Using Network Digital Twin Visualization for Application Traffic Engineering

Felix Stumpf, Leon-Niklas Lux, Sebastian Rieger
Applied Computer Science Department
University of Applied Sciences Fulda, Germany
{felix.stumpf, leon-niklas.lux, sebastian.rieger}@cs.hs-fulda.de

***Abstract*—Network Digital Twins (NDT) support the management and evaluation of increasingly complex network topologies by virtually replicating and abstracting physical infrastructures. This paper presents a real-time visualization for Containerlab-based NDT environments to assess their applicability for NDTs and identify relevant application scenarios. The implemented design follows the principles outlined in the IRTF draft for NDT, implementing an interactive force-directed visualization with high-fidelity modeling using gNMI topology and telemetry data. By using the data to emulate traffic engineering scenarios based on the real network's state in the NDT, network operators can comprehend and improve application-based network load. Additionally, a time machine feature is introduced that allows users to visually inspect historical network states, circumvent problems regarding human perception of large datasets, providing a deeper understanding of the network's behavior in response to different stimuli, supporting well-informed decision-making.**

***Keywords*—Network Management, Application Traffic Engineering, Network Digital Twin, Real-Time Data, Visualization**

I. Introduction

With the increasing complexity of modern network infrastructures, Network Digital Twins (NDTs) have emerged as a promising tool to replicate real-world topologies in isolated artificial environments. This approach enables experimentation and troubleshooting without disrupting the production environment. Real-time visualization plays an important role in NDTs to support network monitoring, analysis, and optimization, as can also be observed in existing network management and monitoring products. Many network providers and hardware vendors have started to offer NDT-like visualization solutions that integrate real-time data into network topology representations to improve operational insight and decision-making [1][2]. This paper introduces DigSiViz[1], a topology and telemetry visualization tool for Containerlab-based networks. DigSiViz consists of a Python-based backend and a React-based frontend, enabling interactive real-time representation of network states. It aggregates data from the components in the topology via the gRPC Network Management Interface (gNMI) and provides granular insights into real-time and historical network behavior through a time machine feature. This feature allows operators to step through past network states and analyze relevant events in detail, improving troubleshooting and decision-making. Thanks to the combination with an NDT approach, individual traffic scenarios can be analyzed and tested without interfering with the real network. For example, failure or congestion on the control and data plane can be replicated to improve application and service performance utilizing traffic engineering. Based on the DigSiViz prototype, the paper discusses the benefit of real-time visualizations as a key requirement for NDT and applies them in application traffic engineering scenarios.

[1]https://github.com/netlab-hfd/digsiviz

II. Related Work

Digital Twins (DTs) have been established as an effective tool for mirroring real-world environments by creating virtual replicas that enable users to gain a deeper understanding of their real-world counterparts by allowing simulations and experiments without impacting the actual production environment. In addition, DTs offer extensible data mining opportunities that enable the user to gain insights that would otherwise remain hidden in complex systems. [3] already stated in 2022 that the networking community is increasingly interested in building NDTs. They provide examples of use cases such as troubleshooting, what-if analyses, network planning, and anomaly detection. In particular, the challenges regarding data collection and the fine-grained control and management of NDTs were emphasized[4]. The IRTF draft of NDT Concepts and Reference Architecture describes requirements for an NDT environment. The network visualization requirement in chapter 9.3 defines that NDT systems have to use network visibility technology to present data and the network model in the NDT with high fidelity to allow users to discover characteristics otherwise hidden in the real network. Furthermore, network visibility can use algorithms such as force-oriented layout for topology layout[5]. As stated in the previous section, there are also some developments in the commercial sector with regard to the visualization of NDTs, as Nokia promotes its implementation of NDTs with the ability to inspect them through GUIs[1]. Telefónica also advertises that it runs NDTs to optimize their planning processes, offering interactive maps displaying the network topology in combination with live statistics[2]. Compared to these commercial solutions, DigSiViz supports heterogeneous network vendors by leveraging gNMI, presents a new time machine to analyze historical data, and supports what-if and traffic engineering testbeds using the underlying DigSiNet framework. In previous work[6], an environment was presented to run multiple NDTs as siblings. In addition, [7] highlighted the importance of data

