2023 20th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA) | 979-8-3503-1943-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/AICCSA59173.2023.10479236

A Data Collection Platform for Network Management

Amar Abane
NIST
Gaithersburg, USA
amar.abane@nist.gov

Abdella Battou
NIST
Gaithersburg, USA
abdella.battou@nist.gov

Abderrahim Amlou
NIST
Gaithersburg, USA
abderrahim.amlou@nist.gov

Tao Zhang
NIST
Gaithersburg, USA
tao.zhang@nist.gov

***Abstract*—Network management relies on extensive monitoring of network state to analyse network behavior, design optimizations, plan upgrades, and conduct troubleshooting. Network monitoring collects various data from network devices through different protocols and interfaces such as NETCONF and Syslog, and from monitoring tools such as Zeek and Osquery. To unify and automate the monitoring workflow across the network, this paper identifies and discusses the data collection requirements for network management, reviews different monitoring approaches, and proposes an efficient data collection platform that addresses the requirements through an extensible and lightweight protocol. The platform design is demonstrated through an adaptive collection of data for network management based on digital twin technology.**

***Index Terms*—network monitoring, network management, publish-subscribe, digital twin network**

I. Introduction

Recent advancements in network management have led to the development of Network Management Systems (NMS) with inventory management, network topology visualization, configuration assistance, and network diagnostics. However, as network complexity and service diversity increase, configuration and management errors become more frequent and identifying the root cause of issues becomes more challenging.

To address these issues, various network analysis and troubleshooting tools have emerged to improve network management by processing all kinds of network data with artificial intelligence (AI) and machine learning (ML) techniques. Therefore, data collected from the network is crucial for the effectiveness of these techniques. This data may include device configuration and status, alarms, topology, port/link status, activity logs, traffic and flow statistics, and user information, and service performance. Network data is typically collected using monitoring tools through a multistage process that involves measuring, transmitting, aggregating, presenting, and storing the data [1]. However, the current monitoring approaches have limitations in gathering network data (see Section III). For example, measurement platforms [2] concentrate solely on the communication performance metrics, such as end-to-end latency. Telemetry interfaces such as NetFlow are inadequate in capturing device configurations. Management protocols are restricted to device configuration data and do not cater to devices' performance data, or have a limited support of telemetry such as gNMI [3]. Furthermore, the frequency and conditions for collecting each type of data are different, as well as the data format as discussed later.

Hence, a need arises for a more general and flexible network monitoring methodology. In this paper, we propose a network monitoring platform design that addresses these requirements. The platform allows to gather all necessary network data from diverse sources, without the bottlenecks posed by centralized monitoring. The platform enables flexible and automated data collection with minimal communication overhead.

This paper is structured as follows. Section II identifies the main requirements in modern network monitoring. In Section III, popular monitoring solutions and approaches are discussed. Section IV presents the design of the proposed data collection platform. Section V discusses a use case for the platform considering the emerging concept of Digital Twin for network management. Section VI concludes the paper.

II. Network Monitoring Requirements

Network monitoring starts with data collection, where information is requested from network devices and mapped into an information model, either tool-specific or general (such as JSON). The formatted data is transmitted to the management station where it undergoes aggregation, filtering, and representation according to the network data model. The network data is then used for various management purposes and some of it may be stored for auditing and long-term analysis.

A network monitoring platform should facilitate data collection, aggregation, and storage, including integration of tools that request the data [1]. Moreover, the data to collect varies in type, frequency, volume, and sources [4]. Hence, a suitable monitoring must be able to handle these diverse data types with a uniform workflow for efficient processing and presentation of data. The workflow should also be extensible to support new monitoring tools and parameters.

To minimize resource consumption, the platform must have a lightweight design with minimal communication and processing overheads. Increasing monitoring frequency can lead to higher resource consumption and data generation. Hence, the platform should have the ability to dynamically adapt monitoring frequency and metrics based on resource availability. Monitoring flexibility includes the scheduling of probes and

the ability to choose between periodic, on-demand, and event-based monitoring. To improve efficiency, the data delivery model should be leveraged to avoid duplicate messages.

