

Golang으로 제작한 Network Scanner

발표자 : 이용희
insaner@me.com

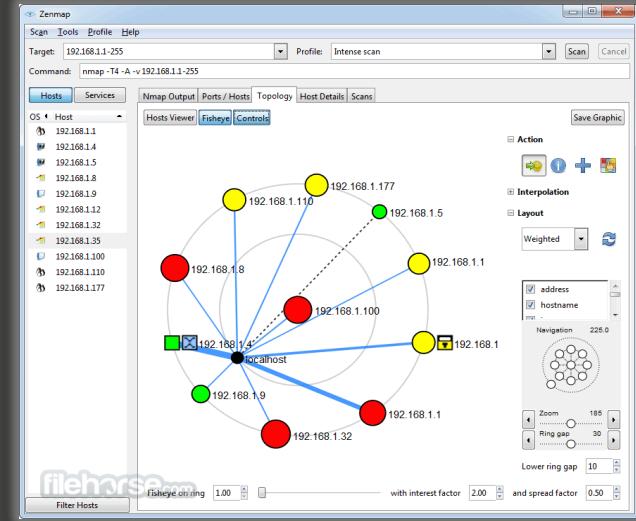
Network Scanning?

```
pi@raspberrypi ~ $ nmap 192.168.1.1-5

Starting Nmap 6.00 ( http://nmap.org ) at 2013-12-24 10:00 UTC
Nmap scan report for 192.168.1.1
Host is up (0.0055s latency).
Not shown: 995 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    filtered ssh
23/tcp    filtered telnet
80/tcp    open  http
8081/tcp  filtered blackice-icecap

Nmap scan report for 192.168.1.4
Host is up (0.0033s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh

Nmap done: 5 IP addresses (2 hosts up) scanned in 16.81 seconds
pi@raspberrypi ~ $
```

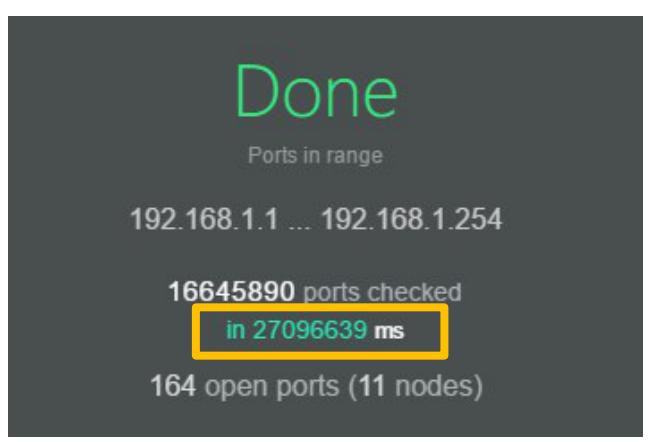
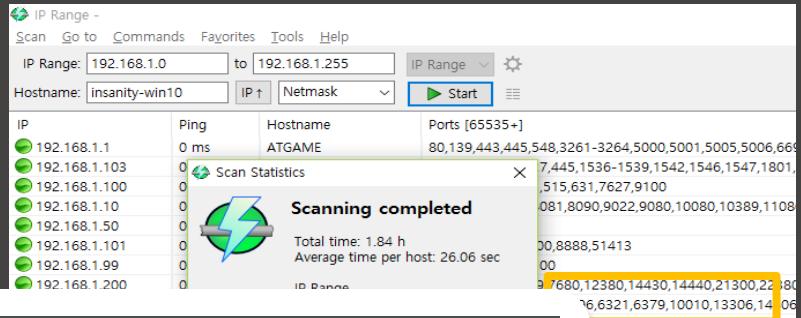


시스템 성능 관리..
네트워크 보안 관리..
APM..



