Student Conference Proceedings
Vol 1, No 1, 2025, Article ID 1994, 4 Pages

Student Conference Lübeck

Proceedings Article

Implementation and Evaluation of a Prototypical Service for the Processing and Provision of Clinical Trend Data in a Distributed System

Isticahayanittamiarzahraa[a*] · Raika Rauterberg[b] · Johannes Kollien[b] · Max Urban[c*]

[a]Student of Biomedical Engineering, Luebeck University of Applied Sciences, Lübeck, Germany
[b]Draegerwerk AG & Co. KGaA, Lübeck, Germany
[c]Technische Hochschule Lübeck, Lübeck, Germany
*Corresponding author, email: i.isticahayanittamiarzahraa@stud.th-luebeck.de; max.urban@th-luebeck.de

Received 13 March 2025; Accepted 23 June 2025; Published online 18 July 2025

Abstract

The ISO/IEEE 11073-SDC standard defines architecture and protocol on interoperability between manufacturer-independent medical devices. In this paper, a prototype implemented this standard to process clinical trend data using Apache Kafka for data storage and GraphQL for its interface. The proposed prototype demonstrated real-time and historical data connected to hundred simulated devices. In the evaluation, the processing time of trend during high-load setting has occasional outliers, but its median is relatively low. A single historical trend query with varied payload and real-time subscriptions were effectively executed, meanwhile simultaneous queries from multiple devices caused high latency due to resource contention.

I. Introduction

IT infrastructures in modern hospitals have multiple different systems and restrictions on the interfaces as well (e.g., unidirectional communication between medical devices). Furthermore, each device generates and utilizes its own physiological data to present it to the caregivers based on individual interpretation and processing, which results in inconsistent and unharmonized data presentations. The ISO/IEEE 11073-SDC (*Service-oriented Device Connectivity*) standard enables secure, vendor-independent interoperability and device-to-device communication in medical products, adding clinical value in high-acuity care settings. This standard ensures standardized storage and access mechanisms to physiological patient data, which can be used by applications that provide improved data views for diagnosis or therapy [1],[2]. The prerequisite for this is a consistent representation of the provided clinical data throughout the entire system. In addition to real-time physiological patient data, historical clinical trend data are especially required to provide comprehensive workflow support for clinical staff. Customer needs consistent historical data visualizations while reducing the implementation efforts in the system components. Therefore, a centralized calculation of clinical trend data and its provision via a clearly defined interface is required.

II. Materials and Methods

II.I. System Architecture

By using the principles of the SDC protocol in the system, seamless data transfer between medical devices is possible. The communication protocol of SDC Standard is specified in *Basic Integrated Clinical Environment Protocol Specification* (BICEPS). It defines the domain information and service model of the medical device. Inside BICEPS there is *Medical Device Information Base* (MDIB), containing an entity-relationship based model for managing medical objects such as patient data, device configurations and remote operations, mapped to an XML format. In the software architecture (Fig. 1), the Data Aggregator receives and collects BICEPS data from medical devices via SDC connection. It then encrypts and stores the data in an Apache Kafka cluster (from now on called "Kafka"), which allows integration of stream processing and persistent logging [3]. There is a built-in Java library in the Data Aggregator, which can detect connected devices and retrieve their MDIB data, creating a clone of the medical device. By accessing this data, the prototype then extracts specific measurement to calculate latest clinical trend data (from now on called "trend") based on domain specific requirements. Through an interface implemented in GraphQL, the processed data can be integrated and streamed live. Using a client application, the end users can interact with the system, displaying clinically relevant data. Nevertheless, these data can also be utilized by other services without an UI (e.g., report generator or user applications).

