Distributed Computing of Management Data in a Telecommunications Network

Ville Kojola[1](✉), Shubham Kapoor[2], and Kimmo Hätönen[3]

[1] Center for Ubiquitous Computing, University of Oulu, Oulu, Finland
ville.kojola@student.oulu.fi

[2] Department of Computer Science, University of Helsinki, Helsinki, Finland
shubham.kapoor@helsinki.fi

[3] Nokia Bell Labs, Espoo, Finland
kimmo.hatonen@nokia-bell-labs.com

Abstract. In this paper, we propose a concept for distributed Management Plane data computation and its delivery in the cellular networks. Architecture for proposed concept is described. Calculation of Key Performance Indicators is distributed to the cellular network edge, close to the managed network elements which reduces the volume of the Management Plane traffic. In this concept, further aggregation and refinement of data is done in the nodes located in the operator's cloud, close to consumers of Management Plane data. Distribution of calculation to the network edge reduces load at the network operator's central database. This paper presents an analysis to the benefits of the proposed concept. Efficient on-demand type streaming data delivery model allows network management functions to be plugged in to receive Management Plane data directly without database access. A demonstrator system has been implemented. The feasibility of the implementation is evaluated in terms of resource consumption and latency.

Keywords: Cellular network · Key performance indicator · Network management

1 Introduction

Cellular network elements provide a wide array of performance metrics called performance counters. The performance counters and the Key Performance Indicators (KPIs) calculated from them are essential in Performance Management (PM) of a cellular network. Network management systems collect data to the operation centers, where it is monitored and analysed to detect any defects or suboptimal states in performance or service quality [12,17].

With the usage growth and development in technology, the number of network elements in cellular networks is growing. The recent developments in Fourth Generation (4G) cellular network has been towards flat Radio Access Network (RAN) hierarchies which simplifies network but also removes hierarchical processing of PM data which was there in earlier cellular generations [15].

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2017
R. Agüero et al. (Eds.): MONAMI 2016, LNICST 191, pp. 146–159, 2017.
DOI: 10.1007/978-3-319-52712-3_11

A centralized processing of this PM data is thus required for monitoring network, which makes centralized models a popular choice for monitoring PM data [7].

For effective performance management and emerging concepts such as Self Organizing Networks (SON) [3] there is a need for higher frequency performance reporting by network elements. In a centrally operated cellular network these factors contribute to a large Management Plane (M-Plane) data volumes and high computational complexity. In past, semantic compression and local processing have been proposed as means to distribute computation in network management and to reduce network management traffic [7].

The motivation of this paper is to demonstrate the concepts of local processing and streaming of data in a telecommunication environment and to show how these concepts could help to solve the problem for processing high volumes of management plane data that is about to arise in cellular networks due to its evolution. This paper presents a demonstrator system, which implements local processing and refinement of cellular network performance reports in near-real-time. The refined or aggregated performance reports are directly and instantly transferred to network management functions in on-demand streams.

Section 2 of this paper presents the concept of our proposed solution. Section 3 presents the architecture of the demonstrator system. In Sect. 4 we discuss about experiments conducted on our system. In Sect. 5 we discuss about achievements of our implemented system with respect to current systems and discuss future direction of our research. Finally, we conclude our paper in Sect. 6.

2 Concept

The objective of proposed concept is twofold: to minimize the amount of transferred data and to maximize the relative amount of up-to date information in it. We propose two major changes to telecommunications networks management systems to achieve this: do the needed computation close to the place where the data is generated and let the data flow directly to applications analyzing it in real time. These changes are inline with recent development in Mobile Edge Computing (MEC) paradigm [2].

To do the distributed processing and to transfer its results to management system applications, we introduce three types of components that are embedded in the network: Data Fetchers next to data sources, Data Switches in the operator's core cloud and Data Hubs next to the Consumers of M-Plane data, i.e., OSS applications (see Fig. 1). This set-up corresponds to a publish-subscribe system [10], where Data Fetchers are data publishers and Data Hubs are subscribers.

A Data Fetcher is a component that does the processing of the data next to the place where it is being generated. In Fig. 1 that is shown to be next to each Base Station. Data Fetcher publishes the computation results to Data Switches that act as data brokers in this publish-subscribe architecture.

