Challenges of Event-based Streaming and Queuing as Data Exchange for Network Digital Twins

Sebastian Rieger, Leon-Niklas Lux, Sven Schickentanz, David Hermann, Thomas Mott, Moritz Freund
Applied Computer Science Department
University of Applied Sciences Fulda, Germany
{sebastian.rieger, leon-niklas.lux, sven.schickentanz1, david.hermann, thomas.mott, moritz.freund}@cs.hs-fulda.de

***Abstract*—Network digital twins offer great potential for network management and monitoring, as they allow to use virtual representations of a real network, e.g., for risk-less tests and what-if analysis regarding fault, configuration, performance and security management. The digital twins take existing network emulation and simulation that is already established for these tasks a step further by allowing contiguous bidirectional state synchronization between the twins and the real network, ideally close to real-time. However, particularly in large networks, this holds significant challenges regarding the volume and velocity of the data to be exchanged between the twins and especially the real network. This paper presents an extension to the DigSiNet network digital twin environment to support message queues and event streaming platforms as scalable near real-time data exchange for the twins. Challenges for different common network management and monitoring data types are identified in an experimental setup and discussed regarding their application in use cases for network digital twins and the DigSiNet prototype. The implementation is provided as an open-source repository.**

***Index Terms*—Network Management, Network Automation, Network Digital Twin, Event Streaming, Message Queues**

I. Introduction

Network Digital Twins (NDT) have been suggested to improve several areas of network and related service management scenarios. Although these concepts can take advantage of existing and well-established network emulation and simulation platforms to implement twins [1], the core requirement of NDTs is to be able to bidirectionally and quickly (i.e., even in near real time) synchronize the state of twins and the connected real network. Computer networking is considered special in this regard, as a lot of data is available and an unsolved question is how these data can be used effectively in NDTs. Especially, this poses challenges regarding the volume and velocity of data to be continuously transferred between the models. The challenges are further amplified by the fact that to replicate the live state of the network, different layers have to be considered. While the replication of configuration state and related changes is feasible and leads to relatively low data volume, synchronizing monitoring or even traffic data between the real network and its twins is challenging at best or close to impossible. This stems not only from the volume and velocity of the data, but also from the fact that, for example, stateful connections would only be replicable by snapshotting the state of the entire network and connected hosts' states (i.e., TCP states) that would have already changed in reality while being synchronized to the twin. Even more complex in volume regarding the synchronization of the twins' states is the redundancy of data being sent across the network. Identical monitoring data, traffic flows and packets traverse multiple links and nodes within the network that would need to be replicated and sent to the twins.

To tackle these problems, we developed an extension of our previously presented prototype [1] that is able to run multiple NDT as so called siblings in individual virtual network environments (VNE). Compared to these VNE and network emulators or simulators, the prototype allows to run multiple network topologies supporting a continuous bidirectional exchange of network management and monitoring data. The proposed enhancements are twofold. First, an adaptive selection and synchronization of only a small set of relevant or desired state data is developed and presented. Second, scalable event-based streaming and queuing solutions are evaluated and adapted for use with NDT state synchronization, including the discussion of potential deduplication and correlation of transferred data. The developed extensions are discussed and evaluated by pulling network management and monitoring data of different granularity from the real network and its siblings, effectively synchronizing parts of the state and characteristics of the network to be able to use them for experiments, e.g., in fault, configuration, administration, performance and security management in the twin networks. As state-of-the-art solutions for event-based streaming and queuing, Apache Kafka and RabbitMQ are used in our prototype. By using these standards, external connections to and from the sibling networks and its contained network elements are enabled. The implementation is provided as open-source in a public GitHub repository[1].

II. Related Work

Digital Twins (DT) have proven to be a valuable tool, enabling abstraction and optimization of complex real world systems [2] and efforts are made to apply the concept of DT to network management [3] [4] [5]. However, cloning a real network into a VNE is hardly feasible for networks at scale [1]. To address this issue, the DigSiNet architecture [1] combines multiple DTs that emulate subsets of the properties of the real network as so-called siblings, leveraging the individual benefits [6] of different VNE types and platforms. A key challenge for DT and hence DigSiNet is data acquisition and processing

[1]https://github.com/srieger1/digsinet

978-3-903176-66-9

[3]. As a network grows, the amount of accumulating data grows too, thus increasing the performance demands of the DT [7] [8]. Using RabbitMQ and Apache Kafka to tackle this challenge is proposed in [9] for optical network telemetry and 5G network digital twins in [10]. Compared to this related work, this paper holds the following contributions:

- Practical testbed with a real-world prototype that can be used to evaluate the practicability of NDT data exchange volume and velocity.
- Assessment of using the prototype for multiple NDT-based network management and monitoring use cases.
- Modular and extensible open-source repository [1] to be used for NDT experiments.