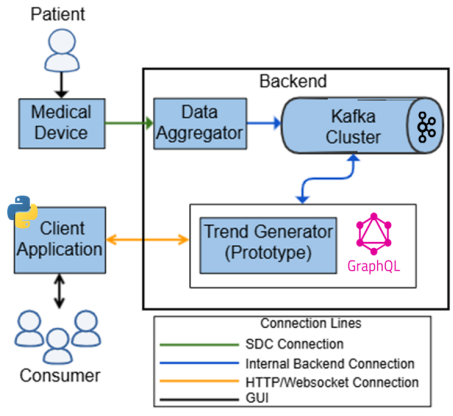

Figure 1: Architecture of the clinical trend data processing prototype with SDC protocol

II.II. Data Acquisition

The MDIB was provided by a patient monitor model, that monitors vital signs (e.g., heart rate, oxygen saturation, and other physiological parameters) in acute care with a sending rate of 0.5s. All backend components (Data Aggregator, Kafka Cluster and Trend Generator) were deployed on a virtual machine Linux 24.04.1 LTS. The machine was equipped with 24 CPUs (QEMU Virtual CPU version 2.5+ @ 2.0GHz), 32 GB of memory, and 246 GB of storage. The services were running for seven days straight with hundred simulated devices connected to the service, hundred measurements were transmitted to each device. This demonstrates the system's ability to manage high data throughput.

II.III. Trend Processing

Through a digital cloned device that is provided by the Java library, the Trend Generator accesses all reports from BICEPS to store and retrieve data. From these reports, relevant trend information is extracted from the BICEPS to monitor and analyze trend. This includes metrics, which are abstract objects in an MDIB that contain measurement, settings, states and contextual information of a medical device [1],[2],[4]. The MDIB database is structured into two components: the descriptive part, defines typically static attributes that include structure of device, services, metrics and coded values which describe the transmitted data; and the state part, which contains real-time dynamic information such as measurement values, corresponding to the objects in descriptive part [1]-[4].

To extract this database, firstly state objects were derived in the MDIB of a dedicated device that has fundamental metadata and are explicitly identified by a handle (without assigning any semantic). For trend extraction, the focus was only on the data measurement, including numeric metrics (e.g., metric value and determination time). Trends are created as entries in a map with the handle of the descriptor as the key and complete measurement state as the value. This allows retrieval of all relevant physiological measurements and their associated descriptive objects (e.g., unit of measure, code, concept description) [4]. Trend sampling occurs within every certain selected time by the prototype and sends it to a Kafka topic as a storage, which contains device-specific information (Fig. 2). If there is a trend update coming from the same device, it will be appended to the partition of the topic in chronological order. Each update has an incremental offset and also a timestamp, which indicates when the trend was stored. To access historical trends, Kafka was configured to retain data for up to 14 days [3].

Offset	Key	Value	Timestamp
1	handle1	[35, BodyTemp]	2025-01-01T07:00:00Z
2	handle2	[64, SpO2]	2025-01-01T07:00:00Z
3	handle1	[36, BodyTemp]	2025-01-01T07:00:15Z
....	....	....	....

medical-device-123_0 partition

Figure 2: Demonstration of partitioned trend in Kafka topic

II.IV. Data Provision using GraphQL

In comparison to traditional APIs, GraphQL is able to provide specific data that clients need, which improves efficiency and also flexibility in data retrieval. GraphQL is integrated into the prototype to interact with and display trend data to a client. GraphQL only uses a single endpoint URL to manage the operations, such as query and subscription [5]. The historical trends can be retrieved by having a query operation. The operation accesses the relevant Kafka topic within the backend. To consume a specified record, its topic, partition, and offset must be known already at the beginning. Because Kafka supports time-based queries by timestamp, a client is able to return the measurement value closest to the specified timestamp retrieved, representing the trend value for that timespan [3]. Operating a subscription allows real-time trend streaming. The Trend Generator access updates directly from the device replica during the streaming instead of Kafka to avoid possible latency and also interruption during trend storage.

