An Extensible Framework For Portable And Distributed Packet Capture

By

Charles de Freitas
Student ID: 1810538
Course: MSci Computer Science
Supervisor: Dr Ian Batten
Word Count: 1421

ABSTRACT

Use of network monitoring solutions as for a long time been standard practice. With many powerful protocols and technologies being adopted; often thanks to their excelent performance, or alternativly their deep integration in a complete network infrastructure solution. New and novel approches to achiving the same results in virtual and dynamic environments, where a unique host may be much more short lived and more focused in it's activity.

The use of network monitoring for both per-host insights and traffic flow statistics opens the way for endless realworld applications of this data. This work implements extensible capture and decoding of network events, which can then be agregated into the broader framework. The

Contents

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.

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.

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.

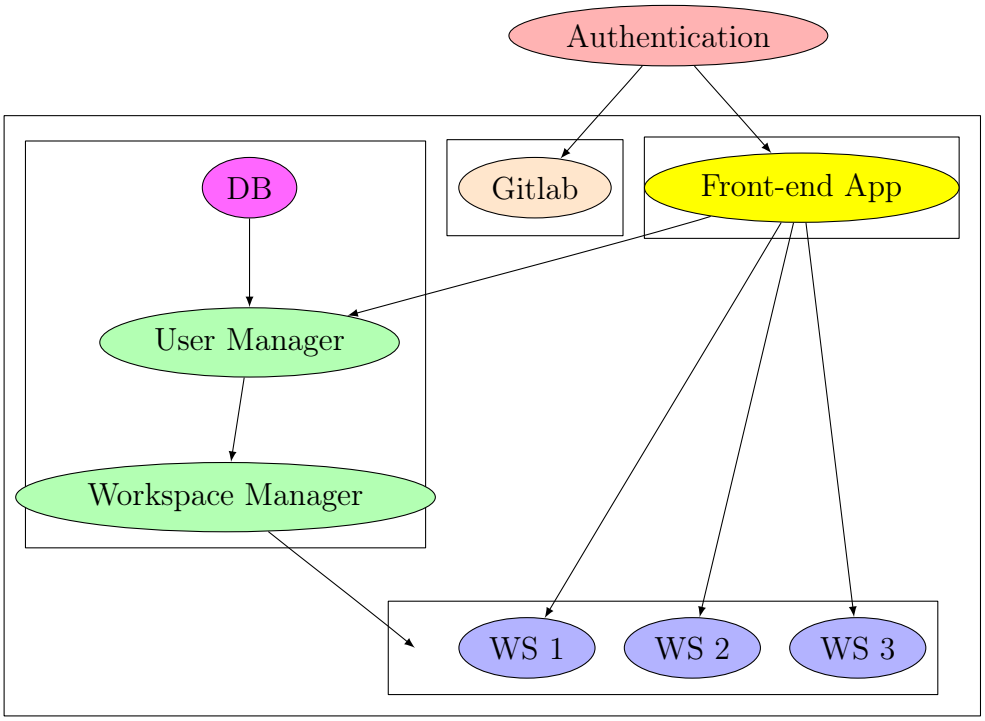

Figure 1.1: Coders

1.1 Previous Work

1.2 Project Aims

1.3 Related Work

1.3.1 Packet capture

1.3.1.1 Current options

1.3.1.2 reading off the wire

1.3.1.3 decoding

1.3.2 Flow monitoring

1.3.2.1 Current options

1.3.2.2 correlating packets

1.3.2.3 Flow Statistics

1.3.2.4 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 [1] ($TableX$). goes in depth on the bottlenecks and achieved performance with flow monitoring.

1.3.2.5 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 [2] protocol. IPFIX implementations MUST support Stream Control Transmission Protocol (SCTP) but may also support TCP and UDP independently [3].

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 [3, 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.

