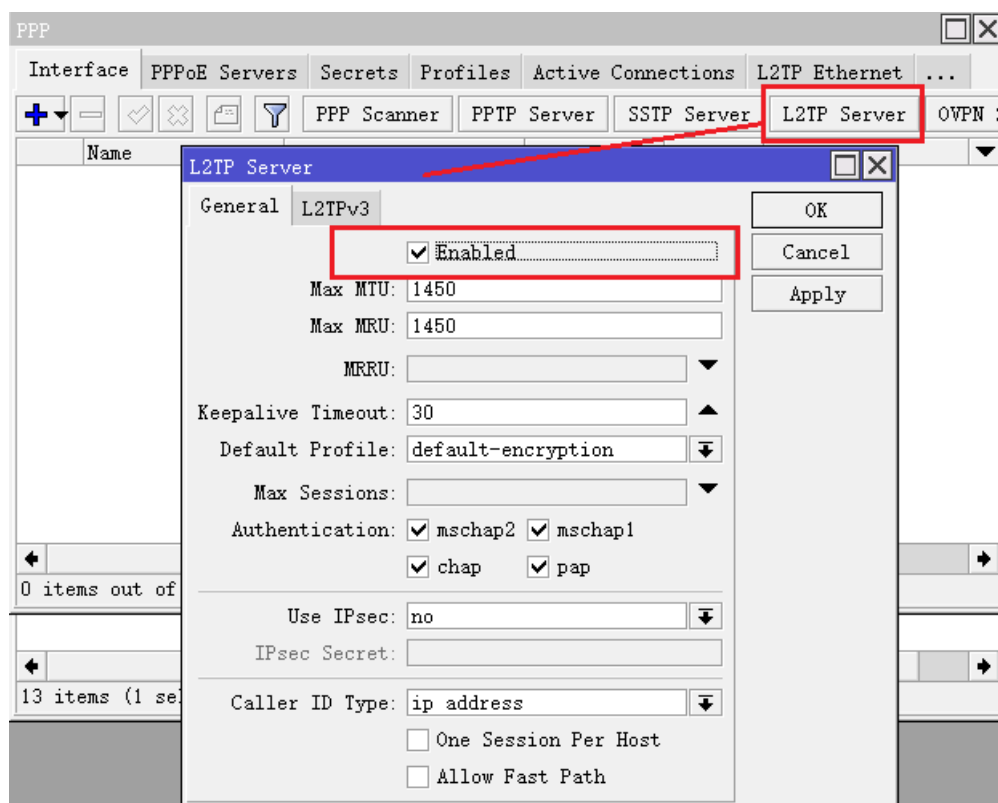
R2 路由器做相同的配置

```
[admin@R2] >/interface bridge
[admin@R2] /interface/bridge>add name=bridge1
[admin@R2] /interface/bridge> port
[admin@R2] /interface/bridge/port>add interface=vxlan1 bridge=bridge1
[admin@R2] /interface/bridge/port>add interface=ether2 bridge=bridge1
```

第十章 L2TPv3

L2TPv3 (rfc3931)通过将协议中 PPP 部分修改移出 L2TP 的核心报文, 允许不同类型的帧通过 L2TP 隧道。对于每种类型的帧被称为 pseudowire(虚拟电路)类型。RouterOS 的 pseudowire 支持 ether 和 PPP。对于 ether 类型使用 L2TPv3, 可将两个异地网络“桥接”在一起, 将它们放在相同的广播域/子网中, 类似 EoIP、BCP 和 VxLAN 技术。

在 Winbox 的 PPP 目录下, 仍然在 L2TP server 启用服务, 在 L2TPv3 Server 选项多了一个 L2TPv3 栏:



但 Winbox 并没有给出 L2TPv3 Server 全部的参数, 在命令行通 print 命令可以查看全部设置:

```
[admin@RouterOS] /interface/l2tp-server/server> print
enabled: no
max-mtu: 1450
max-mru: 1450
mrru: disabled
authentication: pap,chap,mschap1,mschap2
keepalive-timeout: 30
max-sessions: unlimited
default-profile: default-encryption
use-ipsec: no
ipsec-secret:
caller-id-type: ip-address
one-session-per-host: no
allow-fast-path: no
l2tpv3-circuit-id:
l2tpv3-cookie-length: 0
l2tpv3-digest-hash: md5
accept-pseudowire-type: all
accept-proto-version: all
```

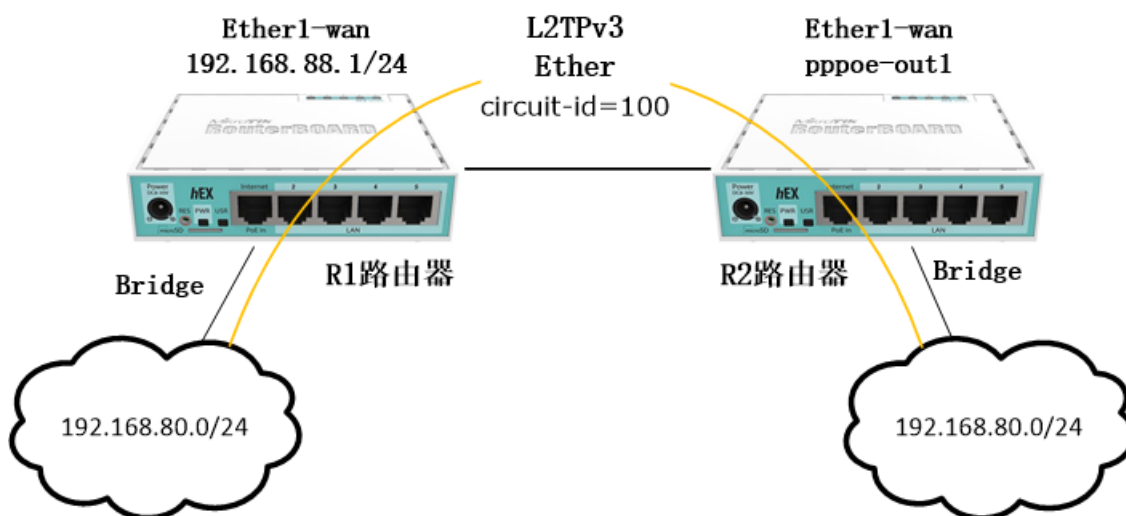
可以看到 `accept-pseudowire-type` 和 `accept-proto-version` 属性在 winbox 没有被显示，默认设置为 `all`，当需要时设置两个属性用于控制连接的类型和协议：

- **accept-pseudowire-type** 包含：all、ether 和 ppp 三个选项，ether 类型代表支持 L2TPv3 ethernet 二层桥接功能，而 PPP 类型代表点对点隧道拨号连接，即通过 L2TP 的账号密码连接方式。
- **accept-proto-version** 包含：all、l2tpv2 和 l2tpv3 三个选项，在协议方面可以选择 l2tpv2 或 l2tpv3。

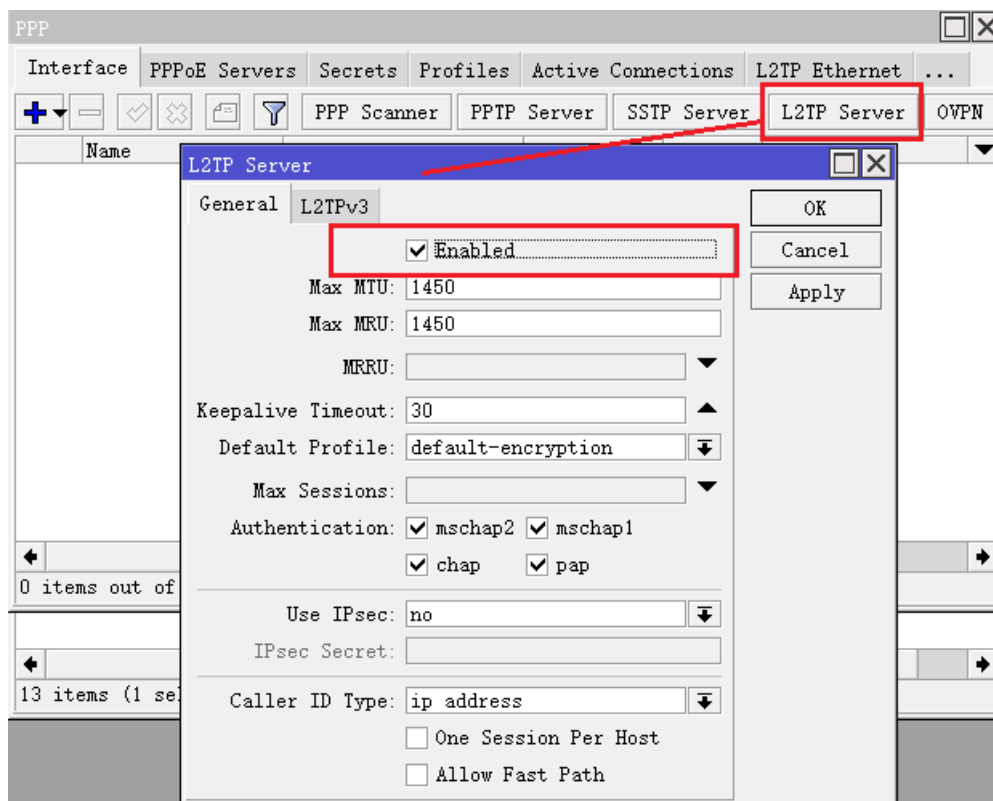
基于 PPP 类型的账号连接就不再讲述，设置与 L2TPv2 操作基本一样，在 server 端可以通过 **accept-proto-version** 选择支持 v2 还是 v3 协议。