II.V. Client Application

In order to connect with the GraphQL server side to retrieve historical data or subscribe to real-time updates, GraphQL-Python-library "gql" was deployed to build a client application. This library supports HTTP and WebSocket transports for flexible data retrieval and subscription [6]. For query operations, such as historical trend retrieval, using HTTP is appropriate for synchronous requests where data is returned in response to a single query. A real-time trend subscription is managed via WebSocket transport, providing a single persistent real-time two-way communication protocol over TCP. Both operations get their responses in a JSON format (Fig. 3). A client can have multiple requests on GraphQL by using async transports. This allow simultaneous subscriptions and queries on multiple devices without any interruption on the system [7].

Query

```
query GetHistoricalTrends(
$device_id: String!,
$handle: String!,
$timePoint: String!) {
  historicalTrends(
timePoint: $timePoint) {
    description
    value
    timestamp
  }
}
```

JSON Response

```
{
  "data": {
    "historicalTrends": [
      {
        "description": "Heart Rate",
        "value": 75,
        "timestamp": "2025-01-17T02:29:59Z"
      }
    ]
  }
}
```

Figure 3: Example of a GraphQL query structure and its response in JSON data format

III. Results and Discussion

The performance of prototype was evaluated considering the processing time from trend emission by the medical device to its storage in Kafka and the response time from a client request to trend retrieval on a client. Fig. 4 shows trend storage to Kafka and trend processing were efficient with minimal variability. The initial creation of a cloned device, involving access to and transmission of MDIB, accounted for most of the processing time out of all services. Its median is around 300 ms with occasional outliers exceeding 800 ms. These outliers occurred during the initial setup, when 100 new devices were detected and trend topic was created in Kafka for each device. With a median of 300 ms on a response time for overall service, this indicates a fast trend processing which is under half a second.

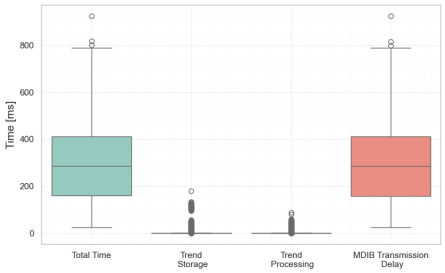

Figure 4: Time measurement of each service on prototype

For historical trend retrieval (Fig. 5 and 6), the response time can be important for the caregiver in the hospital to monitor a patient. The shorter the wait time to retrieve all needed trend of a patient, the better. Each query is measured 100 times per mutation on their response time to ensure reliability and exclude outliers. As shown in Fig. 5, median response times increase with a

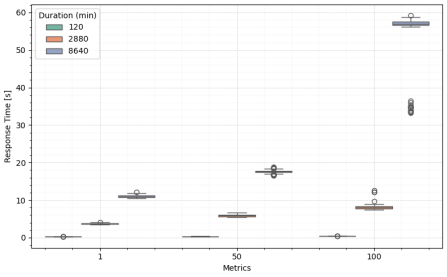

Figure 5: Response times of historical trend queries for multiple metrics from a single medical device (interval sampling time = 30s)

higher number of metrics and longer durations. For 100 metrics over 8640 minutes, some outliers are below Q1, which indicate occasional faster response times under heavy load. In contrast, Fig. 6 shows larger increases in response times with many outliers when multiple devices run simultaneously, even with reduced payload (one metric). It became more challenging as more devices requested data for longer duration at the same time. 50 devices do have lots of outliers when the duration is

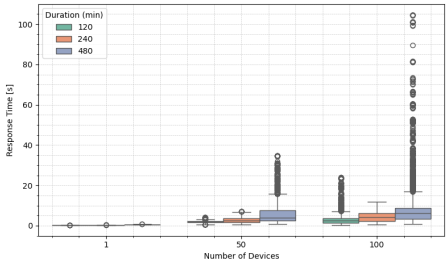

Figure 6: Response times of historical trend queries for one metric from multiple medical device in parallel (interval sampling time = 30s)