Data Switch is responsible for routing data coming from Data Fetchers to those Data Hubs that are serving applications that have requested the data. Data Switch does this so that data flows are not unnecessarily duplicated in

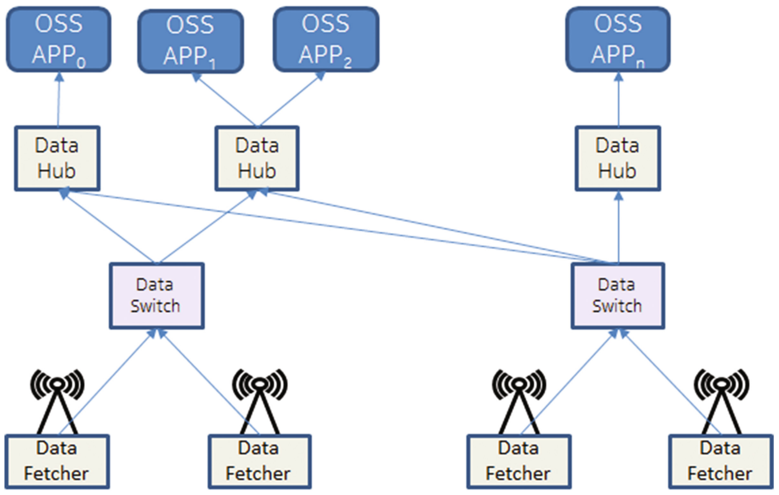


Fig. 1. Elements of the proposed concept. The novel elements of the concept are the Data Fetcher, the Data Switch and the Data Hub.

the network and the content is published only once by a Data Fetcher. In Fig. 1 there are two Data Switches, of which the left one is routing data from two Data Fetchers towards OSS applications 1, 2 and 3, and the right one transfers the data flows from two Data Fetchers to the same applications but also mirrors those flows to OSS application n in upper right corner.

Data Hub serves applications that make requests for data. In Fig. 1 three Data Hubs are shown serving four OSS applications. A Data Hub has an interface, to which an application can send a subscription, in which it specifies what content, from where and how often it wants to receive from the network. The Data Hub sets up the corresponding data collection and processing in appropriate Data Fetchers or, if the similar request has been placed already by someone else, mirroring of the data flow to Data Hub where the request was made. When the Data Hub receives the data, it delivers that to applications via specified interface.

2.1 Streams

The system aims to optimize data collection and delivery from a managed device, such as base stations, by implementing a streaming pattern, where consumers only needs to subscribe to a stream to start receiving data. Inspiration is drawn from the publish/subscribe communication pattern [10] and Information-centric networking [18], which abstract away from the traditional point-to-point connectivity. Publish/subscribe pattern reduces interaction required in collecting the data by removing the need for repeated data queries [9]. Publish/subscribe allows publisher and subscriber to be fully decoupled in time, space and synchronization, which leads to increased scalability [10]. Information-centric networking trades host centric view into a one where network mediates named content based on the interests advertised by the nodes [18].

Two methods are used to optimize data collection, assuming that data needs to be collected repeatedly and that multiple consumers require the data. Firstly, request interaction is minimized with the subscription pattern. A stream of some specific data is created when a consumer makes a subscription for the data. This request starts data processing and delivery in a Data Fetcher, which will publish stream content either sporadically or periodically, depending on the type of the request. No further requests or queries are required and the Data Fetcher continues publishing stream content on its own without the need for further interaction.

Secondly, multiple consumers may subscribe to the same stream, which minimizes the need to send duplicate queries to a Data Fetcher. Therefore, the Data Fetcher needs to send the same data only once using its limited bandwidth resources. Further replication of the data is done in the Data Switch in operator cloud where abundant network resources are expected. A Data Fetcher can publish the data with minimal delay, because it is in control of the publishing and does not depend on polling from other elements. Consumers are decoupled from Data Fetchers and each other, but they can still receive the subscribed data with low latency.

Ideas are lent from the publish/subscribe pattern, which can also be part of the implementation. However some changes are made. Decouplings provided by publish/subscribe system are limited to some extent: Data Fetchers will publish only after a stream has been created (no full decoupling in time) and streams may be tied to some specific Data Fetchers (no full decoupling in space). Subscription model used is based on the idea from information-centric networking of named content which can be requested on demand.

