

VPN over QUIC システム (RQST) 設定マニュアル ver 0.20

1. RQST の動作説明	3
2. RQST のインストール (Windows 版)	5
2.1 仮想 I/F の設定 (クライアント・サーバ共通)	5
2.2 仮想 I/F の追加 (サーバのみ)	6
2.3 仮想 I/F のメトリックの設定 (クライアントのみ)	6
2.4 ICS の設定 (サーバのみ)	6
2.5 RQST の動作テスト	7
3. RQST のインストール (Linux 版)	10
3.1 仮想 I/F の設定 (クライアント・サーバ共通)	10
3.2 ルーティングの設定 (クライアントのみ)	11
3.3 仮想 I/F 等の設定 (サーバのみ)	11
3.4 RQST の動作テスト	12
4. 証明書の設定	13
4.1 認証局を作成する	13
4.2 サーバの秘密鍵・証明書の作成	16
4.3 クライアントの秘密鍵・証明書の作成	17
4.4 秘密鍵・証明書のコピー	17
4.5 証明書検証モードでの RQST の起動	18
4.6 サービスとしてのサーバの起動	18
5. ソースコードの展開	19
6. バイナリのビルド (Windows 版)	20
6.1 Visual Studio Community 2019 のインストール	20
6.2 Rust のインストール	20
6.3 NASM のインストール	20
6.4 ビルドの実行	20
7. バイナリのビルド (Linux 版)	21
7.1 apt パッケージからのインストール	21
7.2 rust のインストール	21
7.3 ビルドの実行	21
8. RQST を構成する各種モジュールの説明	22
8.1 パス・フィルタ機能の仕様・設計	22
8.2 パス・グルーピング機能の仕様・設計	22
8.3 パス・セレクション機能の仕様・設計	22

1. RQST の動作説明

RQST クライアントから RQST サーバに対して QUIC 接続を確立することで、クライアント・サーバ間に仮想のネットワークを作成します (図 1)。仮想ネットワークを流れるパケットは、クライアント・サーバ間の QUIC 接続を通じて Datagram Frame でカプセル化されてクライアント・サーバに届けられます。[QUIC Multipath 拡張](#)に対応していますので、クライアントが複数のネットワークに接続されている場合には QUIC 接続の確立に用いたネットワーク (の IP アドレス) 以外のネットワーク (の IP アドレス) を用いて通信を行うことが可能です。複数のネットワークに接続している場合にどのネットワーク (の IP アドレス) を用いて送受信を行うのかは設定ファイルによって制御することが可能になっています。

仮想ネットワークは仮想 Ethernet として動作する L2 となっているので、サーバ上で DHCP サーバ等を動かすことで、クライアントの仮想 I/F に自動的に IP アドレスを付与することができます。また、サーバをルータとして動作させた上で NAT を行うようにしたり、サーバの仮想 I/F をルータと接続されている I/F とブリッジ接続を行うようにしたりすることで、クライアントがサーバ側のネットワークを通じて通信を行うようにすることができます。

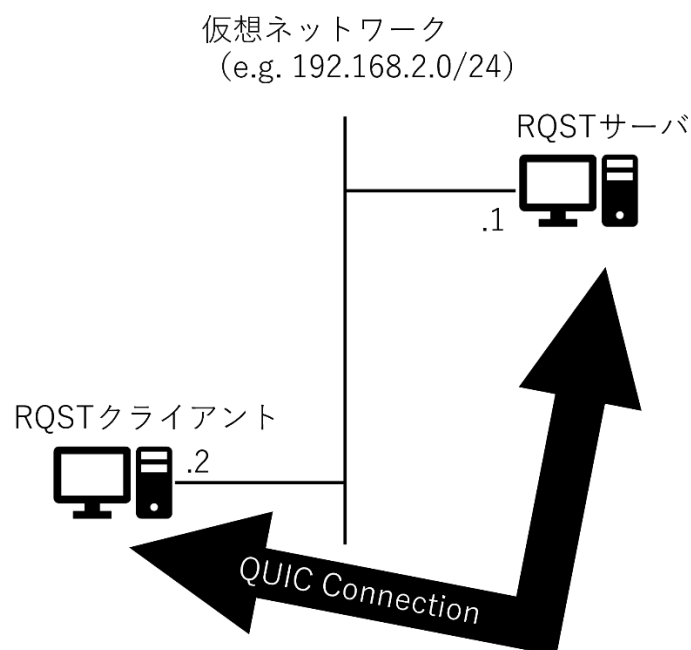


図 1

クライアント・サーバ間の QUIC 接続の確立にあたっては、互いに証明書を検証するようになっており、これにより、クライアントが偽のサーバに接続してしまったり、サーバが無関係の第三者が動作させているクライアントからの接続を許可してしまったりすることが防止されます。もっとも、動作試験のために証明書の検証を省略するモードで動作させる

