试用wireguard

WireGuard简介：

官方介绍如下：

WireGuard ® 是一款极其简单但快速且现代的 VPN，采用最先进的加密技术。它的目标是比 IPsec
更快、更简单、更精简、更有用，同时避免令人头疼的问题。它的性能远高于 OpenVPN。

WireGuard 被设计为通用 VPN，可在嵌入式接口和超级计算机上运行，适合许多不同的情况。它最初针对 Linux
内核发布，现在已跨平台（Windows、macOS、BSD、iOS、Android）且可广泛部署。它目前正在大力开发中，但它可能已被视为业内最安全、最易于使用且最简单的VPN 解决方案。

我们可以用一句话概括它：

WGuard是一款可以组建虚拟私人局域网（VPN）的软件，允许用户通过公共网络（如互联网）安全地传输数据，同时保持数据的机密性和完整性。

WireGuard特点与优势：

一会具体查阅与每一个软件的区别，便于讲解

---

1. 轻便性：以Linux内核模块的形式运行，资源占用小

- **内核模块形式**：WireGuard被设计为Linux内核的一个模块，这意味着它直接在内核空间中运行，减少了用户空间和内核空间之间的上下文切换开销。
- **资源占用小**：由于代码库小且高效，WireGuard在运行时占用的系统资源（如CPU、内存）非常少，适合在资源受限的环境中部署。

---

2. 高效性：相比目前主流的IPSec、OpenVPN等协议，WireGuard的效率要更高

- **高效加密算法**：WireGuard使用Curve25519进行密钥交换，ChaCha20进行加密，Poly1305进行消息认证，这些算法在性能和安全性之间取得了良好的平衡。
- **简化协议设计**：WireGuard的协议设计非常简洁，没有复杂的配置选项和冗余的功能，这使得它在处理数据包时更加高效。

---

3. 快速性：比目前主流的VPN协议，连接速度要更快

- **快速握手**：WireGuard的握手过程非常快速，因为它只交换必要的公钥和会话密钥信息，没有复杂的协商过程。

- **低延迟**：由于协议简洁且高效，WireGuard在传输数据时具有更低的延迟，适合对实时性要求较高的应用。

---

4. 安全性：使用了更先进的加密技术

- **现代加密标准**：WireGuard使用的加密技术（如Curve25519、ChaCha20、Poly1305）都是当前被认为非常安全的加密标准。
- **完美前向保密**：WireGuard支持完美前向保密（PFS），即使长期密钥被泄露，过去的通信也不会被解密。
- **代码审计**：由于代码库小且简洁，WireGuard更容易进行安全审计和漏洞修复。

---

5. 易搭建性：部署难度相对更低

- **简洁的配置**：WireGuard的配置文件非常简洁，通常只需要几个命令行参数或配置文件条目即可完成配置。
- **跨平台支持**：WireGuard支持多种操作系统（如Linux、Windows、macOS等），且在不同平台上的部署方式相似，降低了部署难度。
- **丰富的文档和社区支持**：WireGuard拥有详细的文档和活跃的社区支持，用户可以在遇到问题时快速找到解决方案。

---

6. 隐蔽性：以UDP协议进行数据传输，比TCP协议更低调

- **UDP协议**：WireGuard使用UDP协议进行数据传输，相比TCP协议，UDP协议更加轻量级且没有连接建立和维护的开销。
- **更低调**：由于UDP协议通常用于实时性要求较高的应用（如视频流、在线游戏等），因此使用UDP协议的WireGuard在传输数据时更加低调，不易被网络监控工具识别为VPN流量。

---

7. 不易被封锁性：TCP阻断对WireGuard无效，IP被墙的情况下仍然可用

- **UDP协议的优势**：由于WireGuard使用UDP协议，它不受TCP阻断的影响。即使网络管理员或ISP试图通过阻断TCP流量来限制VPN使用，WireGuard仍然可以正常工作。
- **灵活的端口选择**：WireGuard允许用户自定义监听端口，这使得它可以在被墙的情况下通过更换端口来绕过限制。