Each data item obtained from monitoring should have unique identification and associated metadata about the originating request [4]. This information is used by storage systems to retrieve data when needed. Enriching collected data with user-specific labels is also useful to improve search capabilities.

Security is critical and encompasses authenticating data sources and consumers, and managing which users are authorized to access each set of collected data. Whereas securing monitoring workflow is manageable in environments with a limited set of data sources and consumers, it becomes more challenging as the platform flexibility increases. A schema to define and enforce policies is required to provide fine-grained control over authorization and access control while keeping a reasonable complexity for certificate and key management.

III. Background

Four broad categories of monitoring approaches gained recognition in recent years. These approaches are inspiring for the design of a network monitoring platform that meet the demands outlined above.

A. Internet measurement

There are several platforms that offer public monitoring on the global scale of Internet. One such platform is RIPE Atlas [5], which leverages probe devices hosted by users across the Internet to collect data on network connectivity using predefined probes. The data collected is made publicly accessible and users can conduct custom measurements. Each probe relies on registration servers to identify its controller, which manages the probe by sending a schedule for measurements and receiving results. The results are then centrally processed, enriched, and stored.

Another popular network measurement toolkit is PerfSONAR [2], designed to identify end-to-end network problems through measures such as bandwidth utilization, latency, and packet loss. Its architecture is divided into three layers, with various types of probes at the lowest layer, web services to invoke probes in the middle layer, and a user API at the highest layer to trigger measurements and access results.

While RIPE Atlas and PerfSONAR offer valuable network monitoring capabilities, their functionality is limited by predefined probes and do not offer an architectural foundation for efficient data distribution among multiple producers and consumers.

B. Measurement facilitators

Several platforms provide flexible large scale network monitoring solutions by addressing specific aspects such as storage, interoperability, or scalability.

M-Lab [6] is a server infrastructure that facilitates measurement data exchange through effective resource allocation policies. Several network monitoring tools leverage M-Lab servers for measurement coordination and data ingestion.

mPlane [7] is a scalable infrastructure for distributed Internet measurement. The platform offers flexibility in monitoring through its support for single, iterative, and coordinated measurements, and enables dynamic integration of user-defined measurements through a probe's capability description and request mechanism. However, its point-to-point communication design limits the potential of its workflow and message scheme.

The authors in [4] propose a data collection method for Digital Twin Network (DTN), where the data streaming component informs the DTN of the data it can collect from network devices. The DTN sends commands to the data streaming component to request the desired data. However, this approach does not address other critical considerations such as efficient data delivery and data identification.

C. Standardization efforts

The standardization of network monitoring is being advanced through the efforts of consortiums and working groups. One such effort is the gNMI protocol [3], which offers a vendor-neutral interface for device management. It provides a unified service for both configuration and telemetry, enabling clients to exchange capabilities and retrieve data or subscribe to events from devices. However, the use of the same interface for both telemetry and configuration may result in suboptimal data delivery. While monitoring data can tolerate best-effort delivery with some data loss and duplication, management commands necessitate reliable and consistent data delivery. Additionally, gNMI currently lacks support for essential network diagnostic tools such as Ping and Traceroute, despite their availability through the gNOI protocol, the gNMI complement for network operation.

D. Cloud monitoring and logging

The utilization of monitoring tools integrated within cloud platforms [8] has become widespread for monitoring applications and resources, including virtual private cloud (VPC) networks. These monitoring tools gather performance data, resource utilization metrics, and logs from various sources including the cloud provider's systems, managed products, applications, and VMs with agents installed. The collected data is pushed and processed through a monitoring suite, where it undergoes filtering, ingestion, labeling, and storage. The stored data can be further analyzed, visualized, and processed through user-defined alerts and metrics.

The data collection process is typically achieved via HTTP endpoints to which the monitored sources continuously push data. Although this approach is simple, with a centralized REST API, it does not offer a control over data collection beyond filtering the data at ingestion stage. On the other hand, cloud monitoring tools benefit from the security provided by cloud platforms through the use of flexible identity and access management (IAM). IAM allows for precise control over users and services access to data and resources.

