原 荐 使用Golang实现简单Ping过程

Micooz 发表于 2年前 阅读 9142 收藏 148 点赞 26 评论 13

收藏

摘要: Ping的基本原理是发送和接受ICMP请求回显报文，利用Go语言可以轻松实现这一过程，较之C/C++语言，Go的实现过程十分简单，效率和安全性也十分完美，本文将一步一步带着大家来实现Ping

引言

关于各种语言实现Ping已经是大家喜闻乐见的事情了，网络上利用Golang实现Ping已经有比较详细的代码示例，但大多是仅仅是实现了Request过程，而对Response的回显内容并没有做接收。而Ping程序不仅仅是发送一个ICMP，更重要的是如何接收并进行统计。

下面是网络上几篇关于Ping的实现代码：

https://github.com/paulstuart/ping/blob/master/ping.go

http://blog.csdn.net/gophers/article/details/21481447

http://blog.csdn.net/laputa73/article/details/17226337

本文借鉴了第二个链接里面的部分代码。

准备

1. 安装最新的Go

   由于Google被墙的原因，如果没有VPN的话，就到这里下载：

   http://www.golangtc.com/download

2. 使用任意文本编辑器，或者LiteIDE会比较方便编译和调试，下面是LiteIDE的下载地址

   https://github.com/visualfc/liteide

编码

要用到的package：

```
import (
        "bytes"
        "container/list"
        "encoding/binary"
        "fmt"
        "net"
        "os"
        "time"
)
```

1. 使用Golang提供的net包中的相关函数可以快速构造一个IP包并自定义其中一些关键参数，而不需要再自己手动填充IP报文。

2. 使用encoding/binary包可以轻松获取结构体struct的内存数据并且可以规定字节序（这里要用网络字节序BigEndian），而不需要自己去转换字节序。之前的一片文中使用boost，还要自己去实现转换过程，详见：关于蹭网检查的原理及实现

3. 使用container/list包，方便进行结果统计
4. 使用time包实现耗时和超时处理

ICMP报文struct：

```
type ICMP struct {
        Type        uint8
        Code        uint8
        Checksum    uint16
        Identifier  uint16
        SequenceNum uint16
}
```

Usage提示：

```
arg_num := len(os.Args)

        if arg_num < 2 {
                fmt.Print(
                        "Please runAs [super user] in [terminal].\n",
                        "Usage:\n",
                        "\tgoping url\n",
                        "\texample: goping www.baidu.com",
                )
                time.Sleep(5e9)
                return
        }
```

注意这个ping程序，包括之前的ARP程序都必须使用系统最高权限执行，所以这里先给出提示，使用time.Sleep(5e9)，暂停5秒，是为了使双击执行者看到提示，避免控制台一闪而过。

关键net对象的创建和初始化：

```
var (
                icmp     ICMP
                laddr    = net.IPAddr{IP: net.ParseIP("0.0.0.0")}
                raddr, _ = net.ResolveIPAddr("ip", os.Args[1])
        )

        conn, err := net.DialIP("ip4:icmp", &laddr, raddr)

        if err != nil {
                fmt.Println(err.Error())
                return
        }

        defer conn.Close()
```

引言

net.DialIP表示生成一个IP报文，版本号是v4，协议是ICMP（这里字符串ip4:icmp会把IP报文的协议字段设为1表示ICMP协议），

源地址laddr可以是0.0.0.0也可以是自己的ip，这个并不影响ICMP的工作。

目的地址raddr是一个URL，这里使用Resolve进行DNS解析，注意返回值是一个指针，所以下面的DialIP方法中参数表示没有取地址符。

这样一个完整的IP报文就装配好了，我们并没有去操心IP中的其他一些字段，Go已经为我们处理好了。

通过返回的conn *net.IPConn对象可以进行后续操作。

defer conn.Close() 表示该函数将在Return时被执行，确保不会忘记关闭。

下面需要构造ICMP报文了：

```
icmp.Type = 8
        icmp.Code = 0
        icmp.Checksum = 0
        icmp.Identifier = 0
        icmp.SequenceNum = 0

        var buffer bytes.Buffer
        binary.Write(&buffer, binary.BigEndian, icmp)
```

```
        icmp.Checksum = CheckSum(buffer.Bytes())
        buffer.Reset()
        binary.Write(&buffer, binary.BigEndian, icmp)
```

仍然非常简单，利用binary可以把一个结构体数据按照指定的字节序读到缓冲区里面，计算校验和后，再读进去。

检验和算法参考上面给出的URL中的实现：

```
func CheckSum(data []byte) uint16 {
        var (
                sum    uint32
                length int = len(data)
                index  int
        )
        for length > 1 {
                sum += uint32(data[index])<<8 + uint32(data[index+1])
                index += 2
                length -= 2
        }
        if length > 0 {
                sum += uint32(data[index])
        }
        sum += (sum >> 16)

        return uint16(^sum)
}
```

下面是Ping的Request过程，这里仿照Windows的ping，默认只进行4次：

```
fmt.Printf("\n正在 Ping %s 具有 0 字节的数据:\n", raddr.String())
        recv := make([]byte, 1024)

        statistic := list.New()
        sended_packets := 0

        for i := 4; i > 0; i-- {

                if _, err := conn.Write(buffer.Bytes()); err != nil {
                        fmt.Println(err.Error())
                        return
                }
                sended_packets++
                t_start := time.Now()

                conn.SetReadDeadline((time.Now().Add(time.Second * 5)))
                _, err := conn.Read(recv)

                if err != nil {
                        fmt.Println("请求超时")
                        continue
                }

                t_end := time.Now()

                dur := t_end.Sub(t_start).Nanoseconds() / 1e6

                fmt.Printf("来自 %s 的回复: 时间 = %dms\n", raddr.String(), dur)

                statistic.PushBack(dur)

                //for i := 0; i < recvsize; i++ {
                //      if i%16 == 0 {
                //              fmt.Println("")
                //      }
                //      fmt.Printf("%.2x ", recv[i])
                //}
                //fmt.Println("")

        }
```

