Todo list

Chapter One

Introduction

Network monitoring to some degree has long been a widespread practice. Ranging from capturing a single stream of packets, to watching 1000s of devices communicate across a vast corporate network. Having access to structured network traffic can help an individual track down issues in their own connectivity or that of an application, right through to being just as applicable to performing complex and automated analysis of traffic, in an attempt to detect DDOS attacks or measure QoS. This data could simply be a dump of packets, or sampled data that was observed from multiple points over a network.

Introduction: container glossary entry

List of "IPFIX" projects list

The process for capturing traffic on a single host are readily available and accessible to the average user. Anyone can spin up Wireshark and tell it to give them all of the DNS requests made, which can then be explored through the structured breakdown of each individual packet; alternatively saving this data to a file for later examination. The same logic hold for monitoring the performance of a website, one could simply collect packets on the computer running the website and use this to make an educated guess as to where issues could be arising in the network.

However, this type of approach will immediately run into issues However, problems can quickly begin to crop up. Wireshark will seriously struggle to capture a high throughput of data, while loading large numbers of packets will cause issues with the GUI and exploring the breakdown of each packet or flow. Let alone sifting through ever growing records of traffic. This approach is also fundamentally limited by the scope of the traffic seen, you can only get a somewhat limited view from the perspective of a single computer on a network. This is where purpose build solutions come into play. Enterprise Network switches and routers that come with built in support for capturing traffic are readily available (albeit less accessible to the individual user). By shifting this responsibility to the infrastructure, we have also shifted the point of observation to one that sees traffic destined for multiple hosts. Often also taking advantage of specialized hardware which can deal with much larger volumes of traffic. When collecting from multiple points, this data can be cross-referenced to create much more contextually aware outputs.

The most immediate issue that crops up with dedicated solutions is one of cost. To get the full advantages of a wide observation domain, compatible devices will need to be deployed throughout the network; otherwise, what is effectively the same issue as mentioned before when capturing from a single host. The scope of the data is limited to that one point.

As previously mentioned, most all enterprise network devices will support some form of network monitoring. With protocols such as: sFlow, Network (plus jFlow if you're feeling fancy) and more recently IPFIX being the de facto choices in industry. While these protocols have been designed for enterprise scale and there is an abundance of solutions for processing this data, supported hardware can be vastly expensive; and standalone applications for actually producing this data are scarce or expensive in their own right.

Introduction - what flow stuff does or doesn't

Flow monitoring protocols offer the facility for distributed traffic monitoring, enabling a plethora of use cases for the dataset produced.

Specialised tools do exist, they may take the form of x or y However there may not be one for all use cases. You may end up needing multiple solutions.

What if we could create a generic framework for handling the collection and processing of packets, for the purposes of distributed monitoring of a network domain? Could we build upon the strong properties of industry tried and tested protocols. Taking advantage of the excellent tooling of modern stuff, along with making it generic enough to be applied to an extremely broad range of target platforms and environments.

This project explores the current solutions to network observability, where these show their strengths and weaknesses. It addresses the requirements and expectations such a framework would need. And then implements a full end-to-end solution for distributed network flow and event monitoring which demonstrate the performance and portability needed, while additionally making it trivial to extend the system to support structured processing of new (or the same) protocols.

1.1 Previous Work

Previous Work - What was coders

- I made coders
- It was cool

I wanted to explore this area after working on a project that focused on provisioning container based Integrated Development Environments (IDEs). While this was a somewhat contrived use case, I had hoped to be able to view the same type of 'per-host' statistics that you would get from a normal monitoring solution.

The idea for this project primarily stemmed from a previous system I developed. The purpose of which was to offer fast access to pre-configured Integrated Development Environment (IDE) on demand. By using kubernetes (K8S) as a container orchestrator, the system was capable of dynamically placing each new IDE in the same isolated network (or separately). This was with the intention of allowing further expansion to include networked resources that could be shared among a team. See Figure 1.1 on Page 9 Here I want to talk about the previous project I worked on and the issues it raised.

Previous Work - What was missing

something

Previous Work - some filler

Since the power of these monitoring solutions come from having multiple points of reference, I wanted a platform independent solution for aggregating network events and statistics. Making it easy to monitor traffic with reference to any existing or new hosts you might have. Along with being able to apply this to anything from a physical host to a container.

[Figure 1 about here.]

1.2 Project Aims

Project Aims - 2nd draft

- Provide a mechanism for collecting traffic from as broad a range of platforms as possible
- Build a light weight application for capturing and transforming network traffic into structured events
- portability!!!

The aim of this project is to build a framework for easily implementing aggregated traffic capture and the subsequent structuring and collating of this data.

The focus of this project will be specifically on the portability of the final implementation for traffic capture.

repetitive sentence start

The framework will also be Secure by design (SBD).

While existing protocols such as IPFIX set out that “sensitive data **should** be transmitted to the Collecting Process using a means that secures its contents against eavesdropping” [4, p. 55].