ことは可能です。

なお、サーバは複数のクライアントの接続を受け付けることができますが、それぞれ独立の仮想 L2 ネットワークが作成されます。そのため、デフォルトでは複数のクライアントが接続している状態であってもクライアント同士では直接通信を行うことはできないようになっています。クライアント同士で接続するためにはサーバ上で仮想インターフェースを接続する仮想ブリッジ・インターフェースに接続することが必要です。

RQST は現在、クライアント、サーバともに Windows と Linux で動作します。Windows は 10 および 11 に対応しています。Linux での動作確認は Ubuntu 22.04 LTS で行っていますが、他のディストリビューションでも本マニュアルの記述を若干変更することでビルドや設定をすることが可能です。

2. RQST のインストール (Windows 版)

2.1 仮想 I/F の設定 (クライアント・サーバ共通)

仮想 I/F を作成できるようにするため、OpenVPN Windows 版を[公式サイト](#)からダウンロードしてインストールします。OpenVPN 自体は使用しないため、インストール後にタスクトレイにある”OpenVPN GUI”を右クリックして表示されるメニューから”設定”を選択し、その後表示される設定ウィンドウ (図 2) にある”Windows 起動時に開始”のチェックを外し、”OK”をおしてウィンドウを閉じた後、タスクトレイにある”OpenVPN GUI”を右クリックして表示されるメニューから”終了”を選択して終了させます。

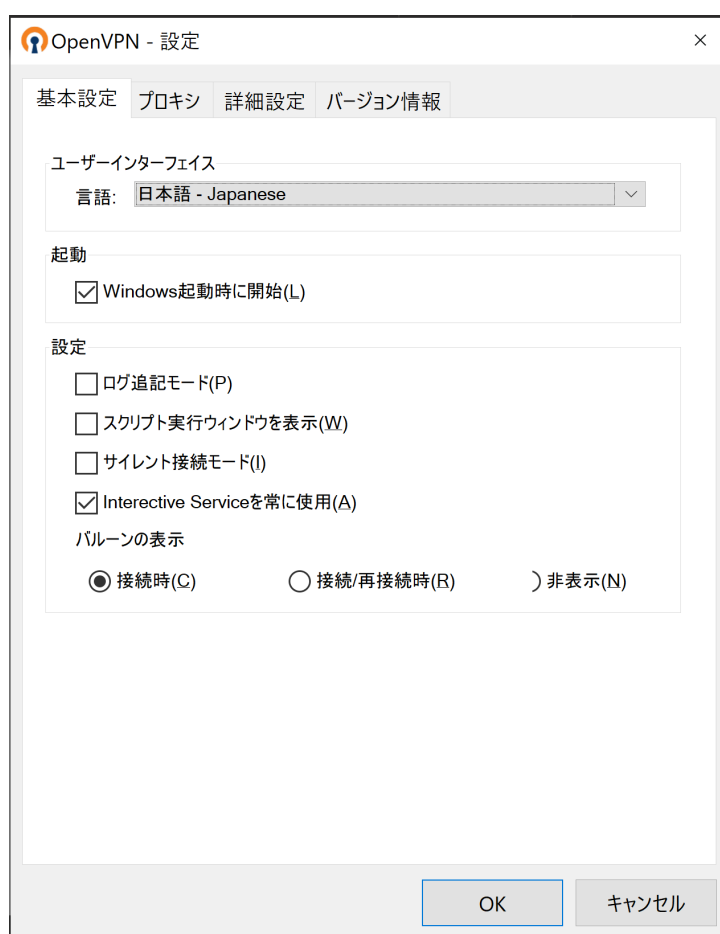


図 2

RQST の最適な動作には仮想 I/F の MTU が 1280 に設定されている必要があります。まず、コマンドプロンプトを管理者権限で実行し、次のコマンドを実行して仮想 I/F の Idx を調べます。対象の仮想 I/F の名前は、デフォルトでは、”OpenVPN Tap-Windows6”です。

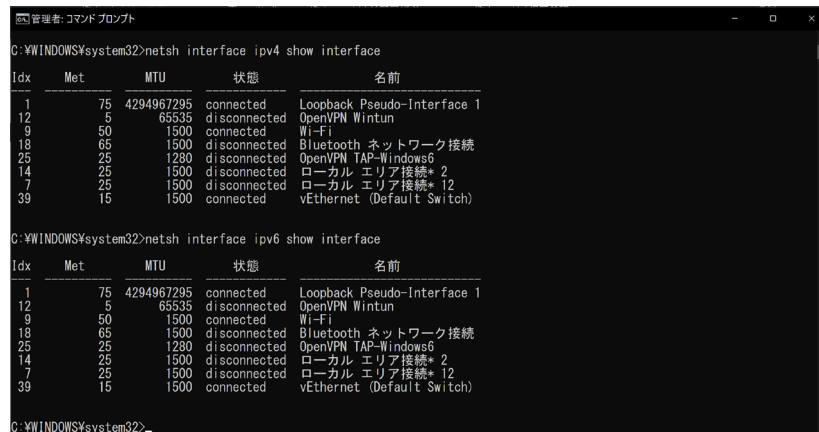
```
C:\WINDOWS\system32>netsh interface ipv4 show interface
```

