Netflow Data Streams Analysis Tool

Network Innovations Inc.

Author(s): Alex Lewis | Kaitlin Culligan | Kunal Dhawan | Karl Winkler | Henry Wu | Thomas Kusinski
Schulich School of Engineering, University of Calgary

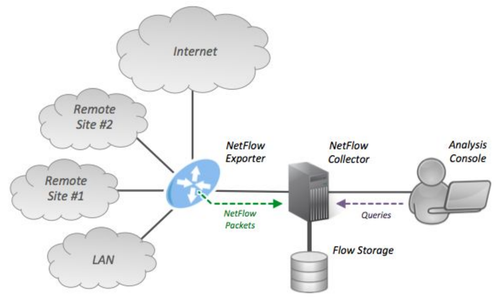
Abstract

Network Innovations is currently looking for a scalable, robust solution to aggregate all the data from these distributed systems, extract key session information, and analyze trends across their network. Our solution aggregates, stores, and provides real-time analysis of the geographically dispersed NetFlow data streams, providing an easy-to-understand representation of current network trends and metrics, while focusing on modularity and effortless expandability.

Our solution can be split into 4 modules: NetFlow collector, Amazon SQS, Analysis Engine, and Solr Database. Each module has been designed as a standalone functional unit, allowing ultimate flexibility when it comes to scaling our solution to larger networks.

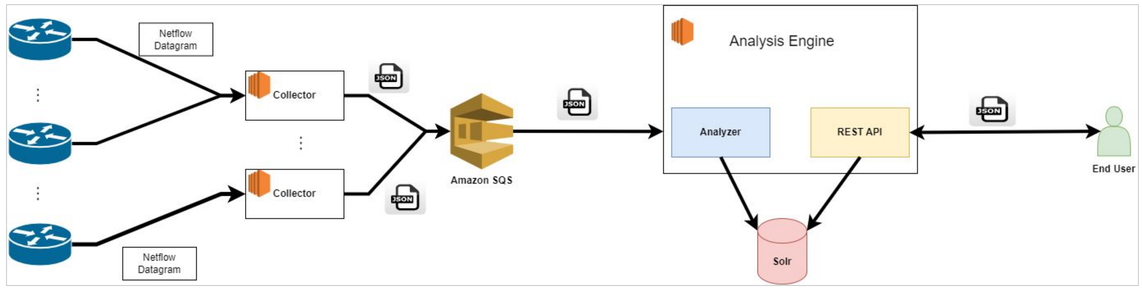
Introduction

In a geographically dispersed network, there are multiple devices such as firewalls and routers located at each point of presence. Network traffic is logged from these devices as NetFlow data streams. NetFlow is a protocol designed by Cisco as an easy way to communicate session information.



The Solution

NetFlow collector: Interfaces NetFlow V5 data stream and prepares packets for the analysis engine.
Amazon SQS: Stores NetFlow packets that back up before the analysis engine to ensure no packets get lost in our system.
Analysis Engine: Extracts key data from the packet, and performs primary analysis of the data.
Solr Database: Solr provides a free, open source, easy to integrate database for us to store processed data.
API: Used to move data to/from the database to visualize collected data, and help ease the integration process of our system into pre-existing systems.



Conclusions

This project has been successful as a proof of concept, however in order for this solution to be implemented in a real world scenario it will need to be expanded upon. For this project, we were constrained to only using the free tier of AWS, which drastically limits many factors that attribute to our systems throughput. Some of these limiting factors include: core speed, input data rates, and the level of parallelism we can achieve.

By simply moving our solution to a more powerful version of AWS the performance will drastically increase. However, by analyzing the behavior of our system at lower input rates we can evaluate the most optimal way to scale our solution up to an industrial level.

Results

Our initial goal for this project was to create a system that could process 100 concurrent data streams while maintaining a real-time throughput of under 10 seconds end-to-end. While this requirement seems simple, due to the nature of Network communication, the number of concurrent data streams is actually a function of two metrics:

- **Packets/Second:** This is simply the rate at which our system receives input data.
- **NetFlow Streams/Packet:** This is the number of Netflow Flow Records present in an incoming packet.

Average End-to-End Time (100 Concurrent Streams):

1 Packet/Sec: 0.058s
10 Packet/Sec: 1.453s
20 Packet/Sec: 1.546s
30 Packet/Sec: 3.772s

For more information on testing methodology and results, a team member would be happy to discuss!

Methods and Materials

Our project was written entirely in Java as it interfaces nicely with a broad range of open source tools. Our sponsor currently operates their existing system on Amazon Web Services (AWS), and requested that our solution also be done with AWS to allow easier integration into their system. Additionally, it was requested that our solution be compatible with a Linux environment.

Our design methodology revolved around modularity and flexibility. We designed each component as a standalone functional unit, then connected each component using AWS and JSON tools. In doing this, we allow each component to be removed, modified or expanded as needed, providing a great amount of scalability and customizability for future developers trying to integrate our solution into a larger system.

Key Tools:

- Java Spring: Java Framework
- JUnit: Unit testing framework
- Apache Solr: NoSQL Database
- Amazon SQS: Managed message queue system
- Amazon EC2: Cloud computing system

References

1. https://en.wikipedia.org/wiki/NetFlow
2. https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html

CONTACT
Alex Lewis
Email: alexander.lewis@ucalgary.ca
Phone: 403-837-5620