10.1 L2TPv3 以太网桥接配置实例

对于 L2TPv3 点对点以太网隧道建立，不要两端具有公网 IP 地址，只需要启用一端的 Server，另外一端作为 client 客户端连接，即可不像 EoIP 和 VxLAN 要求双方都必须填写对方 IP，拓扑如下：



首先在 R1 路由器作为 Server，启用 L2TP server



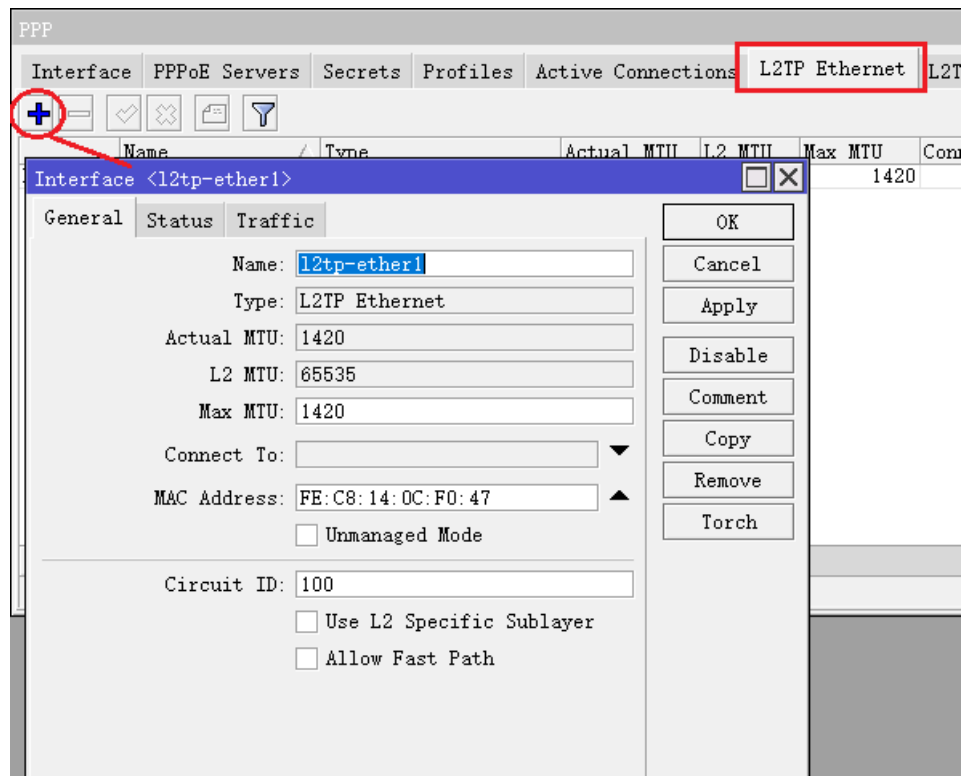
提示： 如果网络仅提供 L2TPv3 ether 类型连接，可在命令行设置 `accept-pseudowire-type=ether`

```
[admin@MikroTik] /interface/l2tp-server/server> set accept-pseudowire-type=ether
```

如果想兼顾点对点的 PPP 账号认证连接拨号，可以修改为 `accept-pseudowire-type=all`

```
[admin@MikroTik] /interface/l2tp-server/server> set accept-pseudowire-type=all
```

在 L2TP-ethernet 路径下创建 `l2tp-ether1` 接口，并设置 `circuit-id=100`（电路 ID，类似 vlan 号，相同 ID 才能通信）



R2 路由器上，同样创建 l2tp-ether1 接口，设置 connect to=192.168.88.1，circuit-id=100

PPP

Interface PPPoE Servers Secrets Profiles Active Connections L2TP Ethernet

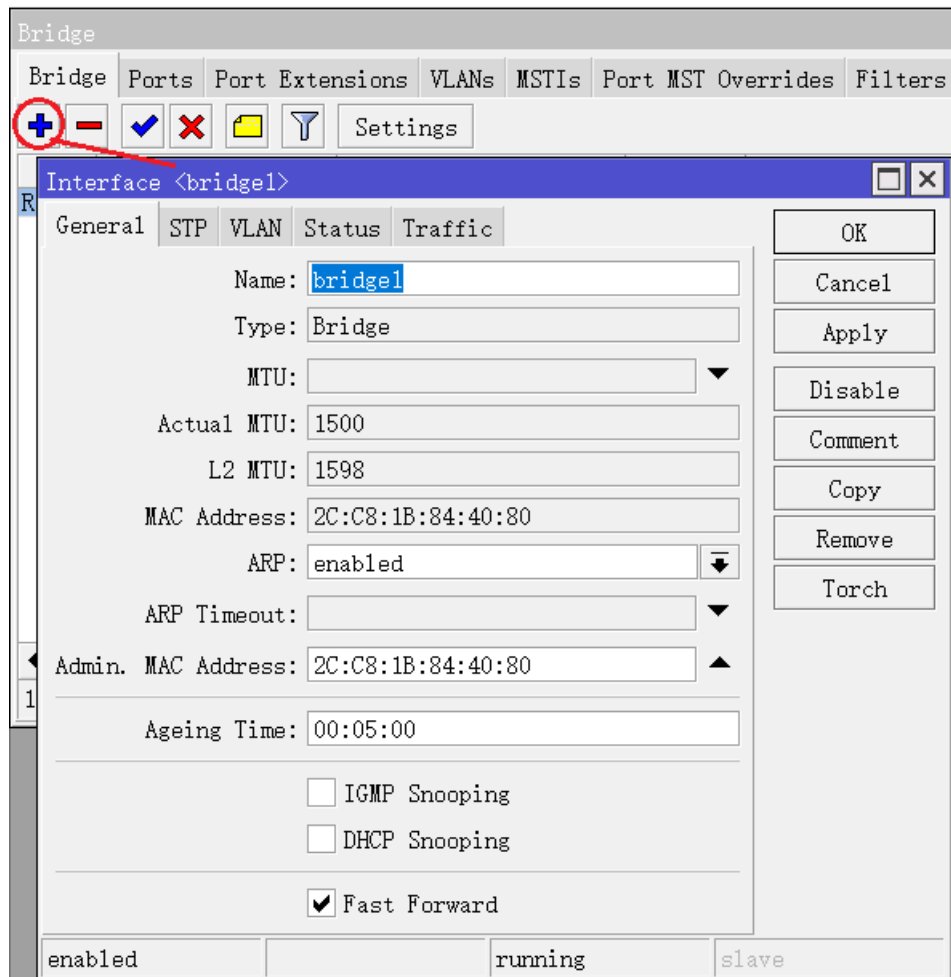
+ - ✓ ✗ 📁 🏠

Name	Type	Actual MTU	L2 MTU	Max MTU
Interface <12tp-ether1>				
General Status Traffic				
Name:	12tp-ether1			
Type:	L2TP Ethernet			
Actual MTU:				
L2 MTU:	65535			
Max MTU:	1420			
Connect To:	192.168.88.1			
MAC Address:	FE:03:14:38:28:41			
<input type="checkbox"/> Unmanaged Mode				
<input type="checkbox"/> Use Ipsec				
Ipsec Secret:				
L2TP Protocol Version: 12tpv3 udp				
Cookie Length: 0				
Digest Hash: md5				
Circuit ID: 100				
<input type="checkbox"/> Use L2 Specific Sublayer				
<input type="checkbox"/> Allow Fast Path				