978-3-903176-75-1 ©2025 IFIP

visualization for NDTs and identified challenges for different data granularities for network management and telemetry. [8] designed a real-time network monitoring system with SNMP featuring visualization functionality. Unlike other related work, they do not focus on (force-)graphs but on multiple dashboard-like displays that visualize multiple key figures of the network, such as the traffic of the connected network devices. They consider a 30-second polling interval as real-time, enabling network administrators to inspect the current condition of the network.

This paper presents an approach for real-time visualization of an NDT topology operating within a virtual Containerlab environment. Using gNMI, the solution can achieve fine granularity with a polling rate of less than one second. Additionally, it features a time machine functionality that allows users to recall and visualize historical data for further analysis and inspection. To evaluate the benefit and practical impact of the approach, a traffic engineering use case is presented and discussed. It allows network operators to analyze and experiment with real-world traffic characteristics observed in real-time and historical network topology and telemetry data. This also holds potential for current path-aware and transport-oriented approaches[9] to test, e.g., application performance in multipath scenarios or application-, service- or flow-based routing. The main research goal of this paper is therefore to enable the evaluation and improvement of the fidelity of data modeling and visualization in NDT.

III. Requirements for Real-Time Visualization

The proposed approach for real-time visualization of a DigSiNet NDT, operating within a Containerlab environment, enables interactive presentation of telemetry data streamed from the virtual devices in the topology. The management data to monitor and visualize is retrieved from the gNMI interfaces of the network devices, as gNMI provides a modern standard with secure transport that allows efficient and fast serialization and data access[10].

DigSiNet is developed in Python using the pyGNMI client library[11]. It abstracts Containerlab to read a reference network model and adaptations, then creates topologies as digital siblings[6]. Monitoring data coming from the devices in the topology is polled in real-time. Besides the real-time polling, the approach presented in this paper features a time machine functionality that allows navigation backward through the visualized historical network management data. Historical data is stored and remains searchable and retrievable based on unique timestamps. The user can then pause the real-time visualization and browse the saved data incrementally. With fixed timestamps and a paused state, data analysis becomes easier and more user-friendly, enabling in-depth analysis of the network's reaction with fine granularity. The visualization features a force-directed layout with network devices visualized as nodes and links between the devices visualized as edges, as outlined in the IRTF draft[5]. In addition, the data processing of the visualization application should not add much latency to the chain. Consequently, it features real-time communication to transfer data in a decoupled way by using WebSockets.

IV. Real-Time NDT Visualization in DigSiNet

The proposed application architecture processes and aggregates router data through gNMI interfaces from a given Containerlab network topology defined in YAML files [12]. This definition will be used directly during the initialization of the visualization application. The topology definition used is parsed in order to obtain full knowledge of the topology. The application is implemented following the client-server architecture principle, decoupling the aggregation of network information from the visualization of the gathered information. This enables the backend to run in the background, pulling router data using gNMI, even if no visualization is currently accessed. Frontend and backend are interconnected through WebSockets and REST APIs. Static information that has to be passed only once while initializing the visualization is passed through the REST API, while real-time data is streamed through WebSockets. The backend thus dynamically pushes updates to the frontend. The frontend automatically reacts to these updates by refreshing its components and the visualization. The frontend can also alter the state on the server if needed, which is required to achieve the time machine functionality, allowing to go back in time in a granular manner to inspect and analyze the historical states of the network. Figure 1 shows the conceptual layout of the application.

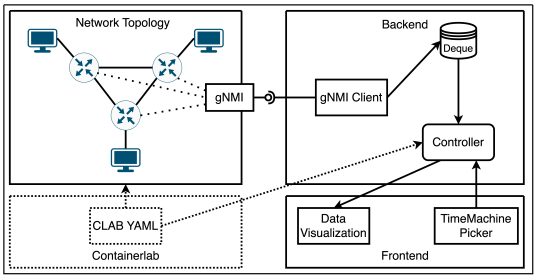

Fig. 1. Conceptual overview of the DigSiViz architecture.