その後、次のコマンドを実行して IPv4/IPv6 のそれぞれで MTU を 1280 に変更します (“25”の部分は先ほど調べた Idx の数値に変更してください)。

```
C:\WINDOWS\system32>netsh interface ipv4 set interface 25 mtu=1280
```

```
C:\WINDOWS\system32>netsh interface ipv6 set interface 25 mtu=1280
```

最後に確認のため、インターフェース一覧をもう一度表示させます。図 3 のように対象の仮想 I/F の MTU が 1280 と表示されていれば成功です。



Idx	Met	MTU	状態	名前
1	75	4294967295	connected	Loopback Pseudo-Interface 1
12	5	65535	disconnected	OpenVPN Wintun
9	50	1500	connected	Wi-Fi
18	65	1500	disconnected	Bluetooth ネットワーク接続
25	25	1280	disconnected	OpenVPN TAP-Windows6
14	25	1500	disconnected	ローカル エリア接続* 2
7	25	1500	disconnected	ローカル エリア接続* 12
39	15	1500	connected	vEthernet (Default Switch)

Idx	Met	MTU	状態	名前
1	75	4294967295	connected	Loopback Pseudo-Interface 1
12	5	65535	disconnected	OpenVPN Wintun
9	50	1500	connected	Wi-Fi
18	65	1500	disconnected	Bluetooth ネットワーク接続
25	25	1280	disconnected	OpenVPN TAP-Windows6
14	25	1500	disconnected	ローカル エリア接続* 2
7	25	1500	disconnected	ローカル エリア接続* 12
39	15	1500	connected	vEthernet (Default Switch)

図 3

2.2 仮想 I/F の追加（サーバのみ）

サーバが複数のクライアントからの接続を受け付ける場合には仮想 I/F を追加する必要があります。管理者権限のあるコマンドプロンプトで tapctl.exe を次のように実行して必要な数だけ追加します。なお、追加した後は 2.1 の手順に従って MTU を 1280 に設定する必要があります。

```
C:\WINDOWS\system32>cd C:\Program Files\OpenVPN\bin
```

```
C:\Program Files\OpenVPN\bin>tapctl create --name "TAP-Windows6 2"
```

2.3 仮想 I/F のメトリックの設定（クライアントのみ）

クライアントが仮想ネットワーク経由でサーバを通じて通信を行うためには、仮想 I/F のメトリックを小さな値に設定しておく必要があります。この設定を行うためには管理者権限のあるコマンドプロンプトで次のようにコマンドを実行します（"25"の部分は 2.1 の場合と同様に対象の仮想 I/F の Idx の数値に変更してください）。

```
C:\WINDOWS\system32> netsh interface ipv4 set interface 25 metric=1
```

```
C:\WINDOWS\system32> netsh interface ipv6 set interface 25 metric=1
```

2.4 ICS の設定（サーバのみ）

サーバ上で、ICS の設定を行ってクライアントがサーバを経由して通信できるようにします。サーバの外部との通信に用いるインターフェースのプロパティを開き、共有のページを開きます（図 4）。その後、"ネットワークのほかのユーザーに、このコンピューターのインターネット接続を通しての接続を許可する（N）"にチェックをいれ、"ホームネットワーク接続（H）"ではクライアントとの接続に用いる仮想 I/F を選択し、"OK"を押します。

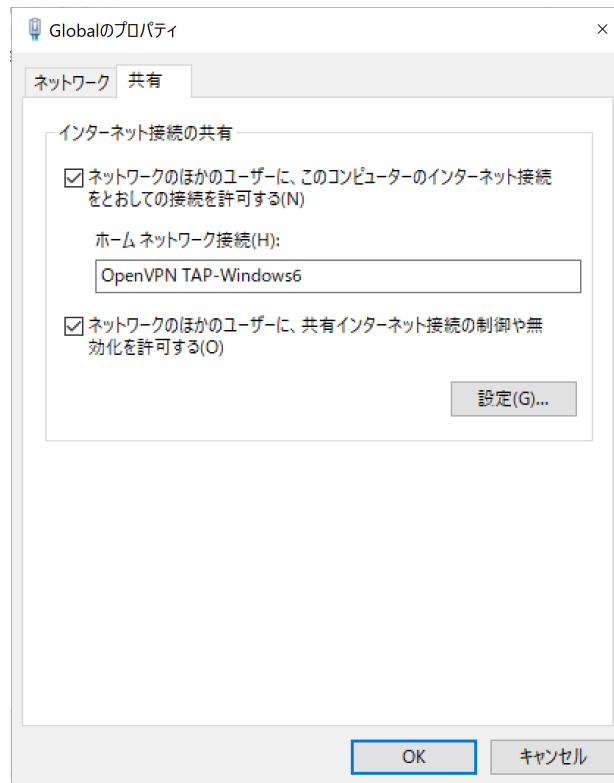


図 4

2.5 RQST の動作テスト

rqst-bin-v020.zip を適切なフォルダに展開します（以下では C:\rqst を使用）。クライアントのバイナリとサーバのバイナリはそれぞれ、C:\rqst\client\vpn-client.exe、C:\rqst\server\vpn-server.exe となります。バイナリと併せて、クライアントの設定ファイルである C:\rqst\client\rqst.toml および動作試験用の TLS 証明書と TLS 鍵も展開されます。ファイル名は、クライアント用の TLS 証明書が C:\rqst\client\client.crt、TLS 鍵が C:\rqst\client\client.key、サーバ用の TLS 証明書が C:\rqst\server\server.crt、TLS 鍵が C:\rqst\server\server.key です。