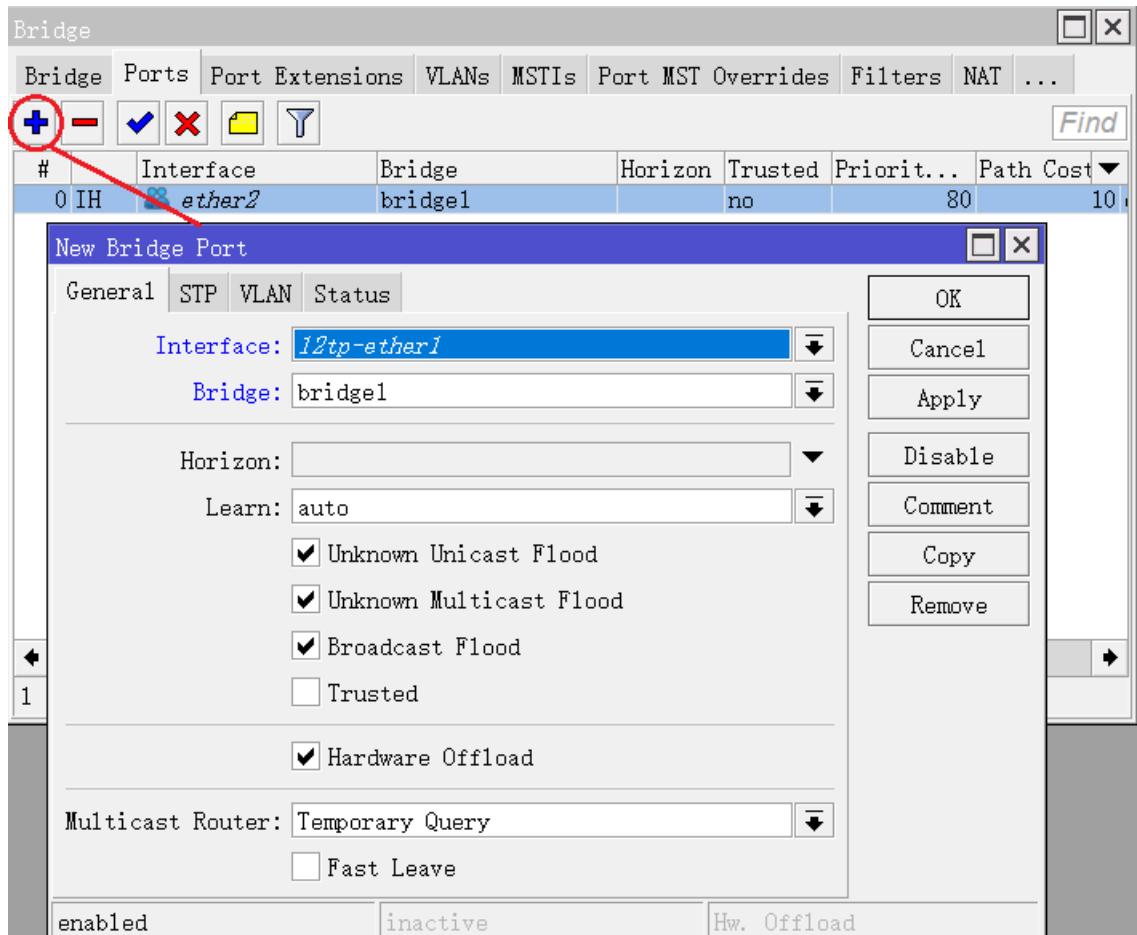
OK Cancel Apply Disable Comment Copy Remove Torch

1 item

同时为两台路由器 R1 和 R2 创建 Bridge1 接口，



同时在两台路由器的 Bridge Port 下，指定 ether2 和 l2tp-ether1 接口加入到 Bridge1 中，

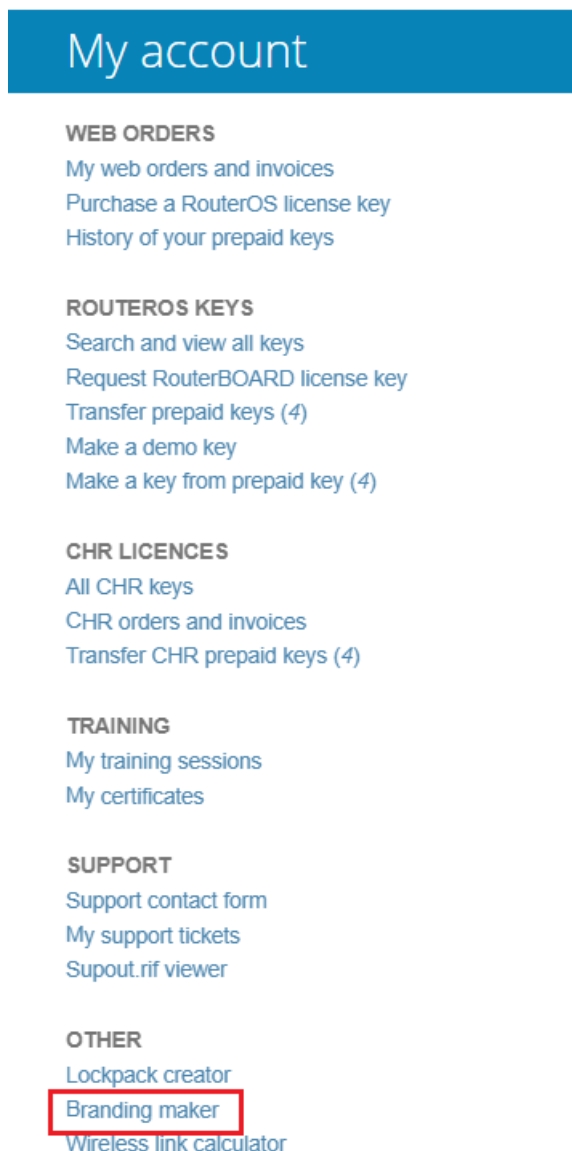


两台路由器 R1 和 R2 完成配置后，ether2 和 l2tp-ether1 建立了二层桥接。

第十一章 Branding 定制

RouterOS 支持系统界面的微定制，可以通过一个 branding 功能进行操作。这是一个特殊的功能，需要在 <https://mikrotik.com/client> 官网注册帐户，进入帐户管理部分的“Branding maker”中生成，生成完成后，文件扩展名为.dpk，并从网站下载。生成的 dpk 包可以安装在任何 RouterOS 版本中，在 RouterOS 上安装与.npk 包相同的方法，上传到 file 根目录，并重启生效。也可以使用 Netinstall 工具安装。

登录官方页面后，找到 branding maker



进入后，如下界面，按照自己的需求修改，然后点击 make 生成.dpk 文件

Enter your custom information

Router name

iRouterOS

Company URL

www.irouteros.com

Manual URL

http://www.irouteros.com/

Telnet ASCII Logo

iRouterOS.com

☐ Hide "Mikrotik" from SNMP information.

Upload custom files

Choose file

 未选择任何文件



Select category:

WebFig logo ▾

Uploaded files

No files uploaded.

Your custom package

Result file	Action
 iRouterOS.dpk	



LCD Logo

Default

11.1 ASCII logo

这是登录到命令行界面时显示的品牌文本 logo 标志，例如 Telnet、SSH、Winbox 终端等。ASCII 标志可以在浏览器中创建，也可以从任何其他明文编辑器复制。请确保它不宽的形式在品牌制造商页面，否则您的徽标将扭曲。

11.2 默认页面

RouterOS 提供 www 服务，当在浏览器输入 RouterOS 的 IP，可以打开一个默认 RouterOS 信息页面，用于管理或登录 webfig 配置。该页面可以自定义 HTML 文件，可以使用以下变量：

- **%version%** 值可修改路由器的当前版本
- **%host%** 值获取路由器的 IP 地址。在 Telnet 链接或标头中使用该变量。

该文件编辑后，必须命名为“index2.html”。确保 HTML 正确编辑，以便兼容浏览器。也可以上传图片或 JavaScript 文件，它们必须引用与索引文件相同的路径，不能使用自定义文件夹名称。如果只希望更改

MikroTik 的 logo，上传整个 HTML 文件，则可以上传默认图片文件名称，名称为 mikrotik_logo.png，仅需上传一个同名的文件，将覆盖即可。

11.3 其他参数

"Router name": 这是 RouterOS 的 Identity 值 (/system identity)，只能是一个“单词”，不能使用空格或特殊字符。

"LCD logo": 该 logo 将显示在装有液晶屏的设备上。logo 要求: 宽度不超过 160px, 高度不超过 72px。CCR 系列有白色(0xffffffff)背景，2011 系列有黑色(0x000000)背景。

"Default configuration": 注意使用默认配置文件时，配置是在简单安装包时附加的，但使用 reset 按钮或者系统复位命令后，只会使用你文件中的配置，重置时不会使用所有其他标准默认配置。该文件必须是一个 text/rsc 文件，每行包含一个 RouterOS 命令。您可以使用控制台的导出文件作为起点，但是我们建议只保留您想要运行的确切命令。即使在 RouterOS 重置后，该配置也会保留。