네트워크 상에 무엇이 존재하는지,
관리 대상이 무엇인지
아는 것이 그 시작

Network Scanner



최대 8시간 까지 걸리는 기존 제품들



목표

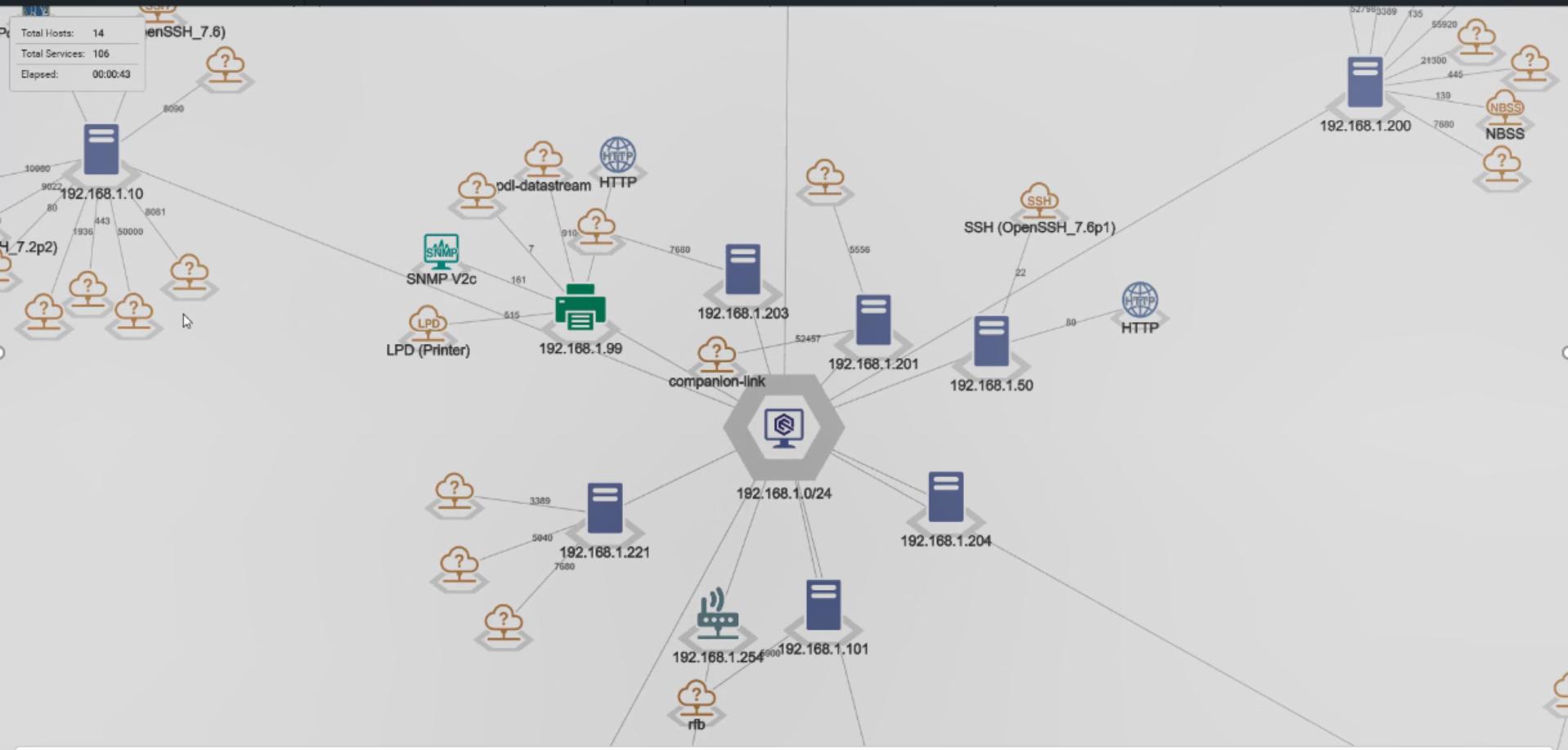
1. 모든 호스트와 그 정체를 찾아내자.
2. 열려있는 모든 포트를 찾아내자.
3. 포트가 열린 목적을 알아내자.
4. 이 모든 과정을 최대한 빠르게 수행하자.