---

8. 低能耗性：不使用时不进行数据传输，移动端更省电

- **按需传输**：WireGuard只在有数据传输需求时才会建立连接并进行数据传输，这减少了不必要的网络活动和电量消耗。
- **移动端优化**：对于移动设备来说，省电是一个非常重要的考虑因素。WireGuard的按需传输特性使得它在移动端使用时更加省电，延长了设备的续航时间。

WireGuard原理简述

WireGuard 就像是一个超级加密的“隧道”，它可以让两台或多台电脑，即使它们在不同的网络里，也能像在同一局域网内一样安全地通信。

1. 基础概念

- **Peer（对等点）**：就是参与通信的电脑或设备。
- **公钥和私钥**：就像是你家里的钥匙和锁。公钥是你可以公开分享的（比如锁），而私钥是你要保密的（比如钥匙）。只有拥有正确私钥的人才能打开对应的锁。
- **隧道接口**：WireGuard 在你的电脑上创建了一个新的网络接口，就像是你电脑上的一个“虚拟网卡”。

2. 加密通信

当你通过WireGuard发送数据时：

- **加密**：你的电脑（Peer A）会先用对方的公钥把要发送的数据包加密成一个密文。这样，只有拥有对应私钥的对方电脑（Peer B）才能解密这个数据包。
- **发送**：加密后的数据包会通过互联网发送到对方的电脑。
- **解密**：对方的电脑（Peer B）收到加密的数据包后，用自己的私钥解密，得到原始的数据。

3. 路由和连接

- **路由**：每个Peer都有一个或多个IP地址，WireGuard会设置路由规则，告诉操作系统，哪些数据包应该通过WireGuard隧道发送。
- **连接**：当你配置好两个Peer后，它们会自动尝试建立连接。一旦连接成功，它们就可以开始安全地传输数据了。

4. 安全性

- **先进加密技术**：WireGuard使用了非常先进的加密技术，确保数据在传输过程中的安全性。
- **简洁高效**：相比其他VPN技术，WireGuard更加简洁和高效，性能更好，资源占用更少。

5. 跨平台

WireGuard可以在很多不同的操作系统上运行，包括Windows、macOS、Linux，甚至是智能手机和路由器。

6. 易于配置

虽然WireGuard的工作原理听起来很复杂，但实际上它的配置非常简单。你只需要生成公钥和私钥，填写一些基本的配置信息，然后启动服务就可以了。

WireGuard功能与试用：

WireGuard 需要至少**一台具有公网 IP 的服务器**作为 VPN 服务端（中继节点），否则无法实现内网穿透。

既然校园网没有公网IP且不支持IPv6，我尝试通过在自己的同一个虚拟机既当客户端又当服务端（既当爹又当妈）的方式在本地环境中试用WireGuard：

环境搭建步骤：

1. **准备虚拟机环境**
   - 使用VMware创建两台虚拟机
   - 建议系统：Ubuntu 22
2. **软件安装和网络配置**

   安装 WireGuard

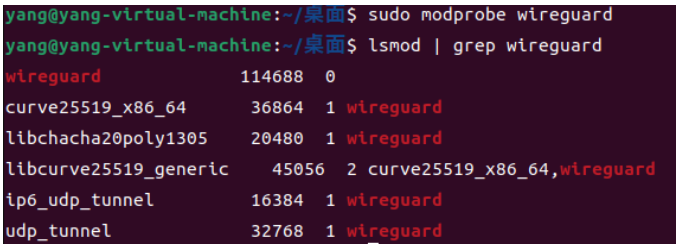
   ```
   sudo apt update
   sudo apt install wireguard resolvconf  # 安装WireGuard和DNS工具
   ```

   - `resolvconf` 用于管理 DNS（可选但推荐）
   - **内核模块**：现代 Ubuntu（20.04+）已内置 WireGuard 内核模块，无需额外编译。