III. DigSiNet Network Digital Twin Environment

Figure 1 shows the DigSiNet architecture that was initially presented in [1] and extended for this paper.

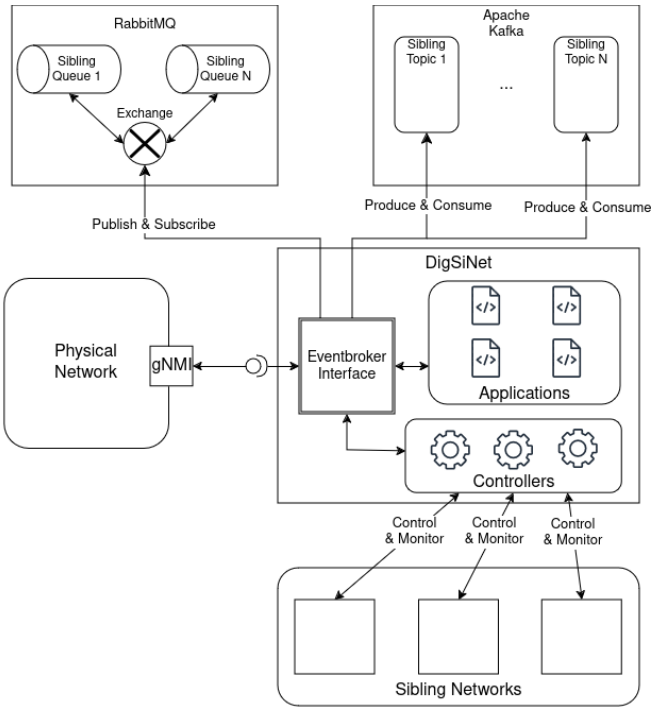

Fig. 1. Event-based Data Exchange for Network Digital Twins in DigSiNet

DigSiNet's architecture consists of a *Real Network* (e.g., physical or virtual reference network) and multiple *Siblings* that mimic, emulate or simulate (depending on the used VNE) parts and individual characteristics of the real network in the form of twins. Further components of the DigSiNet architecture are *Controllers*, that manage and monitor the siblings and the state of the real network. Tasks and functionality of controllers can be influenced by user-defined *Applications* that the controllers run, similar to northbound applications in SDN controllers. For example, an application can react on a certain change or event in the real network or a sibling and modify the state or configuration of other siblings or the real network. Changes and events are detected by *Interfaces* as another component of DigSiNet. Currently gNMI[2] is supported as an interface to detect state and configuration changes in the real network and/or its siblings. However, DigSiNet offers a modular architecture, meaning that custom interfaces, controllers, applications etc. can be implemented besides the ones provided by default (e.g. SNMP, CLI interfaces etc.).

The final additional component in the DigSiNet architecture are *Queues*. While initially Python queues were used in the prototype for the data exchange between Python multiprocessing processes, the extensions presented in this paper implement a modular approach, utilizing RabbitMQ or Apache Kafka to enable event-based streaming and queuing. This way, a core requirement for NDT regarding efficient and scalable data acquisition and processing [3] is addressed. Also, by leveraging state-of-the-art message queues and event streaming platforms, DigSiNet NDTs can now access external resources or vice-versa be controlled by them. For example, sibling configuration or state can now be reported to external network management platforms as well as accessed by them. Furthermore, especially state-of-the-art event streaming platforms like Apache Kafka, do not only support collecting current but also persist and process historical data. Therefore, previous state and configuration data of the network can be used to leverage the siblings for fault or performance analysis or what-if experiments as well as configuration drift assessment.

Keeping the siblings and the real network in sync is challenging if not only low volume and velocity configuration data is to be synchronized, but rather also monitoring, state and traffic data. Ensuring timely and proper state transfer between the siblings and the real network is essential regarding the twins' fidelity and accuracy of the desired use case and network model. This includes being able to change the simulation or emulation in the sibling and restore an old state of the network or inject traffic patterns, e.g., to detect and forecast anomalies or simulate effects of potential changes in what-if or fault analysis scenarios. As a result, bulk updates of the siblings' network models have to be supported to simulate a state of the real network or reconcile drift between them. To avoid drift, continuous feedback loops can be established to reconcile the siblings' and the real network. By extending DigSiNet to support message queues and event streaming, it can be used as a testbed to assess the applicability of NDT for the synchronization of network management and monitoring as well as traffic data. A corresponding testbed and its functional components are described in the following subsections.

