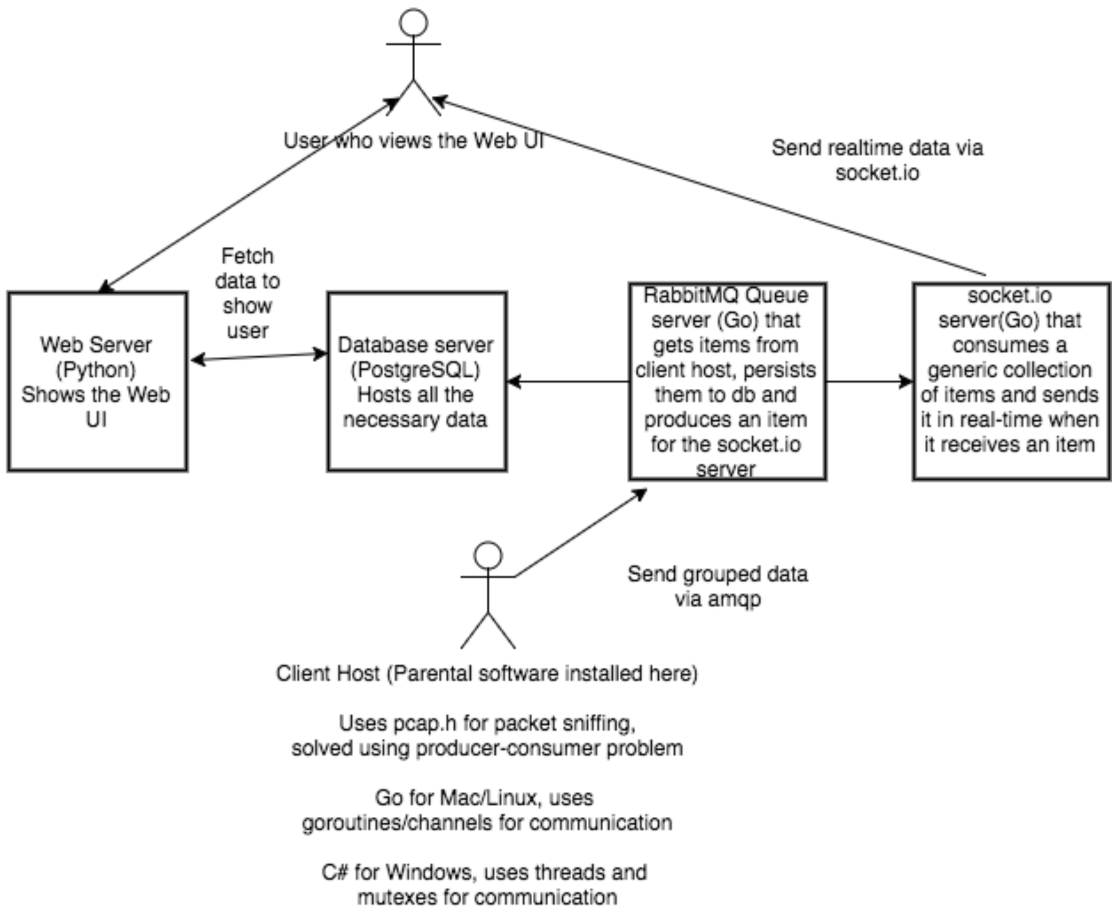
Guardian

My newest project is a Packet Capture application intended for Digital Parental Supervision.

Before diving in, the architecture looks as follows;

