Network Fundamentals for Cloud

BITS Pilani
Pilani Campus

Nishit Narang
WILPD-CSIS

innovate achieve lead

BITS Pilani

Pilani Campus

CC ZG503: Network Fundamentals for Cloud

Lecture No. 16 : Network Observability

One of the dirtiest secrets in systems engineering is just how many outages are never really fully explained or understood. Or how many can't actually be explained or understood given existing telemetry.

—Charity Majors

Co-founder and CTO of Honeycomb

Need for Network Observability

- Distributed systems are hard to understand, hard to control, and always frustrating when things go wrong
- Sandwiched in the middle between the endpoints is the network operator
- The modern data center with its scale and the ever increasing distributed nature of its applications only makes it more difficult to answer the questions that network operators have been dealing with since the dawn of distributed applications.
- ***Observability*** represents the operator's latest attempt to respond adequately to the questions.
- Along with automation, observability has become one of the central pillars of the cloud native data center.

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

What we will cover?

- Observability and how it is different from monitoring
- Understanding of the importance of observability
- The challenges of network observability

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

What is Observability?

- Term coined/used in current form by Twitter!
- Observability can be defined as the property of a system that provides answers to questions ranging from the mundane (*e.g. What's the state of BGP?)* to the existential (*e.g. Is my network happy?*).
- More precisely, observability is a way for an operator to understand what's going on in a system by examining the outputs provided by the system.
- As a concrete example of this, without having examined BGP process' data structures, packet exchanges, state machine, and so on, we can use the ***show bgp summary*** command to infer the state of BGP as a whole on a node.

"The Observability Engineering team at Twitter provides full-stack libraries and multiple services to our internal engineering teams to ***monitor*** service health, ***alert*** on issues, ***support root cause investigation*** by providing distributed systems call traces, and ***support diagnosis*** by creating a searchable index of aggregated application/system logs." **– Twitter Announcement**

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Observability: A different view!

Q1: "*Are all my BGP sessions in Established state?*"

- You can tell that by scanning a listing of the state of all BGP sessions.

However, a better and different question might be

Q2: "Which of my BGP sessions did not reach the Established state?"

- Unfortunately, you cannot answer this question unless the system provides you with a list of failed sessions.
- Of course, providing a list of all peering sessions and using that to determine which peerings failed to establish successfully is possible, but not as good as it does not support ***Usability*** and therefore, does not lead to a ***well-observed system***

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Observability: A different view! (Contd.)

In other words:

- How easily you can answer your question is a measure of how observable a system is.
- The more easily you can gather the information from the commands, the better you can grasp the crucial information, and the less information you need to keep in your short-term memory to build a map of the network as it is currently functioning.

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Observability vs Monitoring

"*Monitoring tells you whether the system works. Observability lets you ask why it's not working.*" -- Baron Schwartz, founder and CTO of VividCortex software

- For example, monitoring can tell you a BGP session is down, but observability can help you answer why.
- In other words, monitoring assumes we know what to monitor, implying also that we know the acceptable and abnormal values for the things we monitor. But the data center and the modern application architecture are large and complex enough to leave many things unknown.
- To reuse a popular cliché, "*Monitoring is sufficient for capturing the known knowns, whereas observability is required for tracking down what went wrong when encountering the unknown unknowns.*"

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

History of Network Observability

- Monitoring of switches (routers and bridges) arose in a time when those devices were appliances that only the networking vendor could modify.
- Therefore, accessing any information about the device required a protocol specifically developed for the purpose.
- That protocol was SNMP.
- It was one of the first technologies developed to allow for gathering data from remote systems.
- In SNMP, every feature has its own Management Information Base (MIB).
  - MIBs were designed to make discovery of the structure easy.
  - By doing something called an *MIB walk*, you could visit every element of the MIB supported by a device and obtain its value.

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Issues with SNMP-driven Observability

- The definition of a MIB for each new feature took two if not more years to stabilize.
  - vendors started implementing their own versions of the MIB while the standards bodies hammered out an agreement → **proprietary implementations and vendor-specific extensions!!**
- SNMP was also defined with a pull model, where the NMS pulls the information rather than the device pushing it.
  - a MIB walk risked overloading the CPU, either crashing it or starving other higher-priority processes such as routing protocols.
  - This meant that monitoring affected network stability
  - A solution to this was to space out the MIB walks
  - Unfortunately, when data is gathered over such long intervals, problems that occur over more granular time are not reflected in the data.