   验证安装

   ```
   sudo modprobe wireguard  # 加载内核模块
   lsmod | grep wireguard  # 检查是否加载成功
   ```

   如果输出包含 `wireguard`，说明安装成功。



为什么要检查内核中是否有wireguard的存在：WireGuard 的独特之处在于它深度集成到 Linux 内核，它的核心逻辑直接运行在 Linux 内核网络栈中，这与传统 VPN（如 OpenVPN）的用户态实现有本质区别。

WireGuard 作为内核模块运行，直接处理网络数据包，避免了用户态-内核态切换的开销，因此：延迟更低（比 OpenVPN 快约 30%）

而且内核模块可以严格遵循网络协议栈的安全规范，减少潜在的攻击面。

3. **模拟公网环境**
    - 使用Host-Only网络模式创建隔离网络
    - 或使用NAT网络+端口转发模拟公网环境

配置示例：

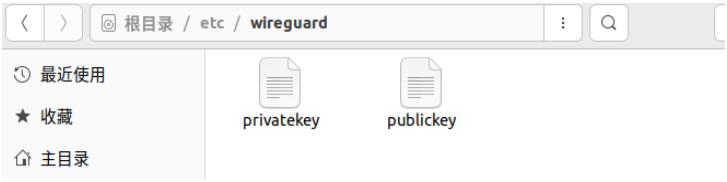
服务端虚拟机：

```
# 生成密钥
wg genkey | sudo tee /etc/wireguard/privatekey | wg
```

```
pubkey | sudo tee /etc/wireguard/publickey

# 配置文件
sudo nano /etc/wireguard/wg0.conf
```

```
[Interface]
Address = 10.0.0.1/24
ListenPort = 51820
PrivateKey = <服务端私钥
>WAZZbmPITcsAK8c0UmZmKzh52LMboJ3Q+q+NVXH6U2w=
```



成功生成私钥和公钥，根据生成的私钥和公钥进行下面配置

服务端私钥：
WAZZbmPITcsAK8c0UmZmKzh52LMboJ3Q+q+NVXH6U2w=

服务端公钥：
tmliXASxdknW0sbY3ANoQvJc5nVBu1BuRMmsQDHHbV8=

客户端虚拟机：

同样的方法生成客户端的公钥和私钥

```
#生成客户端私钥和公钥
wg genkey | sudo tee
/etc/wireguard/client_privatekey | wg pubkey | sudo
tee /etc/wireguard/client_publickey


# 配置客户端文件
sudo nano /etc/wireguard/wg_client.conf
```

客户端私钥：
gB11z+zjlNuk0/iE4X0WEa2mN2ekI7yuHG5akzHNkls=

客户端公钥：
BDj51qleUGBYCsqE/QGlInfX3hb/wH35KbI/KTqnqgg=

```
[Interface]
Address = 10.0.0.2/24
PrivateKey = <客户端私钥
>gB11z+zjlNuk0/iE4X0WEa2mN2ekI7yuHG5akzHNkls=

[Peer]
PublicKey = <服务端公钥
>tmliXASxdknW0sbY3ANoQvJc5nVBu1BuRMmsQDHHbV8=
Endpoint = 192.168.42.128:51820  # 服务端虚拟机IP
AllowedIPs = 10.0.0.0/24
```

1. 启动服务端接口

假设服务端配置文件为 `/etc/wireguard/wg0.conf`：

```
sudo wg-quick up wg0
```

- **`wg0`**：对应配置文件名 `wg0.conf` 中的接口名称。

启动客户端接口

假设客户端配置文件为 `/etc/wireguard/wg_client.conf`：

```
sudo wg-quick up wg_client
```

- **`wg_client`**：对应配置文件名 `wg_client.conf` 中的接口名称（如果配置文件中指定了 `[Interface]` 的 `Address` 等，但未指定其他名称，则默认使用配置文件名中的前缀）。

```
yang@yang-virtual-machine:~/桌面$ sudo wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
[#] ip -4 address add 10.0.0.1/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
yang@yang-virtual-machine:~/桌面$ sudo wg-quick up wg_client
[#] ip link add wg_client type wireguard
[#] wg setconf wg_client /dev/fd/63
[#] ip -4 address add 10.0.0.2/24 dev wg_client
[#] ip link set mtu 65456 up dev wg_client
```