クライアントでは、設定ファイル (rqst.toml) の編集を行います。デフォルトの設定ファイルは次のようになっています。

```
## Configuration for rqst VPN client

# 使用しないローカルのネットワークの IPv4 アドレスを指定
[exclude-ipv4net]
# loopback アドレスと private アドレスは一旦すべて除外
#exclude-ipnets = ["127.0.0.0/8", "10.0.0.0/8", "172.16.0.0/12", "192.168.0.0/16"]
# loopback アドレスのみ除外。後述する exclude-iframe セクションを"inclusive"モードで
```

記述する場合に便利

```
exclude-ipnets = ["127.0.0.0/8"]
# 必要な場合は NAT 配下で用いている Private アドレスを含める
#include-ipnets = ["192.168.1.0/24", "192.168.179.0/24"]

# 使用しないローカルのネットワークの IPv6 アドレスを指定
[exclude-ipv6net]
# loopback アドレスと link local アドレスは除外
exclude-ipnets = [ "::1/128", "fe80::/64"]

# 列挙されたローカルネットワークのインターフェースに属するアドレス以外は除外する
[exclude-ifname]
kind = "inclusive"
# Windows の場合、インターフェースの名前は GUID で指定する。
ifnames = ["{9061C62B-AFA7-4C35-B94D-FF47070DF9A3}", "{0798E57C-C9B4-447C-AE66-9E344D5DF9D6}"]
# Linux の場合
#ifnames = ["eth0"]

# 列挙されたローカルネットワークのインターフェースに属するアドレスは除外する
#[exclude-ifname]
#kind = "exclusive"
#ifnames = ["{4B1E624A-2DEA-4205-8F5F-596A45BECF24}"]

# 使用しないローカルのネットワークを指定（そのネットワークに属する IPv4・IPv6 アドレスが除外される）
[exclude-iftyp]
# 指定できるのは、"metered"または"not-metered"
iftypes = ["metered"]

# 以下で 3 つのパスグループを定義する
[[path-groups]]
# name はこのパスグループの名前を指定。tunnels エントリで用いる
name = "localnet"
ipnets = ["192.168.1.0/24", "192.168.179.0/24", "2001:db8::/32"]
```



```

[[path-groups]]
name = "localnet-not-metered"
ipnets = ["192.168.1.0/24", "192.168.179.0/24", "2001:db8::/32"]
# "localnet"のアドレス範囲に加えて、ローカルネットワークが従量課金でないことを指
定
iftypes = ["not-metered"]

[[path-groups]]
name = "any-metered"
# 従量課金のネットワークであればどのアドレスでも含める
iftypes = ["metered"]

# DSCP フィールドが 0 または 40 である IP パケットは"localnet-not-metered"パスグル
ープで送受信
[[tunnels]]
dscp = [0, 40]
path-group = "localnet-not-metered"

```

動作に用いるローカルネットワークの指定にはインターフェースの名前を用いるのが便利です。どのインターフェースがどのような名前を付与されているのかは、次のコマンドで調べることができます。

```
C:¥rqst>wmic nicconfig list full
```

パス・グループの設定やどのパケットをどのパス・グループで送受信するのかの設定は、それぞれ、path-groups と tunnels セクションで設定します。具体的な記述の仕方は設定ファイルの内容およびコメントを参照してください。

設定が終わったら、サーバ上で通常のコマンドプロンプトを開き、動作確認のため、次のコマンドを入力して vpn-server.exe を証明書の検証を行わないモードで実行します。-v をつけるとログを標準エラーに出力します。つけないとプログラムを置いているフォルダに起動ごとにログファイルを作成し、そこに出力します。他の細かい挙動等は--help をつけて起動すると説明が表示されますので参考にしてください。

```
C:¥rqst¥server>vpn-server -d -v
```

続いてクライアント上でもコマンドプロンプトを開き、次のコマンドを入力して vpn-client.exe を証明書の検証を行わないモードで実行します。なお、接続先のサーバ名は適切に変更してください。

```
C:¥rqst¥client>vpn-client -d -v rqst://rqst.example.com:3456
```

これで RQST が動作している限り、クライアントの通信はサーバを経由して行われることになります。プログラムを終了するには Ctrl+C を入力してください。

3. RQST のインストール (Linux 版)

Linux 版はクライアント・サーバともにソースコードからビルドして動作させます。ソースコードの展開方法は 5 において後述します。以下は、ソースコードを /home/seera/rqst に展開しているという前提で記述します。

3.1 仮想 I/F の設定 (クライアント・サーバ共通)

