O'REILLY®
Technical Guide

Practical Guide to Modern Networking Telemetry

How Telemetry Can Be Used to See Into Your Network's Performance and Usage Patterns

Early Release
RAW & UNEDITED

Compliments of
kentik®

Avi Freedman & Leon Adato

Practical Guide To Modern Networking Telemetry

How Telemetry Can Be Used to See Into Your Network's Performance and Usage Patterns

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Avi Freedman and Leon Adato

O'REILLY®

Practical Guide To Modern Networking Telemetry
by Avi Freedman and Leon Adato

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Megan Laddusaw
Development Editor: Gary O'Brien
Production Editor: Katherine Tozer
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

August 2025: First Edition

Revision History for the Early Release
2025-03-26: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9798341608900* for release details.

979-8-341-60888-7

[FILL IN]

Table of Contents

Brief Table of Contents (*Not Yet Final*)

Introduction

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the Introduction of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *gobrien@oreilly.com*.

About Modern Network Telemetry

Right at the outset, you may be asking yourself, "What's so important about Network Telemetry? What can it really do to make my work or network measurably better? This is not only an understandable question, it's a pretty common one that we hear. A lot. After all, the most important work done by IT practitioners is designing, implementing, and maintaining systems and architectures. Much of monitoring and observability needs to be set up in advance, with the benefit often coming into play only after something has gone wrong (and sometimes only after it's gone catastrophically wrong!).

So before diving into the tools, techniques, and technologies, we'll take a minute to talk about "the network" and step back and discuss

the benefits and return on investment that building robust observability options into your infrastructure can bring.

What Is "the network," exactly?

Back in 2023, in his book "The Ultimate Guide to Network Observability", Avi wrote,

> "People often refer to 'the network' in their organizations, but in most cases the network isn't one entity. It's a complex, diverse, fragmented, and loosely interconnected set of physical and virtual links and equipment, and it's housed in a variety of places, including data centers, corporate wide area networks (WANs), private and public clouds, the internet, container environments, and even inside hosts. Organizations own and control some of those resources but simply pay to use others."

Since that time, things remain largely the same. Sure, the cloud has gotten more cloudy (meaning opaque to users) even as it has become more ubiquitous within organizations large and small. And at the same time corporate LAN and WAN environments have not, by and large, become less complex (or sprawling, or expensive). In fact, the opposite has been true. Even as company's investment in cloud has increased, so too has their investment in on-premises networking gone up.

What Is "network telemetry," exactly?

Network telemetry refers to the data *FROM* and *ABOUT* your network, both from your network elements and from watching data moving through your network.

That includes simple things like device types, basic hardware metrics, and so on. But it also extends to status and performance information about the data moving around the network: source, destination, protocol, and more.

All of this makes network telemetry important (and valuable) for several reasons:

Infrastructure Performance and Health - Uptime, Health, Planning
: In order for user and application traffic to flow well across the network, the devices themselves need to be functioning well,

with enough CPU and RAM capacity, healthy hardware and optics, and links that work and aren't too full.

Watching logs, metrics, traffic flow, configurations, and other network device telemetry (plus a bit of active performance testing) is critical to running a great network.

Network and Application Performance
: All of the traffic for users and applications becomes packets flowing across the network, and the network is a great source of truth for shining light on both the net "composed" performance of the system, and even into applications that are having performance problems.

Cost
: At current infrastructure scale, cost can be enormous for many companies, and optimizing the network infrastructure often is a full time job - or more. Combining network telemetry with business data about cost can drive huge savings that often fund the entire network observability stack.

Life Cycle Automation Support
: Everyone's eager to move their application to the cloud, but is it really performing better there? Network telemetry helps you see the before, during, and after changes from more minor to whole migrations so you can be certain you're getting the improvement you think you're paying for - and didn't break anything!

Security
: Network telemetry remains a fast and great way to identify most cybersecurity issues, such as DDoS attempts, compromise and lateral movement, and the impact of botnets.

About You ("Is this book for me?")

This book is for you if any of the following things are (or might be) true:

- You'd describe yourself as a "learn and do" kind of person.
- You are comfortable with application monitoring and observability, but not networking, and you'd like to find out how

network monitoring and observability are different (and beneficial!)

- You are comfortable with networking, but not monitoring and observability, and you'd like to find out how network *monitoring and observability are different* (and beneficial!)
- You build, maintain, support, or are simply curious about "the network" and the ways in which network performance impacts everything that rides on top of it, from the data to the application to the overall user experience.
- You know how to look at a dashboard and interpret data presented in charts and graphs, but you want to understand how monitoring and observability data are represented in those forms.

On the flip side, what does this book presume you already know? To be honest, there's not a lot of requirements. Throughout this guide, we'll not only provide detailed information on terms and technologies, we'll point you to external content when we think some readers might appreciate a deeper dive than we have pages to cover.

That said, you will be most comfortable with the information we're sharing if the following things are generally true about you:

- You're familiar with the basic network devices - routers, switches, and firewalls - and what they do.
- You have a general understanding of cloud infrastructure concepts like virtual machines and cloud providers.
- You're aware of typical network security issues and threats, like DDoS attacks

We'll If you aren't rock-solid on those topics, DO NOT PANIC (also, don't put this book back on the shelf. We're not done paying off our kids' orthodontist yet.). Throughout this guide we'll offer information, instruction, and examples. And if you need more, we'll also provide links to background and deeper dives on these and other topics as we cover them.

About This Book: What Will I Learn?

We know that a book of this nature is an investment of time and attention. As such we wanted to suggest the return you may enjoy for spending some of your precious time here. By the end of this book the reader will gain a better understanding about:

Types of network telemetry
: Including traffic data, device metrics, events, synthetic measurements, routing information, configuration data, and business/operational data.

Network telemetry sources
: Including physical and virtual network equipment, servers, clients, cloud environments, and more.

The different planes
: (Management, control, data) that serve as points of contact where network telemetry can be gathered..