都成功启动了

2. 验证接口状态

验证服务端接口

```
sudo wg show wg0
```

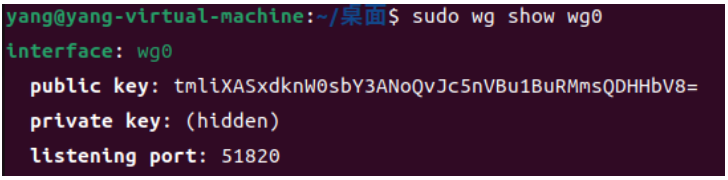
预期输出:

```
interface: wg0
  public key: <服务端公钥>
  private key: (hidden)
  listening port: 51820

peer: <客户端公钥>
  endpoint: <客户端IP（如果适用）>:<端口（如果适用）>
  allowed ips: <客户端允许访问的IP范围>
```

```
  latest handshake: <时间戳>
  transfer: <数据量> received, <数据量> sent
```

实际输出：



验证客户端接口

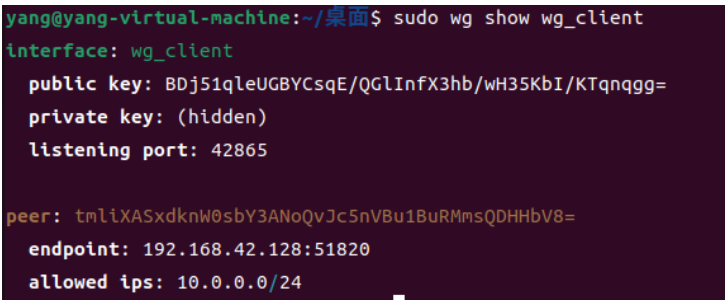
```
sudo wg show wg_client
```

预期输出：

```
interface: wg_client
  public key: <客户端公钥>
  private key: (hidden)
  listening port: <随机端口或指定端口>

peer: <服务端公钥>
  endpoint: <服务端IP>:51820
  allowed ips: <通过VPN访问的远程子网>
  latest handshake: <时间戳>
  transfer: <数据量> received, <数据量> sent
```