ネットワークインターフェースの設定を行うプログラムが起動するようになっているかどうかをチェックします。”enable”と表示された場合は次のコマンドは実行する必要はありません。

```
$ systemctl is-enabled systemd-networkd.service
```

プログラムを起動するように設定を変更します。

```
$ sudo systemctl enable systemd-networkd.service
```

ネットワークインターフェースの設定を行うプログラムの準備ができれば続いて仮想インターフェースを作成するための設定を行います。/etc/systemd/network/tap0.netdev という名前で次のような内容のファイルを新しく作成します。User の部分は RQST を動作させるユーザー名を指定します。

```
[NetDev]
Name=tap0
Kind=tap
[Tap]
User=seera
```

引き続きネットワークインターフェースの設定を行う netplan の設定ファイル (ここでは /etc/netplan/01-network-manager-all.yaml) を編集します。次のような内容に変更してください。

```
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    tap0:
      mtu: 1280
```

設定ファイルの準備ができれば、次の 2 つのコマンドを実行します。

```
$ sudo netplan apply
```

```
$ sudo ip link set dev tap0 up
```

3.2 ルーティングの設定（クライアントのみ）

次のコマンドを入力してサーバのアドレス宛の静的経路を設定します（ここではサーバのアドレスは 203.0.113.1 とします。IPv6 も用いる場合は適宜設定してください）。

```
$ sudo ip route add 203.0.113.1 via 172.22.48.1 dev eth0
```

3.3 仮想 I/F 等の設定（サーバのみ）

netplan の設定ファイルを次のように編集します。

```
network:
  version: 2
  renderer: NetworkManager
  ethernets:
    tap0:
      mtu: 1280
  bridges:
    vpn:
      addresses: [10.255.255.1/24]
      mtu: 1280
```

編集が終わったら、次のコマンドを実行してブリッジ・インターフェースを作成・設定します。

```
$ sudo netplan apply
$ sudo ip link set dev tap0 master vpn
```

引き続き、DHCP サーバの設定を行います。次のコマンドで apt パッケージからインストールできます。

```
$ sudo apt install isc-dhcp-server
```

その後、/etc/default/isc-dhcp-server ファイルを開き、

```
INTERFACESv4=""
```

を

```
INTERFACESv4="vpn"
```

に変更し、さらに、/etc/dhcp/dhcpd.conf に次の記述を加えます。

```
subnet 10.255.255.0 netmask 255.255.255.0 {
  range 10.255.255.101 10.255.255.200;
  option domain-name-servers 8.8.8.8;
  option routers 10.255.255.1;
}
```

設定ファイルの変更が終わったら、次のコマンドを実行して起動設定および起動を行います。

す。

```
$ sudo systemctl enable isc-dhcp-server
```

```
$ sudo systemctl restart isc-dhcp-server
```

さらにサーバーを NAT ルーターとして動作させる変更を行います。まず、`/etc/sysctl.conf` ファイルのうち

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

を

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

に変更し、その後に次のコマンドを実行します。

```
$ sudo sysctl -p
```

次に NAT の設定を次のように追加します。

```
$ sudo iptables -t nat -A POSTROUTING -s 10.255.255.0/24 -o eth0 -j MASQUERADE
```

最後に NAT 設定を永続化するためのプログラムをインストール等しておきます。

```
$ sudo apt install iptables-persistent
```

内容は `/etc/iptables/rules.v4` に保存されます。

3.4 RQST の動作テスト

クライアントの設定ファイルを編集します。編集の仕方は Windows 版の記述を参考にしてください。準備ができたなら、サーバ上の `/home/seera/rqst/rqst` で次のコマンドを実行します。

```
$ cargo run --release --bin vpn-server -- -d -v --cert src/cert.crt --key src/cert.key
```

引き続き、クライアント上で次のようにコマンドを実行します。

```
$ cargo run --release --bin vpn-client -- -d -v --cert src/cert.crt --key src/cert.key --config  
src/rqst.toml rqst://rqst.example.com:4433
```

最後に、クライアント上の別のターミナルで DHCP リクエストを出力します。これで `tap0` インターフェイスにアドレスが付与されます。

```
$ sudo dhclient tap0
```

4. 証明書の設定

動作試験が終わったら、認証局を作成し、クライアント・サーバのそれぞれに秘密鍵・証明書を新しく発行します。これらの秘密鍵・証明書を用いると検証を行うモードで RQST を実行できるようになります。以下の記述は Windows 上での手順についてのものですが、Linux 上でも同様の手順で作成が可能です。

4.1 認証局を作成する

[Download Free OpenSSL for Microsoft Windows \(fireDaemon.com\)](https://www.fireDaemon.com/tech/openssl/) から Win 用の OpenSSL バイナリを入手し、インストールします。その後、openssl.exe のあるフォルダを環境変数 Path に追加します。

続いて、CA という名前のフォルダを作り（以下では C:¥CA とします）、その中で以下の内容で setup.bat ファイルを作ります。

```
@mkdir certs
@mkdir crl
@mkdir newcerts
@mkdir private
@mkdir csr
@type nul > index.txt
@type nul > crlnumber
@echo 1000 > serial
```