Ways to wrangle telemetry data
: Such as collecting monitoring information from devices, replicating data to analytics systems, feeding broader observability systems, and understanding data system requirements and trade-offs.

Ways to use network telemetry in situ
: From guided exploration (e.g., starting from a network map and zooming in), to unbounded exploration (e.g., drilling down on specific aspects of network traffic), through using telemetry as part of workflows, issue responses, and automation. Basically how you navigate and display and use network telemetry within the tool(s) you use to collect it.

Using network telemetry as part of your larger observability ecosystem
: Network data extends, enhances, and informs the telemetry you get from application and infrastructure monitoring. In this section we show you how to integrate and correlate the information so you have a better sense of what is happening from the top to the very bottom of the application stack and the OSI model.

But this guide isn't just geared to increasing your awareness. We'd also like to believe that we'll provide you with skills you can actively

apply. Therefore, after reading this book we also hope the reader will be able to:

- Justify (both to colleagues and decision makers) the business case for implementing and using a network observability solution in the workplace.
- Apply the knowledge about the different types of network telemetry, along with each type's strengths and deficits, in order to select the best mix of options when displaying data about a particular network, application, issue, or architecture.
- Design better monitoring and observability solutions by combining data and telemetry from various tools (be honest, we know you've got more than a couple) in ways that provide clarity and uncover issues.
- Build, adapt, and improve monitoring and observability outputs - everything from dashboards and reports to alerts to automated workflows - based on your (perhaps newfound) understanding of how network telemetry works.
- Lead the charge for better network observability by educating team-mates, departments, and even business leaders.

NOT About This Book: What WON'T I Learn?

Well-organized technologists don't just focus on the list of things they want to do, they also maintain healthy boundaries by keeping in mind the things that are NOT part of the roadmap. We'd like to show the same discipline and organizational rigor here by listing some of the things this book is NOT going to teach you about. We hope this will both set your mind at ease and also let you know whether the book you're holding has the answers you're looking for or not.

This book isn't going to teach you about things like:

- Basic networking. We won't explain in detail the OSI model, networking protocols, or how to configure a routing protocol on a layer 3 device (but will provide links to background on routers)
- How to evaluate, select, install, configure, or use a specific observability solution.

- How to evaluate, select, install, configure, or use a specific type of networking gear; or networking protocols; or standard networking architectures.
- How to evaluate, select, install, configure, or use a specific cloud provider; or how to migrate a particular application to (or from, or between) the cloud.

CHAPTER 1

Network and Telemetry Introduction

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *gobrien@oreilly.com*.

What IS Network Telemetry, Redux

We offered a high-level description earlier in the book, but now it's time to really dig in and offer a detailed explanation: *network telemetry* refers to data ABOUT your network, rather than the data that's moving AROUND your network. Network telemetry includes everything from the relatively simple (and relatively unimportant) device information (think: what are the make, model, and sub-components in the gear that make up my network); to the more relevant (and important) performance and state information about

the devices themselves (e.g. metrics related to CPU, RAM, disk, bandwidth, number of connections, and so on).

But where the concept of network telemetry really hits hard is when it tells you detailed information about the network traffic itself. Where are those packets coming from or going to - which might include everything from IP addresses to URLs to countries? How much of the data is one protocol vs. another? How many of the intermediate hops from source to destination are remaining static, and how many are changing? And those intermediate hops - are they the most efficient hops, or is some of the traffic getting hung up in sub-optimal routes?

Those questions are (or should be) deeply interesting to IT practitioners because they speak to aspects of application performance that can't be sussed out using higher level tracing options found in non-network-centric observability solutions.

This makes network telemetry important (and valuable) for several reasons:

Security First
: Saying "security first" is about 12 astronomical units (1.1 billion miles, or over 70 trillion average-lengthed bananas) away from actually *doing* something about security. Even saying "security is everyone's responsibility" is pointless (not to mention insulting) because sure, it is, but how many of us have ever earned a quarterly bonus because we "did good security"? Yeah, that's what I thought.

  Nevertheless, if a technique allowed you to not only become more aware of security issues, but identify their root causes and even address them, you probably wouldn't turn your nose up at it, either.

  Network telemetry allows you to do exactly that with specific types of security issues. First and foremost there are DDoS and botnet attacks. Not only does network telemetry tell you it's happening to (or on) your network, the right tools can show you when those events are happening "near" your network - meaning they're happening to someone else who is also using your ISP's infrastructure, using same routes as your data is traversing or hitting intermediate network gear your traffic is also using.

But it goes further than that. Because it can tell you both source and destination, as well as the volume, protocol breakdown, and applications involved, network telemetry can quickly identify when data is being exfiltrated. Even if no data is involved, having a network observability solution will allow you to put alerts in place that tell you when unexpected or unwanted protocols, ports, or destinations appear in the mix, or appear above a certain baseline.

Understand your network top to bottom and end to end
: Network telemetry gives you a view of your internal network (LAN, whether in the cloud or on premises) but it also tells you how your network traffic is performing once it exits the edge router and hits the WAN. That means you can make informed decisions about traffic and capacity management at the edge of your network. It's also the best way to understand how your precious (and expensive) investment in everything from your Internet provider(s) to your cloud architecture to your SD-WAN is performing.

  Used either as-is, or in combination with those higher-level application observability tools I mentioned above, network telemetry provides a unique view of your infrastructure, allowing you to pinpoint the area that's actually having a problem (versus just knowing that "the application is slow" or "we can't get to the database"), and thus speed resolution.

  All of this allows you to confidently understand your current use (including new and un-expected services and traffic patterns), plan for future growth, and thus control costs.

Make your cloud environment less… foggy?
: Smooth the transition of applications from on-premises to cloud by allowing you to first baseline the current state in terms of performance, load, traffic direction, etc. And then, once it's been transitioned to the cloud, you'll understand when performance differs, and where the breakdown is occurring, and why.