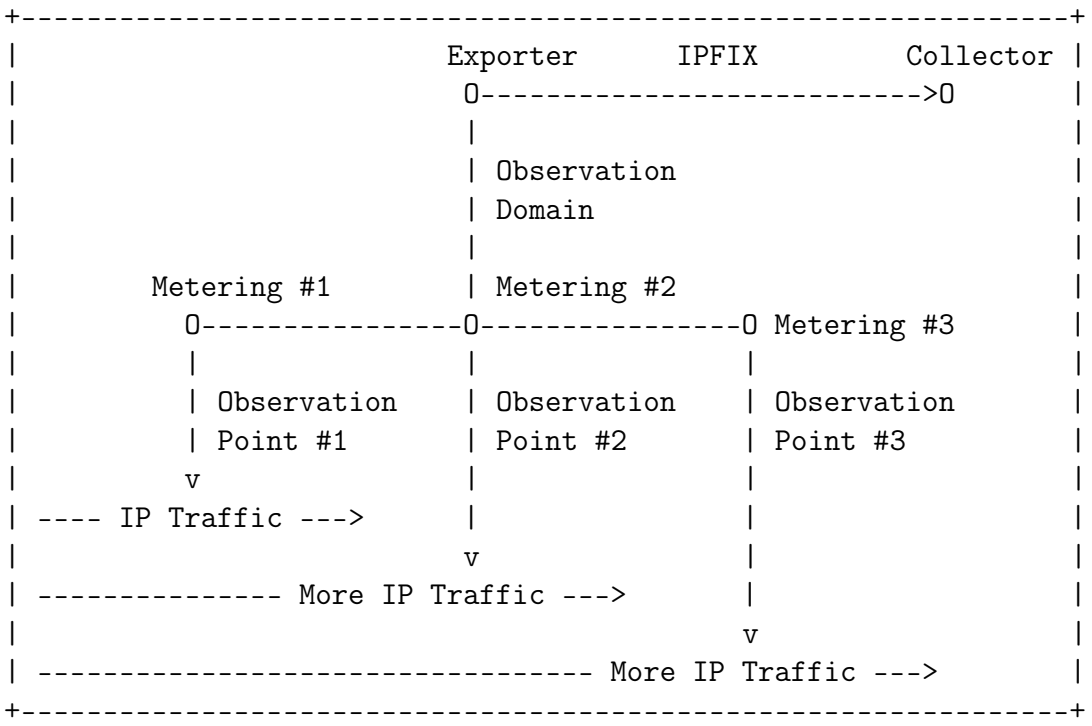

Figure 1.2: IPFIX Architecture (Source: Wikipedia IP Flow Information Export)

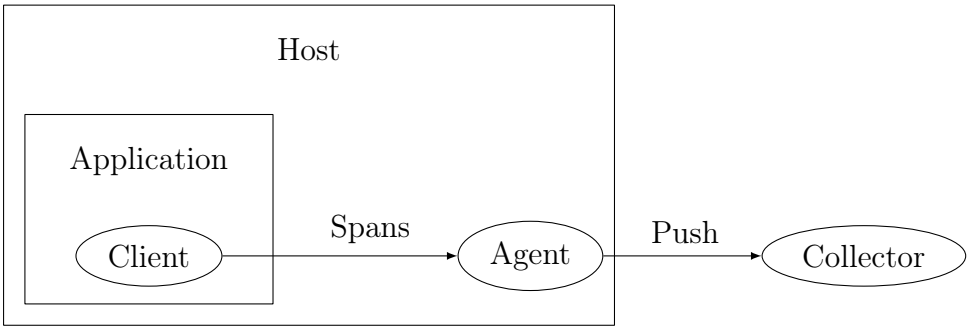

Figure 1.3: Jaeger Architecture

1.3.3 Application vs Network monitoring?

Chapter Two

Design

While working on the project it went through several iterations to further refine the design.

2.1 Planning

As set out in the project aims, there are 3 key points I want to address with this solution:

- Portability, to maximise the variety of platforms we will be able to monitor.
- Extensibility, it should be trivial to extend currently supported protocols; along with adding support for completely new ones.
- Scalability, any real world setting where traffic monitoring is needed will have large volumes of traffic, making operation at scale a fundamental requirement.

2.1.1 Portability

When I say portability, I mean being able to get the code to compile for multiple platforms, as apposed to being able to run the same binary in more places.

As previously mentioned, relying on hardware that supports something like NetFlow removes the need for additional devices but it does limit you to these ones.

The goal is to develop a stand alone binary that, given a sequence of packets will pick out the appropriate packets and extract the desired information. This can be broken down to three main points: collecting, decoding and marshaling; see Figure 2.1 on Page 9 .

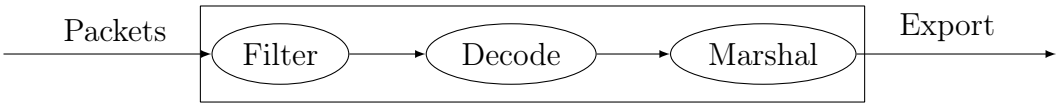

Figure 2.1: Packet pipeline

To minimise the footprint of said program, a compiled language over one which is interpreted seems like the logical choice.