そして、通常のコマンドプロンプトを開き次のコマンドを入力して setup.bat を実行します。

```
C: ¥CA>setup.bat
```

また、以下の内容を C:¥CA¥openssl.cfg というファイル名で保存します。

```
[ ca ]
default_ca      = CA_default

[ CA_default ]
dir              = C:¥¥CA
certs            = C:¥¥CA¥¥certs
crl_dir          = C:¥¥CA¥¥crl
database         = C:¥¥CA¥¥index.txt
new_certs_dir    = C:¥¥CA¥¥newcerts
serial          = C:¥¥CA¥¥serial
crlnumber        = C:¥¥CA¥¥crlnumber
crl              = C:¥¥CA¥¥crl.pem
```

```

certificate      = C:YYCAYYcertsYYca.crt
private_key      = C:YYCAYYprivateYYca.key
name_opt         = ca_default
cert_opt         = ca_default
crl_extensions   = crl_ext
default_days     = 365
default_crl_days = 30
default_md       = sha256
preserve         = no
policy           = policy_match

[ policy_match ]
countryName      = match
stateOrProvinceName = optional
organizationName = optional
organizationalUnitName = optional
commonName       = supplied
emailAddress     = optional

[ policy_anything ]
countryName      = optional
stateOrProvinceName = optional
localityName     = optional
organizationName = optional
organizationalUnitName = optional
commonName       = supplied
emailAddress     = optional

[ req ]
default_bits     = 2048
distinguished_name = req_distinguished_name
x509_extensions  = v3_ca
string_mask      = utf8only
default_md       = sha256

[ req_distinguished_name ]

```

countryName	= Country Name (2 letter code)
countryName_default	= JP
countryName_min	= 2
countryName_max	= 2
stateOrProvinceName	= State or Province Name (full name)
stateOrProvinceName_default	= Okayama
localityName	= Locality Name (eg, city)
localityName_default	= Okayama
0.organizationName	= Organization Name (eg, company)
0.organizationName_default	= SEERA Networks Inc.
organizationalUnitName	= Organizational Unit Name (eg, section)
organizationalUnitName_default	= Okayama Office
commonName	= Common Name (e.g. server FQDN or YOUR name)
commonName_default	= RQST CA
commonName_max	= 64
emailAddress	= Email Address
emailAddress_default	= rqst@example.com
emailAddress_max	= 64

[server_cert]

basicConstraints	= CA:FALSE
nsCertType	= server
nsComment	= "OpenSSL Generated Server Certificate"
subjectKeyIdentifier	= hash
authorityKeyIdentifier	= keyid,issuer:always
keyUsage	= critical, digitalSignature, keyEncipherment
extendedKeyUsage	= serverAuth

[v3_ca]

subjectKeyIdentifier	= hash
authorityKeyIdentifier	= keyid:always,issuer
basicConstraints	= critical,CA:true
keyUsage	= critical, digitalSignature, cRLSign, keyCertSign

[v3_intermediate_ca]

```

subjectKeyIdentifier    = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints        = critical,CA:true, pathlen:0
keyUsage                = critical, digitalSignature, cRLSign, keyCertSign

[ crl_ext ]
authorityKeyIdentifier = keyid:always

```

次に、コマンドプロンプトに次のコマンドを入力し、認証局の秘密鍵を作成します。実行した後は秘密鍵を暗号化するパスワードを入力し、Enter キーを押します。

```
C: ¥CA>openssl genrsa -aes256 -passout stdin -out private¥ca.key 4096
```

そして、コマンドプロンプトに次のコマンドを入力し、認証局の証明書を発行します。実行後はまず秘密鍵のパスワードの入力を待つ状態になっているので、パスワードを入力して Enter キーを押します。その後、認証局の情報を適宜入力します。

```
C: ¥CA>openssl req -config openssl.cfg -key private¥ca.key -passin stdin -new -x509 -days
9999 -sha256 -extensions v3_ca -out certs¥ca.crt
```

4.2 サーバの秘密鍵・証明書の作成

以下の内容を（*_default を適宜変更した上で）C:¥CA¥certreq.cfg に保存します。

```

[ req ]
default_bits          = 2048
distinguished_name = req_distinguished_name

[ req_distinguished_name ]
countryName            = Country Name (2 letter code)
countryName_default    = JP
stateOrProvinceName    = State or Province Name (full name)
stateOrProvinceName_default = Okayama
localityName           = Locality Name (eg, city)
localityName_default   = Okayama
organizationName       = Organization Name (eg, company)
organizationName_default = SEERA Networks Inc.
commonName             = Common Name (e.g. server FQDN or YOUR name)
commonName_default     = rqst.example.com

```

コマンドプロンプトに以下のコマンドを入力し、サーバの秘密鍵を作成します。サーバの秘密鍵にはパスワードを設定しないので入力の必要はありません。


```
C: ¥CA>openssl genrsa -out private¥server.key 2048
```