A. Network Modeling and Interfaces

Applying the concept of DTs to the networking domain poses unique challenges [11]. While the degree of digitalization that is present on network components allows for rather easy acquisition of real-time data necessary for implementing DTs, the amount of data that large-scale networks produce is hard to manage [3]. DigSiNet attempts to provide a solution to

[2]https://github.com/openconfig/reference/blob/master/rpc/gnmi/gnmi-specification.md

this problem following a divide-and-conquer approach: instead of trying to emulate the network as a whole, it attempts to emulate siblings that focus on a distinct characteristic as a part of the real network that is relevant to a corresponding use case. As an example, a sibling that is designed to perform security related experiments could aim to replicate configuration of the devices that form the network as accurately as possible, without trying to emulate traffic-related characteristics [1]. Multiple Siblings can be combined to gain insights that expand on the simulated aspects of individual siblings. Consequently, each sibling in DigSiNet can use an individual network topology (e.g., only a minimal part of the network that is essential for the sibling's use case). DigSiNet currently uses the network model used by containerlab and is able to add or remove nodes and links for each sibling on the fly, as well as fold parts of the network to abstract and handle complex network architecture while focusing on available sibling resources and the intended use case and experiments in the sibling.

B. Data Collection and Exchange

Existing, standardized interfaces for managing and monitoring network devices, such as gNMI, allow the collection of real-time data from the physical network [3]. DigSiNet aims to be interface-agnostic, providing an abstract interface that can be used to implement different management interfaces besides the included reference implementation for gNMI [1]. Data can be acquired from network hardware by either polling (pull) it in a defined interval or by subscribing (push) to specified gNMI paths, automatically receiving updates as changes occur. As it is possible that data arrives quicker than DigSiNet can process it, an event broker extension was implemented, to allow scalable queuing of events.

C. Event-based Streaming and Message Queuing

Two reference implementations have been developed for the event broker. First, RabbitMQ was implemented as a state-of-the-art scalable, flexible and reliable distributed message queuing solution. Second, an Apache Kafka connector was implemented, with Kafka being a de facto standard in scalable distributed event streaming platforms, including data processing and persistence. Both solutions allow decoupling and scale-out of DigSiNet, especially for the siblings. Also, as they are prevalent standard tools, siblings can now connect to external network management and monitoring services and vice versa. By using two different implementations (RabbitMQ and Apache Kafka), more external network management solutions can be connected to DigSiNet. Also, RabbitMQ is better suited for task-oriented sibling interconnections while Apache Kafka offers better support for persistence and data processing. For Apache Kafka, network telemetry streaming standards are also being developed in the IETF, i.e., YANG Push.

D. Experimental Setup

To test the applicability of the proposed event broker for NDT and evaluate possible use cases, a Ubuntu 22.04 VM (10GB RAM, 4 vCPU) running DigSiNet and containerlab[3] was set up. The real network is running as a virtual network using a minimalistic topology with 2 switches connected directly to each other in containerlab. The switches run Arista containerized Extensible Operating System (cEOS) 4.30.1F. Containerlab also supports importing real network topologies[4].

Three siblings are automatically created as twin topologies in containerlab by DigSiNet, as shown in 1. These siblings run adapted network topologies[5], with the first sibling (named *continuous_integration*) replacing a switch and link in the topology and the second sibling (named: *security*) removing one switch and replacing it with a Linux container while the third sibling (named: *traffic_engineering*) is not automatically started for the experiments, as previously presented in [1]. While the real network intentionally uses a simplistic network topology, this still leads to 5 running cEOS containers (2 in the real network, 2 in *continuous_integration* and 1 in the *security* sibling) requiring approx. 5 GB of RAM in total. Additionally, a single-node Kafka and Zookeeper container setup is running in the VM, accounting for about 2 GB of RAM. DigSiNet runs as a Python process that uses the proposed event broker extensions to connect to Kafka with topics for the 3 siblings and the real network. Additionally, 3 controllers and 5 gNMI interface subscription handler processes for each cEOS container are running. Remaining CPU and memory resources are enough to prevent capping and swapping during experiment runs. Besides CPU and memory resources, latency of change notifications received using gNMI *subscribe* on paths of the cEOS nodes are sampled, measured and tracked during experiment runs.

While the topology is intentionally simplistic and Kafka as well as DigSiNet are using a single node, measurements were reproducible without significant deviations. Consequently, assumptions regarding requirements and challenges for the data exchange between the twins and the real network are also applicable to larger topologies by scaling out the containerlab, Kafka and DigSiNet environment. Based on the number of events and transferred data described in the next section, an extrapolation to larger and common real-world networks is possible. Also, especially based on the observed changes, the load for bidirectional state transfer can be estimated.