A. Backend

The backend parses the topology definition in the form of defined nodes and links and converts it into a structure that defines the graph later shown in the frontend. As router IP addresses can be dynamically assigned during the start of Containerlab, the backend also gathers their real management IP addresses by directly accessing the Containerlab Command Line Interface (CLI). When the frontend connects, the backend switches to real-time mode. By triggering the WebSocket connect event, data acquisition starts through a background task. The backend then polls the gNMI data from every router in the given topology through the gNMI client with a polling interval of 0.5 seconds. To fetch gNMI data, the gNMI client needs the management IP and the credentials of the routers, the gNMI port, and the gNMI paths to be monitored. To gather the needed real-time data of the routers, for example, the interface paths of each router are monitored by the gNMI client. To save bandwidth and time, in the following examples, only the

router interfaces configured in the YAML topology definition are monitored. Furthermore, the polling is executed in parallel to achieve an even finer granularity of the values through a higher possible polling frequency. In the test setup, a gNMI polling cycle takes approx. 200 ms per router. Measured from the beginning of the execution of the pyGNMI GET request until the data is successfully saved by the backend, a complete polling cycle of the entire topology will take approx. 250 to 300 milliseconds when executed in parallel. Executed in a single thread, the time for each router is summed, leading to a significantly longer polling cycle. These measurements confirm that a 0.5-second polling interval is a suitable and safe assumption for our test setup when polling is executed in parallel. The aforementioned timeframes can vary depending on the topology size, container image used, CPU load, and hardware specifications.

To implement the time machine function that enables the user to go back in time and inspect previous network states, the polling function is chained onto the time machine function, meaning that every gNMI query round is timestamped and saved to a double-ended queue in Python. This way, the oldest value will be deleted from the queue when the number of entries hits the previously set limit. The time machine function returns the last value of the queue, which is then directly transmitted via WebSocket to the frontend after polling completion. All available timestamps are transmitted to the frontend via another WebSocket endpoint. The user can pause the real-time data stream and select one of the stored timestamps. The selected timestamp is then sent to the backend. Instead of continuing the polling from the gNMI interfaces, the time machine function switches to streaming the data of the selected timestamp as its return value. This allows the chosen historical data of the topology to be transmitted to the frontend, enabling visualization of the corresponding historical states. The queue implementation, effectively storing up to 120 historical states, serves as a proof-of-concept but lacks scalability in terms of CPU load compared to pipeline-driven approaches such as Apache Kafka with time-series databases. Pipeline-driven solutions offer significant advantages in terms of storage capacity, persistent data retention, and parallelization, mitigating timing dependencies and synchronization bottlenecks.

B. Frontend

The frontend is developed in TypeScript using the React framework, enabling low-latency UI updates and extensibility, a key requirement for DigSiViz's real-time visualization[13]. For the force graph visualization of the network topology, the D3 and the react-force-graph library are used, while Tailwind CSS V3 delivers a streamlined styling approach, and Socket.IO client provides the WebSocket implementation. The interconnection between frontend components and backend endpoints can be seen in Figure 2.

The main component on which the entire visualization relies is *TopologyVisualization*. It binds to the topology endpoint of the backend to gather information about the network topology to be observed before receiving router data through the

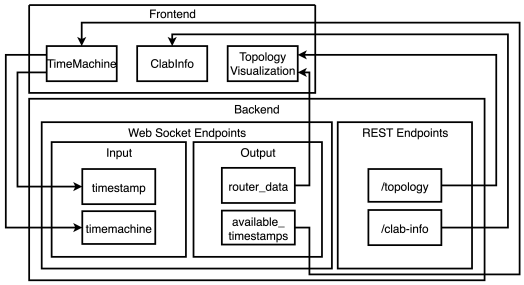

Fig. 2. Implemented interconnection between frontend and backend.