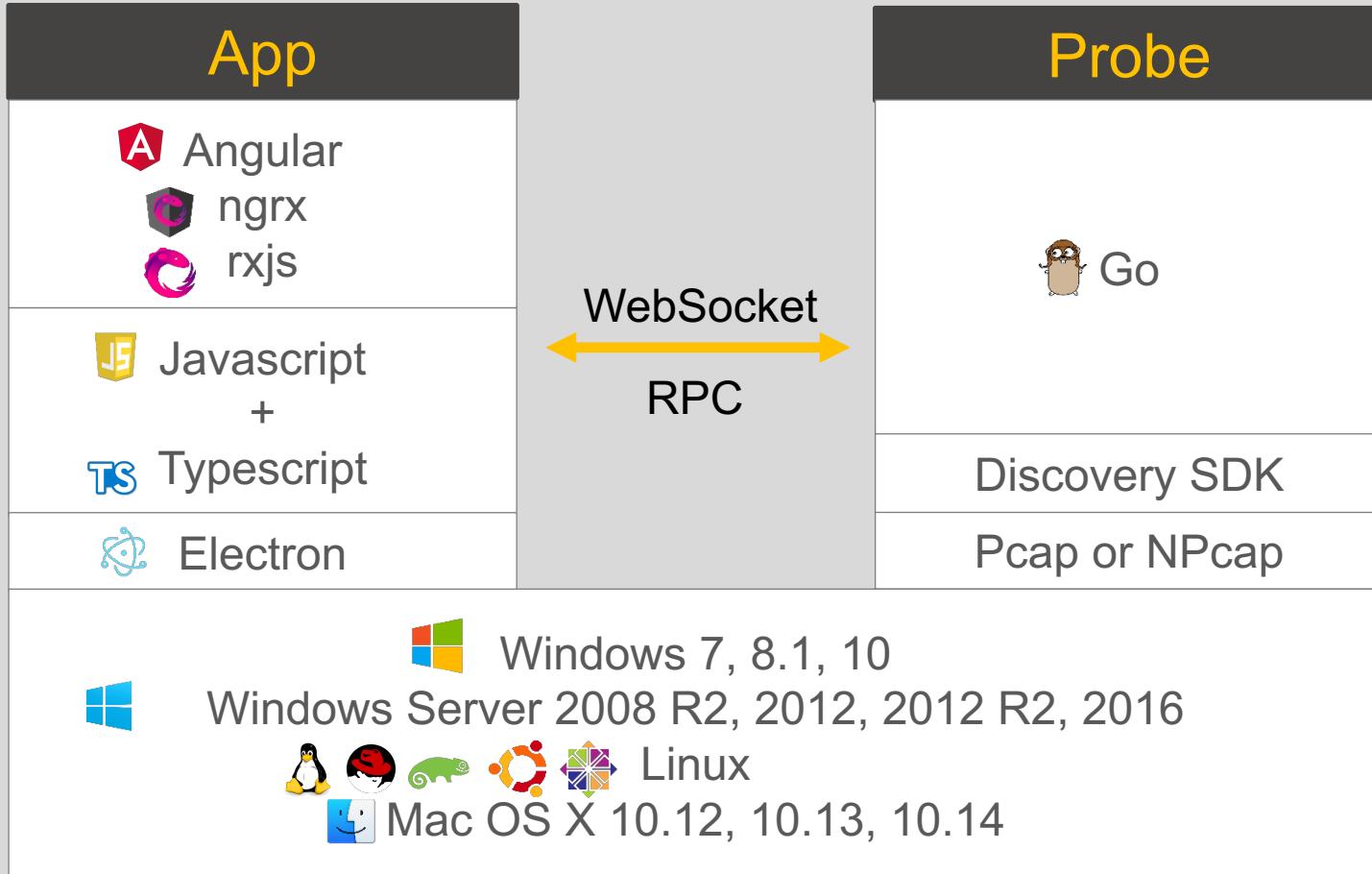
192.168.1.0/24

enp3s0