"Skins": 对于 webfig 的皮肤，是默认文件".Json"必须上传到"Skins"目录下。




























"Custom files": RouterOS 6.48.3 及以上版本也支持自定义文件的上传，将被简单复制到一个名为"branding"的文件夹中，并在 RouterOS 内部访问。

官方有视频介绍关于 [Branding](#)

第十二章 Device-Mode

Device-mode 是一种对设备功能限制属性，即限制对特定功能配置的访问。有两种模式：enterprise 和 home（企业模式和家庭模式）。

提示：由于 RouterOS v7 把大部分功能集成到了 system 功能包中，更多功能包不再单独列出，如下面是 v6.49 和 v7.1rc4 的功能包对比：

 advanced-tools-6.49-mmips.npk	 calea-7.1rc4.npk
 calea-6.49-mmips.npk	 container-7.1rc4.npk
 dhcp-6.49-mmips.npk	 gps-7.1rc4.npk
 gps-6.49-mmips.npk	 iot-7.1rc4.npk
 hotspot-6.49-mmips.npk	 routeros-7.1rc4.npk
 iot-6.49-mmips.npk	 tr069-client-7.1rc4.npk
 ipv6-6.49-mmips.npk	 user-manager-7.1rc4.npk
 lora-6.49-mmips.npk	
 mpls-6.49-mmips.npk	
 multicast-6.49-mmips.npk	
 ntp-6.49-mmips.npk	
 openflow-6.49-mmips.npk	
 ppp-6.49-mmips.npk	
 routing-6.49-mmips.npk	
 security-6.49-mmips.npk	
 system-6.49-mmips.npk	
 tr069-client-6.49-mmips.npk	
 ups-6.49-mmips.npk	
 user-manager-6.49-mmips.npk	
 wireless-6.49-mmips.npk	

也就是说 RouterOS v7 的功能包不能再通过/system package 进行更多功能的控制，只能对几个较大的功能包进行操作，现在通过 device-mode 来细化更多的功能控制。

默认情况下，所有设备都使用 enterprise，该模式允许除 container（容器）外的所有功能。home 模式禁用以下功能：scheduler, socks, fetch, bandwidth-test, traffic-gen, sniffer, romon, proxy, hotspot, email, zerotier, container。

12.1 Device-mode 基本配置

下面通过命令确认你的设备工作模式

```
[admin@MikroTik] > system/device-mode/print
mode: enterprise
```

Device-mode 可以由用户来改变，但是通过远程访问设备（winbox、SSH 或 Web）是不能改变 Device-mode。执行改变 Device-mode 后，需要确认：可按设备的 reset 按钮，或者进行“冷重启”，即按下电源（远程命令重启无效）

```
[admin@MikroTik] > system/device-mode/update mode=home
update: please activate by turning power off or pressing reset or mode button
        in 5m00s
-- [Q quit|D dump|C-z pause]
```

在 5 分钟内，拔插电源重启设备或者按设备的 reset 按钮确认执行 container 模式，设备会重启生效进入 container 模式。如果在规定的时间内没有关机或按下 reset 按钮，则取消模式切换。如果同时运行另一个更新命令，则两个更新命令都将被取消。

重启之后，查看 device-mode

```
[admin@MikroTik] /system> device-mode/print
mode: home
[admin@MikroTik] /system>
```

在 **system/device-mode/** 路径下可以获得以下命令：

命令	描述
get	通过此命令返回一个指定变量值，并可显示在控制台，或赋值给脚本
print	用于打印显示当前路径下的配置参数和属性
update	更改并应用指定的参数属性，具体请参见下面的内容。

属性描述

属性	描述
container, fetch, scheduler, traffic-gen, ipsec, pptp, smb, l2tp, proxy, sniffer, zerotier, bandwidth-test, email, hotspot, romon, socks. (yes / no; 默认 : yes, enterprise 模式)	可用功能列表，能通过 device-mode 选项进行控制。
activation-timeout (默认: 5m);	reset 按钮或电源拔插关机激活，超时时间设置范围在 00:00:10 至 1d 00:00:00。如果在此时间间隔内未按下 reset 按钮或未执行冷重启，则 update 执行将被取消。
flagging-enabled (yes / no; 默认: yes)	启用或禁用标记状态。
flagged (yes / no; Default: no)	RouterOS 使用各种机制来检测对其系统文件的篡改。如果系统检测到对 RouterOS 的非法访问，状态" flags "将设置为 yes。如

属性	描述
	果“flags”设置为 yes，为了您的安全，将设置某些限制。
mode: (home, enterprise; 默认: enterprise);	<p>默 允许选择可用的模式，将限制设备的功能。在未来，可能会添加各种模式。</p> <p>默认情况下，enterprise 模式允许除 container 外的所有功能。因此，要使用 container，需通过执行 device-mode 的 update 来启用。</p> <p>默认情况下，home 模式禁用以下特性： scheduler, socks, fetch, bandwidth-test, traffic-gen, sniffer, romon, proxy, hotspot, email, zerotier, container.</p>

对可用功能特性进行更具体的控制。device-mode 控制的每一个功能都可以被特定地打开或关闭，例如：

```
[admin@MikroTik] > system/device-mode/update mode=home email=yes
[admin@MikroTik] > system/device-mode/update mode=enterprise zerotier=no
```

如果 update 命令指定 mode 和任意功能参数，此时该 update 命令将替换整个 device-mode 配置。在这种情况下，所有“per-feature”，即预设值模式参数设置都将失效，除了使用该命令指定的那些功能参数设置。例如：

```
[admin@MikroTik] > system/device-mode/update mode=home email=yes fetch=yes
-- reboot --
[admin@MikroTik] > system/device-mode/print
mode: home
fetch: yes
email: yes
```

update mode=home，并指定了 fetch=yes 和 email=yes 两个参数，即在 home 模式下除了 fetch 和 email 开启，其他功能将关闭（**scheduler, socks, bandwidth-test, traffic-gen, sniffer, romon, proxy, hotspot, zerotier, container.**）

下面的命令代表在 enterprise 模式下 sniffer 关闭

```
[admin@MikroTik] > system/device-mode/update mode=enterprise sniffer=no
-- reboot --
[admin@MikroTik] > system/device-mode/print
mode: enterprise
sniffer: no
```

关闭 hotspot 功能

```
[admin@MikroTik] > system/device-mode/update hotspot=no
```

```
-- reboot --
```

当重启后，通过 `print` 命令查看：

```
[admin@MikroTik] > system/device-mode/print
mode: enterprise
sniffer: no
hotspot: no
```

当使用 `sniffer` 命令抓取网络数据时，将提示 `device-mode` 不允许执行

```
[admin@MikroTik] > tool/sniffer/quick
failure: not allowed by device-mode
```

在 `hotspot` 下虽然能添加配置，但会有一个注释显示：无法激活，在 `device-mode` 下被禁用：

```
[admin@MikroTik] > ip hotspot/add interface=ether1
[admin@MikroTik] > ip hotspot/print
Flags: X, S - HTTPS
Columns: NAME, INTERFACE, PROFILE, IDLE-TIMEOUT
#   NAME      INTERFACE  PROFILE  IDLE-TIMEOUT
;;; inactivated, not allowed by device-mode
0 X hotspot1 ether1      default  5m
```

12.2 Flagged 状态

除了 `device-mode` 功能，RouterOS 现在可以在系统启动时分析整个配置，以确定是否有任何未经授权访问路由器的迹象。如果检测到可疑配置，则禁用这些可疑配置，**flagged** 属性会设置为“yes”。

设备现在有一个 **flagged** 标记状态，并强制限制某些功能，如下：

```
[admin@MikroTik] > system/device-mode/print
mode: enterprise
flagged: yes
sniffer: no
hotspot: no
```

如果系统有此标记状态，则当前配置正常工作，但无法执行以下操作：

`bandwidth-test`, `traffic-generator`, `sniffer`，以及为以下程序启用或创建新的配置项的配置操作(仍然可以禁用或删除它们): `system scheduler`, `SOCKS proxy`, `pptp`, `l2tp`, `ipsec`, `proxy`, `smb`。

当路由器处于标记状态时执行上述操作，你会收到一条错误消息：

```
[admin@MikroTik] > /tool sniffer/quick  
failure: configuration flagged, check all router configuration for unauthorized changes and update  
device-mode  
[admin@MikroTik] > /inter l2tp-client/add connect-to=1.1.1.1 user=user
```

如果要退出 **flagged** 状态，必须执行命令“/system/device-mode/update flags =no”。系统要求按下 **reset** 按钮或硬重启(拔插电源，或关闭 CHR)。

警告！尽管系统已经禁用了带有 **flagged** 标记的恶意功能，但在退出标记状态之前，检查所有配置，检查是否有未知的配置内容。如果当前系统已经被 **flagged**，假设你的系统已经被入侵，并在重新启用 RouterOS 系统使用之前对所有设置进行全面检查。检查完成后，修改 RouterOS 的所有登录密码，并升级到最新的 RouterOS 版本。