IV. Proposed Platform

The proposed platform is named "CaSpeR", which stands for Capability-Specification-Result/Receipt, reflecting the message sequence that outlines the data collection workflow. This section describes the design of the platform. For the purpose of clarity, technical considerations such as encoding format and a detailed discussion of message structure have been omitted.

A. Overview

The data collection platform design aims to streamline the acquisition of heterogeneous data through three key features. Firstly, it encompasses source discovery, data request and retrieval, and automated data processing to efficiently describe and integrate the data. Secondly, the design offers a scalable solution through a flexible scheme that balances the level of granularity in data request and the associated overhead. Lastly, the architecture is designed for easy implementation and minimal impact on network resources by allowing for seamless integration with the management and control plane.

The architecture of the proposed platform shares the design principles of the mPlane [7]. These principles include adopting a unified protocol for data description, requests, and results. Its protocol facilitates the discovery of monitoring capabilities and enables seamless coordination of their execution. Additionally, the architecture leverages self-contained and idempotent messages, ensuring that every message carries sufficient information to identify the monitoring task it relates to and can be easily detected and ignored in case of duplicity.

While these design principles simplify the architecture and provide flexibility in controlling monitoring tasks, they do not address all the requirements for effective data collection. To fully realize the potential of this approach, crucial enhancements have been introduced to increase flexibility, enhance data semantics, and improve data distribution. These enhancements include: (i) the use of a publish-subscribe model for exchanging messages, which reduces communication overhead and enables diverse data dissemination options compared to point-to-point protocols, (ii) allowing data sources to manage the local execution of monitoring tasks through request aggregation and adjustment based on the solicitation level, and (iii) providing expressive data description through the use of semantics and application-defined labels.

B. Workflow

The platform comprises two main components that communicate through messaging: *services* and *clients*. The service collects data and the client requests it.

Three types of services are considered in the platform:

- Probe services (or agents) perform basic data collection tasks, such as track the status of a component, run measurements, or read data from a device.
- Sink services interface with a data store to save and retrieve data results or provide graphical visualization.
- Aggregators are services that also act as clients to other services. They collect data from multiple services, breaking down a complex data collection task into simpler ones, and producing aggregated results. Depending on their level of intelligence, aggregators may also provide automated iterative monitoring, data transformation and correlation, etc.

In this platform, services broadcast *capability* messages to describe the data they are able to collect and the information required for data retrieval. Each data collection task should be represented by a separate capability. Clients receive capabilities and use them to request data by sending a *specification* message to the relevant service. The service responds with a *receipt* message indicating acceptance or rejection. If the specification is accepted, the collected data is disseminated through one or multiple *result* messages. The service executes the specification to the best of its ability and may adjust the execution.

Clients and services interact in the platform without establishing end-to-end sessions. Messages are exchanged via publish-subscribe topics. This model is chosen for its efficiency in disseminating messages to large groups of clients and services, reducing data duplication, and minimizing control messages.

Multiple services can offer the same capability, and a single client can submit specifications to multiple services. Similarly, a single service can distribute results to multiple clients. This decoupled interaction allows each service to manage the local execution of specifications to optimize resource utilization.

C. Message types

Each message conveys all necessary information for its processing, including the derivation of the topic name to receive or publish the next message (see SectionIV-F). Figure 1 depicts an abstracted structure of the message types. The *type* attribute refers to the nature of the data collection task being described by the capability, which can range from real-time measurements (measure) to reading static data (collect) or database retrieval (query). The *endpoint* is a structured name that contains the namespace in which the capability is defined (e.g., /casper/useast-1/datacenter-1), the name of the capability (e.g., probe-port), and the identifier of the service or group of services providing the capability (e.g., switch-1).

Execution parameters supported by the capability are listed in the *parameters* section, which is a map containing parameter names and types. The allowed temporal scope is defined in the *schedule* section, which is a formatted string indicating start and stop time, period, etc. Parameters and schedule are filled with actual values by the specification message. The *result-keys* section defines the metrics or attributes that can be returned by the capability, and the specification message selects the metrics requested from the service. In result messages, *result-values* is a two-dimensional array containing values corresponding to the *result-keys*. Remaining fields will be introduced in later sections.