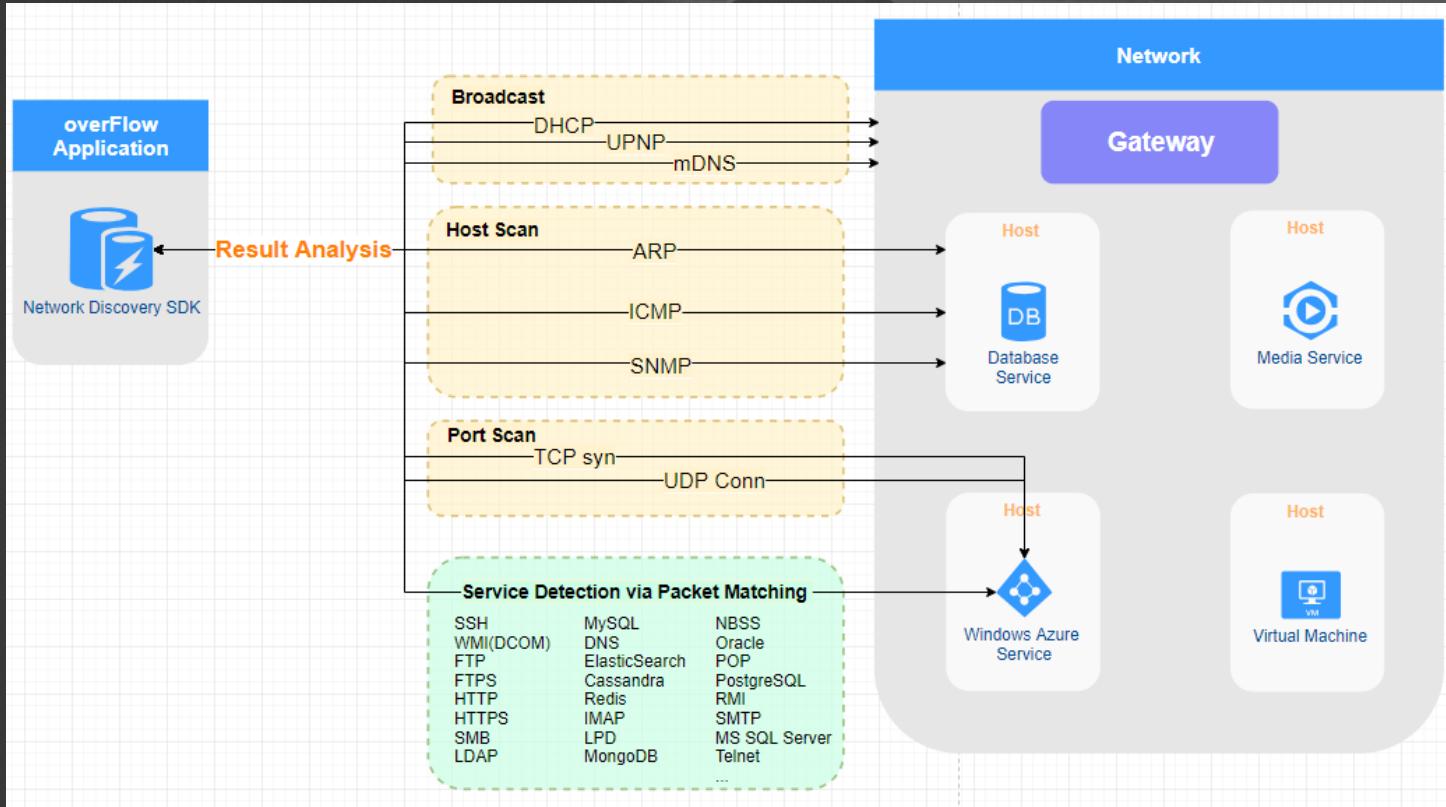
IP V4 (192.168.1.0 - 192.168.1.255)
Port TCP, UDP (1 - 65535)