Delivery mechanism to consumers is also simplified. Instead of consumers needing to poll for updates and a database struggling to fulfil those requests, a Data Hub can immediately forward each new data record it receives to subscribed consumers. Subscribing, streaming and data replication are depicted in Fig. 2. First, in quadrant (a) in upper left corner, an OSS application places an order for some data coming from all three Data Fetchers in the figure. In quadrant (b) in upper right corner, the system sets up streams of data flowing from all Data Fetchers to the application that subscribed for it. Later another OSS application makes the same request as is depicted in lower left quadrant (c). This makes Data Switch in the middle to mirror established flows to the Data Hub serving that OSS application. This is depicted in lower right quadrant (d) of Fig. 2.

2.2 Distributed Computing

As mentioned earlier, the system aims to reduce the management traffic and the need for centralized computation of the management data in telecommunications network. These two goals are considered in regards to two points of interest, which are identified as potential bottlenecks. Firstly, management traffic is mainly of concern in the link between the network element and the operator

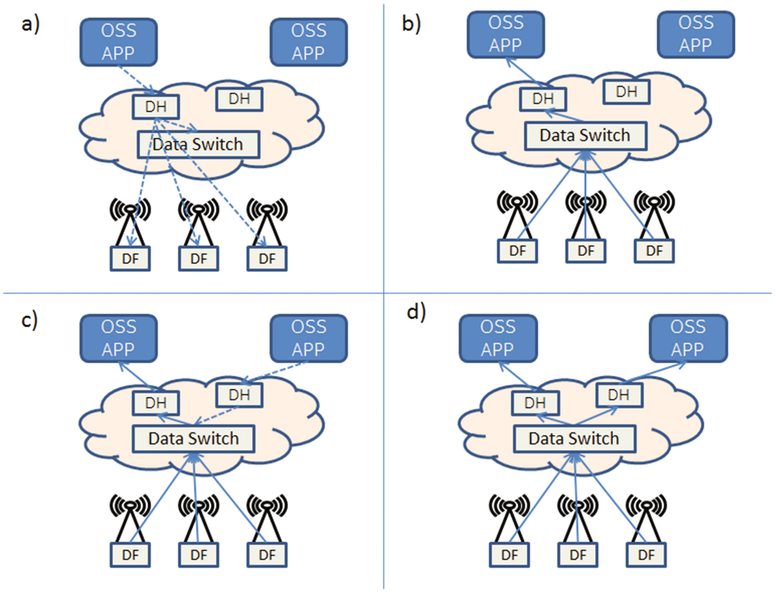


Fig. 2. Stream initialization and delivery. In stream subscription a Data Hub forwards stream initialization request to Data Fetchers. Stream contents are delivered through Data Switch. Pre-existing streams can be replicated to other Data Hubs from Data Switch.

cloud, because of the assumption of a limited bandwidth. Secondly, computational concerns relate to the centralized OSS applications, which may need to process large volumes of data. By moving computation from a central OSS application to Data Fetchers at the edges of the network, the data can be refined already at the source [16].

There are various kinds of data that the network operator collects from a network element. [12,17] Distributed computing of that data close to the source could be leveraged in many ways. One example is raw counter data that is measured by the network element. The network element measures thousands of counters, which need to be collected periodically and then KPIs are calculated from the counters. This calculation is done in the central server before inserting KPIs to the central database. From there, the KPI data can be used to manage the network element using the closed-loop principle [5,13].