コマンドプロンプトに以下を入力し、サーバの証明書要求を発行します。なお、commonName には RQST のサーバを運用する予定の FQDN のどれかを指定し、subjectAltName には RQST のサーバを運用する予定の FQDN をすべて列挙します（ここでは、rqst.example.com と rqst4.example.com の両方を用いる例とします）。

```
C: ¥CA>openssl req -config certreq.cfg -addext "subjectAltName=DNS:
rqst.example.com,DNS: rqst4.example.com" -key private¥server.key -new -sha256 -out
csr¥server.csr
```

次に subjectAltName に指定した内容に応じて次のような内容で san.txt を作成します。

```
subjectAltName = DNS: rqst.example.com,DNS: rqst4.example.com
```

そしてコマンドプロンプトに以下を入力し、サーバ証明書を発行します。実行後はまず秘密鍵のパスワードの入力を待つ状態になっているので、パスワードを入力して Enter キーを押します。

```
C: ¥CA>openssl ca -config openssl.cfg -passin stdin -days 5000 -notext -md sha256 -in
csr¥server.csr -extfile san.txt -out certs¥server.crt
```

4.3 クライアントの秘密鍵・証明書の作成

コマンドプロンプトに以下のコマンドを入力し、クライアントの秘密鍵を作成します。クライアントの秘密鍵にはパスワードを設定しないので入力はありません。

```
C: ¥CA>openssl genrsa -out private¥client.key 2048
```

次にコマンドプロンプトに以下を入力し、クライアントの証明書要求を発行します。なお、commonName には何を入力してもかまいません。

```
C:¥CA>openssl req -config certreq.cfg -key private¥client.key -new -sha256 -out
csr¥client.csr
```

そしてコマンドプロンプトに以下を入力し、クライアント証明書を発行します。実行後はまず秘密鍵のパスワードの入力を待つ状態になっているので、パスワードを入力して Enter キーを押します。

```
C: ¥CA>openssl ca -config openssl.cfg -passin stdin -days 5000 -notext -md sha256 -in
csr¥client.csr -out certs¥client.crt
```

3.4 秘密鍵・証明書のコピー

サーバ上で認証局の証明書 (ca.crt)、サーバの秘密鍵・証明書 (server.key, server.crt) をバイナリと同じフォルダ (C:¥rqst¥server) に上書きコピーします。また、クライアント上で認証局の証明書 (ca.crt)、クライアントの秘密鍵・証明書 (client.key, client.crt) をバイ

ナリと同じフォルダ (C:¥rqst¥client) に上書きコピーします。

4.4 証明書検証モードでの RQST の起動

サーバ上でコマンドプロンプトを開き、次を入力して実行します。

```
C:¥rqst¥server>vpn-server -v
```

続いてクライアント上でもコマンドプロンプトを開き、次のコマンドを入力して実行します。なお、接続先のサーバ名はサーバ証明書作成時に指定した FQDN のどれかにする必要があります。

```
C:¥rqst¥client>vpn-client -v vpn://rqst.example.com:3456
```

4.5 サービスとしてのサーバの起動

RQST サーバは次のように設定することで Windows のサービスとしても実行することができます (Windows 版のみ)。

まず、管理者権限のあるコマンドプロンプトを開き、次のように入力してサービスとしてインストールします。

```
C:¥rqst¥server>vpn-server.exe install
```

そしてコマンドプロンプトに次のように入力をして、サービスを起動します。

```
C:¥rqst¥server>sc start quic_vpn_server
```

停止する場合は次のように入力します。

```
C:¥rqst¥server>sc stop quic_vpn_server
```

5. ソースコードの展開

RQST および quiche の拡張は Rust で書かれています。ソースコードは rqst-src-v020.tar.gz に含まれているパッチを [Cloudflare の quiche](#) および [オリジナルの RQST](#) のソースコードに適用して作成します。以下では Linux 上で、/home/seera/rqst 以下に展開する前提で説明します。

まず、quiche のソースコードを github から入手します。

```
$ git clone --recursive https://github.com/cloudflare/quiche.git
```

次に以下のコマンドを実行して Multipath 拡張を有効にします。

```
$ cd quiche
$ git fetch origin pull/1310/head:multipath
$ git checkout multipath
```

このとき、Multipath 拡張のソースコードがパッチ作成時のままなら、次のコマンドは実行不要です。git log の結果の先頭が以下のようになっていれば作成時のままになっています。

```
commit 9350e386390f2d95dd22bf7d213b72a730fbc005 (HEAD -> multipath)
Author: Quentin De Coninck <quentin.deconinck@uclouvain.be>
Date:   Wed Mar 1 10:50:44 2023 +0100

    no need for public API `find_scid_seq()`
```

もし先頭が上の内容と異なっていれば次のコマンドを実行してください。

```
$ git checkout 9350e386390f2d95dd22bf7d213b72a730fbc005
```

パッチを適用するために次のコマンドを実行します。

```
$ patch -p1 < ../quiche.diff
```

引き続き、RQST のソースコードも github から入手します。

```
$ cd ..
$ git clone https://github.com/cityroam/rqst.git
```