Privileged Mode





Network Process Flow



Raw Socket Programming

7 Layers of the OSI Model

Application

- End User layer
- HTTP, FTP, IRC, SSH, DNS

Presentation

- Syntax layer
- SSL, SSH, IMAP, FTP, MPEG, JPEG

Session

- Synch & send to port
- API's, Sockets, WinSock

Transport

- End-to-end connections
- TCP, UDP

Network

- Packets
- IP, ICMP, IPSec, IGMP

Data Link

- Frames
- Ethernet, PPP, Switch, Bridge

Physical

- Physical structure
- Coax, Fiber, Wireless, Hubs, Repeaters

Kernel이 헤더를 자동으로 관리하는 일반적인 Socket으로는 네트워크 계층의 프로토콜 사용 불가

**Raw Socket –
가공되지 않은 소켓.**

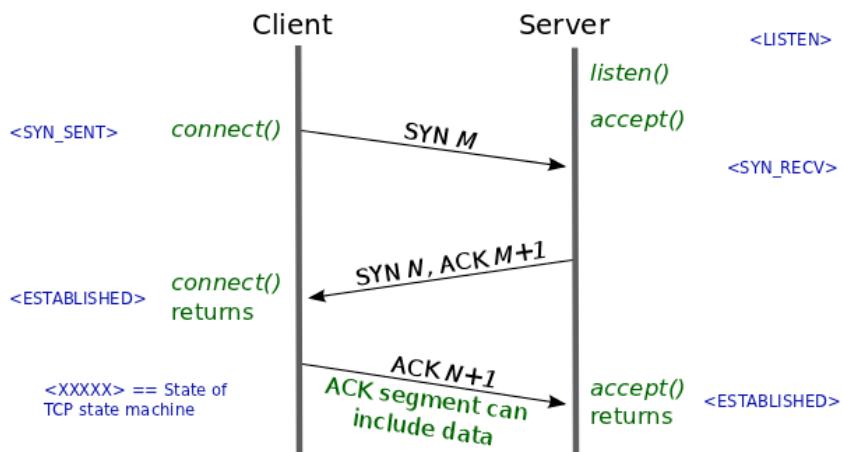
네트워크/전송 계층 헤더를 직접 제어 가능

Raw Socket Programming

Q. “패킷의 헤더를 왜 조작해야하지??”

- ⇒ NAT나 Spoofing이 대표적
- ⇒ 패킷의 목적지 주소나 출발지 주소를 바꾸는 것이 가능
- ⇒ 선의적 / 악의적인 목적 모두에 쓰임
- ⇒ 네트워크 스캐닝 기술 중에서는 대표적으로 SYN scan에 사용

TCP 3way-handshaking

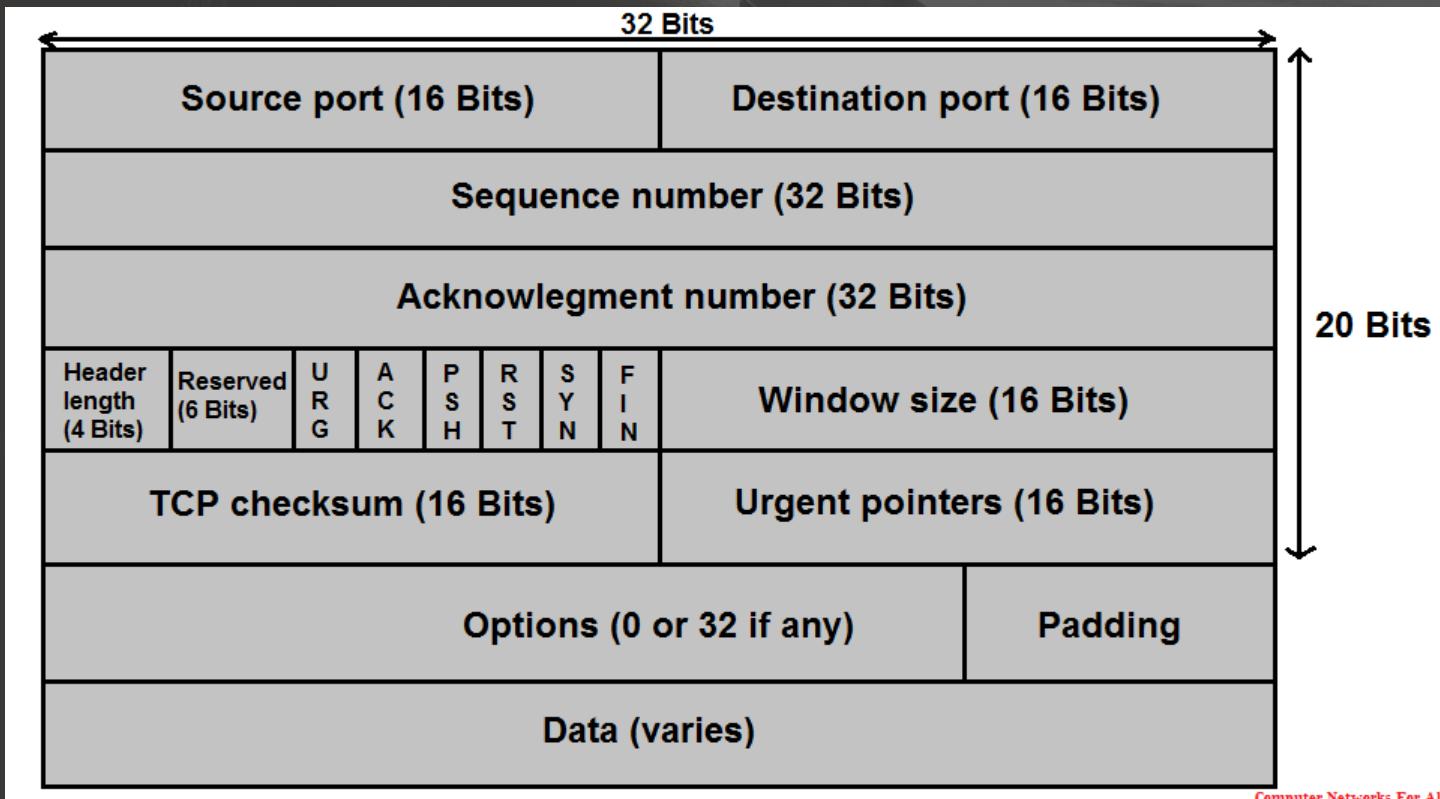


불완전한 연결 기반 (half-open)