Like the STP, SNMP is widely regarded as a technology whose time is done.

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Network Observability Challenges

Why Is Observability Difficult with Networking? (vis-a-vis compute nodes)

- Two primary reasons:
  - the design of routers and bridges as appliances rather than as well-defined (and open) platforms → limited and proprietary tools for observability
  - the design of packet switching silicon. On a compute node, you can single-step through a function. It is not possible to single-step through packet forwarding silicon. The speed and the capabilities of the switch have been the deterrents to this transparency → switching silicon remains relatively opaque

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Network Observability in DCNs

- First, the challenges of Network Observability in DCNs:
  - *Multipathing*
    - DCNs are by comparison highly dense networks. Multipathing is the norm, not the exception.
    - Many operators use ***ping*** to check connectivity between endpoints, but ping does not test reachability over all paths.
    - Another common network troubleshooting tool is ***traceroute***. However, many vendor versions of the tool do not support multipathing
  - *Obscured by tunnels*
    - When network virtualization is deployed, no popular implementation of the ***traceroute*** command can reveal the multiple paths via the underlay to get to the destination
  - *Containers and microservices*
    - Network observability tools today are siloed between compute and network (and storage).
    - Further, the short lives of containers, makes it harder for network administrators to perform post-mortem analysis

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Network Observability in DCNs

- Next, the factors that make observability easier in DCNs:
  - *Simpler Nodes and use of COTS*
    - Clos topology frees up the network to be built entirely with a single kind of devices
    - Smaller & simpler boxes means that we can poll the device more frequently for information
  - *Network disaggregation*
    - cloud native data center have elevated the need to build switches as platforms, not as appliances.
    - As a result, the disaggregated switches run Linux as the network OS, permitting the unification of tooling

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Decomposing Observability

- As the Twitter announcement indicated, Observability can be decomposed into the following pieces:
  - Telemetry, or the gathering of data
  - Storage, or how the data gathered is stored
  - Monitoring
  - Alerting
  - Data analysis and ad hoc queries

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Decomposing Observability – Telemetry

- The process of telemetry can be broken down into:
  - what (what is collected)
  - when (when it is collected), and
  - how (how it is transported).
- In networks today, SNMP controls the answer to all the questions.
  - What is gathered is limited to the SNMP data models supported by the device.
  - When is determined by the frequency that doesn't cause SNMP to overload and bring down the system.
  - How is simply SNMP.
- However, even traditional network devices support some form of sending logs to a central location
  - Usually via the syslog protocol.

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Telemetry – What to collect?

- Brendan Gregg's (Netflix) **USE** model focuses on Utilization, Saturation and Errors:
  - monitoring receive packets, transmit packets, and byte counts on the interface determines utilization
  - drop and error counts determine errors
  - buffer utilization determines saturation
- Can gather state of all the hardware tables such as ACL tables, MAC table, and the tunnel table to monitor how full they are
- Data-points to calculate *Polarization Ratio*:
  - In Clos topology, high capacity is achieved when the bandwidth used on each uplink is about the same.
  - If this isn't happening, it could be an indication that either flow hashing isn't working well or that there are lots of elephant flows that have ended up getting hashed onto the same link.
  - If the polarization ratio isn't very close to 1, applications could be suffering suboptimal performance
- Google's famous book **Site Reliability Engineering** adds latency as another important metric
- Packet captures using sFlow are another kind of useful data for tracking what sort of traffic is flowing through the network.
  - Example, sFlow packet sampling using 1 in 30,000 packets in certain large networks can be useful.

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Telemetry – How to gather?

- Use of Streaming Telemetry tools like Prometheus and InfluxDB
- Big data tools such as Kafka are also increasingly being used to stream the data from various endpoints to a collector
- For logging, the most common model is to use *syslog* to push data to a remote *syslog* server. From there, data can be pushed into powerful log analyzers such as Splunk or the open source equivalent, ELK (Elasticache, Logstash, and Kibana)
- Pull Model vs Push Model? Depends on:
  - Need for *Just-in-time monitoring*
  - Requirement of Agents: With push, you need specific agents for each telemetry solution
  - Need for change of collection frequency → poll is easier to change
  - A hybrid approach could also be used. Example:
    - SNMP used a push model using what it called *traps* to notify the collector of critical, time-sensitive events
    - It also used the pull model for gathering data that was less time-sensitive

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Telemetry – When to gather?