  In essence, network telemetry empowers you to gain a deeper understanding of your network's health, performance, and usage patterns. This knowledge allows you to proactively manage your network, optimize its performance, and ensure a smooth digital experience for users.

Anatomy of a Network

Let's be honest - networks are composed of simple base components but at scale *anything* but simple. Sure, most of the network diagrams you see in class are 3 routers (inevitably named "Spring", "Summer", and "Fall"), connected to a switch, which is connected to "the cloud".

And yet, in the real world, networks are composed of many (MANY!) different device types in multiple configurations and use cases.

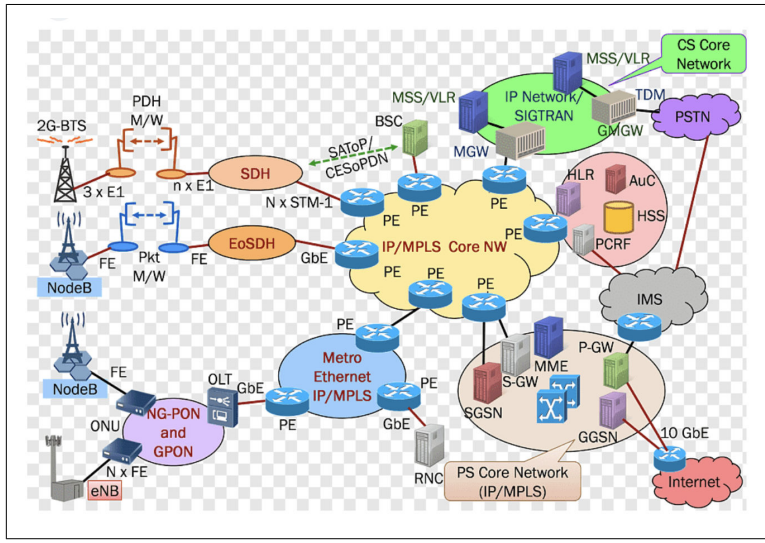

Figure 1-1. Title placeholder

So we wanted to take a moment to identify and define the most common devices and functions you might see, as these are going to be the sources of network telemetry that we discuss later in the book.

Broadly categorized, these include:

Routers, switches, and access points
: Physical and virtual equipment responsible for moving your traffic in your data centers and clouds; across your campuses and wide-area networks; and over the internet and broadband and mobile networks.

Servers, clients, and IoT endpoints
: Physical and virtual equipment that connects to your network, such as servers, clients like PCs, laptops, and mobile devices, and even internet-of-things (IoT) endpoints.

Cloud VPCs
: Virtual private clouds in your public cloud infrastructure. Includes subnets and the container environments where you deploy your microservices-enabled applications.

Controllers, service meshes, load balancers, and firewalls
: Physical and virtual devices that control, orchestrate, load balance, program network configuration, and filter inbound and outbound network and application traffic based on policies and threat intelligence across data centers, cloud, WANs, and the internet.

Transport devices
: Many modern transport devices (layers 1 and 2 in the OSI Model[1] in fiber, broadband, and mobile networks now support active and passive telemetry.

TAP/SPAN/NPB devices
: Physical and virtual test access points (TAP), switch port analyzers (SPAN), and network packet brokers (NPB) that provide port mirroring, testing, and monitoring.

But a few of those deserve a more detailed description. Once again, the purpose of this guide is not to teach you every aspect of network design, architecture, implementation or management. Instead, we want to describe the devices below in terms of the telemetry they emit and the insights that telemetry provides.

It's also important to note at the outset that each of these devices has their own hardware metrics that shed light on the network - everything from

- up/down status of individual components
- to CPU and RAM to fan, temperature, and power supply data

1 For those who need a reminder, the layers are: Physical, Data Link Layer, Network, Transport, Session, Presentation, Application. A popular mnemonic for this is: "Please Do Not Throw Sausage Pizza Away". For those who need more than a reminder, there's always wikipedia (LINK TO: *https://en.wikipedia.org/wiki/OSI_model*)

- to disk activity and configuration changes

To a greater or lesser extent, combining that insight with the other details can tell you where problems are occurring or, conversely, when a problem is actually downstream of a device which seems to be complaining but is in actuality simply unable to communicate with the next hop in the chain.

The final caveat before diving into the specifics of each device type is that even something as seemingly innocuous as inventory (especially when visualized as a map) can have a profound impact on your ability to understand how a network is performing and where the root cause of a problem may lie.

Network Routing/Switching Primitives

There are four foundational primitives of networking that most networking elements use in various combinations for IP networking:

Link-Layer Forwarding (usually Ethernet)
: Most of this guide will focus on IP networking - watching IPv4 and IPv6 traffic. But underneath the IP layer there are link layers that think in "frames" - not "packets". Those frames are usually forwarded by learning dynamically what addresses (called MAC addresses for Ethernet) are at various places. There used to be more link-layer protocols for wired networking, but Ethernet has dominated for decades, perhaps with a smattering of Infiniband for supercomping and AI data centers.

IP Forwarding (also called "routing")
: Taking packets from one interface (physical or logical) to another. Routing tables are populated that describe how packets get from one place to another. Usually it's done mostly by looking at the destination IP address of the packet, but it can get more complex and look at the source IP address or other parts of the packet.

Access Control Lists / Firewall Rules
: Another core routing primitive allows filtering and rate-limiting traffic according to specified policies. These policies are often called Access Control Lists, or ACLs, in routers and switches; Firewall Rules in security elements; and sometimes just "policy" in more cloud-y networking layers.

Tunnels / VPNs

: Originally used for more "exotic" configurations, tunnels are now commonplace and are protocol-based wormholes that connect different parts of a network together. Common tunneling protocols include GRE, IP (in) IP, and Wireguard. When these are exposed to users often they're just called VPNs, but people raised in networking often think of them as tunnels.