第十三章 ROSE-存储功能

ROSE--为 RouterOS 增加了额外的企业数据中心功能,包括支持磁盘监控、改进的格式化、RAID、rsync、iSCSI、基于 TCP 的 NVMe、NFS 和改进的 SMB。该功能目前支持 arm, arm64, x86 和 tile 平台。需 RouterOS v7.8beta2 以上版本 (v7.8beta3 对分区做了调整)。

要求功能包: rose-storage

在使用此功能前,需要单独安装 RouterOS 扩展功能包 rose-storage,下载扩展功能包访问 <https://mikrotik.com/download>,选择对应设备硬件构建,下载 **Extra packages**,获得 zip 文件,解压后,找到 rose-storage-xxx-xxx.npk 文件上传到 RouterOS 设备 File 的根目录下,重启设备。具体操作可以参见《RouterOS 入门到精通 v6》版本的 2.8 章节。

操作路径: /disk

RAID(独立磁盘冗余阵列)技术允许将数据存储多个驱动器上-通过将它们组合成逻辑单元来提高数据传输性能,数据保护或两者兼有。

RouterOS 当前支持软 RAID 0, 1, 4, 5, 6 和嵌套 RAID10,关于 RAID 知识的相关资料大家可以自行查阅。

13.1 RAID 配置

在配置 RAID 前,存储磁盘的大小必须相同,或必须设置在相同大小的分区中,或使用 raid-max-component-size 参数限制更大的磁盘卷大小,以适应其他存储。

理论上,RAID 性能优化应该使用正确的 stride 步长和 stripe-width 条带宽度。这些依赖于“RAID -chunk-size”、文件系统块大小和磁盘数量。

下面是如何计算 stride 和 stripe_width 参数:

```
stride=raid-chunk-size/block_size
stripe_width=disks*stride
```

在本例中,将创建包含 10 个磁盘的 RAID 6,RouterOS 在格式化本地 RAID 设备时自动执行此操作。

创建 RAID6,设置 slot=raid1,磁盘存储数量 10

```
/disk add type=raid raid-type=6 raid-device-count=10 slot=raid1
```

添加存储设备到该 RAID,该实例的存储设备都是 pcie1-nvme,

```
/disk set pcie1-nvme1 raid-master=raid1 raid-role=0
/disk set pcie1-nvme2 raid-master=raid1 raid-role=1
/disk set pcie1-nvme3 raid-master=raid1 raid-role=2
```

```
/disk set pcie1-nvme4 raid-master=raid1 raid-role=3
/disk set pcie1-nvme5 raid-master=raid1 raid-role=4
/disk set pcie1-nvme6 raid-master=raid1 raid-role=5
/disk set pcie1-nvme7 raid-master=raid1 raid-role=6
/disk set pcie1-nvme8 raid-master=raid1 raid-role=7
/disk set pcie1-nvme9 raid-master=raid1 raid-role=8
/disk set pcie1-nvme10 raid-master=raid1 raid-role=9
```

raid-role 是手动设置项，但强烈推荐启用。如果存储设备从未加入到 RAID 中，那么 **superblock** 是空，并且将自动设定为 **raid-role**，如果已经使用相同的 **raid-role** 可能会出现错误。

RAID 将自动同步，通过 **disk print detail** 命令查看

```
/disk print detail
...
20 bM      type=raid slot="raid1" slot-default="" parent=none device="md0" uuid="3b4d4ec9-e7413ae8-37e7e397-9cd9152e"
fs=ext4 model="RAID5 1-parity-disk" size=8 641 770 946 560 free=8 572 463 624 192 raid-type=5
raid-device-count=10 raid-max-component-size=none raid-chunk-size=1M raid-master=none
raid-state="clean, resync = 1.8% (17498368/937692160) finish=45.2min speed=339148K/sec"
nvme-tcp-export=no iscsi-export=no nfs-export=no smb-export=no
```

13.2 iSCSI

iSCSI 允许通过基于 IP 的网络访问存储设备。在启动器上 iSCSI 设备将显示为块设备。RouterOS 支持 **target** 和 **initiator** 模式

Target (Host) 模式配置:

```
/disk
set pcie1-nvme1 iscsi-export=yes
```

Initiator (client)模式配置

```
/disk
add type=iscsi iscsi-address=192.168.1.1 iscsi-iqn=pcie1-nvme1
```

Iscsi-iqn 需要匹配目标设备上的 slot 名称，iscsi-address 为目标地址。

13.3 NFS

NFS 允许分享本地文件路径到网络，RouterOS 当前支持 NFS v4 模式

RouterOS 配置

Target (Host)模式启用 NFS 配置

```
/disk
set pcie1-nvme1 nfs-export=yes
```

Initiator (client)连接配置:

```
/disk
add type=nfs nfs-address=192.168.1.1
```

Linux 配置:

```
mkdir /mnt/files
mount -t nfs 192.168.1.1:/ /mnt/files
```

13.4 SMB

SMB 是目前比较流行的文件共享协议。ROSE 扩展功能包目前支持 SMB2.1、SMB3.0、SMB3.1.1(由于安全漏洞, 不支持 SMB1)

提示: RouterOS 在未安装 ROSE 扩展功能包, 仍然支持旧 SMB 功能。

开启 SMB 服务配置

```
/disk
set pcie1-nvme1 smb-export=yes
```

客户端连接 RouterOS 搭建的 SMB 服务器配置

```
/disk
add type=smb smb-address=192.168.1.1 smb-share=pcie1-nvme1
```

Smb-share 需要匹配目标设备上的 slot 名, smb-address 为目标 IP 地址。

13.5 NVMe over TCP

NVMe -tcp 使用简单高效的 TCP/IP 协议将 NVMe 扩展到网络存储。对于目标存储设备而言, 该设备可以是 hdd/ssd/nvme 甚至是 raid 阵列

Target (Host) 配置:

```
/disk
set pcie1-nvme2 nvme-tcp-export=yes nvme-tcp-port=4420
```

Initiator (client)的 RouterOS 配置:

```
/disk
add type=nvme-tcp nvme-tcp-address=192.168.1.1 nvme-tcp-name=pcie1-nvme1
```

nvme-tcp-name 需设置在 target 设备的 slot 名称

Linux 配置:

```
load kernel module
modprobe nvme_tcp
```

在 Linux 探测可用 nvme-tcp targets:

```
nvme discover -t tcp -a 192.168.1.1 -s 4420
...
Discovery Log Number of Records 1, Generation counter 2
=====Discovery Log Entry 0=====
trtype: tcp
adrfam: ipv4
subtype: nvme subsystem
treq: not specified, sq flow control disable supported
portid: 4420
trsvcid: 4420
subnqn: pcie1-nvme1
traddr: 10.155.166.7
sectype: none
```

subnqn 需匹配 slot 名称, 使用 -n 参数

```
nvme connect -t tcp -a 192.168.1.1 -s 4420 -n pcie1-nvme1
```

查看可以获取的设备

```
ls /dev/nvme*
...
/dev/nvme0 /dev/nvme0n1 /dev/nvme-fabrics
```

13.6 RAMdisk

RAMdisk - 允许使用部分 RAM 作为存储设备。如果与 tmpfs 相比——这允许使用 RAM 作为 RAID 的一部分,或者作为一个存储设备,而不是文件夹。

```
/disk
```

```
disk add type=ramdisk ramdisk-size=500M
```

RAMdisk 将在 RouterOS 重启或断电后被删除

13.7 数据加密

目前 RouterOS 支持 SED (self - encryption drives)和 dm_crypt 驱动器加密。

Self-Encrypting Drives

对于使用 SED 驱动器必须是 opal 兼容。在购买驱动器之前，请参考驱动器制造商的文档，以了解该驱动器是否支持此功能。

RouterOS 通过 Flags，来显示支持情况 o(inactive)或 O(active)标志：

```
/disk print
Flags: B - BLOCK-DEVICE; M, F - FORMATTING; o - TCG-OPAL-SELF-ENCRYPTION-SUPPORTED
Columns: SLOT, MODEL, SERIAL, INTERFACE, SIZE, FREE, FS, RAID-MASTER
```

#	SLOT	MODEL	SERIAL	INTERFACE	SIZE	FREE	FS	RAID
0	BMO sata1	Samsung SSD 860 2.5in	S3Z9NX0N414510L	SATA 6.0 Gbps	1 000 204 886 016 983 351 111 680	ext4	none	
1	BMO sata2	Samsung SSD 860	S5GENG0N307602J	SATA 6.0 Gbps	1 000 204 886 016 983 351 128 064	ext4	none	
2	BMO sata3	Samsung SSD 860	S5GENG0N307604H	SATA 6.0 Gbps	1 000 204 886 016 983 351 128 064	ext4	none	
3	BMO sata4	Samsung SSD 860 2.5in	S4CSNX0N838150B	SATA 6.0 Gbps	1 000 204 886 016 983 351 128 064	ext4	none	