- Trade-off involved:
  - Gathering data too infrequently makes it difficult to catch problems and understand your system well.
  - But gathering data too frequently can also be unnecessary (e.g. Checking for disk failures every 15 seconds)
- In general, a default period of 15 seconds seems appropriate across both network and compute
  - it is important to make the measurement interval for a specific service uniform across all nodes: compute and network
  - 15 seconds is also the default time used in Prometheus

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Decomposing Observability – Storing the Data

- Most tools use a time-series database
- when storage is expensive:
  - data can be frequently *rolled up* or aggregated after a time period (say hourly)
  - when aggregating data, average for the period, along with some other information such as min, max, and sometimes standard deviation can be stored
- However, with cheap storage and more experience in troubleshooting systems, rolling up data is generally frowned upon
  - Raw data, captured say every 15 seconds, is often stored for days and rolled up only after a week or two. This is the new norm.

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O'Reilly 2020

Decomposing Observability – Monitoring, Alerting & Data Analysis

- All data gathering has traditionally been used for Alerting and building Dashboards
- **Alerting** involves notifying the operator proactively of critical events, hopefully before the effects of the problems become widespread or catastrophic
  - With the increasing use of tools such as HipChat, PagerDuty, and Slack, any modern alerting framework must interact with these tools instead of merely sending email or lighting up a dashboard.
  - Disadvantage: “alert fatigue” - like the boy who cried wolf once too often. Can be solved by:
    - Reducing False Positives: watch for an anomaly to persist over a few cycles before triggering an alert
    - Limited audience: Alert only the required audience
    - Avoid Duplicate Alerts
- **Dashboard** consists of graphs that are supposed to reveal at a glance whether something is wrong with the system
  - Modern graphing tools such as Grafana allow you to pull data from multiple backends and customize your own dashboard
- **Data Analysis** tools such as Python’s pandas, and data query languages such as SQL, are emerging to help operators ask questions of the gathered data and look for correlations

Source: Dinesh G. Dutt. Cloud Native Data Center Networking: Architecture, Protocols and Tools, O’Reilly 2020

Observability: Scenarios / Use Cases

- Network Visualization Scenarios:
  - Simple Bytes Vs Ports visualization
  - Bytes Vs Times Visualization
  - Humans Vs Bots in Network Traffic
  - Weekdays Vs Weekends Traffic
- Cloud Use Cases:
  - Cloud gateways and load balancers can benefit from data-driven optimization
    - Just like auto-scalers use monitoring data to control the number of instances, gateways and load balancers use traffic data to optimally use these instances within a DC
  - Measurements/Analysis can help to better configure CDN/Caches/Routing paths across DCs/Edge systems in a Cloud