IV. Evaluation

Figure 2 and 3 show the results from the experiments as gNMI event rate and cumulative throughput in a logarithmic scale using Grafana visualizations. The legend shows five cEOS (2 in the real network, 2 in the continuous integration and 1 in the security sibling, as presented previously in [1]) nodes that were continuously monitored for changes. All presented experiments were run 10 times without significant deviations compared to the measurements shown in the figures. Samples are collected over 5 minutes, followed by a 2 minute cool-down, forming each sample block in the figure. The first sample block around timestamp 8:51 in the diagram uses

[3]https://containerlab.dev

[4]https://github.com/jbemmel/netsandbox

[5]https://github.com/srieger1/digsinet/blob/60e5b373/digsinet.yml#L33-L89

gNMI *get* to pull all configuration values (gNMI path: /, data type: config) from the 5 running cEOS nodes, sends the gNMI notifications to three Apache Kafka topics (*realnet*, *continuous_integration* and *security*) and reports the measurements shown in the figure to metrics in Prometheus as a time series database and monitoring system. gNMI *get* was used to pull the samples from the nodes every second. Consequently, the first four sample blocks that used gNMI *get* show 5 events per second (one poll for each node) in Figure 2.

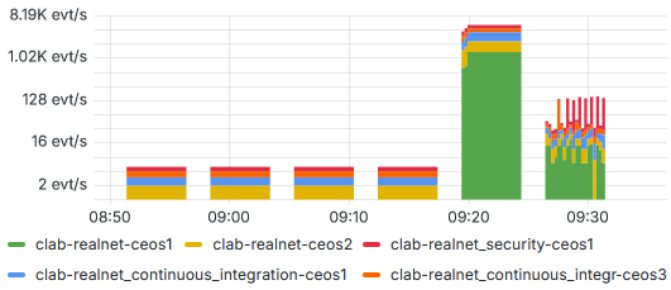

Fig. 2. Number of events measured using gNMI get versus subscribe.

In the second run starting at 8:58, again gNMI *get* was used to pull the information, but in this case only state/monitoring data was polled (path: /, type: state). Third run starting around 9:05 pulled the operational state (interface status, etc.) from the nodes (path: /, type: operational). The fourth run, starting shortly before 9:12, pulled all data types from the nodes (path: /, type: all). For the fifth block at 9:19, gNMI *subscribe* was used with a subscription to push all information from a node each second (path: /, subscription mode: sample, delay: 1s).

As can be observed in Figure 2, the notifications for the subscription contain separate updates for each subscribed information. The last, sixth block starting 9:26 used gNMI *subscribe* to only receive values that have changed (path: /, subscription mode: on_change). Consequently, the number of events as well as their size shown in Figure 3 are significantly lower and fluctuate as only changes for dynamic values (i.e., primarily state and operational values as config values are typically constant during runtime) are received. This also relates to the load on the Kafka topics and their increase in size as the throughput defines the ingestion rate of events.

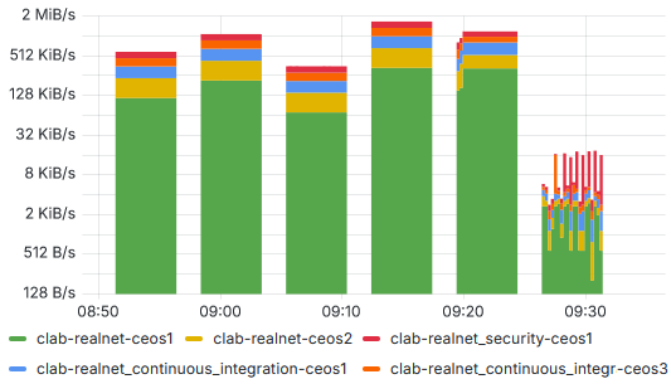

Fig. 3. Size of collected gNMI notifications from real network and siblings.

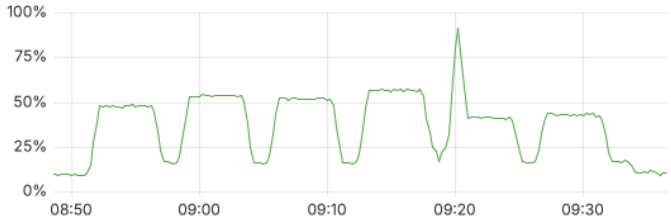

Fig. 4. CPU load during the data transfer between real network and siblings.