While relying on a complete infrastructure and monitoring solution, you are bound to encounter some amount of lock-in.

Fundamentally, by being able to target (build for) a greater range of Operation Systems (OSs), it will allow for more flexible deployments.

```
message Flow {
  uint32 src_port = 1;
  uint32 dst_port = 2;
  string src_address = 3;
  string dst_address = 4;
  google.protobuf.Timestamp timestamp = 5;
  uint64 id = 6;
  repeated Packet packets = 7;
  uint64 packet_count = 8;
}
```

Listing 1: Flow Definition

2.1.2 Extensibility

2.1.3 Scalability

2.2 Protocol

To replicate the rich details that are available through Wireshark [4, Seciont 6.1] without generating prohibitively large network overhead, the data must be extracted and encoded at the point of collection. Take the case of summarising a TCP stream. Including full packets would amplify traffic.

2.2.1 event types

2.2.1.1 single packet

DNS / TLS

2.2.1.2 multi packet

Flow statistics

gRPC uses HTTP/2

Protocol buffers are a language-neutral, platform-neutral, extensible mechanism for serializing structured data .

At the core of the framework is the message definitions. The base purpose being to define the details that can be associated with a given event. The structure of a messages is intentionally very loose, allowing for it to be tailored to any level of detail. By using protocol buffer as a base throughout, I'm able to update exisiting definitions without affecting any existing functionality.

Given the goals of **extensiblity** and **performance** I ended up with some key similarities with IPFIX.

2.3 Message Design

Since I wanted to be able to export detailed protocol information without the overhead of sending all the captured traffic, the message structure needed to be able to carry these additional details.

2.3.1 Flows

2.3.2 Rich Details

The risk of data amplification. Packet capture on low end / embedded devices my be CPU or memory bound

2.3.3 Sampling

Chapter Three

Implementation

3.1 Architecture

3.2 Agent

Based on the existing solutions that have been discussed, I chose to start with a push based agent.

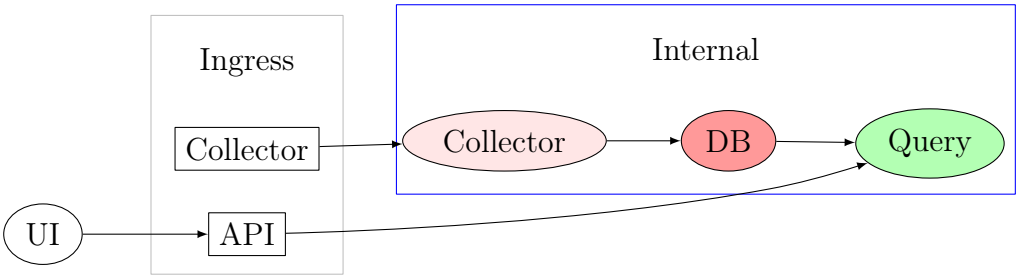

Figure 3.1: Component Architecture

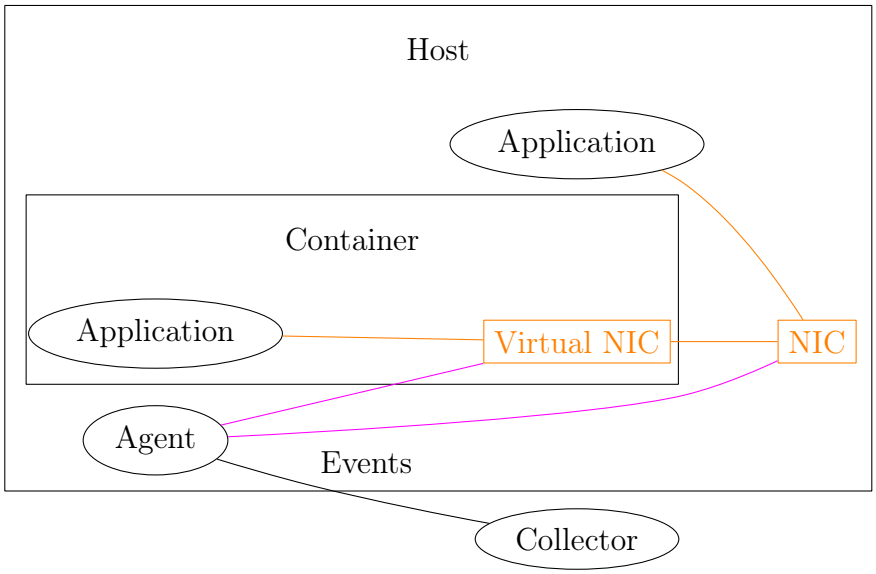

Figure 3.2: Agent

3.2.1 Cross Platform Support

3.2.2 Packet Capture

3.2.3 Decoding

3.2.3.1 Performance

3.3 Collector

3.3.1 RPC Service

3.3.2 Database Connections

3.4 Query API

3.4.1 Request Mapping

3.4.2 Performance

3.5 UI

Chapter Four

Evaluation

When it came to evaluation, I wanted to bring it back to each point set out in the project planning.