Since this project encompasses both and implementation, it will make no assumptions about the integrity of the network [1]

Project Aims - 1st draft

The aim of this project was to build a system for easy collection and aggregation of network related events and statistics. With a specific focus on the ability to target a wide range of platforms / architectures; while also making it straightforward to

deploy in both virtual and physical environments.
The next core focus was that the data captured should have the potential to carry protocol specific information. e.g. including the raw query provisioning container based IDEs. While this was a somewhat contrived use case, I had hoped to be able to view the same type of 'per-host' statistics that you would get from a normal monitoring solution. in DNS events.

expand on reason for rich events

1.3 Related Work

1.3.1 Packet capture

Current options

Missing figure

$networkcard- > libpcap/af_packet/pf_ring- > gopacket- > output$

reading off the wire

decoding

1.3.2 Flow monitoring

Current options

correlating packets

Flow Statistics

Overhead

Due to the volumes of data a network may be subjected to, there is the risk of each component of the collection process becoming overloaded. Sampling of the packets can significantly reduce the volume of traffic produced [2] ($TableX$). goes in depth on the bottlenecks and achieved performance with flow monitoring.

Overload

Overhead: Existing solution architectures

IPFIX

Internet Protocol Flow Information Export (IPFIX) is a protocol used in the transmission of traffic flow information across a network. By purely describing how information should be structured, regardless of transport [3] protocol. IPFIX implementations MUST support Stream Control Transmission Protocol (SCTP) but may also support TCP and UDP independently [4].

this isn't the same as my current def of flows

What was previously described as a flow differs from what is referred to here. A Flow is defined as a set of packets or frames passing an Observation Point in the network during a certain time interval. All packets belonging to a particular Flow have a set of common properties [4, p. 8] .

The core principle of capturing traffic flows is observing traffic from multiple points, as to collect it and process as one data-set.

[Figure 2 about here.]

[Figure 3 about here.]

1.3.3 Application vs Network monitoring?

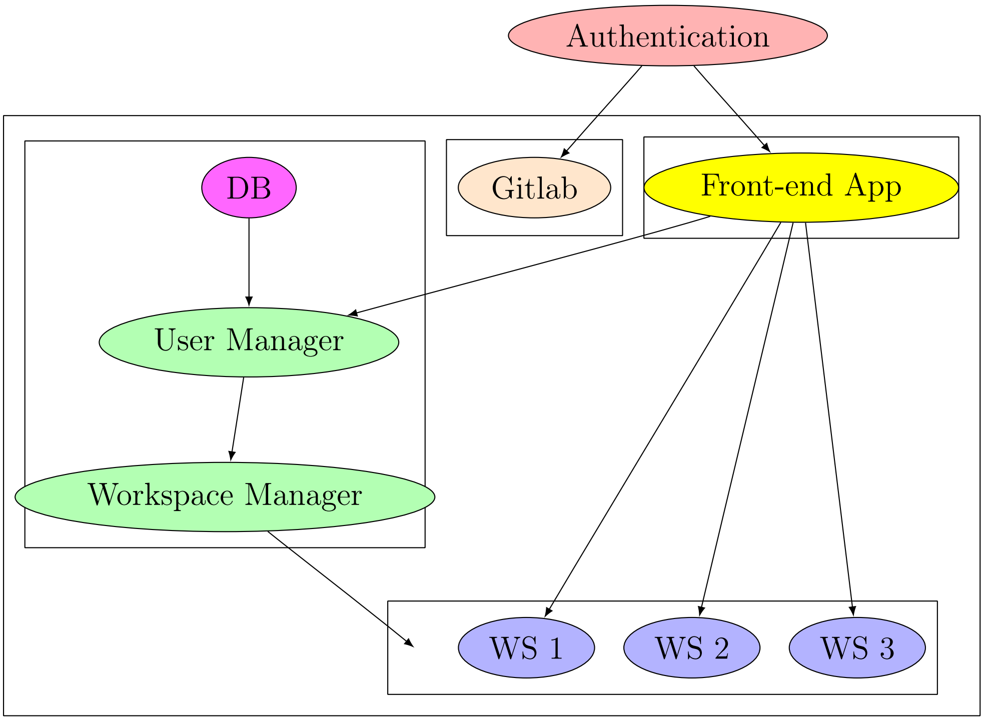

Figure 1.1: Coders

```
+----------------------------------------------------------------+
|                          Exporter      IPFIX        Collector |
|                           O--------------------------->O       |
|                           |                                    |
|                           | Observation                        |
|                           | Domain                             |
|                           |                                    |
|        Metering #1        | Metering #2                        |
|          O----------------O----------------O Metering #3       |
|          |                |                |                   |
|          | Observation    | Observation    | Observation       |
|          | Point #1       | Point #2       | Point #3          |
|          v                |                |                   |
| ---- IP Traffic --->      |                |                   |
|                           v                |                   |
| --------------- More IP Traffic --->       |                   |
|                                            v                   |
| ---------------------------------- More IP Traffic --->        |
+----------------------------------------------------------------+
```

Figure 1.2: IPFIX Architecture (Source: Wikipedia IP Flow Information Export)