WebSocket endpoint. With this pre-defined structure, the react-force-graph library can directly create a ForceGraph2D object of this topology. ForceGraph2D is based on the D3 library, which means that the forces between nodes can be modified by defining custom D3 forces. By default, the forces between the nodes are too weak, leading to disorder, overlapping connections, and entanglements. To fix this, custom forces [14] are set with higher values based on the given topology, as depicted in Table I. In addition, each node is assigned a network symbol determined by its kind to increase usability. Labels and descriptions are provided dynamically. The graph links are provided with canvas objects that directly name the interconnection based on the source and target interfaces. Furthermore, the name and container image of each node are displayed directly. When hovering over node icons, node labels filled with the nodes' static information are displayed.

TABLE I
D3 CUSTOM FORCES FOR FORCEGRAPH2D.

Force	Configuration	Effect
charge	forceManyBody().strength(-300)	Strong repulsion between nodes
link	forceLink().distance(90)	Force distance to approx. 90 px
collision	forceCollide(30)	30 px safety radius around nodes

By clicking on a router node, only data from this specific node is displayed in that information box. When clicking on a link between two nodes, the displayed data shows information from both connected interfaces. The data can also be filtered. By default, all available information is shown. It is also possible to view only statistics, data rates, or transceiver information. As the *TopologyVisualization* component visualizes router data emitted from the backend, it is also able to visualize historical data when enabling the time machine function, as both share the same JSON format. The *TimeMachine* component receives all available timestamps from the backend. In *time machine* mode, the component presents a drop-down menu with a list of available sorted timestamps and buttons to perform a step forward or backward. The component then transmits the selected timestamp to the backend via the designated WebSocket, as shown in Figure 3.

V. EXPERIMENTAL SETUP

To discuss the benefits of the presented approach, an example Containerlab topology is deployed for evaluation. The example contains a triangular topology in which three

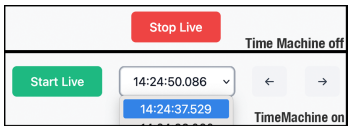

Fig. 3. Time machine UI element states.

routers running Nokia SR Linux are interconnected with each other. A host running Alpine Linux is connected to each router. As an example, iPerf3[15] is run between two hosts. Although the absolute measured values do not matter for visualization evaluation, the reaction of the network topology can be visualized and saved for inspection using the DigSiViz application. To determine valid functionality, iPerf3 is executed both in TCP and UDP mode with host h1 as the client and h2 acting as the server. In Figure 4 the results of a TCP

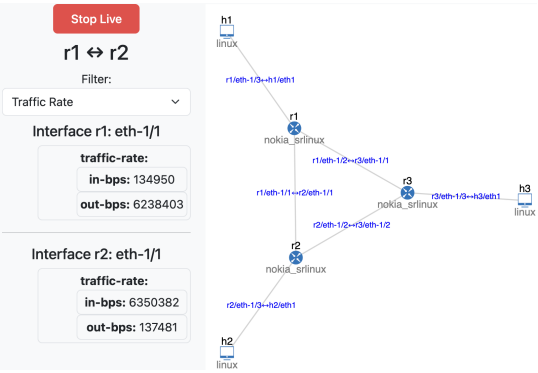

Fig. 4. Interface statistics for iPerf3 test example using TCP.

iPerf3 test are shown. The distinguishable characteristic of TCP traffic is clearly visible in the visualization application, as the acknowledgments from the receiving host are observable as *in-bps* on *r2:eth-1/1* and *out-bps* on *r1:eth-1/1*.