Network Device "Layers"

(Caveate: we're being really broad in our use of the word "device". We could say "element" but that's even more vague and prone to overlap with other technical elements… see what we mean? In any case, for our purposes the word "device" might mean a physical or virtual object which receives, processes, and/or emits data.)

Network devices forward packets or frames but how do you configure those devices, and how do the rules get loaded and the forwarding executed?

In the old world packets ran through the CPU but at current scales there just aren't general-purpose computers fast enough for that for many core network devices, so accelerated forwarding hardware is used.

Vendors and practitioners generally talk about multiple "planes" we interact with in network devices. With regard to monitoring and observability, a *plane* is really an abstract concept that broadly refers to a distinct *layer* of the network architecture where a specific process takes place.

Most commonly (in networking) you'll read about the management plane, the control plane and the data plane.

The *management plane* is concerned with device-specific information and tasks, such as the configuration of the device and its subcomponents. This is also the part of the architecture responsible for updates to firmware and operating system, security features, and (most importantly to us), monitoring.

The *control plane* is the part of the architecture responsible for how data (packets) are moved from one place to another - forwarding, refusing access, etc. Thus, the act of building a routing table is part of the control pane's functionality.

The *data plane* is the area of the network architecture that actually *does* the forwarding. Thus, the data plane takes a packet and sends it out a specific interface, based on the routing map assembled by the control plane. It's also the data plane that performs tasks like modifying a packet with additional header information, or applying a Quality of Service (QoS) rule.

Each of these planes has different kinds of telemetry we can consume or pull.

For example, CPU utilization from the management plane; table utilization from the control plane; and traffic speeds, summaries, and even detailed records of traffic sent/received from the data plane.

We'll map these in more detail in this module.

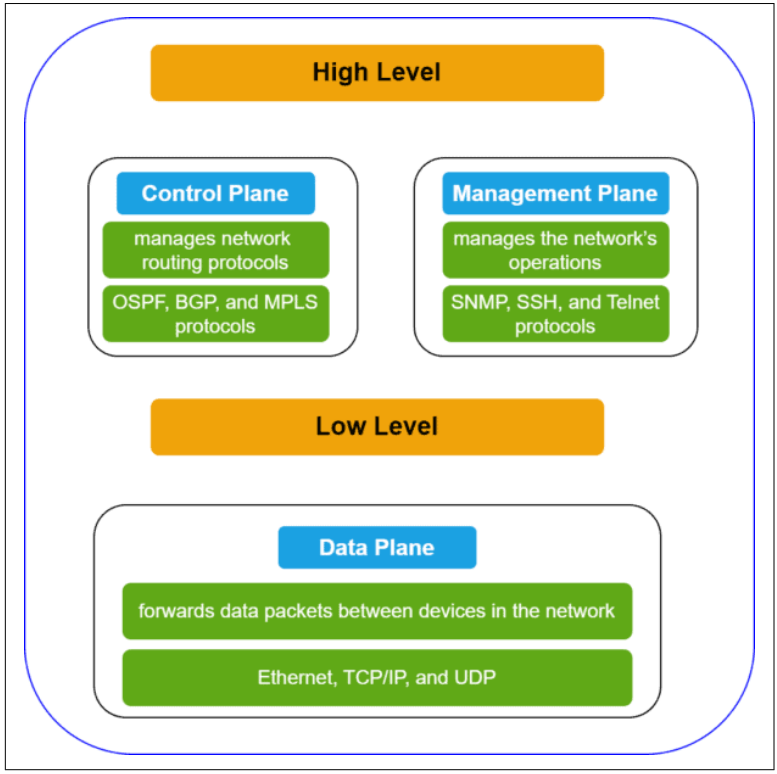

Figure 1-2. Title placeholder

Device Types

Routers
: Routers are hardware that generally run a Unix-based OS that interacts with users and other networking elements, and instructs specialized hardware (if present) how to forward, filter, and report on the packets going through.

  Routers have interfaces - physical or logical - and the physical interfaces usually have optics or wired ports that can be monitored.

  Many modern routers can do switch-like Layer 2 forwarding themselves, but generally, (unlike a switch), a router segregates Layer 2 forwarding unless told to do otherwise via configuration.

Layer 2 Switches
: Layer 2 switches move (typically) Ethernet frames around, but also have OSes, CLI, protocols, tables, and telemetry they generate.

Layer 3 switches
: Most switches today are very close to being routers, and do IP forwarding as well at line rate.

Virtual Private Clouds
: In cloud computing and cloud networking, Virtual Private Clouds, or VPCs, are primitives that behave like router interfaces.

Web Logs
: Web servers can emit log lines per transaction that describe various actions and transactions, both success and failure. These logs can shed a great deal of light on the network since they show source and destination IP addresses, application context, and performance information about both application and TCP-layer performance.

Load balancers
: With regard to the devices we're exploring in this section, load balancers may be the most fundamentally different as well as functionally narrow-focused. They're really Layer 4+ routers in some sense, and the metrics and telemetry you should look for

from them bears at least a passing resemblance to that of routers and switches.

Unlike routers, since they are part of the Layer 4+ transactions, they actually see and can report on response time, latency, throughput, error information, and even the traffic patterns of the data being handled. They can be great observation points but are often left out of network telemetry sources by teams.

Service Meshes
: Broadly speaking, *service meshes* are load balancers designed to talk to other backend software elements, not browsers/users. They can do health checks, load balancing, content rewriting, policy enforcement, and telemetry just like load balancers, and are almost always delivered as a software layer or service, unlike load balancers which are sometimes still physical appliances.

Firewalls
: When discussing routers, we mentioned that there are security controls that might be involved. Firewalls take that behavior and make it their entire raison d'etre. At the same time, there are still routing elements involved.

Hosts, Containers, and Kubernetes
: No, these are not (usually, but see below) network devices. Yes, they still matter - even in the context of "network observability".