实际输出：



---

3. 测试连接

在服务端测试到客户端的连通性

如果客户端在 VPN 网络中的 IP 地址是 `10.0.0.2`，可以在服务端上执行：

```
ping 10.0.0.2
```

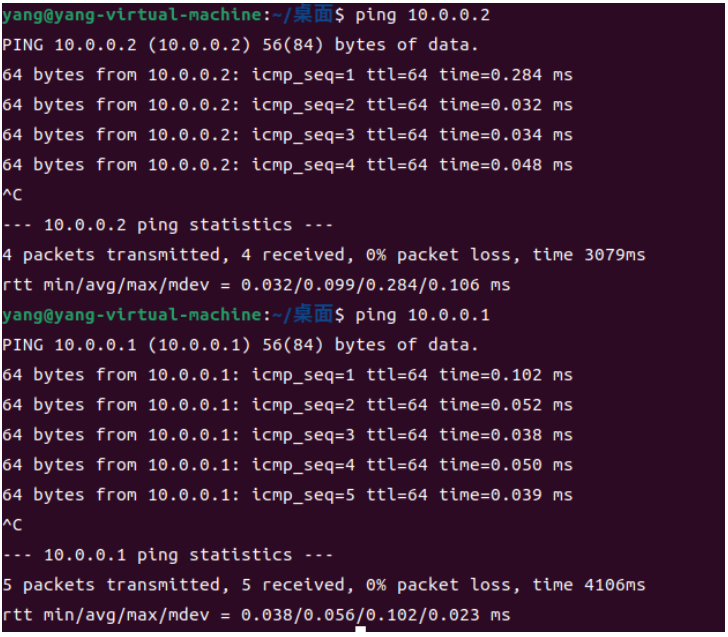
在客户端测试到服务端的连通性

如果服务端在 VPN 网络中的 IP 地址是 `10.0.0.1`，可以在客户端上执行：

```
ping 10.0.0.1
```

- **成功**：如果能够收到回复，说明 VPN 连接成功。
- **失败**：检查配置文件的 `PublicKey`、`Endpoint`、`AllowedIPs` 等是否正确，以及防火墙和网络连通性。

因为我这是客户端服务端一体的，所以我同时进行测试



都能成功收到回复！VPN设置成功了，虽然只能自己和自己玩，下面附上一张没成功的对比图：

```
yang@yang-virtual-machine:~/桌面$ ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
^C
--- 10.0.0.1 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6149ms

yang@yang-virtual-machine:~/桌面$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6174ms
```

4. 单机多namespace测试（高级点）

使用Linux网络命名空间（总体步骤）：

```
# 创建两个网络命名空间
sudo ip netns add ns1
sudo ip netns add ns2

# 在每个命名空间中启动WireGuard
sudo ip netns exec ns1 wg-quick up ./wg0-ns1.conf
sudo ip netns exec ns2 wg-quick up ./wg0-ns2.conf

# 测试连通性
sudo ip netns exec ns1 ping 10.0.0.2
```

命名空间 `ns1` 的配置文件 `wg0-ns1.conf`

```
sudo bash -c 'cat > /etc/wireguard/wg0-ns1.conf'
<<EOF
[Interface]
PrivateKey =
gB11z+zjlNuk0/iE4X0WEa2mN2ekI7yuHG5akzHNkls=
Address = 10.0.0.101/24  # 为 ns1 分配一个独立的 IP 地
址
ListenPort = 51820

[Peer]
PublicKey =
tmliXASxdknW0sbY3ANoQvJc5nVBu1BuRMmsQDHHbV8=
Endpoint = 192.168.42.128:51820
AllowedIPs = 0.0.0.0/0
PersistentKeepalive = 25
EOF
```

命名空间 `ns2` 的配置文件 `wg0-ns2.conf`

```
sudo bash -c 'cat > /etc/wireguard/wg0-ns2.conf'
<<EOF
[Interface]
PrivateKey =
gB11z+zjlNuk0/iE4X0WEa2mN2ekI7yuHG5akzHNkls=  # 如果
ns2 需要独立密钥，请生成新的密钥对
Address = 10.0.0.102/24  # 为 ns2 分配一个独立的 IP 地
址
ListenPort = 51820  # 如果在同一台机器上运行多个实例，确
```

```
保端口不冲突或使用不同接口

[Peer]
PublicKey =
tmliXASxdknW0sbY3ANoQvJc5nVBu1BuRMmsQDHHbV8=
Endpoint = 192.168.42.128:51820
AllowedIPs = 0.0.0.0/0
PersistentKeepalive = 25
EOF
```

在每个命名空间中启动 WireGuard 接口，并指定对应的配置文件。

```
# 在 ns1 中启动 WireGuard
sudo ip netns exec ns1 wg-quick up
/etc/wireguard/wg0-ns1.conf

# 在 ns2 中启动 WireGuard
sudo ip netns exec ns2 wg-quick up
/etc/wireguard/wg0-ns2.conf
```