VI. NDT-based Application Traffic Engineering

Replicating application traffic from real networks to NDT is challenging, as in addition to packets and flows transferred, the host state needs to be consistent esp. for connection-oriented protocols[7]. In emulated virtual network environments, for example, iPerf3, as shown in the previous section, can be used to mimic traffic patterns from the real network. However, virtualized real-world NOSes like Nokia SR Linux focus on offering configuration features instead of high throughput or accurate latency. Also, high link speeds, for example, above 10 Gbit/s hold significant CPU load for virtual devices. To be able to evaluate traffic engineering scenarios in NDT, while still using virtual NOSes with realistic functionality, the prototype uses traffic shaping to proportionally reduce link speeds. This way, e.g., real network 10 and 1 Gbit/s links can be represented as 10 and 1 Mbit/s links in the virtual NDT network, avoiding transmission rate capping and burstiness stemming from software-based traffic emulation and corresponding shaping. However, fluctuations related to the applied traffic shaper and CPU load in the virtualized environment itself must be considered. DigSiViz allows visualizing these effects, enabling effective testing of application-based traffic engineering scenarios. As DigSiNet allows running multiple NDTs (siblings) of the same real network topology, individual tools and link speed reductions can be used to have, e.g., one NDT that is on par with configuration features of the real network devices and another one that allows realistic traffic engineering tests.

We used the NDT to evaluate the use of multiple paths to distribute network traffic of an application with policy-based routing (PBR). Similar evaluations can be performed using Segment Routing or intents in multipath-aware applications and transport services[9], enabling ”intent-based routing”. The approach can be combined with packet injection or flow emulation in the NDT or using traffic generators instead of iPerf3 if more complex traffic characteristics are required. In our use case we were able to evaluate the overall utilization of the network for the application compared to using the standard shortest path route and transfer the PBR setup from the NDT to the real network. Figure 5 shows a screenshot of the real-time visualization of the link load collected using gNMI for all interfaces on FRR routers running in Containerlab while running iPerf3 TCP from leftmost *h3-1* to rightmost *h1-1*. The achieved total bitrate is 7 Mbit/s resulting from 1 Mbit/s max. on the shortest path via r3 and r1 (2 hops), a second path with 2 Mbit/s max. from r3 over r2 to r1 (3 hops) and an additional path with 4 Mbit/s max. from r3 over r5, r4 and r1 (4 hops). The shortest path uses plain BGP, while the additional paths for the multipath setup are configured using PBR.

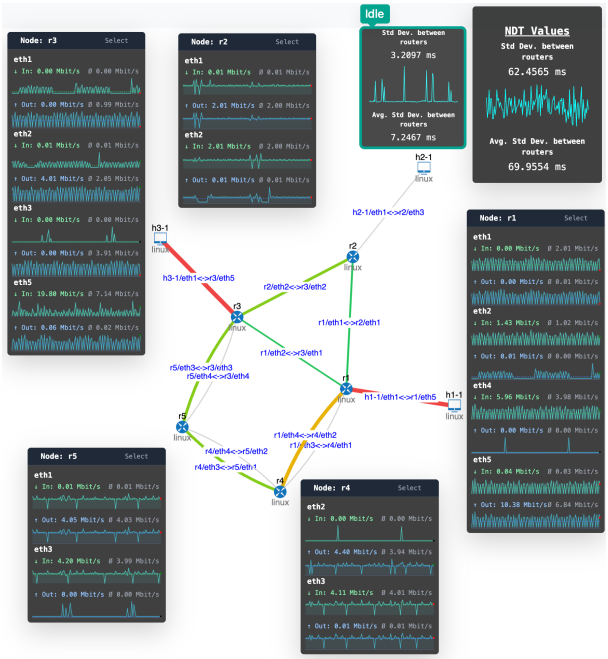

Fig. 5. Visualizing and evaluating multipath PBR traffic in DigSiViz.