  First and foremost, hosts, containers, etc are usually the ultimate and final end-points between which all network traffic flows. But second, they often have networking elements - real, honest-to-goodness, route-and-switch-type-data - contained within them, and thus they are (yet another) amazing and often-overlooked source of insight and telemetry.syslog

  Other host types - from servers to containers and beyond may also be (openly or secretly) acting as routers. All it takes is two network adapters (physical or virtual) connected to two separate network subnets, and - voila! - network routing is happening.

  In a modern way hosts get combined to run workloads, Kubernetes (K8s) is a hot topic!

  K8s has *namespaces* that are just collections of containers that perform various tasks - some in service of the K8s orchestra-

tion functions itself, and more (most) related to the application which forms the raison d'etre for the K8s pod in the first place.

But inevitably, the K8s namespace will include a system running system primitives like forwarding tables and traffic filters, and most interestingly nowadays, Container Native Interfaces, or CNIs, that are the "routing" controller for the underlying systems.

Getting network telemetry on hosts can be done broadly the "traditional" way - running an SNMP daemon to expose interface, CPU, and other metrics, as well as exporting traffic summaries via a flow-exporting daemon.

There is also a newer way of observing traffic and metrics on hosts via eBPF, which was developed as a modern kernel instrumentation infrastructure that can observe and control what the Linux kernel is doing. Generally, eBPF agents that look at the network are TCP and application-decode focused and can provide enriched flow-like summaries with performance and application context - but are often architected to ignore lower layers, so may miss errors or attacks at lower protocol layers.

Tap/SPAN/Network Packet Brokers (NPB) devices
: When network devices themselves can't process or emit traffic (flow-like) telemetry of their own, one solution is to use network taps (optical splitters), SPAN (Switch Port Analyzer) functionality on switches, or Network Packet Brokers that are "super SPAN" devices with more complex filtering and policy. Many of these devices just copy packets to the things that will generate telemetry, but some of them can generate flow summaries themselves.

Common Telemetry Types: The Four Pillars

If you've been anywhere near a conversation on social media about "observability", you will inevitably hear someone mention the three four pillars of observability (really, telemetry). These have been the subject of great debate, passionate argument, internecine flame wars, and (most of all) confusion.

At the heart, the goal was (and still is) an attempt to both list and then group the essential techniques that make up observability, such

that customers could have a single reference by which they could evaluate the vendors and their products in this space.

Of course, it could be argued that the reason for some of the disagreements boils down to vendors not accepting a definition which would put their own products into a less-than-glowing category.

Arguments aside, the pillars seem to have settled themselves into some mix of "M.E.L.T.":

Metrics
: Individual (usually numerical) data points which can be gathered, graphed, averaged, etc to show a trend.

Events
: Strings of text and numbers stamped with a time and a source to show that such-and-such occurred to so-and-so system and thus-and-this time.

Logs
: Sources of messages and other output which might be aggregated across many systems and comprise multiple layers of the architecture, from low-level hardware to high level application.

Traces
: A coherent collection of information that show how a "transaction" within an application traverses multiple systems, and the ways the transaction is performed at every step along that path - usually augmented with a transaction or trade ID to allow correlation of all of the steps in that transaction.

These categorizations are, by and large, fine. However they are (and always have been) biased toward application telemetry. And that's fine, but it doesn't work completely when discussing network telemetry.

Therefore, Avi and I are presenting a different framework for understanding the different types of data that you'll commonly encounter when monitoring a network infrastructure.

Types of NETWORK Telemetry

Admittedly, monitoring and observability data comprise a really long list of a wide range of data types. A quick glance at the next section will show that more than a few aren't even relevant to network

devices (looking at you, PerfMon counters). So let's take a moment to describe the essential telemetry types that will make up the bulk of our focus in this book:

Traffic
: This describes any data element (metric, trace, etc) that shows how your traffic is flowing across networks. Sample formats include NetFlow, sFlow, IPFIX, VPC flow logs, traffic data in JSON, and packet data via PCAP and eBPF for connections, process, and container context in cloud-native environments.

Device metrics
: These tell you the state or health of your physical and logical network equipment. Sample formats include SNMP, syslog, and streaming telemetry.

Events
: This indicates events like an attempted login, a threshold has been met, or a configuration has been changed. Sample formats include SNMP trap and syslog.

Tables
: Snapshots/state of the various tables in a router, mostly for forwarding/routing.

Synthetic
: These "synthetic" measurements reveal performance metrics such as latency, packet loss, and jitter, and can be triggered or collected via device telemetry interfaces. They span client and server endpoints, network equipment, and internet-wide locations at both the network and application layers.

Configuration
: This (typically static) data represents the operating intent for all configurable network elements such as topology information, IP addresses, access control lists, location data, and even device details such as hardware and software versions. Sample formats include XML, YAML, and JSON files.

Business or operational
: Often called "layer 8," this data provides business, application, and operational context about what the network is being used, and can be added to telemetry to help network pros measure

impact, understand value of certain traffic, and prioritize their work.

DNS
: DNS telemetry helps put other network data into context by indicating from or to where traffic is coming or going. Most DNS information comes in text-based files.

Drill-Down: Telemetry Types

The previous list of types of telemetry is concise and focused rather than comprehensive. But now that we understand the definition and the value of network telemetry, as well as the devices that make up a typical network, we need to take a moment to list out all of the various data types available for monitoring and observability.

This section will go into both the protocols themselves and, in some cases, touch on the Device Health and Status: Syslog.

Syslog is a protocol which allows one machine to send a message ("log") to a server listening on TCP or UDP port 514. This is more often used and at higher volume when monitoring network and *nix (Unix, Linux) devices, but network and security devices such as firewalls and IDS/IPS systems send system and component logs - and can be configured to send even more detailed logs, though care is needed not to overwhelm the CPU (control plane).

Syslog messages are similar to SNMP traps, but differ in that syslog messages are relatively freeform and don't depend on the MIB-OID structure required by SNMP.