设置 TCG-OPAL-SELF-ENCRYPTION:

```
/disk
disk set sata1 self-encryption-password=securepassword
```

取消设置

```
/disk
disk unset sata1 self-encryption-password
```

或者通过！取消设置

```
/disk
disk set sata1 !self-encryption-password
```

13.8 File Sync

ROSE 扩展功能包支持文件上传/下载和同步工具。将本地文件夹内容同步(推送)到其他 RouterOS 设备，设置文件上传同步

```
/file sync add local-path=pcie1-nvme1/myfolder/ remote-addr=192.168.1.1 mode=upload user=admin
password="" remote-path=test/
```

使用 user/password 方式，使用对端 RouterOS 设备的用户名和密码。对接时需要打开 Winbox 端口

设置文件下载同步

```
/file sync add local-path=pcie1-nvme1/myfolder/ remote-addr=192.168.1.1 mode=download user=admin
password="" remote-path=test/
```

配置完成后，文件夹下的内容将同步，所有对文件的更改将在设备之间同步。一个文件夹不应该有下载和上传类型到同一个目标，以避免不明确的行为。

13.9 SMB 与 NFS 配置实例

使用一台 CHR 的 RouterOS，添加外部存储，使用 NFS 分享给 Linux，用 SMB 分享给 CHR，首先在 winbox 的 /system/disk 下进行格式化操作，从 v7.8beta3 开始支持 GPT 分区，需要先将扩展的磁盘单独添加一个分区，不能直接对扩展的磁盘分区格式话否则会报错。

进入 disk 目录，添加一个 500M 的分区

```
[admin@MikroTik] /disk> add type=partition parent=sata1 partition-size=500M
[admin@MikroTik] /disk> print
Flags: B - BLOCK-DEVICE; M, F - FORMATTING; g, p - PARTITION
Columns: SLOT, MODEL, SERIAL, INTERFACE, SIZE, FREE
#   SLOT   MODEL      SERIAL          INTERFACE  SIZE   FREE
0 B g sata1  QEMU HARDDISK QM00007        SATA 1.5 Gbps 8589934592
1 Bm p sata1-part1 @65'536-500'065'280 499999744 457854976
```

然后使用 ext4 格式对分区格式化

```
[admin@MikroTik] /disk> format-drive sata1-part1 file-system=ext4
Columns: OUTPUT
OUTPUT
file-system: ext4
mke2fs -t ext4 /dev/sdb1
mke2fs 1.46.5 (30-Dec-2021)
Discarding device blocks: done
Creating filesystem with 488280 1k blocks and 121920 inodes
Filesystem UUID: c88ad052-d5ba-4a04-b30d-d13bf2a3d84d
Superblock backups stored on blocks:
8193, 24577, 40961, 57345, 73729, 204801, 221185, 401409
Allocating group tables: done
```

```
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

[admin@MikroTik] /disk>
```

查看格式化结果

```
[admin@MikroTik] /disk> print
Flags: B - BLOCK-DEVICE; M, F - FORMATTING; g, p - PARTITION
Columns: SLOT, MODEL, SERIAL, INTERFACE, SIZE, FREE, FS, RAID-MASTER
#   SLOT          MODEL      SERIAL          INTERFACE      SIZE  FREE  FS   RAID
0 B g sata1      QEMU HARDDISK QM00007        SATA1.5Gbps    8589934592      none
1 B Mp sata1-part1 @65'536-500'065'280  499999744 457854976 ext4 none
```

可以看到 Slot 下有两项, 编号 1 的是新键 **sata1-part1** 分区, 这里我们通过指定编号, 开启 **sata1-part1** 的 **smb** 和 **nfs** 服务

```
[admin@MikroTik] /disk> set 1 smb-export=yes nfs-export=yes
```

使用 **print detail** 命令查看详情。

```
[admin@MikroTik] /disk> print detail
Flags: X - disabled, E - empty, B - block-device; M - mounted, F - formatting; f - raid-member-failed;
r - raid-member, c - encrypted; g - guid-partition-table, p - partition;
t - nvme-tcp-export, i - iscsi-export, s - smb-export; n - nfs-export;
O - tcg-opal-self-encryption-enabled, o - tcg-opal-self-encryption-supported
0 B   g   slot="sata1" slot-default="sata1" parent=none device="sdb" model="QEMU HARDDISK"
serial="QM00007"
fw-version="2.5+" size=8 589 934 592 interface="SATA 1.5 Gbps" interface-speed=1200.0Mbps
raid-master=none nvme-tcp-export=no iscsi-export=no nfs-export=no smb-export=no

1 B M psn type=partition slot="sata1-part1" slot-default="sata1-part1" parent=sata1 device="sdb1"
uuid="c88ad052-d5ba-4a04-b30d-d13bf2a3d84d" fs=ext4 serial="@65'536-500'065'280" size=499 999 744
free=457 854 976 partition-number=1 partition-offset=65 536 partition-size=499 999 744
raid-master=none nvme-tcp-export=no iscsi-export=no nfs-export=yes smb-export=yes
smb-user="GUEST"
smb-password="" smb-encryption=no

[admin@MikroTik] /disk>
```

查看 **file** 下的文件情况, 在 **sata1-part1** 下存放一个 **backup** 文件

```
[admin@MikroTik] /disk> print detail
Flags: X - disabled, E - empty, B - block-device; M - mounted, F - formatting; f - raid-member-failed;
r - raid-member, c - encrypted; g - guid-partition-table, p - partition;
t - nvme-tcp-export, i - iscsi-export, s - smb-export; n - nfs-export;
O - tcg-opal-self-encryption-enabled, o - tcg-opal-self-encryption-supported
0 B   g   slot="sata1" slot-default="sata1" parent=none device="sdb" model="QEMU HARDDISK"
serial="QM00007"
fw-version="2.5+" size=8 589 934 592 interface="SATA 1.5 Gbps" interface-speed=1200.0Mbps
raid-master=none nvme-tcp-export=no iscsi-export=no nfs-export=no smb-export=no

1 BM   psn  type=partition slot="sata1-part1" slot-default="sata1-part1" parent=sata1 device="sdb1"
uuid="c88ad052-d5ba-4a04-b30d-d13bf2a3d84d" fs=ext4 serial="@65'536-500'065'280" size=499 999 744
free=457 854 976 partition-number=1 partition-offset=65 536 partition-size=499 999 744
raid-master=none nvme-tcp-export=no iscsi-export=no nfs-export=yes smb-export=yes
smb-user="GUEST"
smb-password="" smb-encryption=no

[admin@MikroTik] /disk>
```

查看 **file** 下的文件情况，在 **sata1-part1** 下存放一个 **backup** 文件

```
[admin@MikroTik] /file> print
Columns: NAME, TYPE, SIZE
#  NAME                                     TYPE      SIZE
0  skins                                     directory
1  sata1-part1                             disk
2  sata1-part1/lost+found                   directory
3  sata1-part1/MikroTik-20230227-1350.backup backup     19.2KiB
```

在 PVE 的 linux 测试连接 RouterOS 的 NFS，在 **home** 下创建一个 **nfs** 目录，将远程的 **nfs** 挂载到 **/home/nfs** 目录下，并查看目录下的文件

```
root@pve:/home# mkdir nfs
root@pve:/home# mount -t nfs 192.168.99.30:/ /home/nfs/
root@pve:/home# cd nfs/
root@pve:/home/nfs# ls
usb1-part1
root@pve:/home/nfs# cd usb1-part1/
root@pve:/home/nfs/usb1-part1# ls
53#MikroTik-20230227-1350.backup lost+found
root@pve:/home/nfs/usb1-part1#
```


在另外一台 CHR 上测试 SMB 连接，添加路径是 `smb-share=sata1-part1`，具体配置如下：

```
[admin@MikroTik] /disk> add type=smb smb-address=192.168.99.30 smb-share=sata1-part1
```

