

















 master ▾





Go to file

 Code ▾

		
 examples		
 nogomobile		
 payloads		
 tests		
 .gitignore		
 .travis.yml		
 LICENSE		
 Makefile		
 README.md		
 client.go		
 config.go		
 doc.go		
 error.go		
 go.mod		
 go.sum		

## About

Go implementation of NKN client and wallet

-  Readme
-  Apache-2.0 license
-  Activity
-  Custom properties
-  59 stars
-  9 watching
-  28 forks

Report repository

## Releases 29

 **v1.4.8** Latest  
on May 29

[+ 28 releases](#)

## Packages

No packages published

## Contributors 9



message.go		
multiclient.go		
nanopay.go		
resolver.go		
rpc.go		
type.go		
util.go		
wallet.go		



## Languages



README Apache-2.0 license

# nkn-sdk-go

reference license Apache 2.0 go report A+ build passing  
 PRs welcome

Go implementation of NKN client and wallet SDK. The SDK consists of a few components:

- [NKN Client](#): Send and receive data for free between any NKN clients regardless their network condition without setting up a server or relying on any third party services. Data are end to end encrypted by default. Typically you might want to use [multiclient](#) instead of using client directly.
- [NKN MultiClient](#): Send and receive data using multiple NKN clients concurrently to improve reliability and latency. In addition, it supports session mode, a reliable streaming protocol similar to TCP based on [ncp](#).

- [NKN Wallet](#): Wallet SDK for [NKN blockchain](#). It can be used to create wallet, transfer token to NKN wallet address, register name, subscribe to topic, etc.

Advantages of using NKN client/multiclient for data transmission:

- Network agnostic: Neither sender nor receiver needs to have public IP address or port forwarding. NKN clients only establish outbound (websocket) connections, so Internet access is all they need. This is ideal for client side peer to peer communication.
- Top level security: All data are end to end authenticated and encrypted. No one else in the world except sender and receiver can see or modify the content of the data. The same public key is used for both routing and encryption, eliminating the possibility of man in the middle attack.
- Decent performance: By aggregating multiple overlay paths concurrently, multiclient can get ~100ms end to end latency and 10+mbps end to end session throughput between international devices.
- Everything is free, open source and decentralized. (If you are curious, node relay traffic for clients for free to earn mining rewards in NKN blockchain.)

## Documentation

---

Full documentation can be found at [GoDoc](#).

## Usage

---

### Client

NKN Client provides low level p2p messaging through NKN network. For most applications, it's more suitable to use

multiclient (see [multiclient](#) section below) for better reliability, lower latency, and session mode support.

Create a multiclient with a generated key pair and default config:

```
account, err := NewAccount(nil)
client, err := NewMultiClientV2(account, "", nil)
```



Or with an identifier (used to distinguish different clients sharing the same key pair):

```
client, err := NewMultiClientV2(account, "any s
```



Get client key pair:

```
fmt.Println(account.Seed(), account.PubKey())
```



Create a multiclient using an existing secret seed:

```
seed, err := hex.DecodeStrings("039e481266e5a05:
account, err := NewAccount(seed)
client, err := NewMultiClientV2(account, "any s
```



Secret seed should be kept **SECRET!** Never put it in version control system like here.

By default the client will use bootstrap RPC server (for getting node address) provided by NKN. Any NKN full node can serve as a bootstrap RPC server. To create a multiclient using customized bootstrap RPC server:

```
conf := &ClientConfig{SeedRPCServerAddr: NewStr:
client, err := NewMultiClientV2(account, "any s
```



Get client NKN address, which is used to receive data from other clients:

```
fmt.Println(client.Address())
```



Listen for connection established:

```
<- client.OnConnect.C  
fmt.Println("Connection opened.")
```



Send text message to other clients:

```
response, err := client.Send(NewStringArray("and
```



You can also send byte array directly:

```
response, err := client.Send(NewStringArray("and
```



Or publish a message to a specified topic (see wallet section for subscribing to topics):

```
client.Publish("topic", []byte("hello world!"),
```



Receive data from other clients:

```
msg := <- client.OnMessage.C  
fmt.Println("Receive message from", msg.Src + "  
msg.Reply([]byte("response"))
```



Get 100 subscribers of specified topic starting from 0 offset, including those in tx pool (fetch meta):

```
subscribers, err := client.GetSubscribers("topic  
fmt.Println(subscribers.Subscribers, subscriber:
```



Get subscription:

```
subscription, err := client.GetSubscription("topic")  
fmt.Printf("%+v\n", subscription) // &{Meta:meta:topic:topic}
```

## Session

Multiclient supports a reliable transmit protocol called session. It will be responsible for retransmission and ordering just like TCP. It uses multiple clients to send and receive data in multiple path to achieve better throughput. Unlike regular multiclient message, no redundant data will be sent (other than retransmission when packets are lost).