The CPU load of the host running the virtual networks, Kafka and Prometheus, is shown in Figure 4. Lastly, Figure 5 shows the latency of gNMI changes between the real network and the siblings. To measure the latency, a change in an interface description containing a timestamp is made on *clab-realnet-ceos1* using gNMI *set*. Another process waits for incoming changes in the description using gNMI *subscribe* (path: interfaces/interface[name=Ethernet1]/config/description, mode: on_change). As soon as the change with the timestamp comes in, the process measures the delay and sends it to Prometheus.

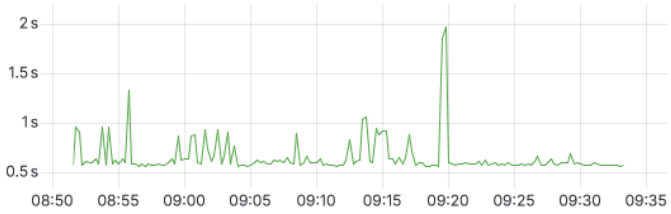

Fig. 5. Latency of gNMI operations during data transfer to and from siblings.

As expected, the samples using gNMI *subscribe* show a significantly higher event rate compared to pulling the information using gNMI *get*. Also, the size of subscribed events in bytes is only a small fraction (around 1/64) of the amount of bytes to be transferred and stored in the topics when polling the same gNMI path with gNMI *get*. This does not only compress the data to be stored in the topics, but also increases the performance by filtering redundant data when setting the sibling network to a historical state consumed from Kafka.

However, subscription only reports state delta, so a full view of the network's state requires also gathering unchanged static data for config, state and operational values. Since the CPU load is less when using *subscribe*-based push instead of pulling, a solution for this is a low-frequency (e.g., every 5 minutes) sample-based subscription, as shown in the fifth sample block around 9:19 in Figure 2, together with a more frequent on_change subscription. This way, the highest CPU load (and gNMI "ping" latency) will only occur during the initial setup of the full subscription and not for subsequent samples coming in every 5 minutes. Lower CPU load would especially be valuable for real-world physical network devices compared to the experimental virtual setup used in this paper, as transferring data to the twins increases the load on control plane resources, especially CPU.

As can be seen in the figure for the first three sample blocks

using polling, exporting mostly static config data has the lowest CPU load, operational data causes a slightly higher one and state data accounts for the highest CPU load. Depending on the use case of the sibling, also only full state and/or operational data can be subscribed to. Operational data (e.g., port status) has the smallest size and throughput, followed by config (settings on the virtual switch), with state data (packet, error, byte counter of interfaces, etc.) accounting for the biggest size, as shown in the figure.

For the fidelity of the twins, besides these challenges regarding the granularity of exported data with respect to its volume and velocity, also the latency for changes in the real network to be detected and propagated to the siblings is relevant, especially if near real-time operation is desired, e.g., to detect fault, performance or security issues and evaluate as well as mitigate them using the siblings. Baseline for this latency was approx. 0.6s in the experimental testbed, when no other gNMI operations were performed. When pulling gNMI data, the ping went up to 1.3s. While pulling state instead of config data, spikes to 1.0s occurred significantly more frequently. Pulling only operational data from the nodes kept the latency close to its baseline around 0.7s, consequently allowing high-frequency operational data updates (e.g., more frequent updates of overall link status compared to individual fine-grained counters). During the initial setup of the subscription to all available gNMI data from the nodes in the beginning of the fifth sample block, the latency briefly increased to a maximum of 1.96s in the experiments.

Experimental results were used for an assessment of the challenges of NDT data exchange. Different data types, as already stated in this section for config, state and operational data, have to be differentiated regarding their volume and velocity. This is not only important to evaluate the feasibility of transmitting them near real-time between the real network and the sibling networks, but also when setting the twins to a historical network state based on these data, e.g., to perform fault, configuration, performance or security analysis. The same holds true when setting the sibling networks to an artificially generated state for test or benchmark scenarios (e.g., traffic engineering or prediction). As explained above, differential state data serves as a significant compression, also removing duplicate data. However, also static data is needed to present a complete representation of the network's state. In the experiments, this full state data accounted for approx. 2 MiB/s, as shown in Figure 3, though in our experiments only 5 nodes were exporting data and the network was nearly idle. This suggests that, as discussed in [1], only small portions of real-world networks' state can be used timely and accurately in siblings especially with limited hardware resources.

As experiments were run in a virtual testbed, traffic in the network was not measured and exported to the siblings, which is considered as a next step. However, as previously shown for real-world campus network traffic in [12], flow and packet level data is even more redundant and volumetric given the average size of flows and current network bandwidths.