KPIs are calculated from the counter values by using simple formulas where one or more counter values are summed, subtracted, multiplied by a constant or divided. By computing all the needed KPIs in the Data Fetcher, the data to be transferred can be reduced in volume. The reduction depends on the complexity of KPI formulas used. Some may consist of only one counter while others may consist of a dozen. A KPI consisting of 6 counters roughly reduces the data volume needed to transfer to a sixth. Equations 1 and 2 shows equation for a typical KPI calculation from raw counter values [4].

$$InitialEPSBEstabSR = \times \frac{\sum_{cause} RRC.SuccConnEstab.[cause]}{\sum_{cause} RRC.AttConnEstab.[cause]} \times \frac{\sum RRC.SuccConnEstab}{\sum SuccConnAtt} \times \frac{\sum_{QCI} SAEB:NbrSuccEstabInit.[QCI]}{\sum_{QCI} SAEB.NbrAttEstabInit.[QCI]} \times 100 \quad (1)$$

$$MobilitySuccessRate_{QCI=x} = \frac{HO.ExeSucc}{HO.ExeAtt} \times \frac{HO.PrepSucc.QCI_{QCI=x}}{HO.PrepAtt.QCI_{QCI=x}} \times 100[\%] \quad (2)$$

3 Implementation

The system implements a middleware between the OSS applications and the base stations. As described earlier, the main components of the system are: Data Fetchers, Data Switches and Data Hubs. For controlling the system, there are also Global Repositories, that help in coordinating subscriptions in the system. The system architecture is depicted in Fig. 3. The Data Hubs communicate directly with data consumers, i.e., OSS applications. They receive requests from the applications and forward resulting data to those. Each Data Fetcher is paired with a base station and communicates directly with the base station. Communication streams, such as performance data streams or log streams, are relayed through a Data Switch layer from Data Fetchers to Data Hubs. Data Switch layer is a network of Data Switches, publisher/subscriber brokers, which temporarily caches data and delivers it efficiently.

We chose to use Apache Kafka [6,11] as the Data Switch layer. It includes many features and functions that are needed in our system. We interfaced it with base stations and OSS applications by Data Fetcher and Data Hub layers. We do management of data streams and book keeping of their subscribers by using a combination of Apache Zookeeper [6] and our Global Repositories.

3.1 Data Fetcher

Data Fetcher is the element that resides at the network edge close to a network element. The computing environment of Data Fetcher is characterized by limited CPU, RAM, bandwidth and disk resources. Data Fetcher can be in one-to-one relation or one-to-many relation to managed network elements. Close proximity to network elements reduces network latency and allows Data Fetcher to minimize the network load resulting from the transfer of data. Data Fetcher consist of Fetcher and Sender. Fetcher receives requests for streams and fetches data

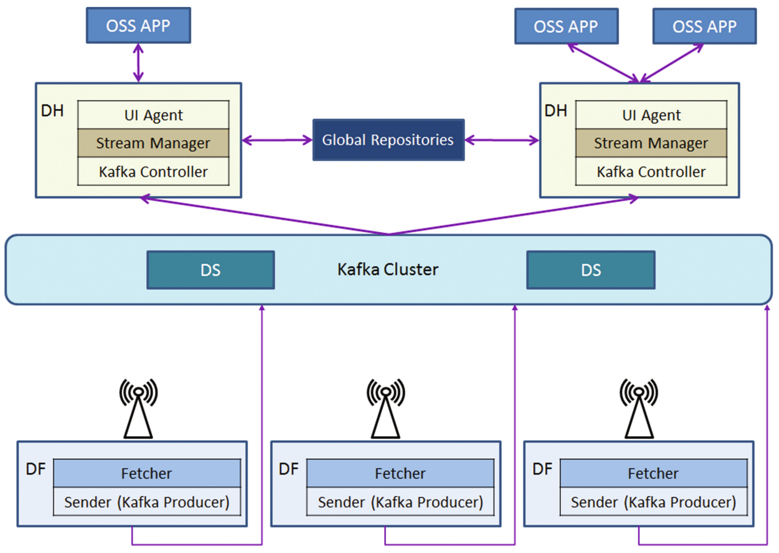


Fig. 3. System architecture. Data Switches are implemented as Apache Kafka cluster. Data Fetcher and Data Hub consist of subcomponents. Sender and Kafka controller contain Apache Kafka producer and consumer clients.

from managed network element it is connected to. Fetcher fetches raw counter data. Sender is a Kafka producer client which publishes stream contents to Kafka broker.

Based on the consumer requests Data Fetcher receives stream requests from Data Hubs. Requests can, for example, initiate collecting and processing of data or cancel it. Start request contains list of names of KPIs that need to be collected. For each KPI name a script is started which consumes counters and produces a KPI value. When the Data Fetcher collects a new set of counters it forwards the relevant counters to KPI scripts which then calculate the KPI values. KPI values for a topic are then grouped together and published to Kafka. Cancel request stops all processes involved in producing a particular stream.