"具有0字节的数据"表示ICMP报文中没有数据字段，这和Windows里面32字节的数据的略有不同。

conn.Write方法执行之后也就发送了一条ICMP请求，同时进行计时和计次。

conn.SetReadDeadline可以在未收到数据的指定时间内停止Read等待，并返回错误err，然后判定请求超时。否则，收到回应后，计算来回所用时间，并放入一个list方便后续统计。

注释部分内容是我在探索返回数据时的代码，读者可以试试看Read到的数据是哪个数据包的？

统计工作将在循环结束时进行，这里使用了defer其实是希望按了Ctrl+C之后能return执行，但是控制台确实不给力，直接给杀掉了。。

```
defer func() {
                fmt.Println("")
                //信息统计
                var min, max, sum int64
                if statistic.Len() == 0 {
                        min, max, sum = 0, 0, 0
                } else {
                        min, max, sum = statistic.Front().Value.(int64), statistic.Front().Value.(int64), int6
                }

                for v := statistic.Front(); v != nil; v = v.Next() {

                        val := v.Value.(int64)

                        switch {
                        case val < min:
                                min = val
                        case val > max:
                                max = val
                        }

                        sum = sum + val
                }
                recved, losted := statistic.Len(), sended_packets-statistic.Len()
                fmt.Printf("%s 的 Ping 统计信息: \n  数据包: 已发送 = %d, 已接收 = %d, 丢失 = %d (%.1f%% 丢失), \n
                        raddr.String(),
                        sended_packets, recved, losted, float32(losted)/float32(sended_packets)*100,
                        min, max, float32(sum)/float32(recved),
                )
        }()
```

统计过程注意类型的转换和格式化就行了。

全部代码就这些，执行结果大概是这个样子的：

注意每次Ping后都没有"休息"，不像Windows或者Linux的会停顿几秒再Ping下一轮。

收尾

Golang实现整个Ping比我想象中的还要简单很多，静态编译速度是十分快速，相比C而言，你需要更多得了解底层，甚至要从链路层开始，你需要写更多更复杂的代码来完成相同的工作，但究其根本，C语言仍然是鼻祖，功不可没，很多原理和思想都要继承和发展，这一点Golang做的很好。

分类：Go 字数：1611

标签： Golang ICMP Ping

Micooz

个人站长 成都

+ 关注

粉丝 38 | 博文 33 | 码字总数 51094 | 作品 1

相关博客

ping 原理与ICMP协议
xiangxw
320 0

关于icmp协议的理解以及ping命令的实现
胡栋梁
338 1

利用WireShark分析由Ping产生的Internet 控制报文协议(ICMP)
Micooz
467 0

评论 (13)

Ctrl+Enter 发表评论

包菜兄
1楼 2014/08/12 08:54
赞，回家跟着实现一下看看

ninja911
2楼 2014/08/12 09:05
作者写得好，文字思路清晰，我们读者看到就很爽……感谢

yunfound
3楼 2014/08/12 10:32
非常易懂的文章，简单却又详细！赞！！！
希望作者继续发表一些类似的文章。

jemygraw
4楼 2014/08/12 12:42
> 引用来自“包菜兄”的评论
> 赞，回家跟着实现一下看看

实现好了，分享一下。

Silvery
5楼 2014/08/12 12:52
"相比C而言，你需要更多得了解底层，甚至要从链路层开始" -- 无语。

C代码：https://github.com/Silveryfu/Projects-SFU/blob/master/CMPT471/RawSocket.cpp

Micooz
6楼 2014/08/12 13:09
> 引用来自“Silvery”的评论
> "相比C而言，你需要更多得了解底层，甚至要从链路层开始" -- 无语。
>
> C代码：https://github.com/Silveryfu/Projects-SFU/blob/master/CMPT471/RawSocket.cpp

这个代码的确是从链路层开始封装的：ether_header->ip_header ->icmp_header，上述go代码仅涉及icmp_header

yunfound

7楼 2014/08/12 14:51

> 引用来自“Silvery”的评论
> "相比C而言，你需要更多得了解底层，甚至要从链路层开始" -- 无语。
>
> C代码：https://github.com/Silveryfu/Projects-SFU/blob/master/CMPT471/RawSocket.cpp

太复杂了，没看懂！！！

LongRaindy

8楼 2014/08/13 00:21

对于我这种go新手实用啊，谢了。

Silvery

9楼 2014/08/13 00:27

> 引用来自“Silvery”的评论
> "相比C而言，你需要更多得了解底层，甚至要从链路层开始" -- 无语。
>
> C代码：https://github.com/Silveryfu/Projects-SFU/blob/master/CMPT471/RawSocket.cpp

> 引用来自“Micooz”的评论
> 这个代码的确是从链路层开始封装的：ether_header->ip_header ->icmp_header，上述go代码仅涉及icmp_header

哈哈, sorry 我这才意识到自己误会你原文的意思了。

yangjh_chs

10楼 2014/08/16 22:30

谢谢，很清晰

L_world_125

11楼 2015/02/23 21:49

golang是个什么鬼。

Micooz

12楼 2015/02/23 22:49

> 引用来自“L_world_125”的评论
> golang是个什么鬼。

Go语言

OSC首席键客

13楼 2015/03/30 00:00

ttl怎么搞？