TCP 3 way-handshake 과정을 제어하여 syn 패킷을 보내고 받는 것만으로 연결 가능 여부를 검사

Standard socket을 이용한 연결 기반의 검사도 가능하지만 속도가 느릴 수 밖에 없음

Raw Socket Programming



Raw Socket Programming

```
// TCP is the layer for TCP headers.  
type TCP struct {  
    BaseLayer  
    SrcPort, DstPort          TCPPort  
    Seq                      uint32  
    Ack                      uint32  
    DataOffset               uint8  
    FIN, SYN, RST, PSH, ACK, URG, ECE, CWR, NS bool  
    Window                   uint16  
    Checksum                uint16  
    Urgent                  uint16  
    sPort, dPort              []byte  
    Options                 []TCPOption  
    Padding                 []byte  
    opts                     [4]TCPOption  
    tcpipchecksum  
}
```

일일이 구조체를 다 정의한 후
패킷 생성

TCP / UDP / ICMP / ARP...
너무 번거롭다!!

<https://github.com/google/gopacket>

Raw Socket Programming

```
func makePacketARP(hw net.HardwareAddr, ip net.IP) layers.ARP {  
    return layers.ARP{  
        AddrType:         layers.LinkTypeEthernet,  
        Protocol:        layers.EthernetTypeIPv4,  
        HwAddressSize:   6,  
        ProtAddressSize: 4,  
        Operation:       layers.ARPPRequest,  
        SourceHwAddress: []byte(hw),  
        SourceProtAddress: []byte(ip),  
        DstHwAddress:     []byte{0, 0, 0, 0, 0, 0},  
    }  
}  
  
func makePacketTCP() layers.TCP {  
    return layers.TCP{  
        SrcPort: 54321,  
        DstPort: 0,  
        SYN: true,  
    }  
}
```

<https://github.com/google/gopacket>

Gopacket을 이용한 packet processing

Raw Socket Programming

```
func Socket(  
    domain, // 1. 프로토콜 체계 (protocol family)  
    type,   // 2. 소켓의 타입  
    proto int // 3. 프로토콜 체계 중 실제로 사용할 프로토콜  
) (fd int, err error)
```

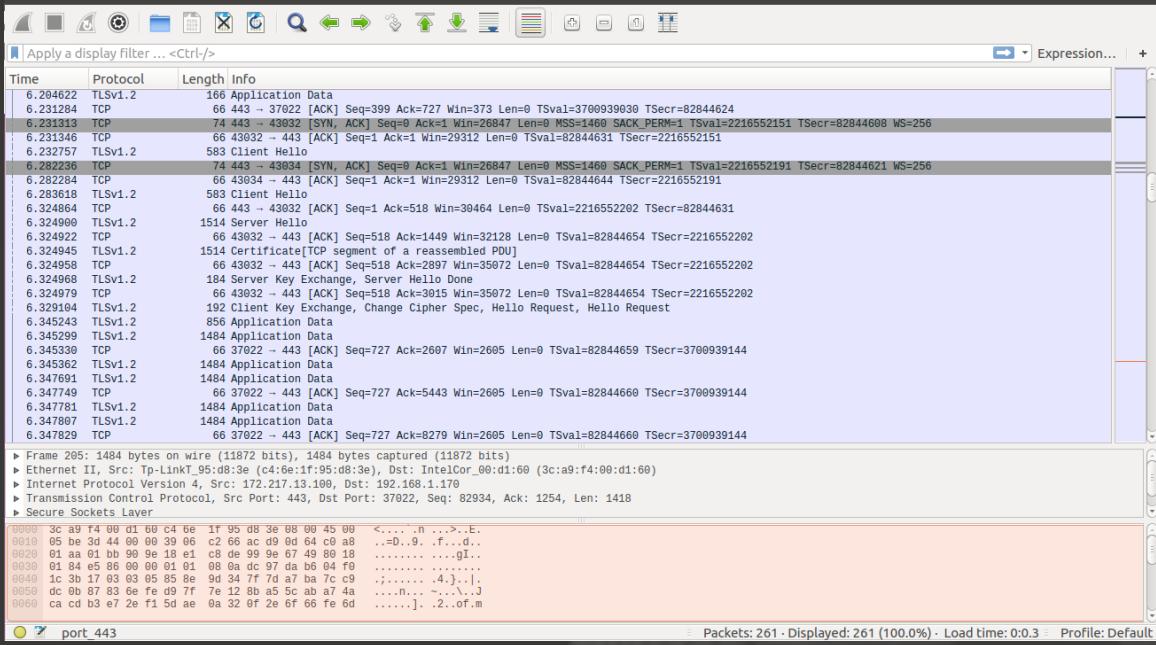
이름	프로토콜 체계 (protocol family)
PF_INET	IPv4 인터넷 프로토콜 체계
PF_INET6	IPv6 인터넷 프로토콜 체계
PF_LOCAL	로컬 통신을 위한 UNIX 프로토콜 체계
PF_PACKET	Low Level 소켓을 위한 프로토콜 체계
PF_IPX	IPX 노벨 프로토콜 체계
...	...