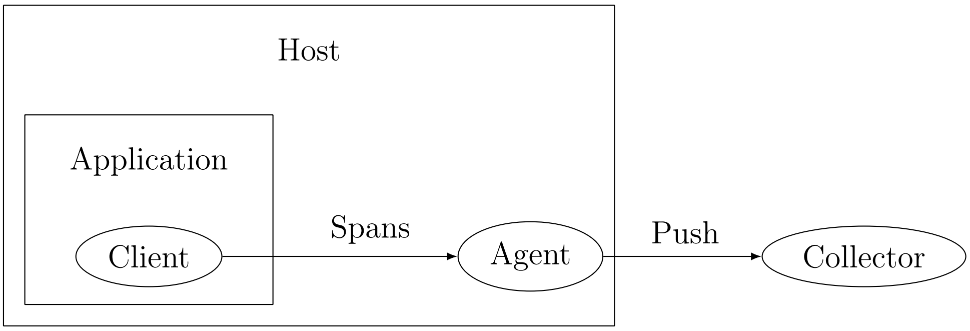

Figure 1.3: Jaeger Architecture

Chapter Two

WIP

Eval

To evaluate the effectiveness of my solution, I explored a series of realistic use-cases, along with generated benchmarks.

I explored the limitations that arise due to external components that the system must interact with.

Also testing how performance varies on the different platforms that I have been able to target. For the most part, I used libpcap for the platform dependant packet capture. This adds its own set of limits and advantages.

Benchmarks

A critical requirement at both the point of observation and when ingesting is the speed at which we can transform the data.

Firstly there is capturing packets, since this is something handled by the underlying implementation it isn't worth an in-depth review.

Next is the decoding of packets after they were captured. This handled by the gopacket library, which has flexible support for only attempting to decode packets to the desired layers. Additionally, it is able to do TCP stream reassembly without copying/allocations (assuming packets arrive in order)

I had to get the best performance out of gopacket (use it properly)

Having decoded the packets, the appropriate information needs to be carried over to the new format.

Marshall The first benchmark is marshalling performance.

Here you can see the results for the currently supported protocols and features...

These are generated from the protobufs, and work by populating the structure with seeded random data. With the results consisting of...

Unmarshall

Discussion

The main advantage of this approach is how the amount of code that needs to be maintained is very small.

Adding protocols

Add support for decoding the packets Define the message to represent it Add or description to the message Define any additional API mappings

API performance

I wrote a post-processor that reads the database definitions and generates a set of translations for web requests to SQL queries.

Either the request path or the query parameters can be used to bind a request to filter the resulting query.

Additional parameters like id_eq or id_ne can also be mapped to SQL clauses.

All these possible clauses generated at compile time as what is effectively an O(1) Time complexity lookup due to it being a static hashmap. Show some traces to demonstrate performance.

UI

The ui is a responsive progressive web app built using react admin.

This provides an easy starting point for displaying a basic table view of resources exposed by the api

It consists of:

Resources

A resource describes an individual type that the api exposes, e.g. DNS.

For a resource I define a list view. Which describes how to represent one record in a table.

Data provider This handles mapping ui elements and views to specific api endpoints.

Compared to ipfix or netflow...

Fundamentally there are a lot of similarities between my solution and the protocol design of ipfix. Both are routed in the idea that the underlying structure should be easily extensible allowing for arbitrary levels of detail or to focus down the information collected.

In the case of ipfix, the rfc sets out that an implementation MUST support sctp but MAY support tcp or udp. With this flexibility it allows for an implementation to be purpose built for use cases where one transport might make more sense. While SCTP is a great idea, (whhhhyyy)

While rfc 7011 (section 11) comprehensively addresses the security considerations for the communication between exporters and collectors, highlighting both why the traffic should be confidential but also should not be interrupted (for accounting & forensic purposes)

gRPC (literally has testimonials from cisco and juniper) While ipfix does set out requirments for how to secure traffic, it explicitly states "Information Element containing end-user payload information is exported, it SHOULD be transmitted to the Collecting Process using a means that secures its contents against eavesdropping."

I chose to use gRPC as my primary transport implementation.

It offers Streaming Blocking / non blocking Cancellation / timeout Lots of language support Pluggable authentication

texcount

```
File: introduction.tex
Encoding: utf8
Sum count: 1373
Words in text: 1323
Words in headers: 30
Words outside text captions, etc.: 18
Number of headers: 15
Number of floats/tables/figures: 3
Number of math inlines: 2
Number of math displayed: 0
Subcounts:
  text+headers+captions #headers/#floats/#inlines/#displayed
  720+1+0 1/0/0/0 Chapter: Introduction
  214+2+1 1/1/0/0 Section: Previous Work
  249+2+0 1/0/0/0 Section: Project Aims
  0+2+0 1/0/0/0 Section: Related Work
  0+9+0 4/0/1/0 Subsection: Packet capture
  140+10+17 6/2/1/0 Subsection: Flow monitoring
  0+4+0 1/0/0/0 Subsection: Application vs Network monitoring?
```