也是在网络空间成功配置了

```
yang@yang-virtual-machine:~/桌面$ sudo ip netns exec ns1 wg-quick up /etc/wireguard/wg0-ns1.conf
[#] ip link add wg0-ns1 type wireguard
[#] wg setconf wg0-ns1 /dev/fd/63
[#] ip -4 address add 10.0.0.101/24 dev wg0-ns1
RTNETLINK answers: Network is unreachable
[#] ip link set mtu 1420 up dev wg0-ns1
[#] wg set wg0-ns1 fwmark 51820
[#] ip -4 route add 0.0.0.0/0 dev wg0-ns1 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
[#] nft -f /dev/fd/63
yang@yang-virtual-machine:~/桌面$ sudo ip netns exec ns2 wg-quick up /etc/wireguard/wg0-ns2.conf
[#] ip link add wg0-ns2 type wireguard
[#] wg setconf wg0-ns2 /dev/fd/63
[#] ip -4 address add 10.0.0.102/24 dev wg0-ns2
RTNETLINK answers: Network is unreachable
[#] ip link set mtu 1420 up dev wg0-ns2
[#] wg set wg0-ns2 fwmark 51820
[#] ip -4 route add 0.0.0.0/0 dev wg0-ns2 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
[#] nft -f /dev/fd/63
```

配置路由

确保每个命名空间的路由表正确，以便流量通过 WireGuard 接口。

- **查看路由表**：

  ```
  sudo ip netns exec ns1 ip route
  sudo ip netns exec ns2 ip route
  ```

  ```
  yang@yang-virtual-machine:~/桌面$ sudo ip netns exec ns1 ip route
  10.0.0.0/24 dev wg0-ns1 proto kernel scope link src 10.0.0.101
  yang@yang-virtual-machine:~/桌面$ sudo ip netns exec ns2 ip route
  10.0.0.0/24 dev wg0-ns2 proto kernel scope link src 10.0.0.102
  ```

  只有自己的路由

- **添加路由（如果需要）**：

  例如，如果需要访问外部网络，可能需要添加默认路由：

```
sudo ip netns exec ns1 ip route add default via
10.0.0.1  # 替换为实际的网关 IP 地址
sudo ip netns exec ns2 ip route add default via
10.0.0.1
```

连通性测试

- **从 `ns1` ping `ns2` 的 VPN IP 地址**（如果它们在同一子网内）：

```
sudo ip netns exec ns1 ping 10.0.0.102
```

- **从 `ns2` ping `ns1` 的 VPN IP 地址**：

```
sudo ip netns exec ns2 ping 10.0.0.101
```

- **测试访问外部网络**：

```
sudo ip netns exec ns1 ping 8.8.8.8  # 测试通过
VPN 访问外部网络
sudo ip netns exec ns2 ping 8.8.8.8
```

这个丢包率100%，不知道哪里没配好，暂时失败了

5. 容器化测试（Docker方案）

使用Docker-compose：

```
version: '3'
services:
  wg-server:
    image: linuxserver/wireguard
    cap_add:
      - NET_ADMIN
    ports:
      - "51820:51820/udp"
    volumes:
      - ./config:/config
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=Asia/Shanghai

  wg-client:
    image: linuxserver/wireguard
    cap_add:
      - NET_ADMIN
    volumes:
      - ./client-config:/config
    depends_on:
      - wg-server
```

测试验证方法

1. **基础连通性测试**：

```
ping 10.0.0.1  # 从客户端ping服务端
traceroute 10.0.0.1
```

2. **流量转发测试**：

```
# 在服务端启用IP转发
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward

# 测试通过VPN访问外部网络
curl --interface wg0 ifconfig.me
```

3. **性能测试**：

bash

```
iperf3 -c 10.0.0.1 -t 30 -R  # 测试带宽
ping -f 10.0.0.1 -c 1000     # 测试延迟和丢包
```

注意事项

1. 测试环境与生产环境的区别：
   - 无需考虑NAT穿透
   - 无需担心防火墙限制
   - 带宽和延迟指标仅供参考

2. 建议测试内容：

   - 配置文件语法
   - 密钥生成和管理
   - 路由规则设置
   - 多客户端配置
   - 日志查看和分析

通过这些本地化测试方案，你可以在受限网络环境中全面掌握WireGuard的配置和使用方法，为将来在实际网络环境中部署打下坚实基础。