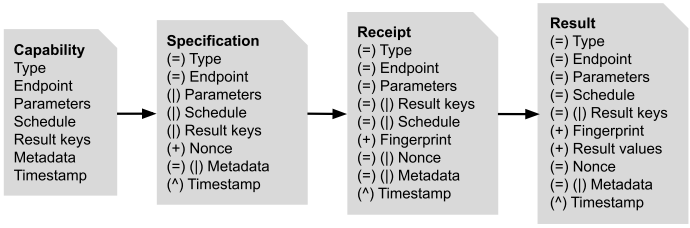

Fig. 1. Structure of the main messages. (=) means that the field and its value are copied from the previous message, (+) means that the field can be added in the current message, (|) means that the field is kept from the previous message and its value is defined/updated in the current message, (∧) denotes a field with a specific value for each message. Combination of signs represents an alternative.

Figure 2 displays the relationships between messages. A specification carries all relevant information from the referenced capability. A receipt includes information from the linked specification and updated information about the expected result messages. A result contains information from the specification used to generate it. The interrupt, redemption, result, and termination messages all provide information on the task requested by the original specification. Beside capability, specification, result, and receipt, the workflow includes other message types. An *Interrupt* is used to inform a service to terminate the execution of a specification. A client asks a service to resend results of a specification by sending a *Redemption* message. A *Termination* message is published by a service to inform specification execution is terminated. An *Exception* is sent by clients/services to signal workflow errors.

D. Data Collection Management

An *operation* is the data collection process requested by a specification. Each operation has a unique *fingerprint*, which is a hash of the type, endpoint, parameters, and result-keys defined in the specification. The fingerprint is used to group messages related to a specific operation. If the fingerprint cannot be computed from a message due to a modified field, its value is explicitly included in the message.

Each operation has an implicit *id* which is generated by combining the fingerprint with a client-generated *nonce*, allowing the client to differentiate between multiple executions of the same operation. The combination of the fingerprint and id, along with the client's identity information, is known as the *session*, and is used by the service to manage operations execution.

The use of the fingerprint, id, and session in messages between the service and client allows for a balance between resource consumption, monitoring accuracy, and scalability. The service can adjust the requested operation as long as it complies with the specification. For example, if a specification requests a probe every 10 minutes, the service can fulfill it with an operation that produces results every 5 minutes. The service can determine if a similar operation is already running by using the fingerprint, and if so, adjust its schedule to meet the new specification. This is known as **schedule adjustment**.

The receipt informs the client of the expected result-keys and the topic on which the result messages will be published. If the service performs schedule adjustment, it updates the nonce in the receipt, and publishes the results for all specifications that are aggregated in the same task, either via the same topic or in separate topics for each specification. The service has the option to skip schedule adjustment or to perform it and still publish results in separate topics.

E. Result Management

The receipt is used by the clients to associate the results with a specific operation id. To present results in a concise format, the service may opt to split the result-keys across multiple result messages, a process referred to as **result splitting**. In this case, the service updates the result-keys in each message to match the corresponding result-values and includes the original operation fingerprint for identification purposes (see Figure 1).

The *flow* section is used to control the publishing of results. The service can set the flow to "stream" in the capability to indicate real-time streaming of results or "batch" to indicate that results will be published once the operation is completed, either through a single or multiple messages. Depending on the nature of the operation and the available resources, a service can enforce one flow option, or allow the client to select the delivery mode in the specification. Upon receipt of result messages, the client can organize and reassemble the results based on the operation type, fingerprint, and id.

The *metadata* section helps in handling the results. The metadata *type* can be set to "point" to indicate that each result message represents a single point of data from the operation. In this case, the client can reconstruct the full operation data using the operation id. If the metadata type is set to "table", it indicates that each result message contains the complete data collected during the operation. The metadata *format* specifies how result-keys and result-values should be displayed in a chart, using chart definition languages such as Vega-lite [9]. The metadata *labels* carry user-defined key-value information for tagging results. Labels defined by the service are included in subsequent specifications, receipts, and results. User-defined labels are set in the specification, kept in the corresponding receipt but not in the results as they may be shared among multiple clients.