이름	소켓 타입
SOCK_STREAM	연결 지향형(TCP)
SOCK_DGRAM	비연결 지향형(UDP)
SOCK_RAW	사용자 정의형

이름	프로토콜
IPPROTO_TCP	TCP 프로토콜
IPPROTO_UDP	UDP 프로토콜
IPPROTO_RAW	RAW
...	...

Raw Socket Programming

Raw Socket을 이용하는 대표적인 프로그램 Wireshark



* 오픈 소스 패킷 분석 툴

* Pcap을 이용한 패킷 스니핑
이 주요 기능

* 루프백을 포함하여, 특정 NIC
를 통하는 모든 네트워크 패킷
을 볼 수 있음.

* 네트워크 학습용으로도 추천

Architecture

Socket API를 지원하는 대부분의 언어는
Raw Socket도 지원함

그 중 C++을 이용해 모듈 1차 구현 완료.

너무 높은 코드 복잡도
동시성 처리의 어려움

결국 Golang으로 재작업 시작!

C++ 코드 리뷰 후 실제 상황



Architecture

Golang 선택 배경

- ⇒ 훌륭한 Learning curve
- ⇒ 낮은 비용 그러나 고효율의 동시성 처리
- ⇒ 뛰어난 코드 가독성
- ⇒ Google의 gopacket lib 존재
- ⇒ 크로스플랫폼 지원

비로소 생긴 마음의 여유
가자! 몬스터헌터의 세계로



Issues

다수의 goroutine 사용에서 오는 Race condition 문제

- ⇒ Race Detector를 활용하자!
- ⇒ -race 옵션만으로 Deadlock이나 Race condition full stack tracing 가능

```
package main

import (
    "fmt"
)

func main() {
    c := make(chan bool)
    m := make(map[string]string)
    go func() {
        m["1"] = "a"
        c <- true
    }()
    m["2"] = "b"
    <-c
    for k, v := range m {
        fmt.Println(k, v)
    }
}
```

```
D:\project\go\src\git.loafle.net\prototype\test>go run -race main.go
=====
WARNING: DATA RACE
Write at 0x00c0420781e0 by goroutine 6:
  runtime.mapassign_faststr()
      D:/develop/go/src/runtime/hashmap_fast.go:694 +0x0
  main.main.func1()
      D:/project/go/src/git.loafle.net/prototype/test/main.go:11 +0x64

Previous write at 0x00c0420781e0 by main goroutine:
  runtime.mapassign_faststr()
      D:/develop/go/src/runtime/hashmap_fast.go:694 +0x0
  main.main()
      D:/project/go/src/git.loafle.net/prototype/test/main.go:14 +0xd0

Goroutine 6 (running) created at:
  main.main()
      D:/project/go/src/git.loafle.net/prototype/test/main.go:10 +0xa1
=====

2 b
1 a
Found 1 data race(s)
exit status 66
```

Issues

정확한 원인을 알 수 없는 connection 이상 현상

연결 가능한 상태의 대상임에도 간헐적으로 응답을 주지 않는 경우
리소스를 만들다가 hang이 걸리는 경우

- ⇒ 모든 연결에 Timeout을 걸어 다른 작업에 끼치는 영향을 최소화하자.
- ⇒ Context를 이용한 Goroutine Cancellation도 활용

```
ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
defer cancel()

go func() {
    res <- c.connect(ctx, param)
}()

// ...
```

```
select {
    case <- ctx.Done():
        // ...
}
```