In addition to being more freeform, syslog tends to be "chattier" than SNMP traps. However, it's more flexible because many applications can be set up to send syslog messages, whereas SNMP traps are generally used much more sparingly, and most companies have much more broad and robust log collection ability and scale than they do for SNMP traps.

Traffic: Flow Monitoring (sFlow, NetFlow, VPC Flow Logs, eBPF, and others)

Standard device metrics can tell you the WAN interface on your router is passing 1.4Mbps of traffic. But who's using the traffic? What kind of data is being passed? Is it all HTTP, SSH, or something else?

You probably recognize the term "NetFlow". It's been around for a while. But what we're referring to is really the broader category of traffic data, or "flow", in general. Examples of these protocols that report on network traffic details include NetFlow, sFlow, JFlow, IPFIX, and VPC Flow Logs, among others. Despite the differences in strengths or weaknesses, implementation specifics, and more, they all have similar aims.

Flow monitoring answers those questions. It exports in terms of "conversations"—loosely defined as one period of data transfer between two computers using the same protocol. If DesktopComputer_123 is sending a file to Server_ABC via FTP, you may see a snippet or snippets of it via flow monitoring. Flow monitoring usually at least includes protocol, source and destination IP addresses and ports, and the number of bytes and packets observed. Note that if the flow records are sampled, the bytes and packets need to be multiplied back by the sample rate to form the approximate total traffic actually observed…

While people usually refer to both sFlow and the more connection-oriented NetFlow/IPFIX/VPC Flow Log protocols as flow monitoring, and once parsed, the data their records contain look pretty similar, the way they observe and report on traffic actually is quite different. sFlow exports headers from a sample of packets and is much easier to implement and takes less resources. It is also usually much more real-time. The traditional connection-oriented protocols need to build software or hardware tables of the connections (called flows), accumulate data like bytes and packets seen, and every so often pick records to export, usually according to a flow expiry timer.

eBPF-observed traffic can be a bit of a mix of those, and is done on hosts that run native processes, containers, and VMs. Most eBPF traffic exporters are concerned primarily with TCP and UDP connections that terminate on the kernel of the server they are running on. This enables adding a lot of context not possible from outside the compute layer - for example, process ID, command line arguments, process memory usage, sometimes application decodes, and if containerized, pod and namespace. On the other hand the traditional approaches often don't watch all the packets and can miss ICMP, non-IP protocols, bad layer 2 framing, and especially, traffic in hosted VMs.

While some flow sources only support un-sampled export (typically, firewalls and many VPC Flow Log sources), generally flow protocols and their software and hardware implementations will require sampling. And even if they don't, you may want to enable sampling to reduce the amount of compute and storage you need to receive, process, store, and query the traffic data.

Flow data is often captured by a network device located somewhere in the middle of the conversation—usually one or more routers near a local edge or core of the network (or one router at each remote location if there are site-to-site communications which don't go through the core). In prior decades, many routers that looked like they supported flow export protocols actually had real problems with platform stability and/or flow accuracy, but most platforms now support it fairly well.

Note the two machines in the conversation (DesktopComputer_123 and Server_ABC, in my example) do NOT need to be monitored. Just the network devices that they are either attached to, or that might see their conversations..

Traffic and Policy: Packet Monitoring

For higher granularity monitoring than flow allows, packet monitoring is sometimes used, though much less than years ago. This is generally enabled with optical taps that split a fraction of light off to be observed; by asking switches or routers to copy packets (called now generically SPAN, or Switch Port Analyzer); or by using specialized switching devices called packet brokers to do this.

The packet copies generally go to servers that can generate flow, store the raw packets, and/or perform DPI (Deep Packet Inspection) to go even deeper than most flow protocols allow.

While instrumenting modern infrastructure with full packet monitoring can be incredibly expensive and is not usual in most greenfield builds, packet analysis can generate flow-like data with application decoding and performance data simply not available in most flow-based traffic monitoring implementations, and storing all of the packets allows for more detailed security inspection and investigation.

A Device-to-Telemetry Rosetta Stone

Having gone over the types of devices you might meet in your travels through the network; and the types of telemetry you might encounter in your observability solution, I thought we'd take a moment to identify which types of telemetry you will be able to coax out of various device types.

Table 1-1. Title placeholder

Device type	Telemetry Summary
Routers	Just as important is the information about the traffic itself - in this we're grouping the volume, errors, discards, packet counts, packet errors; as well as the sources, destinations, URLs, ports, and protocols involved. Along with the essential service of routing traffic, routers might also do a fair bit of security functions, especially if ACLs or other permission-based handling is involved. And all of those functions are going to carry with them their own sets of metrics, logs, and other telemetry which further informs your picture of the infrastructure.
Layer 2 Switches	Just like routers, switches provide a full complement of metrics under the broad umbrella called "bandwidth" - sources and destinations (this time in the form of MAC addresses rather than IPs or ports - along with bandwidth volume, errors, discards, packet counts, and packet errors.
Layer 3 Switches	From a network telemetry point of view, these devices combine everything you might find in a router and an L2 switch. Thus, the very thing that makes them more powerful in terms of network architecture, is what makes them more complex with regard to monitoring and observability.
Virtual Private Clouds	VPC flow logs are equivalent to the flow records (e.g., NetFlow, sFlow, etc.) in a traditional (on premises) network. Various cloud network components such as a VPC, a subnet, a network interface, internet gateway, or a transit gateway, can generate flow logs which are used in the same way as NetFlow data. These started with the core cloud network functionality (VPC, NSG, VNET) but have versions now available for many other cloud services like storage, firewalls, and other layers of the stack.
Load Balancers	On the other hand, there are some very specific data types, like connections (active, passive, etc), request count, information on healthy/unhealthy hosts, and even the health and performance of the load balancer infrastructure itself. If the load balancer is in a cloud environment, you might also be interested in request tracing, changes to the load balancer itself (due to elastic compute), and connectivity to other cloud-based elements like storage, content engines, and more. It's also important to note that "load balancer" is a general term for a device which might be specifically designed for application, network, or even gateways (virtual devices like firewalls, or intrusion detection systems) traffic.
Service Meshes	Like load balancers, service meshes see a rich set of data that can be incredibly useful to establish Mean Time to Innocence (MTI) and to debug full-stack issues ("is it the network?").