Using this setup, students can evaluate traffic steering and transfer the BGP and PBR configuration back to the real network. By dynamically changing link colors and sizes, network parameters such as link utilization can be easily determined. Other metrics could also be considered for the color-coding, such as error rates, depending on the specific use case. As

stated above, traffic shaping a 1 Gbit/s link to 1 Mbit/s ensures that traffic emulation in the NDT is not capped due to limited CPU and virtualization resources. However, as can also be seen from the zigzag in the throughput of the interfaces of r1 and r3 (around the indicated trend line in the center), traffic shaping adds fluctuations and ramp-up phases due to the nature of tc-based Linux kernel traffic shaping policing outgoing interface traffic. To evaluate how realistic characteristics of the adapted reduced traffic rate remain, we compared the iPerf3 throughput in the NDT to its counterpart in the real network. The average result over 5 repeated throughput measurements is shown in Figure 6. As can be seen in the figure, initial throughput is higher due to tc buffers in the start phase; moreover, the shaped NDT traffic shows more fluctuations, as expected. However, these effects can be mitigated while transferring the results to the real network, as the bitrate in the virtual environment is close to that of the real network and start phase can be ignored. Furthermore, tc and PBR buffers can be tuned for additional realism. The tc, FRR and PBR testbed is available as open source[2]. Given the real-time visualization, we also evaluated the fidelity of the NDT and how close the telemetry values read from each router are in relation to the real network. The result is shown in Figure 5 as *Std. Dev. between routers*. As expected, the polling timestamps diverge under load. During the PBR experiment shown in the figure, an average of 71 ms was measured as the std. dev. between the timestamps on gNMI values from the routers. With no traffic and therefore no load on the routers, the timestamp std. dev. for collecting data from the routers dropped to an average of 7 ms, as shown in the box with the label *Idle*.

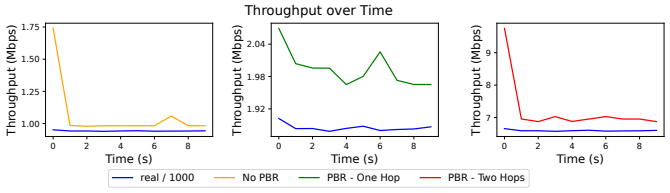

Fig. 6. Accuracy of NDT multipath iPerf3 traffic adaption.

Similar evaluations can be performed when detecting overloaded paths or imbalances in multipath setups during the runtime of real networks. In this way, NDT can be used to analyze and improve the distribution of traffic. Using DigSiViz, network operators can visually analyze and resolve network load issues. Although automatic remediation in a closed-loop setup between the real network and NDT would be possible, it holds the risk of load oscillation or deterioration. In contrast, DigSiViz allows operators to experiment with different traffic steering and verify the configuration before deducing improvements for the real network in the form of an assisted open-loop scenario.

VII. Discussion

Based on the functionality presented by the DigSiViz application, several improvements offered by real-time visualization and scenarios that benefit from them can be identified.

[2]https://github.com/netlab-hfd/ibr-testbed

A. Improvements and Scalability of Real-Time Visualization

The main goal of this paper was to implement a visualization approach for DigSiNet. The real-time visualization offers a dynamic representation of the instantiated virtual network topology. Especially patterns or specific data structures are easier to observe using visualizations than by reading plain text or tables[16]. By rendering the network topology using a force graph, this representation can improve understanding of the topology by revealing structures under the influence of configured forces that would be difficult to recognize in a standard layout [17]. Another improvement is the time machine functionality. As the user cannot inspect an entire topology management dataset within a 0.5-second update interval, the time machine offers an efficient way to analyze previous network states, allowing for a fine-grained step-by-step analysis of every network device's state in a specific timeframe. This avoids blind spots and increases user satisfaction by reducing the total amount of simultaneously displayed information as the monitoring data are separated into subunits that are assigned to the network devices and links. As the filter state is persistently stored in the frontend, the user can browse through filtered data from multiple devices without losing focus in the large amount of data. In contrast to related work, DigSiViz utilizes gNMI instead of SNMP. SNMP has been a standard since 1990. It is known to generate large overhead in retrieving multiple objects[18]. That makes SNMP unsuitable for high-frequency polling or scaling up, especially in a virtualized network that is not capable of providing the bandwidth and performance of a real network. gNMI is a more modern approach that utilizes a structured YANG-based data model. It supports on-demand data retrieval and streaming capabilities, allowing continuous data collection with reduced overhead. It utilizes gRPC that benefits from modern techniques such as protocol buffer serialization and binary data format compression[19]. One of gNMI's advantages is increased scalability achieved through more efficient and timely data transfer for the model updates in the NDT. Although the experimental setup described in Section V only shows two small example topologies, larger setups are also supported. However, larger topologies not only increase CPU, RAM, and I/O requirements, but the amount of data to be fetched also significantly impacts scalability. If all gNMI data are pulled, more latency is observed, as is also the case for a larger node count in the topology. However, DigSiNet and DigSiViz allow to retrieve only a subset of data needed for the use case of the NDT. Furthermore, the leveraged Containerlab can be scaled out to multiple Kubernetes nodes using its clabernetes subproject. Also, DigSiNet and potentially also DigSiViz could be scaled to run on multiple nodes, each collecting and processing a fraction of the nodes in the topology as data collection is based on the scalable event streaming platform Apache Kafka.