F. Messaging topics

Figure 2 displays the relationships between message types and the topics where they are published. Capabilities are published in the "capability" topic, while the specification, interrupt, and redemption messages are published in the topic derived from the capability's endpoint (i.e. "<endpoint>.control"). The receipt is published in the topic derived from the specification (i.e. "<endpoint>.receipt.<fingerprint>.<nonce>.<timestamp>"). The topics where results and termination messages are published are derived from the receipt (i.e. "<endpoint>.results.<fingerprint>").

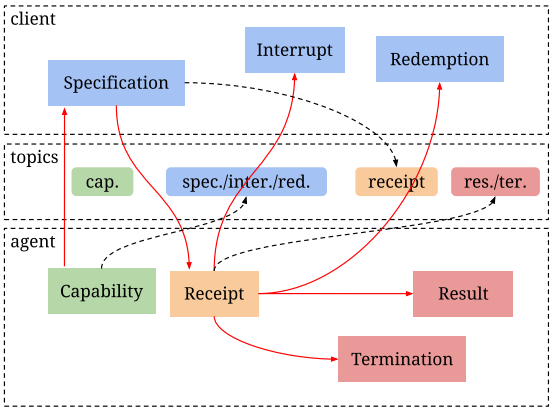

Fig. 2. Message relationships and topics derivation. Red arrow from A to B indicates that message B derives from A. Dashed arrow from A to B indicates that topic B is inferred from message A. Each message is published in the topic represented with the same color.

G. Security

The security of CaSpeR communications is independent of the messaging system being used. As depicted in Figure 3, an Authorization Server (AS) enables the administrator/owner to control which permissions (publish-specification, read-result, publish-result) are granted to each identity (client and service) for each capability. Clients and services are authenticated through their own accounts managed by the administrator on the AS. Access control is based on roles. A role groups together a set of permissions necessary for participating in the workflow.

Three base roles are defined. The *specification-sender* role enables clients to request new operations and includes *publish-specification* and *read-result* permissions. The *result-reader* role, allows clients to only access data from ongoing specifications and includes only the *read-result* permission. The *result-publisher* role, enables services to publish data and includes the *publish-result* permission. Additional roles can be created for more fine-grained access control. The administrator/owner can grant and revoke roles for each identity. A policy links an identity, a role, and a set of capabilities using the hierarchical naming structure of the *endpoint*.

The security scheme combines HTTPS between the AS and clients/services, and self-secured encrypted messages to provide authentication and authorization (see Figure 3). The self-secured encryption scheme is similar to the role-based security framework demonstrated in Named Data Networking [10].

Each client and service has a certificate signed by the AS. Clients and services sign each message along with its endpoint, allowing the receiver to verify its authenticity. A service checks the signature of a specification (or interruption, redemption) and the client's certificate, and uses the verified identity to retrieve the client's role from the AS. The service can then accept or reject the specification (or interruption, redemption) based on the permissions allowed for the namespace to which the specification endpoint belongs. Similarly, a client checks the message signature and verifies that the service is authorized to produce messages for a given endpoint.

The AS manages symmetric content encryption/decryption keys (CK) for each namespace. Services and clients retrieve the CKs for namespaces they have access to based on their roles. Note that, with this scheme, if a client has a result-reader role, it can also decrypt specifications related to the capability. However, this does not pose a significant security threat since message derivation from a capability is clearly defined in the protocol.

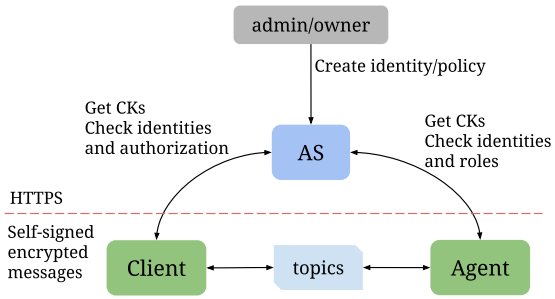

Fig. 3. CaSpeR security scheme.