4.1 Syntetic workload

4.2 deployment

In relation to how to deploy the system (and external components) for a greater load.

4.3 Use Cases

4.3.1 Types of Use cases evaluated

4.3.1.1 Hypervisor

Deployed to a **Physical** host (Hypervisor) which itself is hosting a variaty of virtual machines and containers. What kinds of agents have been deployed?
What scale is this intended to work at?

how does it match aims?

What are the bits that went well / bad

what would change.

Chapter Five

Conclusion

Appendix One

Benchmarks

GOOS	GOARCH	CPU
linux	amd64	AMD Ryzen 5 `5600X 6-Core Processor`

Table A.1: Benchmark Environment

Message	loops	(ns) per loop	I/O (MB/s)
Agent	2003037	590.8	885.28
Endpoint	10596949	108.1	583.01
Flow	810357	1543	516.41
Packet	1721414	671.9	308.07
Http	1533000	804.6	548.07
Dns	1548782	766.8	507.31
Tls	1733568	714.0	473.41
Arp	1178860	998.3	690.18

Table A.2: Unmarshal Benchmarks

Message	loops	(ns) per loop	I/O (MB/s)
Agent	3045099	392.6	1332.12
Endpoint	11553470	116.7	539.83
Flow	1000000	1237	644.46
Packet	1556341	689.0	300.42
Http	1702892	614.0	718.25
Dns	1843177	680.5	571.67
Tls	1961786	661.3	511.14
Arp	1523559	790.6	871.49

Table A.3: Marshal Benchmarks

Message	loops	(ns) per loop	I/O (MB/s)
Agent	63025371	18.83	28043.78
Endpoint	198427028	6.135	10106.47
Flow	5098138	219.5	3607.41
Packet	7151492	175.0	1143.15
Http	8916307	129.6	3440.40
Dns	11554758	119.2	3322.87
Tls	9748363	129.0	2651.38
Arp	10249390	119.8	5742.23

Table A.4: Size Benchmarks

Glossary

flow A sequence of related packets from one source computer to a destination. Also described as . 5

message Stucture used for representing any features the system will report . 11

NetFlow Cisco stuff. 8

protocol e.g. DNS. i, 4, 7

protocol buffer Protocol buffers are a language-neutral, platform-neutral, extensible mechanism for serializing structured data . 10, 11

traffic flow 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 [3, p. 8] . i, 5

Acronyms

DNS Domain Name System. 10

IPFIX Internet Protocol Flow Information Export. ii, 4, 5, 11, 22

OS Operation System. 9

SCTP Stream Control Transmission Protocol. 4

TCP Transmission Control Protocol. 4, 10

TLS Transport Layer Security. 10

UDP User Datagram Protocol. 4

Nomenclature

MB/s Mega Bytes per Second

Figures

Tables

Code Snippets

References

[1] Rick Hofstede et al. "Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX". In: *IEEE Commun. Surv. Tutor.* 16.4 (2014), pp. 2037–2064. ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2321898 (cit. on p. 4).

[2] R. Stewart (Ed.) *Stream Control Transmission Protocol*. RFC 4960. Updated by RFCs 6096, 6335, 7053, 8899. Fremont, CA, USA: RFC Editor, Sept. 2007. DOI: 10.17487/RFC4960. URL: https://www.rfc-editor.org/rfc/rfc4960.txt (cit. on p. 4).

[3] B. Claise (Ed.), B. Trammell (Ed.), and P. Aitken. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. RFC 7011. Fremont, CA, USA: RFC Editor, Sept. 2013. DOI: 10.17487/RFC7011. URL: https://www.rfc-editor.org/rfc/rfc7011.txt (cit. on pp. 4, 5, 20).

[4] *Wireshark*. URL: https://www.wireshark.org/ (visited on 04/28/2021) (cit. on p. 10).