
water

`water` is a native Go library for TUN/TAP interfaces.

`water` is designed to be simple and efficient. It

- wraps almost only syscalls and uses only Go standard types;
- exposes standard interfaces; plays well with standard packages like `io`, `bufio`, etc..
- does not handle memory management (allocating/destructing slice). It's up to user to decide whether/how to reuse buffers.

`water/waterutil` has some useful functions to interpret MAC frame headers and IP packet headers. It also contains some constants such as protocol numbers and ethernet frame types.

See <https://github.com/songgao/packets> for functions for parsing various packets.

Supported Platforms

- Linux
- Windows (experimental; APIs might change)
- macOS (point-to-point TUN only)

Installation

```
1 go get -u github.com/songgao/water
2 go get -u github.com/songgao/water/waterutil
```

Documentation

<http://godoc.org/github.com/songgao/water>

Example

TAP on Linux:

```
1 package main
2
3 import (
4     "log"
5
```

```

6     "github.com/songgao/packets/ethernet"
7     "github.com/songgao/water"
8 )
9
10 func main() {
11     config := water.Config{
12         DeviceType: water.TAP,
13     }
14     config.Name = "0_0"
15
16     ifce, err := water.New(config)
17     if err != nil {
18         log.Fatal(err)
19     }
20     var frame ethernet.Frame
21
22     for {
23         frame.Resize(1500)
24         n, err := ifce.Read([]byte(frame))
25         if err != nil {
26             log.Fatal(err)
27         }
28         frame = frame[:n]
29         log.Printf("Dst: %s\n", frame.Destination())
30         log.Printf("Src: %s\n", frame.Source())
31         log.Printf("Ethertype: % x\n", frame.Ethertype())
32         log.Printf("Payload: % x\n", frame.Payload())
33     }
34 }

```

This piece of code creates a **TAP** interface, and prints some header information for every frame. After pull up the `main.go`, you'll need to bring up the interface and assign an IP address. All of these need root permission.

```
1 sudo go run main.go
```

In a new terminal:

```
1 sudo ip addr add 10.1.0.10/24 dev 0_0
2 sudo ip link set dev 0_0 up
```

Wait until the output `main.go` terminal, try sending some ICMP broadcast message:

```
1 ping -c1 -b 10.1.0.255
```

You'll see output containing the IPv4 ICMP frame:

```
1 2016/10/24 03:18:16 Dst: ff:ff:ff:ff:ff:ff
2 2016/10/24 03:18:16 Src: 72:3c:fc:29:1c:6f
3 2016/10/24 03:18:16 Ethertype: 08 00
```

```
4 2016/10/24 03:18:16 Payload: 45 00 00 54 00 00 40 00 40 01 25 9f 0a 01
    00 0a 0a 01 00 ff 08 00 01 c1 08 49 00 01 78 7d 0d 58 00 00 00 00 a2
    4c 07 00 00 00 00 00 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1
    f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
    36 37
```

TUN on macOS

```
1 package main
2
3 import (
4     "log"
5
6     "github.com/songgao/water"
7 )
8
9 func main() {
10     ifce, err := water.New(water.Config{
11         DeviceType: water.TUN,
12     })
13     if err != nil {
14         log.Fatal(err)
15     }
16
17     log.Printf("Interface Name: %s\n", ifce.Name())
18
19     packet := make([]byte, 2000)
20     for {
21         n, err := ifce.Read(packet)
22         if err != nil {
23             log.Fatal(err)
24         }
25         log.Printf("Packet Received: % x\n", packet[:n])
26     }
27 }
```

Run it!

```
1 $ sudo go run main.go
```

This is a point-to-point only interface. Use `ifconfig` to see its attributes. You need to bring it up and assign IP addresses (apparently replace `utun2` if needed):

```
1 $ sudo ifconfig utun2 10.1.0.10 10.1.0.20 up
```

Now send some ICMP packets to the interface:

```
1 $ ping 10.1.0.20
```

You'd see the ICMP packets printed out:

```
1 2017/03/20 21:17:30 Interface Name: utun2
2 2017/03/20 21:17:40 Packet Received: 45 00 00 54 e9 1d 00 00 40 01 7d 6
   c 0a 01 00 0a 0a 01 00 14 08 00 ee 04 21 15 00 00 58 d0 a9 64 00 08
   fb a5 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c
   1d 1e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32
   33 34 35 36 37
```

Caveats

1. Only Point-to-Point user TUN devices are supported. TAP devices are *not* supported natively by macOS.
2. Custom interface names are not supported by macOS. Interface names are automatically generated serially, using the `utun<#>` naming convention.

TAP on Windows:

To use it with windows, you will need to install a tap driver, or OpenVPN client for windows.

It's compatible with the Linux code.

```
1 package main
2
3 import (
4     "log"
5
6     "github.com/songgao/packets/ethernet"
7     "github.com/songgao/water"
8 )
9
10 func main() {
11     ifce, err := water.New(water.Config{
12         DeviceType: water.TAP,
13     })
14     if err != nil {
15         log.Fatal(err)
16     }
17     var frame ethernet.Frame
18
19     for {
20         frame.Resize(1500)
21         n, err := ifce.Read([]byte(frame))
22         if err != nil {
23             log.Fatal(err)
24         }
25         frame = frame[:n]
```

```
26     log.Printf("Dst: %s\n", frame.Destination())
27     log.Printf("Src: %s\n", frame.Source())
28     log.Printf("Ethertype: % x\n", frame.Ethertype())
29     log.Printf("Payload: % x\n", frame.Payload())
30 }
31 }
```

Same as Linux version, but you don't need to bring up the device by hand, the only thing you need is to assign an IP address to it.

```
1 go run main.go
```

It will output a lot of lines because of some windows services and dhcp. You will need admin right to assign IP.

In a new cmd (admin right):

```
1 # Replace with your device name, it can be achieved by ifce.Name().
2 netsh interface ip set address name="Ethernet 2" source=static addr
   =10.1.0.10 mask=255.255.255.0 gateway=none
```

The `main.go` terminal should be silenced after IP assignment, try sending some ICMP broadcast message:

```
1 ping 10.1.0.255
```

You'll see output containing the IPv4 ICMP frame same as the Linux version.

Specifying interface name If you are going to use multiple TAP devices on the Windows, there is a way to specify an interface name to select the exact device that you need:

```
1     ifce, err := water.New(water.Config{
2         DeviceType: water.TAP,
3         PlatformSpecificParams: water.PlatformSpecificParams{
4             ComponentID: "tap0901",
5             InterfaceName: "Ethernet 3",
6             Network: "192.168.1.10/24",
7         },
8     })
```

TODO

- tuntaposx for TAP on Darwin

Alternatives

tuntap: <https://code.google.com/p/tuntap/>