V. Case Study: Adaptive Data Collection for Digital Twin

The concept of Digital Twin Network (DTN) has emerged to improve network management and automation using modeling, emulation, and AI/ML techniques [11]. A DTN is a real-time digital representation of a physical network, which can be used to design and evaluate network optimization, plan network upgrades, conduct "what-if" analysis, and troubleshoot the network [12].

We discuss in the following how the Casper platform can be used to collect network data for a DTN. In this case study, the DTN is a client and the data is produced by various sources in the network acting as services.

A. Basic data collection

The DTN collects a variety of data from network equipment from different vendors, which use different protocols. The platform provides a uniform interface for DTN services and applications to access this data, hiding the protocol specifics.

To build a digital version of the network, a DTN needs to continuously collect network topology (via port and link status), device configuration, alarms and logs, and various measurements reflecting network performance such as service Key Performance Indicators (KPIs) and device telemetry.

Network performance data is collected periodically. The capability advertised by the corresponding service has the *type* "measure", and uses the "point" metadata type. Data is sent to the DTN as a stream, with the collection frequency defined in the specification's *schedule* section.

Real-time updates of network topology changes are critical for effective operation of the DTN. The corresponding capability has the *type* "measure", and uses the "point "

metadata type. The ”on-event” option in the ”schedule” section of the specification allows for real-time reception of topology changes.

Device configuration data is locally stored on the device and collected by a service, which advertises it using a ”collect” capability with a ”table” metadata type. In a device, some configurations change on a daily basis, and others change rarely or less frequently [13]. To handle that, the DTN sends two specifications for the same capability, one for the infrequent changes and one for the frequent changes. Both specifications set the *flow* section to ”batch”. To collect data from all devices while reducing the number of exchanged messages, one capability can be implemented to collect data from more than one device, using one column in the result-keys for the device name and specifying the device(s) to collect data from in the parameters section.

Logs and alarms are parsed at the service and described using a collect capability with ”table” metadata type. The DTN can have logs published periodically and alarms received in real-time.

B. Adaptive data collection

In a real-world production network, such as an enterprise data center, the amount of data generated can grow rapidly, making it infeasible to continuously collect and process all of it. While flows and packets are effective in capturing network behavior, the resources required to handle them can quickly increase. Connectivity tests like Ping and Traceroute are convenient to handle and offer valuable information for troubleshooting, but they also add to network traffic and require precise source and destination network addresses to generate meaningful results. On the other hand, device alarms and telemetry can be obtained and analyzed with minimal communication and processing overhead, and can be continuously collected for a significant portion of the network.

The Casper platform allows to realize an adaptive data collection strategy that leverages these observations to request expensive-to-collect data only when necessary. The aggregator services perform iterative data collection, and can be rapidly deployed using VMs or containers and integrated with the data collection workflow. The aggregator services first analyze performance telemetry and alarms to detect potential network issues. If a problem is detected, they evaluate and investigate it by testing connectivity using Ping and Traceroute, and then collect specific flows and packets [14].

C. Data replication

Given the limited capacity of the DTN to handle real-time network data, lower-cost storage tiers with data retention policies can be employed for long-term storage of the collected data. This historical data can be useful for training ML models [12] or for replaying previous network events [11].

The platform’s workflow and message idempotency allow for easy replication of the requested data in independent data repository systems without additional processing overhead at the service or the need for explicit client-service connections.

While a DTN system can automate monitoring and operation, human expertise is still crucial in production networks. To aid in this, sink services with graphical user interfaces can be deployed with minimal overhead as they consume copies of the data that is sent to the DTN.

VI. Conclusion

Collecting large amounts of heterogeneous data becomes necessary for modern network management tools, whereas it used to be an additional feature in traditional NMS and Software-Defined Networking (SDN) solutions. Therefore, data collection needs a dedicated workflow instead of being implemented alongside control protocols as it has been so far. This need is addressed by proposing an extensible data collection platform that encapsulates the various interfaces used in network monitoring.

The platform can also be used for other telemetry purposes. For example, it is currently used for optical quantum network metrology [15].