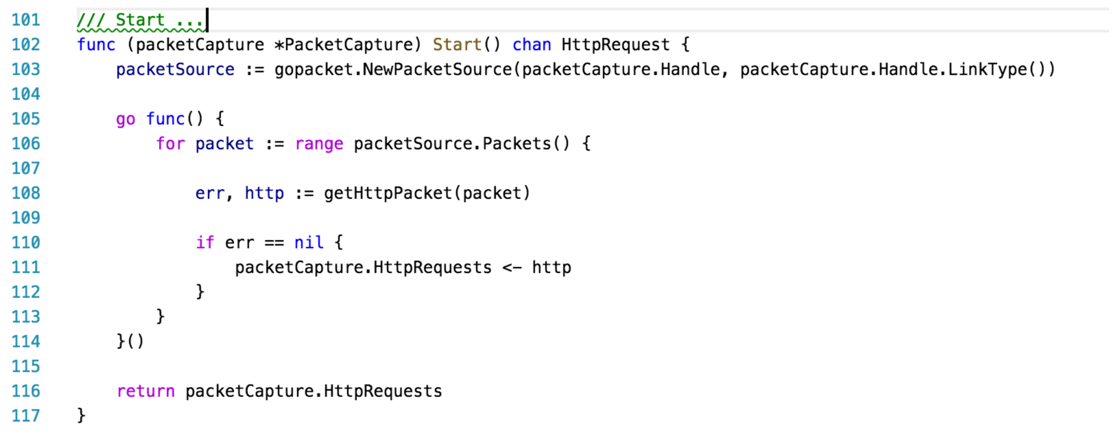

User who views the Web UI
Send realtime data via socket.io
Fetch data to show user
Web Server (Python) Shows the Web UI
Database server (PostgreSQL) Hosts all the necessary data
RabbitMQ Queue server (Go) that gets items from client host, persists them to db and produces an item for the socket.io server
socket.io server(Go) that consumes a generic collection of items and sends it in real-time when it receives an item
Send grouped data via amqp
Client Host (Parental software installed here)
Uses pcap.h for packet sniffing, solved using producer-consumer problem
Go for Mac/Linux, uses goroutines/channels for communication
C# for Windows, uses threads and mutexes for communication

The "clients" are written in Go and C# using pcap.h/winpcap to handle the packet capture. For Go there are two goroutines, one that produces HTTP request packets and another one that consumes items, groups them together and bulk sends them on a defined interval.

For C# instead of Goroutines and channels, threads and mutexes are used for communication and synchronization.

The clients communicate to a Queue Server via RabbitMQ to send the data to the server on a specified interval. The queueserver persists the data in PgSQL and notifies the socket.io server to update his connected clients with the new data. (This means that you can see in real-time what a user is browsing)

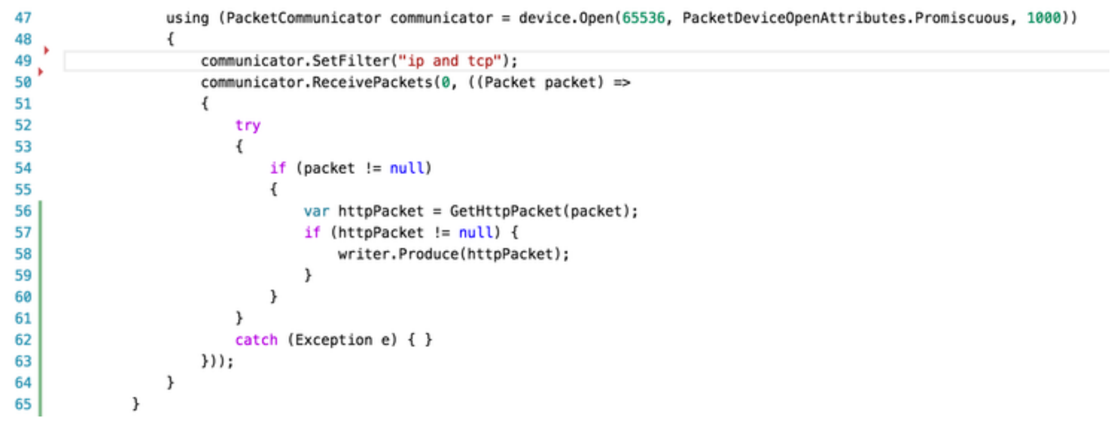
Go packet capture on a host (anonymous goroutine function, another goroutine is waiting for items on the channel):



```go
/// Start ...
func (packetCapture *PacketCapture) Start() chan HttpRequest {
    packetSource := gopacket.NewPacketSource(packetCapture.Handle, packetCapture.Handle.LinkType())

    go func() {
        for packet := range packetSource.Packets() {

            err, http := getHttpPacket(packet)

            if err == nil {
                packetCapture.HttpRequests <- http
            }
        }
    }()

    return packetCapture.HttpRequests
}
```

C# packet capture on a host (threading using a writer that produces items, another thread is using a blocking collection to consume the items):

:

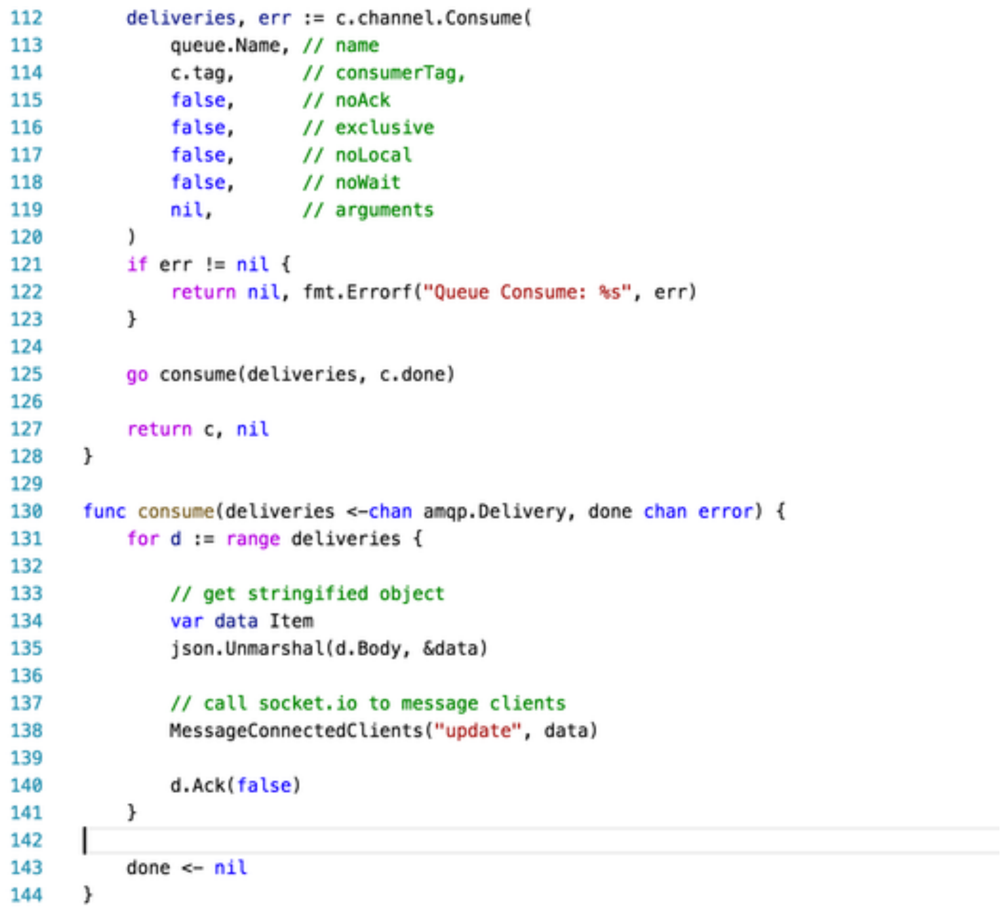


```csharp
            using (PacketCommunicator communicator = device.Open(65536, PacketDeviceOpenAttributes.Promiscuous, 1000))
            {
                communicator.SetFilter("ip and tcp");
                communicator.ReceivePackets(0, ((Packet packet) =>
                {
                    try
                    {
                        if (packet != null)
                        {
                            var httpPacket = GetHttpPacket(packet);
                            if (httpPacket != null) {
                                writer.Produce(httpPacket);
                            }
                        }
                    }
                    catch (Exception e) { }
                }));
            }
        }
```

Output in terminal:

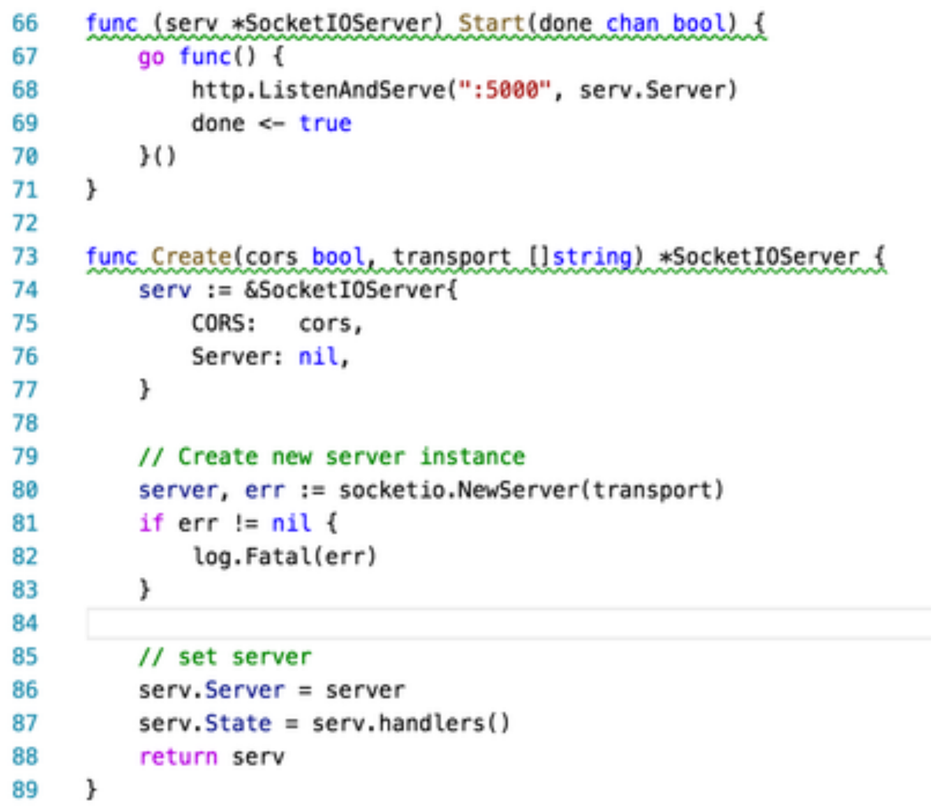
```
{ping.chartbeat.net 54.197.250.167 80(http)}
{www.visir.is 82.221.81.10 80(http)}
{www.tvinna.is 173.255.232.47 80(http)}
{dc.services.visualstudio.com 40.114.241.141 80(http)}
{www.dv.is 82.221.134.41 80(http)}
```

The Queue server is hosted on a Debian x64 server and consumes amqp deliveries, it then communicates with a socket.io server that handles real-time updates to users.



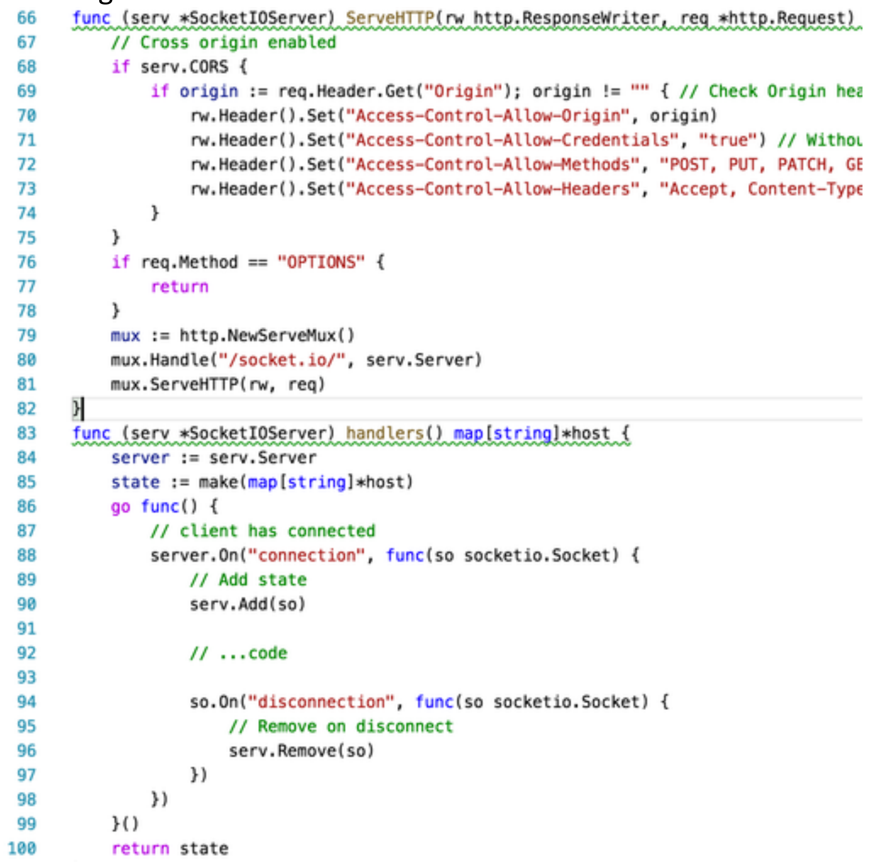
```go
    deliveries, err := c.channel.Consume(
        queue.Name, // name
        c.tag,      // consumerTag,
        false,      // noAck
        false,      // exclusive
        false,      // noLocal
        false,      // noWait
        nil,        // arguments
    )
    if err != nil {
        return nil, fmt.Errorf("Queue Consume: %s", err)
    }

    go consume(deliveries, c.done)

    return c, nil
}

func consume(deliveries <-chan amqp.Delivery, done chan error) {
    for d := range deliveries {

        // get stringified object
        var data Item
        json.Unmarshal(d.Body, &data)

        // call socket.io to message clients
        MessageConnectedClients("update", data)

        d.Ack(false)
    }

    done <- nil
}
```