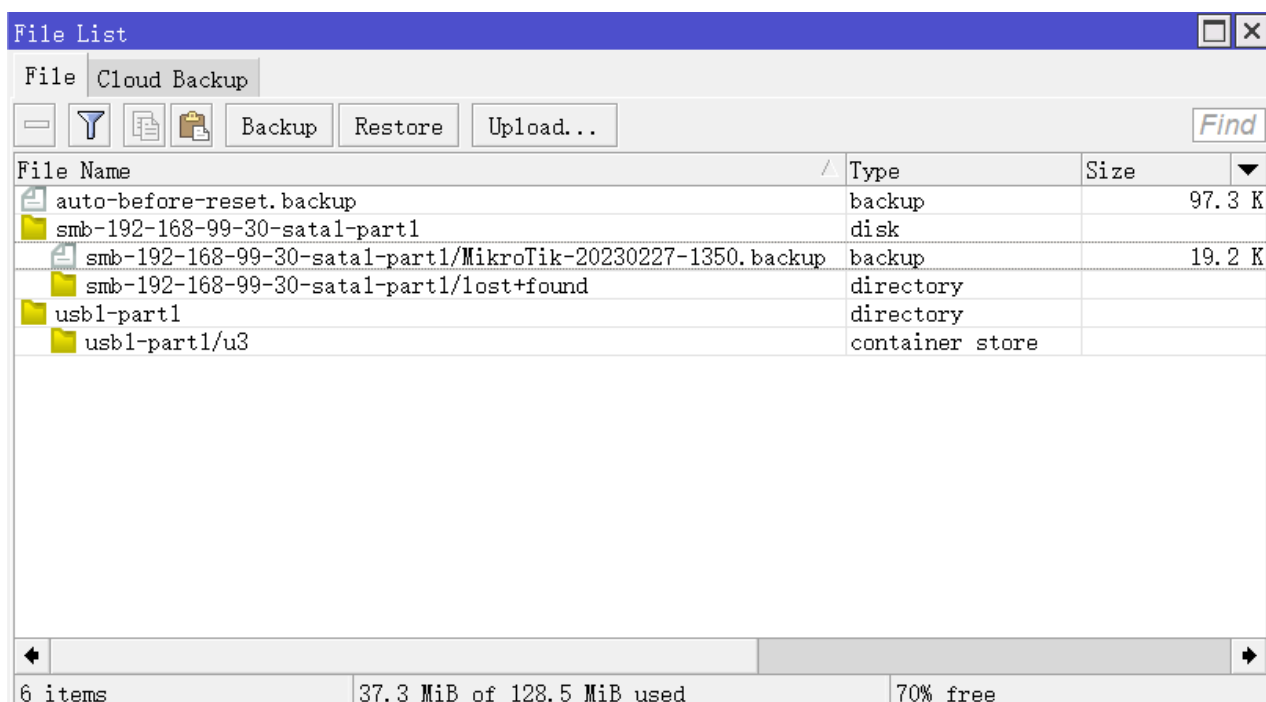
```
[admin@MikroTik] /disk> print
```

Flags: B - BLOCK-DEVICE; M, F - FORMATTING; g, p - PARTITION; s - SMB-EXPORT

Columns: SLOT, MODEL, SERIAL, INTERFACE

#	SLOT	MODEL	SERIAL	INTERFACE
0	smb-192-168-99-30-sata1-part1	smb://192.168.99.30/sata1-part1		network

如果连接成功，在 **file** 下可以看到文件目录



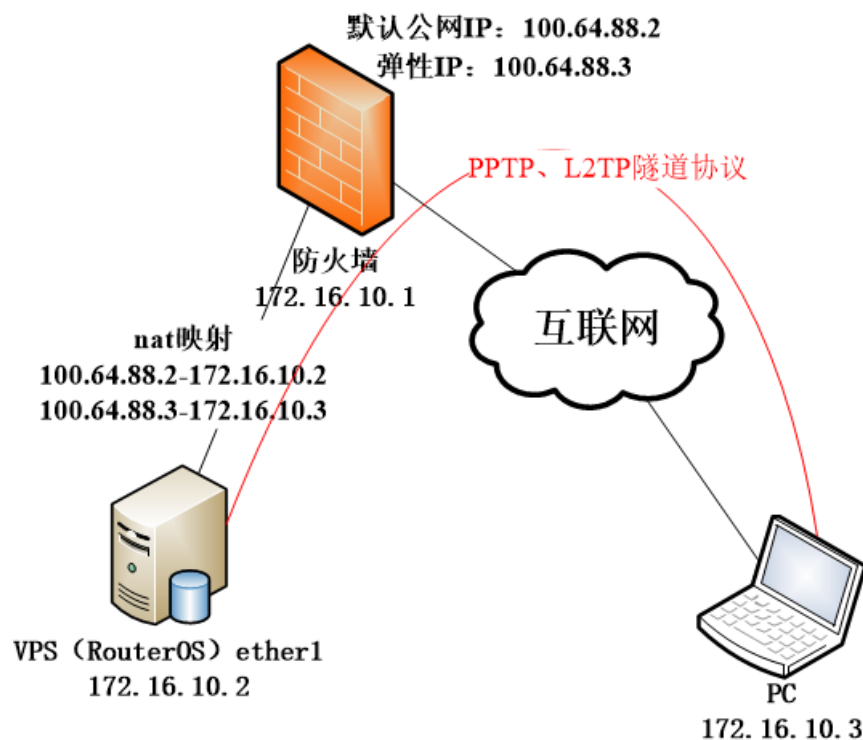
第十一章 各类应用案例

14.1 RouterOS 在 VPS 上的隧道应用

讨论关于 RouterOS 安装到 VPS 后的一些应用，在某些场景需要通过 VPS 建立隧道，下面通过两类场景实例讨论下：

弹性 IP 地址分配通过 proxy-arp:

VPS 主机分配私网 IP 地址，通过防火墙 nat 映射公网 IP 给 VPS 主机，网络拓扑如下：



VPS 已经安装了 RouterOS，并分配了一个私网 IP 地址 172.16.10.2，通过公网 IP: 100.64.88.2 映射到互联网。为了扩展业务，VPS 又申请了一个弹性公网 IP 地址 100.64.88.3，对应内网 IP 是 172.16.10.3，但该 IP 不配置到 VPS 主机，而是希望使用 PPTP 或 L2TP 隧道分配到一台远端的 PC 上，具体配置如下：

在 VPS 的 RouterOS 上，查看 IP 地址配置：

```
[admin@MikroTik] /ip/address> print
```

```
Flags: D - DYNAMIC
```

```
Columns: ADDRESS, NETWORK, INTERFACE
```

#	ADDRESS	NETWORK	INTERFACE
0	172.16.10.2/24	172.16.10.0	ether1

默认网关配置情况：

```
[admin@MikroTik] /ip/route> print
Flags: D - dynamic; X - disabled, I - inactive, A - active;
c - connect, s - static, r - rip, b - bgp, o - ospf, d - dhcp, v - vpn, m - modem,
y - copy;
H - hw-offloaded; + - ecmp
```

#		DST-ADDRESS	GATEWAY	DISTANCE
0	As	0.0.0.0/0	172.16.10.1	1
	DAc	172.16.10.0/24	ether1	0

创建 PPP Profile，设置隧道的 local-address 为 172.16.10.254、DNS 为 114.114.114.114 和每个账号登录唯一性

```
[admin@MikroTik] /ip/route> /ppp/profile/
[admin@MikroTik] /ppp/profile> add name=vps local-address=172.16.10.254 dns-server=114.114.114.114
only-one=yes
```

创建账号和密码为 yus，并为该账号分配 remote-address 为 172.16.10.3

```
[admin@MikroTik] /ppp/profile> ..
[admin@MikroTik] /ppp> secret/
[admin@MikroTik] /ppp/secret> add name=yus password=yus remote-address=172.16.10.3
```

启用 PPTP 服务，选择 profile 为 vps

```
[admin@MikroTik] /ppp/secret> /interface/pptp-server/server/
[admin@MikroTik] /interface/pptp-server/server> set enabled=yes default-profile=vps
```

进入 interface Ethernet 下查看网卡

```
[admin@MikroTik] /interface/pptp-server/server> /interface/ethernet
[admin@MikroTik] /interface/ethernet> print
Flags: R - RUNNING; S - SLAVE
Columns: NAME, MTU, MAC-ADDRESS, ARP, SWITCH
```

#	NAME	MTU	MAC-ADDRESS	ARP
0 R	ether1	1500	E4:8D:8C:BA:79:F6	enabled

修改 ether1 网口的 arp 模式为 proxy-arp，即使用 arp 代理方式响应两端网络的 arp 请求（关于 proxy-arp 请参阅 RouterOS 入门到精通 v6 的 5.3 章节 ARP 模式）

```
[admin@MikroTik] /interface/ethernet> set name=ether1 arp=proxy-arp
[admin@MikroTik] /interface/ethernet> print
Flags: R - RUNNING; S - SLAVE
```