V. Discussion

The experimental setup of DigSiNet using the event broker implementation presented in this paper was used to assess the performance and applicability of message queues and event-based streaming using gNMI in NDT scenarios. Based on the preliminary evaluation results, the DigSiNet environment can be used to evaluate the fidelity and scalability of the siblings, e.g., to measure the data exchange volume and velocity for different network management and monitoring use cases. As the testbed is using real-world network operating systems (NOS) in containerlab, this allows to assess the practical application and value of NDT and characteristics that can be transferred to physical networks and devices. Especially, for challenges inherently connected to NDT data exchange like latency of state changes and possibilities to reconcile diverging state of siblings and the real network. This divergence can be intentionally, e.g., while carrying out what-if or troubleshooting analysis in which the current or a historical state of the real network should be imported as a bulk update.

TABLE I
Data Type Assessment for practical NDT Data Exchange

Data Type	Volume	Velocity	Use Case
Configuration Data	low	very low	synchronization, replication, action
Monitoring Data	medium	medium	partial replication, action *(partially redundant)*
Aggregated Traffic	high	medium	traffic emulation & simulation, action *(redundant)*
Packet-level Traffic	very high	very high	partial traffic emulation & simulation *(infeasible for real-world traffic / only initial packets, highly redundant)*

During the implementation and the experiments, we identified and assessed 4 types of management and monitoring data to be considered in NDT scenario. Table I shows a classification of these data exchange types. Regarding the fidelity and applicability of the siblings compared to the real network, the challenges we identified again primarily relate to the volume and velocity of the exchanged data. The higher either volume, velocity or both, the higher the challenge to use this data type in the practical setup of NDT.

Based on the observed amount of data in our experiments and the resulting classification, we derived possible use cases shown in the last column in Table I. Synchronizing the data between the real network and its siblings using gNMI was only possible for *Configuration Data* as its volume and velocity (frequency of changes within the data) are low. *Configuration Data* of the cEOS nodes (e.g., *openconfig:interfaces/interface/config*, */subinterfaces* etc.) is collected using gNMI *get* or *subscribe* and synchronized using gNMI *set* with nodes in the sibling based on filters defined in DigSiNet's config file. As the OpenConfig specification currently does not support explicitly filtering config values in

subscriptions, though an extension was suggested, a configurable filter to achieve this was implemented in DigSiNet. For *Monitoring Data* that has larger volume and velocity, partial replication is possible. In DigSiNet, this is achieved using gNMI *subscribe* on paths holding state data (e.g., *openconfig:interfaces/interface/state*, port state, in / out bytes & packets, link utilization, error count, device CPU, memory utilization, etc.), and persisting it in the relevant sibling topics in Kafka. This also allows processing the data by consuming it from the topics instantly or streaming historical data for subsequent analysis. Besides replication, both aforementioned data types can be used to trigger actions in the siblings or the real network. The remaining two data types identified for NDT data exchange are related to traffic being sent over the links. *Aggregated Traffic* can be collected on the nodes, e.g., using NetFlow/IPFIX. This way, consecutive packets belonging to the same traffic context can be collected as flows, e.g., periodically (active flows) or after their termination (inactive flows). Even *Aggregated Traffic* already leads to a high traffic volume in real networks like our University's campus network as, e.g., evaluated in [12].

Depending on the active and inactive timeout for flow exports from the switches, the frequency and hence velocity of the data is also at least medium compared to the other data types. However, traffic flows can be used to create artificial packets and resulting transfers in the NDTs as traffic emulation or simulation. Another use case is again to use this data to perform an action in the siblings or, if the *Aggregated Traffic* data was received from tests in the sibling, also on the real network. Lastly, we observed and assessed *Packet-level Traffic* to be considered as data being exchanged between real network and siblings. However, this traffic has very high volume and velocity and is highly redundant. If high-speed transfers are carried out over multiple switches in the real network, to transfer them to the NDTs running in the siblings would require multiple times the bandwidth of the entire real network and as the packets are sent over the switches, the same data would be repeatedly transferred. Therefore, *Packet-level Traffic* can only be used for partial traffic emulation or simulation based on a small size of sampled packets (e.g., initial packets of a flow, in-band network telemetry data, attack detection traffic samples etc.).

Consequently, the main challenge we assessed regarding data exchange between twin networks themselves and especially between twins and the real network was posed by the desired granularity and redundancy of the data type based on the desired NDT use case. While state-of-the-art message queues and event streaming platforms are designed for high volume and velocity of data, the amount, especially of traffic data in real-world scenarios [12], is already too large for the export. Therefore, mechanisms have to be implemented as extensions for DigSiNet to sample only initial packets of a flow based on previously filtered transfers being relevant for sibling use cases. Also, inter- and intra-switch deduplication mechanisms for exported partial traffic (e.g., based on bloom and cuckoo filters) are possible and required.