Device type	Telemetry Summary
Web Logs	Nowadays most web/application servers are behind load balancers or service meshes and those tend to be the observation points. But classically and still sometimes, getting a stream of web transaction logs can be a great way to see performance / latency of requests, whether they succeeded or failed, the source information of the browser that made the request (even if the packet was subsequently bounced off a VPN or other obfuscating device), and cookies and other data harvested by the web application itself. All of that, along with the telemetry from load balancers and routing information, can paint an incredibly clear picture of the "intent" and performance of the traffic on your network.
Firewalls	For the intrepid network telemetry spelunker, that means being prepared to gather all of the bandwidth information (volume, errors, etc) along with information about the device connections themselves - data about blocked connections, intrusion attempts, traffic patterns that have been pre-determined to be "suspicious", and network requests that violate built-in security policies. All of those will have a wide range of specific metrics and logs associated that shed light on who tried to do what, when, and to whom.
Hosts, Containers, and Kubernettes	In K8s-land (and let's be honest, there aren't many other serious contenders at the time of this writing), intra-container and inter-container network telemetry can be just as important to your understanding of performance, availability, fault, and reliability as traditional LAN and WAN insights. It can get a bit complicated because of tunnels and NAT, and from an observability perspective, using IPv6 un-routed space to disambiguate workloads and their IP addresses can be very helpful as a NAT alternative.
Tap/SPAN/ Network Packet Brokers (NPB) devices	In the early days of network devices, many couldn't generate their own traffic telemetry well or stably, but while today most can, they don't do much to observe traffic performance, and often really need to sample to keep up even now. Taps and NPBs can make it possible to send line-rate un-sampled traffic records, and/or send copies of the frames/packets to appliances that can do deeper inspection of traffic performance and contents than routers and switches are capable of doing.

Collecting data, network style

….and (broadly speaking) how to get that data out of your devices.

In the next module, we're going to show you how to "wrangle" your data - meaning take it from its initial form and make it usable in various observability tools. But before getting to that, we thought it would be helpful to offer an overview of how to get the data itself.

To be clear, this section will not offer comprehensive device- or vendor-specific instructions. It will mostly focus on how, generally speaking, to get the various telemetry types (SNMP, streaming telemetry, etc) out of devices, with a few illustrative vendor-specific examples. So more of "how to do an SNMP get" than "how to

configure SNMP on a Cisco IOS switch versus doing it on a Juniper router."

Telemetry Deep Dive: SNMP and Streaming Telemetry

SNMP has been around for decades, and therefore folks interested in learning more about it will find a wealth of information, history, and tutorials at their disposal. For that reason, we're going to keep this section relatively brief, and trust the reader to find the specific instructions they might need..

SNMP can be broadly divided into two types of information, based on the delivery method - push-based or pull-based.

SNMP Traps

This is the pull-based option. They are triggered on the device being monitored, and sent to another device that is listening for those messages (a trap receiver or trap destination). Nothing is needed on the part of the monitoring solution except to receive and store those messages, and then correlate them with telemetry from other sources.

All the configuration for this is done on the device itself, meaning that every device on your network that needs to send traps has to be configured with the trap destination, along with the security elements (community string or username/password). We'll explain more about those options below.

SNMP Get

As hinted at earlier, SNMP Get requests are pull-based. A remote system sends a request to the machine being monitored, requesting one or more pieces of data. The most basic form of this command is:

```
snmpget -v <SNMP version> -c <community string> <machine IP or
name> <SNMP object>
```

What that looks like in practice:

```
~$ snmpget -v 2c -c public 192.168.1.10 1.3.6.1.2.1.1.5.0
iso.3.6.1.2.1.1.5.0  = STRING: "BRW9C305B289C1"
```

Rather than get into the weeds of the various methods of SNMP-Get (which will largely be handled under the hood by whatever monitoring and observability tool you use), the point is that SNMP works to

bring data about a remote system (whether numeric or text) into a local repository for storage, tracking, and visualization

A Note about SNMP versions

Each version of SNMP brought enhancements aimed at addressing the needs and challenges of network management at the time.

SNMP version 1 (SNMPv1) uses community strings, which act as a rudimentary form of authentication.

SNMP version 2c enhances the performance aspects of SNMPv1, including GetBulk requests.

SNMP version 3 strongly emphasizes security and privacy with the inclusion of authentication, privacy (encryption), and access control. The drawback is that configuration of SNMP v3 is somewhat more complex both for each device to be monitored and for the monitoring solution that collects the data.

Consistency is the Key

Or it would be, if SNMP had more of it. It's not just that a particular data point (temperature, or CPU, or bandwidth) is inconsistently presented from one vendor to another.

It might not even be such a glaring issue if the difference existed between two different types of devices (routers and load balancers, for example).

Or between two models of the same device type from the same vendor.

But the fact that there are differences in the way the same data point is presented between two sub-elements (like interfaces) on the same device is simply egregious.

We aren't sharing this to downplay SNMP's continued importance in the landscape of network telemetry options, but rather to alert you that some mapping of different SNMP variables might be needed to get a unified idea of, for example, "interface traffic", even on devices from the same vendor.

Streaming Telemetry

As described in the "Types of Telemetry" section, streaming Telemetry (ST) is notable for having many of the same data points as SNMP, and more; and a far more consistent data structure; and a significantly higher granularity with a significantly lower impact on the device being monitored.

Originally developed as an alternative to SNMP, the goal was to move away from poll-based observation (and each monitoring system separately polling devices), and towards pushing defined data from network devices, to then be consumed by all of those systems.

So what's not to love? Well, getting it set up can be a bit of a challenge. This mostly arises from the newness of the technology and the lack of experience on the part of both engineers developing observability solutions; and network professionals who are implementing ST in their environments.