B. Scenarios Benefiting from Real-Time Visualization

Some commercial vendors already utilize NDTs to visualize and monitor throughput and bandwidth by enriching the NDT

with real-time data[2]. By mapping the actual real-world network to an NDT, the NDT can be used, e.g., for traffic engineering. For example, it can help to analyze under- or over-utilized network paths or identify the root cause of traffic anomalies occurring in the network. The visualization helps network administrators to quickly identify relevant elements in the topology and to get a quick overview of the network health using recognizable colors that represent the node and link states in the topology graph. DigSiNet creates different virtual siblings of the real network environment that can be inspected to gain knowledge about individual topology characteristics [7]. Here, real-time visualization helps to analyze the characteristics. On the one hand, it helps to structure the monitored data by assigning it to the topology layout for better visibility. On the other hand, it enables the user to inspect the reaction of the topology under test step-by-step by going through the saved states in the time machine storage. This allows for application and service performance measurements, e.g., using iPerf3 or similar traffic generators or real applications in different NDT configurations while monitoring overall network load and state. By comparing the results, the user can determine the effect of the changes applied to optimize the network topology and configuration. The same applies to problem solving, as the user is able to quickly identify possible problems by inspecting the visualization, e.g., comparing time machine data. In addition, stability tests performed in DigSiNet, e.g., disabling links in the topology, can be analyzed, as the visualization helps to find bottlenecks in the topology. Furthermore, it can be used in learning environments to allow students to experiment with different network and traffic management techniques.

VIII. Conclusion and Future Work

To build a real-time visualization for the DigSiNet NDT environment, the presented DigSiViz visualization application utilizes gNMI to retrieve monitoring data from the NDT topology. Monitoring data is polled at a 0.5-second frequency, processed, and sent to a frontend visualization application. The frontend visualizes the processed monitoring data in a force-directed graph layout and displays real-time data for the selected node or link. A time machine functionality stores historical states of the network monitoring data to enable users to draw deeper conclusions from the behavior of the NDT. In the context of NDTs, real-time visualization can improve their applicability by providing a GUI that structures monitoring data and reduces the complexity of otherwise purely textual data by assigning data to associated visual elements of the displayed topology graph. This paper presents and discusses multiple scenarios for NDTs that can benefit from such real-time visualizations. Also, the paper evaluates the accuracy and fidelity of virtual twin networks and links compared to their real counterparts. Using colors and dynamic layouts, the human eye can recognize patterns and deviations faster, helping to quickly find problems, optimize network performance, or determine the efficiency of different protocols.

As future work, we consider replacing the queue storing timestamped monitoring data with a time series database such as InfluxDB. This could also help reduce data usage by implementing compression of older historical timeframes and deduplication algorithms to eliminate redundant data. Furthermore, flow monitoring data such as sFlow could be used to gain deeper insights into application traffic flows in the NDT, augmenting the already acquired gNMI data. Finally, automatic tagging of interesting states in the NDT can be implemented. A solution for automatic event-based tagging of stored monitoring data using thresholds or other triggers would greatly improve the applicability of NDTs for traffic engineering, as interesting events are automatically marked to notify the user. This could also include pattern recognition or machine learning.