A. Use Cases

DigSiNet can be used for several use cases within the FCAPS network management paradigm. However, it is particularly suited for fault and configuration management as NDTs offer ways of testing or troubleshooting a network with less risk. For example, it allows performing diagnosis/isolation and correlation/aggregation steps of the aforementioned paradigm on the NDT instead of the real network. This way, the twin allows network administrators to verify the part of the network that is causing the problem as well as simulate future consequences of the fault. DigSiNet can also be used for configuration management use cases, particularly testing software updates and device configuration changes before deploying them to the real network. This reduces the risk of potential, unnecessary downtime or maintenance.

Another particular use case that DigSiNet was developed for is training and higher education courses. In this setting, DigSiNet allows learners and operators to load a specific state of a real network topology into a learning lab and run experiments (e.g., what-if scenarios or fault analysis). As stated above, this can also be used to test/evaluate new firmware, features or protocols in a safe environment. Besides learning labs, a similar concept can be used as an onboarding tool for new employees or trainees. The learning environment frontend being developed for DigSiNet supports multiple views on the same topology. It can display the current state of the real network and also provide multiple variants of a virtualized version of this network running in the siblings. The variants can be selected in the form of layers (also supporting sub-layers for topology information, e.g., on layer 2, VLANs, overlay networks, etc.) and tabs on a web-based dashboard, that also allows to display streaming context information like counters and historical network state or configuration data. As DigSiNet supports different backends, i.e., VNEs, for each sibling, different variants can use individual network emulation or simulation environments best fitting the purpose of the twin's use case. This way, simulators can be used for exact and deterministic timing, e.g., of network traffic, while emulators can be used to realistically mimic the network devices' behavior and connect to external networks and network management, automation, monitoring and operations tools.

B. Data Visualization

The visualization of network data is a critical aspect of fully leveraging the potential of NDTs. Although DigSiNet does not yet offer comprehensive visualization capabilities in its current form, the underlying technology presents the potential for the development of advanced dashboards. Such dashboards are already being adopted in the enterprise sector, as demonstrated by the Nokia Digital Twin project, where they are used to display network architectures and data flows in real-time, thereby supporting improved decision-making and problem analysis. By integrating event-based streaming and message queues, it is possible to create dynamic network maps that are continuously updated. These maps can not only visualize the

current structure of the network, but also make the impact of configuration changes and network events traceable.

A possible extension of this technology could be the implementation of a ”network time machine”. This feature could enable the visualization of past network states to analyze changes over time. This would be particularly useful for fault diagnosis and performance evaluation, as it would allow administrators to investigate the sequence of events and their impacts in detail. Furthermore, such visualizations could provide context-sensitive information that adapts depending on the selected network segment or data point. This would enable deeper exploration and analysis of specific events as well as their causes and effects. The development of such visualization tools could establish DigSiNet not only as a monitoring platform but also as a comprehensive solution for active problem-solving, network optimization or training scenarios. Figure 6 shows DigSiNet's dashboard with gNMI state data for 2 cEOS nodes, including historical data for *openconfig-interfaces:in-octets*.

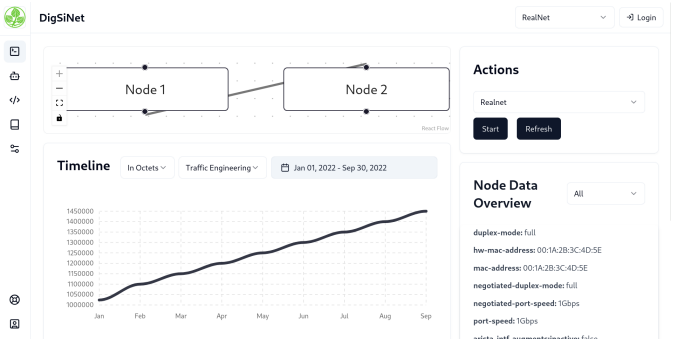

Fig. 6. DigSiNet dashboard showing state data of traffic_engineering sibling.