innovate achieve lead

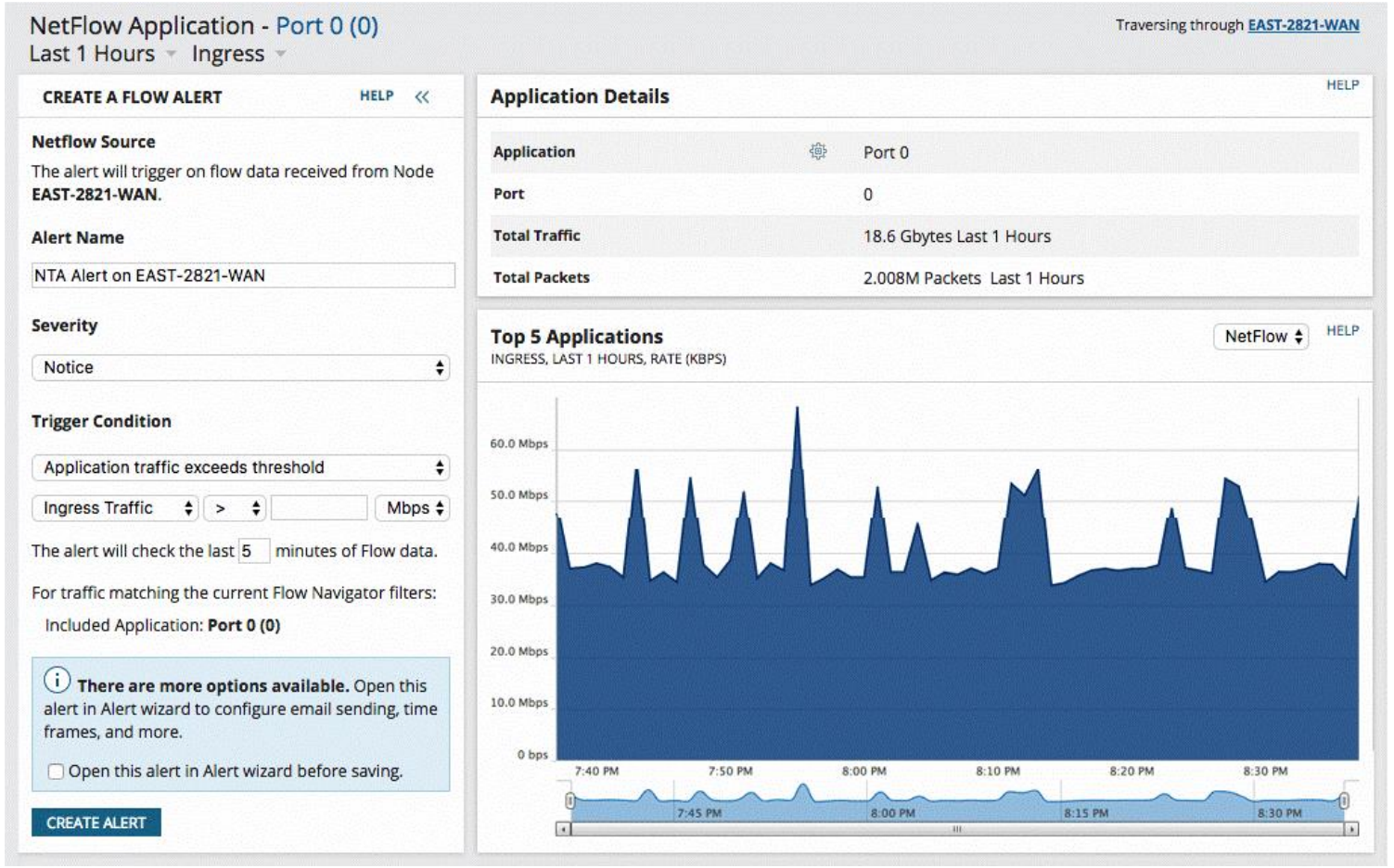
BITS Pilani

Pilani Campus

Case Studies: Network Observability Tools

Netflow Traffic Analyzers

- NetFlow is a network protocol used to monitor the flow of traffic over the network.
- By analyzing NetFlow data, you can get a picture of how network traffic flows across your network, including source, destination, congestion points, and volume.
- Using a NetFlow monitoring solution can help you analyze flow records to understand and optimize traffic within the network, so you don’t spend money on additional bandwidth that is not needed.



Source: [NetFlow Analyzer - Analyze Remote Network Bandwidth Traffic | SolarWinds]

Network Traffic Analysis using SiLK

- SiLK, the System for Internet-Level Knowledge, is a collection of traffic analysis tools developed by the CERT Network Situational Awareness Team (CERT NetSA), Carnegie Mellon University, to facilitate security analysis of large networks.
- The SiLK tool suite supports the efficient collection, storage, and analysis of network flow data, enabling network security analysts to rapidly query large historical traffic data sets.
- A SiLK installation consists of two categories of applications:
  - The packing system: The packing system collects IPFIX, NetFlow v9, or NetFlow v5 and converts the data into a more space efficient format, recording the packed records into service-specific binary flat files.
  - The analysis suite: The analysis suite consists of tools which read these flat files and perform various query operations, ranging from per-record filtering to statistical analysis of groups of records. The analysis tools interoperate using pipes, allowing a user to develop a relatively sophisticated query from a simple beginning.
- The SiLK software components are released under the GNU General Public License V2

Source: SiLK (cert.org)

Thank You!