References

[1] S. Lee, K. Levanti, and H. S. Kim, “Network monitoring: Present and future,” *Computer Networks*, vol. 65, pp. 84–98, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912861400111X

[2] The perfSONAR Project and contributors, “perfSONAR.” [Online]. Available: https://www.perfsonar.net/

[3] R. Shakir, A. Shaikh, P. Borman, M. Hines, C. Lebsack, and C. Morrow, “gRPC Network Management Interface (gNMI),” Internet Engineering Task Force, Internet-Draft draft-openconfig-rtgwg-gnmi-spec-01, Mar. 2018, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-openconfig-rtgwg-gnmi-spec/01/

[4] C. Zhou, D. Chen, P. Martinez-Julia, and Q. Ma, “Data Collection Requirements and Technologies for Digital Twin Network,” Internet Engineering Task Force, Internet-Draft draft-zcz-nmrg-digitaltwin-data-collection-01, Nov. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-zcz-nmrg-digitaltwin-data-collection/01/

[5] RIPE Atlas, “RIPE Atlas.” [Online]. Available: https://atlas.ripe.net/

[6] Measurement Lab, “The M-Lab NDT data set,” https://measurementlab.net/tests/ndt, (2009-02-11 – 2015-12-21), bigquery table `measurement-lab.ndt.download`.

[7] B. Trammell, P. Casas, D. Rossi, A. Bär, Z. B. Houidi, I. Leontiadis, T. Szemethy, and M. Mellia, “mplane: an intelligent measurement plane for the internet,” *IEEE Communications Magazine*, vol. 52, no. 5, pp. 148–156, 2014.

[8] Google, Inc., “Cloud Monitoring.” [Online]. Available: https://cloud.google.com/monitoring

[9] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, “Vega-lite: A grammar of interactive graphics,” *IEEE Trans. Visualization and Comp. Graphics (Proc. InfoVis)*, 2017. [Online]. Available: http://idl.cs.washington.edu/papers/vega-lite

[10] Z. Zhang, Y. Yu, H. Zhang, E. Newberry, S. Mastorakis, Y. Li, A. Afanasyev, and L. Zhang, “An overview of security support in named data networking,” *IEEE Communications Magazine*, vol. 56, no. 11, pp. 62–68, 2018.

[11] C. Zhou, H. Yang, X. Duan, D. Lopez, A. Pastor, Q. Wu, M. Boucadair, and C. Jacquenet, “Digital Twin Network: Concepts and Reference Architecture,” Internet Engineering Task Force, Internet-Draft draft-irtf-nmrg-network-digital-twin-arch-02, Oct. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-irtf-nmrg-network-digital-twin-arch/02/

[12] P. Almasan, M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, D. Perino, D. López, A. A. P. Perales, P. Harvey, L. Ciavaglia, L. Wong, V. Ram, S. Xiao, X. Shi, X. Cheng, A. Cabellos-Aparicio, and P. Barlet-Ros, “Digital twin network: Opportunities and challenges,” 2022. [Online]. Available: https://arxiv.org/abs/2201.01144

[13] H. Hong, Q. Wu, F. Dong, W. Song, R. Sun, T. Han, C. Zhou, and H. Yang, "Netgraph: An intelligent operated digital twin platform for data center networks," in *Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration*, ser. NAI'21. New York, NY, USA: Association for Computing Machinery, 2021, p. 26–32. [Online]. Available: https://doi.org/10.1145/3472727.3472802

[14] Y. Zhu, D. Chen, C. Zhou, L. Lu, and X. Duan, "A knowledge graph based construction method for digital twin network," in *2021 IEEE 1st International Conference on Digital Twins and Parallel Intelligence (DTPI)*, 2021, pp. 362–365.

[15] A. Abane, D. Anand, A. Amlou, L. A. Oucheggou, Y.-S. Li-Baboud, A. Battou, J. Bienfang, I. Burenkov, P. Kuo, A. Migdall, S. Polyakov, A. Rahmouni, P.-S. Shaw, O. Slattery, and T. Gerrits, "Optical quantum network metrology," 2022.