Start and create the socket.io server, this is also hosted on a Debian x64, albeit another machine:



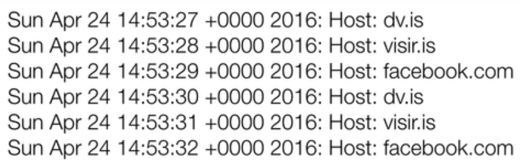
```
func (serv *SocketIOServer) Start(done chan bool) {
    go func() {
        http.ListenAndServe(":5000", serv.Server)
        done <- true
    }()
}

func Create(cors bool, transport []string) *SocketIOServer {
    serv := &SocketIOServer{
        CORS:   cors,
        Server: nil,
    }

    // Create new server instance
    server, err := socketio.NewServer(transport)
    if err != nil {
        log.Fatal(err)
    }

    // set server
    serv.Server = server
    serv.State = serv.handlers()
    return serv
}
```

Handling connected clients in socket.io

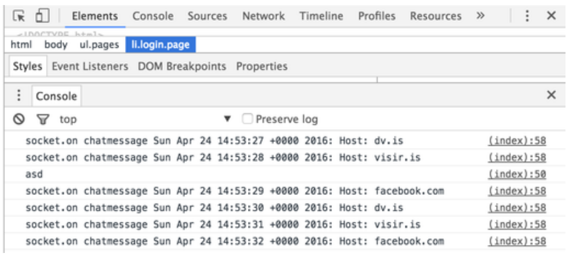


```
func (serv *SocketIOServer) ServeHTTP(rw http.ResponseWriter, req *http.Request)
    // Cross origin enabled
    if serv.CORS {
        if origin := req.Header.Get("Origin"); origin != "" { // Check Origin hea
            rw.Header().Set("Access-Control-Allow-Origin", origin)
            rw.Header().Set("Access-Control-Allow-Credentials", "true") // Withou
            rw.Header().Set("Access-Control-Allow-Methods", "POST, PUT, PATCH, GE
            rw.Header().Set("Access-Control-Allow-Headers", "Accept, Content-Type
        }
    }
    if req.Method == "OPTIONS" {
        return
    }
    mux := http.NewServeMux()
    mux.Handle("/socket.io/", serv.Server)
    mux.ServeHTTP(rw, req)
}
func (serv *SocketIOServer) handlers() map[string]*host {
    server := serv.Server
    state := make(map[string]*host)
    go func() {
        // client has connected
        server.On("connection", func(so socketio.Socket) {
            // Add state
            serv.Add(so)

            // ...code

            so.On("disconnection", func(so socketio.Socket) {
                // Remove on disconnect
                serv.Remove(so)
            })
        })
    }()
    return state
```

Output in web browser, for now I’m only using Pythons SimpleHTTPServer for POC. In the future I will either use Node or Python to show the web UI.



```
Sun Apr 24 14:53:27 +0000 2016: Host: dv.is
Sun Apr 24 14:53:28 +0000 2016: Host: visir.is
Sun Apr 24 14:53:29 +0000 2016: Host: facebook.com
Sun Apr 24 14:53:30 +0000 2016: Host: dv.is
Sun Apr 24 14:53:31 +0000 2016: Host: visir.is
Sun Apr 24 14:53:32 +0000 2016: Host: facebook.com
```



```
socket.on chatmessage Sun Apr 24 14:53:27 +0000 2016: Host: dv.is          (index):58
socket.on chatmessage Sun Apr 24 14:53:28 +0000 2016: Host: visir.is       (index):58
asd                                                                        (index):50
socket.on chatmessage Sun Apr 24 14:53:29 +0000 2016: Host: facebook.com   (index):58
socket.on chatmessage Sun Apr 24 14:53:30 +0000 2016: Host: dv.is          (index):58
socket.on chatmessage Sun Apr 24 14:53:31 +0000 2016: Host: visir.is       (index):58
socket.on chatmessage Sun Apr 24 14:53:32 +0000 2016: Host: facebook.com   (index):58
```