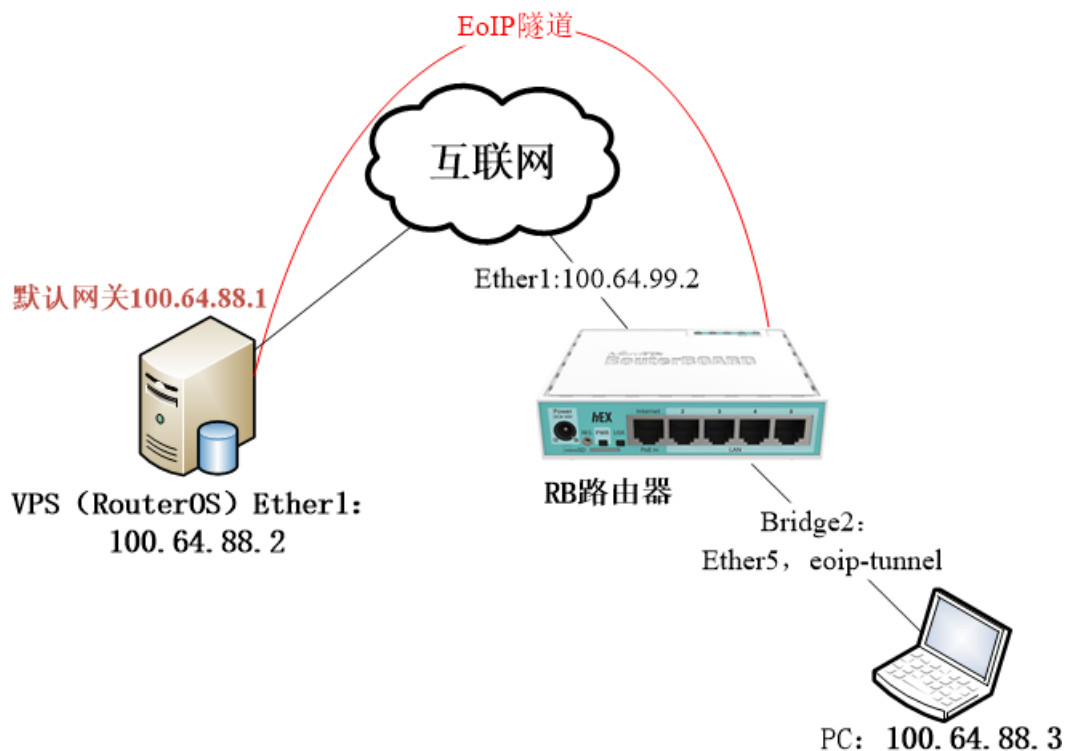
Columns: NAME, MTU, MAC-ADDRESS, ARP, SWITCH

#	NAME	MTU	MAC-ADDRESS	ARP
0 R	ether1	1500	E4:8D:8C:BA:79:F6	proxy-arp

RouterOS 端配置完成，剩下的是在 PC 端建立 PPTP 隧道拨号，PPTP 拨号完成后，将实现分配到 172.16.10.3 的 IP，并映射到 100.64.88.3 的公网 IP，这里就省略 PC 端的 PPTP 拨号操作，当然也可以是 RouterOS 或者其他网络设备。

EoIP 建立二层隧道

该场景不同点在 VPS 直接分配的公网 IP 地址，因此没有防火墙映射，如果希望将 VPS 的公网 IP 给远端使用，除了可以采用之前的 Proxy-arp 方案，还可以通过建立 EoIP 隧道，打通远端 RouterOS 路由器的二层隧道，使用二层隧道也可以选择 VxLAN 和 L2TPv3 ethernet 等，也可以使用 BCP，但效果不理想。



如上拓扑，VPS 有两个公网 IP 地址 100.64.88.2 和 100.64.88.3，

在 VPS 端和 RB 路由器端都需要创建 bridge，将网口和 EoIP-tunnel 隧道绑定起来，建立二层桥接。VPS 操作上需要将 ether1 加入到 bridge，并把 IP 地址 100.64.88.2 分配到 bridge 接口上

VPS 配置

创建 bridge，bridge 和添加 IP 地址建议在控制台完成

```
[admin@VPS] > /interface/bridge/
[admin@VPS] /interface/bridge> add name=bride1
```

将 ether1 加入 bridge1

```
[admin@VPS] /interface/bridge> port
[admin@VPS] /interface/bridge/port> add bridge=bridge1 interface=ether1
```

配置 IP 地址

```
[admin@VPS] /interface/bridge/port> /ip address/
[admin@VPS] /ip/address> add address=100.64.88.2/24 interface=bridge1
```

添加默认路由

```
[admin@ VPS] /ip/address> /ip route/
[admin@ VPS] /ip/route> add gateway=100.64.88.1
```

以上是手动添加 IP 地址和默认路由，另外 VPS 方通常使用 DHCP 分配 IP 地址，这里可以启用 DHCP-client 自动获取 IP，如果是安装的 CHR 默认配置是自带 dhcp-client 仅需要将 interface 修改为 bridge1 即可，这里不再叙述。

以上配置完成 VPS 主机网络应该可以正常访问，下面创建 EoIP 隧道

```
[admin@VPS] /ip/route> /interface/eoip/
[admin@VPS]/interface/eoip> add remote-address=100.64.99.2 tunnel-id=100 mtu=1500
name=eoip-tunnel1
```

将 EoIP 隧道加入 bridge 中

```
[admin@VPS]/interface/eoip> /interface/bridge/port
[admin@VPS] /interface/bridge/port> add bridge=bridge1 interface= eoip-tunnel1
```

RB 路由器配置

查看 IP 地址配置

```
[admin@RB] /ip/address> print
Flags: D - DYNAMIC
Columns: ADDRESS, NETWORK, INTERFACE
#   ADDRESS          NETWORK          INTERFACE
0   100.64.99.2/24     100.64.99.0     ether1
1   192.168.88.1/24   192.168.88.0    bridge1
```

RB 路由器已经创建了一个 bridge1，因此我们需要再创建一个 bridge2，将 VPS 隧道透传的 IP 和现有 192.168.88.0/24 内网区分开，分配到 ether5 口上。

```
[admin@RB] > /interface/bridge/  
[admin@RB] /interface/bridge> add name=bride2
```

将 ether5 加入 bridge2，确保 ether5 不在 bridge1 的 port 下，如果在请从 bridge1 删除掉

```
[admin@RB] /interface/bridge> port  
[admin@RB] /interface/bridge/port> add bridge=bridge2 interface=ether5
```

以上配置完成 VPS 主机网络应该可以正常访问，下面创建 EoIP 隧道

```
[admin@RB] /ip/route> /interface/eoip/  
[admin@RB]/interface/eoip> add remote-address=100.64.88.2 tunnel-id=100 mtu=1500  
name=eoip-tunnel1
```

将 EoIP 隧道加入 bridge 中

```
[admin@RB]/interface/eoip> /interface/bridge/port  
[admin@RB] /interface/bridge/port> add bridge=bridge2 interface= eoip-tunnel1
```

以上操作已经完成，eoip 隧道建立后，两端二层透传建立完成，PC 主机接入 ether5 口，可以获取远端 VPS 分配的公网 IP: 100.64.88.3/24 地址，网关仍然配置 100.64.88.1。

这种方案除了 EoIP 隧道，还可以选择 VxLAN 或 L2TPv3，另外 EoIP 和 VxLAN 隧道建立是对等的，因此两端需要公网 IP 地址，才能建立，如果 RB 一端没有公网 IP，可以先建立一个 wireguard 隧道，在利用固定的隧道 IP 地址，来运行 EoIP 或者 VxLAN 隧道。

Changelog

v7.1

- 2.3 章节, 新增基于 wireguard 的 EoIP 隧道
- 9 章节, 新增 L2TPv3 隧道
- 11 章, 新增各类应用案例
- 6 章节, 调整章节内容
- 6.4 章节, 新增 WiFi 创建 EAP PEAP 认证

v7.2

- 3.1 章节, 更新 Zerotier 基础介绍
- 1.6 章节, 更新 v7 PCC 部分内容
- 6.4 章节, 更新 WiFi 创建 EAP PEAP 认证
- 2.4 章节, 新增与 ubuntu 建立 wireguard
- 4 章节, 更新 7.4beta4 版本后的 Container 内容(docker)
- 11 章节, 新增 Device-Mode 内容

v7.3

- 2.1 章节, 更新防火墙配置介绍
- 4.5 章节, container 新增 shell 介绍
- 7.1 章节, 更新证书简单操作内容
- 8 章节, 新增 Dot1x (基于以太网 802.1x) 章节
- 4.8 章节, 新增 hAP ac3 安装 DNSMasq
- 13 章节, 新增 ROSE 存储功能章节

v7.4

- 2.5 章节, 新增 wireguard 多点组网
- 13.9 章节, 更新 v7.8beta3 在 disk 分区的内容
- 9 章节, 更新 VXLAN 内容

参考文献：

<http://wiki.mikrotik.com>

<https://help.mikrotik.com>

<http://www.routerboard.com>