개발/사용한 Go libraries

종 류	설 명
Server framework	Web, Websocket 서버 프레임워크
Annotation library	Java의 Annotation같은 기능을 제공
Dependency Injection library	Java의 Spring같은 의존성 주입
Bidirectional JSON RPC	기존의 단방향 RPC에서 제공하지 않는 Server push를 제공

Server framework for Go

모든 서버의 Lifecycle은 크게 다르지 않다는 전제로
Handler pattern 기반의 Server framework

```
func (sh *ServerHandlers) Init(serverCtx server.ServerCtx) error {
    if err := sh.ServerHandlers.Init(serverCtx); nil != err {
        return err
    }
    return nil
}

func (sh *ServerHandlers) OnStart(serverCtx server.ServerCtx) error {
    if err := sh.ServerHandlers.OnStart(serverCtx); nil != err {
        return err
    }
    return nil
}

func (sh *ServerHandlers) OnStop(serverCtx server.ServerCtx) {
    for _, _service := range sh.Services {
        as, err := oa.GetMethodAnnotationsByType(reflect.TypeOf(_service), oda.PreDestroyAnnotationType)
        if nil == err || nil != as {
            for k := range as {
                ins := make([]reflect.Value, 0)
                reflect.ValueOf(_service).MethodName(k).Call(ins)
            }
        }
    }
    sh.PubSub.Shutdown()
    sh.ServerHandlers.OnStop(serverCtx)
}

func (sh *ServerHandlers) Destroy(serverCtx server.ServerCtx) {
    sh.ServerHandlers.Destroy(serverCtx)
}
```



<https://github.com/loafle/server-go>

Annotation for Go

개인적으로는 go에서 가장 아쉬운 부분이 Annotation이라고 생각.
Tag와 Reflection을 활용해 제작한 Annotation 라이브러리

```
func init() {
    od.RegisterType(DiscoveryServiceType)
}

var DiscoveryServiceType = reflect.TypeOf((*DiscoveryService)(nil))

type DiscoveryService struct {
    oa.TypeAnnotation `annotation:"@Injectable('name': 'DiscoveryService') @Service()"``

    Discoverer discovery.Discoverer `annotation:"@Inject('name': 'Discoverer')"`
    PubSub     *pubsub.PubSub      `annotation:"@Inject('name': 'PubSub')"`

    _InitService   oa.MethodAnnotation `annotation:"@PostConstruct()"``
    _DestroyService oa.MethodAnnotation `annotation:"@PreDestroy()"``
}
```

<https://github.com/loafle/annotation-go>

Dependency Injection for Go

Spring framework처럼 되고 싶어요
제작한 annotation을 기반으로 의존성을 관리
@inject, @injectable 등 Angular의 형태를 차용

```
od.RegisterSingletonByName("Discoverer", discovery.Instance())
od.RegisterSingletonByName("PubSub", _pubsub)

services, err := od.GetInstancesByAnnotationType(annotation.ServiceAnnotationType)
if nil != err {
    olog.Logger().Panic(err.Error())
}
```

<https://github.com/loafle/di-go>

2018 Global SW 공모대전 국무총리상 수상

제 30회 글로벌SW공모대전 최종 수상작		
훈격	작품명	비고
대상 (대통령상)	G1 SYSTEM(글로벌통합물류시스템)	일반
금상 (국무총리상)	초고속 네트워크 트탈 스캐너 Flip: 빠르고 가벼운 합수형 통계 라이브러리	일반 학생부
은상 (장관상)	Ai기반 캠페인 Assistant 플랫폼 소셜 무직 협업 창작 플랫폼을 위한 저작SW&플랫폼(서버) 스마트 인터랙티브 둘시통역 시스템 SnIB(Scrap note Internet Browser) Cviano Whisperer MOCA	일반 학생부
동상 (기관장상)	SILCROAD v2.0 for Oracle to MySQL CODE MIND Browser 원격 자율 주행을 위한 드론 지상 제어 스테이션 개발 IFTTT를 활용한 통합 모듈 제어 프레임워크 MATRIX-매장 업무 관리 시스템 콘크리트 균열 탐지 소프트웨어	일반 학생부
장려상 (단체장상)	Happy KITCHEN 안전 보행 지원이 BAND C CARE(반드시 케어) Drink-preter (딥러닝 및 영상처리를 통한 실시간 음료 정보 음성 안내 기기)	학생부

최초의 Go 사용 수상작

Golang korea 그룹분들께 많은 도움 받았습니다. 감사합니다.

감사합니다.