References

[1] "Nokia Network Digital Twin | DAC," https://www.dac.nokia.com/nokia-network-digital-twin/, accessed: 2025-08-19.

[2] "Optimized network planning: o2 Telefónica network gets a twin," https://www.telefonica.de/news/press-releases-telefonica-germany/2024/05/optimized-network-planning-o2-telefonicas-network-gets-a-digital-twin.html, accessed: 2025-08-19.

[3] M. Attaran and B. G. Celik, "Digital Twin: Benefits, use cases, challenges, and opportunities," *Decision Analytics Journal*, vol. 6, Mar. 2023.

[4] P. Almasan *et al.*, "Network Digital Twin: Context, Enabling Technologies, and Opportunities," *IEEE Communications Magazine*, vol. 60, no. 11, Nov. 2022, conference Name: IEEE Communications Magazine.

[5] C. Zhou *et al.*, "Network Digital Twin: Concepts and Reference Architecture," Internet Engineering Task Force, Internet Draft draft-irtf-nmrg-network-digital-twin-arch-10, Feb. 2025.

[6] S. Rieger, L.-N. Lux, J. Schmitt, and M. Stiemerling, "DigSiNet: Using Multiple Digital Twins to Provide Rhythmic Network Consistency," in *IEEE Network Operations and Management Symposium (NOMS)*, 2024.

[7] S. Rieger, L.-N. Lux, S. Schickentanz, D. Hermann, T. Mott, and M. Freund, "Challenges of Event-based Streaming and Queuing as Data Exchange for Network Digital Twins," in *20th International Conference on Network and Service Management (CNSM)*, 2024.

[8] E. Safrianti, L. O. Sari, and N. A. Sari, "Real-Time Network Device Monitoring System with Simple Network Management Protocol (SNMP) Model," in *3rd International Conference on Research and Academic Community Services (ICRACOS)*, 2021.

[9] B. Trammell *et al.*, "RFC 9622: An Abstract Application Programming Interface (API) for Transport Services," 2025.

[10] R. Shakir, A. Shaikh, P. Borman, M. Hines, C. Lebsack, and C. Morrow, "gRPC Network Management Interface," https://datatracker.ietf.org/meeting/98/materials/slides-98-rtgwg-gnmi-intro-draft-openconfig-rtgwg-gnmi-spec-00, accessed: 2025-08-19.

[11] A. Karneliuk, "pygnmi: Python gnmi client," https://github.com/akarneliuk/pygnmi, 2025, accessed: 2025-08-19.

[12] R. Dodin, "containerlab - Topology definition," https://containerlab.dev/manual/topo-def-file/, accessed: 2025-08-19.

[13] E. Chopra, "Creating live dashboards for data visualization: Flask vs. React," *Internat. Journal of Engineering Research*, vol. 8, no. 9, 2021.

[14] "d3-force," https://d3js.org/d3-force, accessed: 2025-08-19.

[15] "iPerf doc," https://iperf.fr/iperf-doc.php, accessed: 2025-03-11.

[16] Z. Ruan, Y. Miao, L. Pan, Y. Xiang, and J. Zhang, "Big network traffic data visualization," *Multimedia Tools and Applications*, vol. 77, no. 9, 2018.

[17] L. Braun, M. Volke, J. Schlamp, A. von Bodisco, and G. Carle, "Flow-inspector: a framework for visualizing network flow data using current web technologies," *Computing*, vol. 96, no. 1, 2014.

[18] A. Pras, T. Drevers, R. van de Meent, and D. Quartel, "Comparing the performance of SNMP and Web services-based management," *IEEE Transactions on Network and Service Management*, vol. 1, no. 2, 2004.

[19] F. Paolucci, A. Sgambelluri, F. Cugini, and P. Castoldi, "Network Telemetry Streaming Services in SDN-Based Disaggregated Optical Networks," *Journal of Lightwave Technology*, vol. 36, no. 15, 2018.