パッチを適用するために次のコマンドを実行します。

```
$ cd rqst
$ patch -p1 < ../rqst.diff
```

6. バイナリのビルド（Windows 版）

6.1 Visual Studio Community 2019 のインストール

Visual Studio Community 2019 を [MS のサイト](#) からダウンロードしてインストールします（動作確認を行ったのは 2019 ですが、2022 でもビルドできると考えられます）。インストールの際、ワークロードでは"C++によるデスクトップ開発"を選択し、個別のコンポーネントでは"Windows 用 C++ Cmake ツール"を追加し、言語パックでは"日本語"と"英語"を追加します。

6.2 Rust のインストール

rustup を [公式サイト](#) からダウンロードし、実行してインストールを行います。

6.3 NASM のインストール

NASM を [公式サイト](#) からダウンロードしてインストールします。また、バイナリのインストール先を環境変数の Path に追加します。

6.4 ビルドの実行

スタートメニューから"x64 Native Tools Command Prompt for VS 2019"を選択して開発環境のコマンドプロンプトを立ち上げます。その後、RQST のソースコードを展開したフォルダに移動した後、次のコマンドを入力してビルドを行います。

```
C:¥ rqst-src>cargo build --release
```

7. バイナリのビルド（Linux 版）

7.1 apt パッケージからのインストール

次のコマンドを実行します。

```
$ sudo apt install curl binutils clang cmake
```

7.2 rust のインストール

次のコマンドを実行します。

```
$ curl https://sh.rustup.rs -sSf | sh
```

7.3 ビルドの実行

/home/seera/rqst/rqst で次のコマンドを実行します。

```
$ cargo build-release
```

8. RQST を構成する各種モジュールの説明

8.1 パス・フィルター機能の仕様・設計

ソースコードは/home/seera/rqst/rqst/src/ifwatch/以下に存在します。IfWatcherExt という名前のモジュールになっています。内部的に、if-watch ([公式サイト](#)) と if-addr ([公式サイト](#)) という名前の crate を使用しています。

ローカルネットワークの IP アドレスを通知する機能を持ちます。IP アドレス、属するインターフェースの名前、従量課金かどうかの情報でローカル・ネットワークの IP アドレスをフィルターし、通知できないようにする機能があります。

Windows 版のみ、対象の IP アドレスの属するローカル・ネットワークが従量課金かどうかを判定する機能が実装されています。Linux 版は現在のところ実装されていません。

8.2 パス・グルーピング機能の仕様・設計

ソースコードは/home/seera/rqst/quiche/quiche/src/以下の Cloudflare が作成している QUIC スタックである quiche のソースコードに含まれています。API の仕様および動作等は以下の通りです。

quiche::Connection::insert_group() は、パス・グループの ID (64bit) とそのグループに追加させようとしているパスの 4 つ組 (ローカルネットワークの IP アドレス、ローカルポート番号 (UDP)、リモートネットワークの IP アドレス、リモートポート番号 (UDP)) を与えて呼び出します。この API はクライアントのみが呼ぶことができます。クライアントは insert_group() が呼ばれると PATH_SET_GROUP Frame を作成して、サーバに通知します。

PATH_SET_GROUP Frame にはパス・グループの ID とそのグループに属するパスの ID (サーバがそのパスでの QUIC パケット送信時に用いる DCID のシーケンス番号) が含まれます。

quiche::Connection::remove_group() はパス・グループの ID とパスの 4 つ組を与えると、そのパスをパス・グループから削除します。その後、パス・グループの情報を更新するため、クライアントは PATH_SET_GROUP フレームをサーバに送信します。

なお、パス・グルーピング機能において、パス・グループの ID0 番は QUIC コネクションに属するすべてのパスを含む特殊な扱いになっています。

8.3 パス・セレクション機能の仕様・設計

ソースコードは/home/seera/rqst/quiche/quiche/src/以下の Cloudflare が作成している QUIC スタックである quiche のソースコードに含まれています。API の仕様および動作等は以下の通りです。

クライアントおよびサーバは、Stream Frame または Datagram Frame の送信時に使われ

るパスを限定するために、Stream ごと、または Datagram 送信ごとにパス・グループを指定することができます。

Stream ごとに指定する API は `quiche::Connection::stream_group()` です。引数には Stream ID とパス・グループの ID を指定します。この API を実行するとその Stream の Stream Frame はすべて指定されたパス・グループに属するパスから送信されます。

Datagram 送信ごとに指定する API は `quiche::Connection::dgram_send_group()` です。引数には送信するデータとパス・グループの ID を指定します。この API を実行すると指定されたそのデータを送信する Datagram Frame は指定されたパス・グループに属するパスから送信されます。

以上の API によるパス・グループの指定はアプリケーション層でネゴシエーションすることが想定されています。なお、quiche では Stream ごとの送信に優先順位を指定することができ、優先度が高い Stream は先に送信されます。その際、その Stream にパス・グループが設定されていればそのグループに属するパスを用いて送信されることになります。