Any multiclient can start listening for incoming session where the remote address match any of the given regexp:

```
multiclient, err := NewMultiClientV2(...)  
// Accepting any address, equivalent to multiclient.Listen()  
err = multiclient.Listen(nil)  
// Only accepting pubkey 25d660916021ab1d182fb61...  
err = multiclient.Listen(NewStringArray("25d660916021ab1d182fb61..."))  
// Only accepting address alice.25d660916021ab1d182fb61...  
err = multiclient.Listen(NewStringArray("^alice.25d660916021ab1d182fb61..."))
```

Then it can start accepting sessions:

```
session, err := multiclient.Accept()
```

Multiclient implements `net.Listener` interface, so one can use it as a drop-in replacement when `net.Listener` is needed, e.g. `http.Serve`.

On the other hand, any multiclient can dial a session to a remote NKN address:

```
session, err := multiclient.Dial("another nkn address")
```

Session implements `net.Conn` interface, so it can be used as a drop-in replacement when `net.Conn` is needed:

```
buf := make([]byte, 1024)
n, err := session.Read(buf)
n, err := session.Write(buf)
```



## Wallet

Create wallet SDK:

```
account, err := NewAccount(nil)
wallet, err := NewWallet(account, &nkn.WalletCo
```



By default the wallet will use RPC server provided by `nkn.org`. Any NKN full node can serve as a RPC server. To create a wallet using customized RPC server:

```
conf := &WalletConfig{
    Password: "password",
    SeedRPCServerAddr: NewStringArray("https://ip
}
wallet, err := NewWallet(account, conf)
```



Export wallet to JSON string, where sensitive contents are encrypted by password provided in config:

```
walletJSON, err := wallet.ToJSON()
```



Load wallet from JSON string, note that the password needs to be the same as the one provided when creating wallet:

```
walletFromJSON, err := nkn.WalletFromJSON(walle
```



Verify whether an address is a valid NKN wallet address:

```
err := nkn.VerifyWalletAddress(wallet.Address())
```

Verify password of the wallet:

```
err := wallet.VerifyPassword("password")
```

Query asset balance for this wallet:

```
balance, err := wallet.Balance()
if err == nil {
    log.Println("asset balance:", balance.String())
} else {
    log.Println("query balance fail:", err)
}
```

Query asset balance for address:

```
balance, err := wallet.BalanceByAddress("NKNxxx: ")
```

Transfer asset to some address:

```
txnHash, err := wallet.Transfer(account.WalletAddress, amount)
```

Open nano pay channel to specified address:

```
// you can pass channel duration (in unit of block)
// after expired new channel (with new id) will be created
// this means that receiver need to claim old channel
np, err := wallet.NewNanoPay(address, "0", 4320)
```

Increment channel balance by 100 NKN:

```
txn, err := np.IncrementAmount("100")
```



Then you can pass the transaction to receiver, who can send transaction to on-chain later:

```
txnHash, err := wallet.SendRawTransaction(txn) 
```


Register name for this wallet:

```
txnHash, err = wallet.RegisterName("somename", 1) 
```


Delete name for this wallet:

```
txnHash, err = wallet.DeleteName("somename", nil) 
```

Subscribe to specified topic for this wallet for next 100 blocks:


```
txnHash, err = wallet.Subscribe("identifier", 100) 
```

Unsubscribe from specified topic:

```
txnHash, err = wallet.Unsubscribe("identifier", 1) 
```

## Compiling to iOS/Android native library

This library is designed to work with [gomobile](#) and run natively on iOS/Android without any modification. You can use `gomobile bind` to compile it to Objective-C framework for iOS:

```
gomobile bind -target=ios -ldflags "-s -w" github.com/nknorg/nkn-sdk-go 
```

and Java AAR for Android:

```
gomobile bind -target=android -ldflags "-s -w" github.com/nknorg/nkn-sdk-go 
```

It's recommended to use the latest version of gomobile that supports go modules.

## Contributing

### Can I submit a bug, suggestion or feature request?

Yes. Please open an issue for that.

### Can I contribute patches?

Yes, we appreciate your help! To make contributions, please fork the repo, push your changes to the forked repo with signed-off commits, and open a pull request here.

Please sign off your commit. This means adding a line "Signed-off-by: Name " at the end of each commit, indicating that you wrote the code and have the right to pass it on as an open source patch. This can be done automatically by adding `-s` when committing:

```
git commit -s
```



## Community

[Terms](#) [Privacy](#) [Security](#) [Status](#) [Docs](#) [Contact](#) [Manage cookies](#) [Do not share my personal information](#)



© 2024 GitHub, Inc.