VI. Conclusion and Future Work

The presented assessment of using message queues (e.g., RabbitMQ) and event streaming solutions (e.g., Apache Kafka) as extension for the DigSiNet digital twin environment revealed different use cases and challenges for the data exchange in NDT scenarios. Implemented event broker extensions for the prototype that are able to run multiple twins of a real network concentrating on individual parts and characteristics to be analyzed in each twin (so called sibling) are available as a public open source repository. Use cases of the DigSiNet environment leveraging the data exchange discussed in this paper are network management and monitoring tasks like configuration, fault and performance, as well as security management, e.g., using continuous integration testing for configuration changes, fault or performance analysis within the siblings or security tests to detect vulnerabilities in the twin and mitigate them accordingly in the real network. The main challenge regarding the data exchange for NDT identified in our experiments was posed by the granularity and redundancy of transferred state and config data between the network twins and the real network. As discussed, four different data types were classified regarding their challenges and use cases based on configuration, monitoring (state and operational) and aggregated/packet-level traffic data using gNMI interfaces to pull and push data from the network devices. Though state-of-the-art event streaming and message queue solutions are designed for performance and scalability, the exchange of this raw state data, especially with respect to traffic data, is not only close to infeasible, given the growing bandwidth of network links, but also not useful, as it is highly redundant, as discussed in the paper. To address this challenge, we will extend our experiments and DigSiNet to evaluate further filtering and especially deduplication techniques, leveraging, e.g., bloom and cuckoo filters that track upcoming traffic flows and disseminate them across the network to prevent data from being collected and sent to the event broker and siblings multiple times. In order to provide overall better scalability, a dedicated service that performs data retrieval on network devices and submission to a corresponding message queue could be promising, as it provides the ability to scale this critical part of the experimental setup as needed. This will also allow for bigger networks and use cases like traffic engineering analysis in a sibling to be further evaluated.

References

[1] S. Rieger, L.-N. Lux, J. Schmitt, and M. Stiemerling, “DigSiNet: Using Multiple Digital Twins to Provide Rhythmic Network Consistency,” in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–5.

[2] C. Semeraro, M. Lezoche, H. Panetto, and M. Dassisti, “Digital twin paradigm: A systematic literature review,” *Computers in Industry*, vol. 130, p. 103469, 2021.

[3] C. Zhou, H. Yang, X. Duan, D. Lopez, A. Pastor, Q. Wu, M. Boucadair, and C. Jacquenet, “Network Digital Twin: Concepts and Reference Architecture,” Internet Research Task Force, Tech. Rep. [Online]. Available: https://datatracker.ietf.org/doc/draft-irtf-nmrg-network-digital-twin-arch/06/

[4] P. Almasan, M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, D. Perino, D. López, A. A. P. Perales, P. Harvey, L. Ciavaglia, L. Wong *et al.*, “Network Digital Twin: Context, enabling technologies, and opportunities,” *IEEE Communications Magazine*, vol. 60, no. 11, pp. 22–27, 2022.

[5] A. Hakiri, A. Gokhale, S. B. Yahia, and N. Mellouli, “A comprehensive survey on digital twin for future networks and emerging internet of things industry,” *Computer Networks*, p. 110350, 2024.

[6] J. A. Gomez Gaona, E. Kfoury, J. Crichigno, and G. Srivastava, “A survey on network simulators, emulators, and testbeds used for research and education,” 2023, last accessed: 2023-06-04. [Online]. Available: http://dx.doi.org/10.2139/ssrn.4457366

[7] P. Almasan, M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, D. Perino, D. López, A. A. Pastor Perales, P. Harvey, L. Ciavaglia, L. Wong *et al.*, “Digital twin network: Opportunities and challenges,” 2022. [Online]. Available: https://arxiv.org/abs/2201.01144

[8] M. Liebsch, M. Stiemerling, and N. Schark, “Challenge: Network Digital Twin - Practical Considerations and Thoughts,” Internet Research Task Force, Tech. Rep. [Online]. Available: https://datatracker.ietf.org/doc/draft-liest-nmrg-ndt-challenges/00/

[9] R. Vilalta, R. Casellas, R. Martínez, R. Muñoz, A. González-Muñiz, and J. Fernández-Palacios, “Optical network telemetry with streaming mechanisms using transport API and Kafka,” in *2021 European Conference on Optical Communication (ECOC)*. IEEE, 2021, pp. 1–4.

[10] M. S. Rodrigo, D. Rivera, J. I. Moreno, M. Àlvarez-Campana, and D. R. López, “Digital twins for 5G networks: A modeling and deployment methodology,” *IEEE Access*, vol. 11, pp. 38 112–38 126, 2023.

[11] W. Hu, T. Zhang, X. Deng, Z. Liu, and J. Tan, “Digital twin: A state-of-the-art review of its enabling technologies, applications and challenges,” *Journal of Intell. Manuf. and Special Equip.*, vol. 2, no. 1, 2021.

[12] C. Hardegen, B. Pfülb, S. Rieger, and A. Gepperth, “Predicting network flow characteristics using deep learning and real-world network traffic,” *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2662–2676, 2020.