Oh, and the fact that it's not supported on a wide range of devices yet. That's another critical factor.

Telemetry Deep Dive: NetFlow, sFlow, and Other Traffic Sources

NetFlow (and its variations like sFlow, JFlow, IPFIX, and others, which we'll refer to from hereon out collectively as "NetFlow" unless we're discussing something particular about the other variations) was practically purpose-built with the goals of network telemetry and network observability in mind. While NetFlow isn't the only protocol a network engineer may need, it is almost certainly the primary one they will refer to for the richest level of insight.

NetFlow is push-based, meaning the device observing and generating the telemetry (kind of. More on that in a minute) sends data to a listening device.

As you can see, the single machine is able to report on conversations between devices on the internal network, the internet, and many points along the way such as peer routers.

Telemetry Deep Dive: API

Talking about APIs in the context of network telemetry is a slightly different discussion than the one you might have when describing

APIs for programming, automation, or retrieving cute cat pictures. Or Chuck Norris jokes. Or things that are slightly more useful like the time for sunrise/sunset.

An older way to gather metrics is still also sometimes required - CLI scraping via libraries that log into the device command line, issue commands, and retrieve and parse responses.

But why bother with APIs or CLI scraping?

APIs are attractive because they allow the process of telemetry collection to move into a slightly more modern paradigm, both for individuals and for monitoring solution engineers.

But both API and CLI scraping also allow access to some network device telemetry that is unavailable via SNMP or ST (Streaming Telemetry).

For example, many devices don't expose all the operating parameters of physical optics (temperatures and light levels) via SNMP and ST.

This adds yet another dimension to the unification required to consume all of these metrics, and as in our discussion of ST, is left as an exercise to the observability tools team and/or vendors.

Telemetry Deep Dive: Synthetic Transactions

If there is one drawback to all of the methods we've discussed so far, it's that it requires the active presence (and participation) of those pesky network elements called "users". Meaning: without people actively using an application, system, database, etc there's no (or at least significantly less) data moving through the network, and therefore fewer data points to collect, and therefore fewer opportunities to identify issues.

Also, most sources of network metrics and traffic telemetry don't have any concept of user or traffic performance (latency, loss, jitter, and throughput). So how best to see how well things are performing?

And this is where synthetic transactions come into play. As the name suggests, these are actions which generate traffic by means other than authentic user behavior.

Just for context, and maybe as the simplest example of this type of monitoring, both "ping" and "traceroute" could be catego-

rized as synthetic transactions. However, synthetic transactions are, more broadly speaking, pre-set (you might think of them as "pre-recorded") actions which run at regular intervals. At its most simple, you can think of them as a macro running on your laptop every x minutes. But in reality the technique (and its benefits) go so much further than that.

First, the action can run from multiple locations, but report the results into a single repository. This provides insight as to whether an issue is happening with the target application or system (i.e. the problem is consistent from all locations at the same time) or if the app/system is fine, but certain routes to it are impacted.

Moreover, synthetic transactions can be multi-step. Rather than just checking if a web page "is up" (returns a 200 code), a single transaction can:

1. Go to a website
2. "Click" on the login page
3. Log in with pre-set credentials to a dummy account
4. "Click" on the account balance page
5. Verify that the balance is $2.75

Not only will you be able to determine whether the app/system/site is up or down, you will also gain insight as to the timing of the overall transaction and each of the steps along the way.

The range of applications, systems, and conditions that can be checked via synthetic transactions include network status and performance tests - from the simple checks (ping and traceroute); to more complex for things like BGP, ASNs, and CDNs; Internet protocol tests for DNS and http responsiveness; Web-centric tests for things like page load time or API responsiveness; and multi-step tests like the one described above.

Configuration management

Configurations sit at the border between monitoring and management. On the one hand, a configuration identifies how a thing (a system, application, or operation) works on a fundamental level. On the other hand, knowing that a configuration has changed - and especially when it's changed - can often make the difference

between having no idea why a system is suddenly having an issue; and knowing when the problem REALLY started and where to look.

At a high level, configuration management as it relates to monitoring and observability is the process of first identifying which files or elements count as "configuration", doing an initial scan of that object, and then repeating the process and noting changes.

Generally this is not done by watching files on a filesystem for the "big vendor" routers and switches, though it may be on your Linux or BSD-based services.

So in that case, the monitoring system would have to either use an API (hopefully!) or log into the system using a terminal protocol like ssh (because friends don't let friends use telnet), running some variation of the "show config" command, capturing ("scraping") the results, and saving that information to the monitoring system (whether as a file or a database entry). That process would be repeated, and the two results scanned for differences. But that's not all. Those same configurations can be scanned for everything from syntax errors to security errors.

If issues are found, it could trigger an alert; or it might show up on a periodic report with the new, changed, deleted, or problematic elements highlighted.

To summarize: Configurations may seem at first glance to be outside the scope of what a monitoring and observability tool might care about. It certainly doesn't seem to fit the description of what "network telemetry" includes.

But in truth configurations are so tightly bound up in system, application, and sub-component stability and performance that NOT monitoring this critical aspect of the environment seems foolhardy at the very least, and possibly negligent when you consider the worst-case (which are sadly becoming more common in these days of companies joining the security-breach-of-the-week club) scenarios.

About the Authors

Avi Freedman and **Leon Adato** have, collectively, over 70 years experience in the tech industry, with particular focus on networking, monitoring, and observability. Both recognize that, after the hard work of building a solution is done - whether that be a network, a datacenter, or an application - the hard work of keeping things running starts. And that's usually where the problems really start. Their decision to collaborate on this book arose first and foremost to share all the samples, examples, stories, and lessons they usually share in the booth at conferences, or in talks, or when helping customers; but also to provide a resource to the readers themselves: who might need to articulate those same lessons to colleagues, managers, or the odd (*very* odd) person at a dinner party.