480 minutes with the maximal value of less than 40 seconds. In comparison, 100 devices over 480 minutes could exceed more than 100 seconds to retrieve trend, which is two times the maximal value on 50 devices. This suggests that simultaneously requesting data could disrupt the process of the prototype on storing trend data to Kafka during retrieval and cause delays. Using asynchronous functions for trend retrieval and storing in Kafka could help achieve a non-blocking application, so each service on retrieval and storing could be run without waiting for each service to end. To simulate multiple devices subscribing to trend simultaneously, up to 100 devices were connected to the Trend Generator in parallel for 5 hours. Fig. 7 shows overall low median response times as device subscriptions increase, which is less than 10 ms, as it streams trends directly from the device replica, not through Kafka. Nevertheless, outliers grow with higher device counts due to resource competition with a maximum value of less than 50 ms. However, the limitation on how many devices that can be subscribed to in parallel needs further investigation.

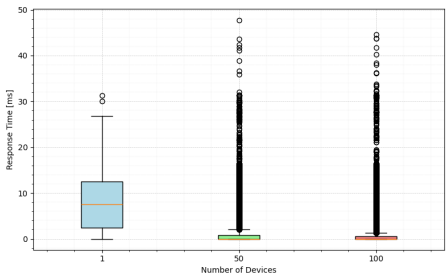

Figure 7: Response times for 5 hours duration of subscription on multiple devices in parallel

IV. Conclusion

The prototype shows an efficient performance on clinical trend data processing and retrieval in a distributed system. A single query of historical trend retrieval demonstrated gradually increasing response times as the payload gets higher. Even when there were up to 100 devices subscribed to the real-time trend, the response times on subscription had a low median overall, which was less than 50 ms. In contrast, as multiple devices, particularly 100 devices, request historical trends at the same time, the prototype resulted in challenges on the response times with delays and increased outliers. This is due to the system's resources being shared by multiple devices at once. In the future, applying asynchronous on retrieving historical trends can help manage simultaneous requests. Therefore, the prototype could deliver reliable clinical trend data in a demanding clinical setting.

Acknowledgments

The work has been carried out at Drägerwerk AG & Co. KGaA, Moislinger Allee 53, 23558 Lübeck and was supervised by the Department of Applied Sciences, Technische Hochschule Lübeck.

Author's statement

Conflict of interest: Authors state no conflict of interest.

References

[1] M. B. et al., Using data distribution service for ieee 11073-10207 medical device communication, in *Wireless Mobile Communication and Healthcare: 6th EAI International Conference, MobiHealth 2016, Milan, Italy, November 14–16, 2016, Revised Selected Papers*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 320, Cham: Springer, 2020, 127–139. doi:10.1007/978-3-030-49289-2_10.

[2] D. G. et al., An approach to integrate distributed systems of medical devices in high acuity environments, in *5th Workshop on Medical Cyber-Physical Systems*, V. T. et al., Ed., ser. Open Access Series in Informatics (OASIcs), 36, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/OASIcs.MCPS.2014.15.

[3] G. W. et al., Consistency and completeness: Rethinking distributed stream processing in apache kafka, in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD '21, 2602–2613, New York, NY, USA: Association for Computing Machinery, 2021. doi:10.1145/3448016.3457556.

[4] ISO/IEC/IEEE. Health informatics — Point-of-care medical device communication Part 10207: Domain Information and Service Model for Service-Oriented Point-of-Care Medical Device Communication. *ISO/IEEE 11073-10207:2019(E)*, pp. 1–24, 2019, doi:10.1109/IEEESTD.2019.8675788.

[5] M. Bryant, Graphql for archival metadata: An overview of the ehri graphql api, in *2017 IEEE International Conference on Big Data (Big Data)*, 2225–2230, 2017. doi:10.1109/BigData.2017.8258173.

[6] G. Python, Graphql documentation, `https://gql.readthedocs.io/en/stable/index.html`, Accessed: 2025-01-20, © Copyright 2020, graphql-python.org, 2020.

[7] G. Foundation. (2025). Graphql: A query language for your api. Accessed: 2025-01-20, URL: `https://graphql.org/learn/`.