Data Fetcher acts as an interface to this network element data so that no OSS application needs to directly request the data from the network element, but all the data that is needed in the process of network operation is fetched by the Data Fetcher. The data that has been fetched is rapidly forwarded by the Data Fetcher to Data Switch, so that the data does not need to be stored for long on Data Fetcher and is fresh when it reaches OSS applications.

Before the data is sent, it can be processed in the Data Fetcher. Volume of the data can be reduced by refining it to a format that better fits OSS application or operator needs, such as calculating key performance indicator from the raw counter data. Refining the data at the Data Fetcher reduces the amount of calculation needed at the center of the operator cloud.

3.2 Data Hub

Data Hub resides in the network operator cloud. The computing environment of Data Hub is characterized by sufficient hardware and network resource. Data Hub consists of UI Agent, Stream Manager and Kafka Controller. Kafka Controller manages creation and deletion of Kafka topics. Kafka Controller communicates with Global Repositories to synchronize subscription counts by Data Hubs to topics. Kafka Controller also embeds Kafka consumer. Stream Manager corresponds to Fetcher in DF and can be used to further aggregate data, for example by geography. UI Agent is the consumer side interface of DH, which serves requests by consumers and delivers stream contents to the consumers. Redundancy of Data Hub instances is possible. Data Hub receives and serves requests made by consumers by doing book keeping in Global Repositories and forwarding requests to appropriate Data Fetchers.

Requests can be either of stream subscription type or of one-off request-response type. Traditional request-response model can be used to pull data from Data Fetchers on-demand. Stream subscription subscribes a consumer to regular or irregular stream of packets from one or more Data Fetchers which the Data Hub delivers to the consumer.

Data delivery in the streaming model is optimized by reducing the amount of communication required between the Data Hub and Data Fetcher. A stream is started when a Data Hub issues a start command to one or more Data Fetchers. When the stream is started the stream delivery continues Data Hub and Data Fetchers decoupled. Multiple consumers can receive the same streamed data through the same Data Hub. Multiple Data Hubs can also receive same data from the same Data Fetchers, yet Data Fetcher only needs to send the data once.

3.3 Data Switch

Data Switch is a publish/subscribe broker residing in the operator cloud. In our implementation it is a Kafka broker and we implement our data streams with Kafka topics. Task of broker is to deliver stream contents from a Data Fetcher to one or more Data Hubs. Data published by Data Fetcher to a Data Switch is available to Data Hubs and consumers from Data Switches so that direct requests to the Data Fetcher can be limited.

3.4 Coordination

Information about Data Fetchers, Data Hubs and running streams is stored in Global Repositories. Global Repositories contains the global view of the components running in the system and is used to coordinate the different requests. When a consumer requests a stream the involved Data Hub will communicate with Global Repositories to decide whether a request needs to be sent to a Data Fetcher or the stream already exists and can be subscribed to in the Data Switches.

4 Experiments

The feasibility of the proposed system is measured in an experimental setup. In the experimental configuration 12 Data Fetcher instances are run in separate virtual machines of a cloud environment. Each virtual machine has 1 GB of RAM and a 2400 MHz dual core processor. In the experiment the virtual machines are not communicating with actual network elements, but are replaying recorded M-Plane data of real LTE Cells which are the part of live LTE network hosted by Nokia for research and development purposes. ZooKeeper and Kafka broker are deployed on a separate virtual machine. Data Hubs receive data from the Data Fetchers through the Kafka broker. Data Fetchers produce two streams of data. One to measure the resource consumption of the Data Fetcher and the other to emulate management data computing and collection. The usage of network, CPU and memory resources by Data Fetchers are monitored to evaluate the feasibility of the Data Fetcher and distributed management data computation concepts.

Data Fetchers were set to produce different numbers of KPIs. Resource utilization in each case was recorded. Mean, median and maximum values for each resource statistic were calculated for each Data Fetcher instance. By analyzing the results the resource requirements to produce KPIs can be estimated.

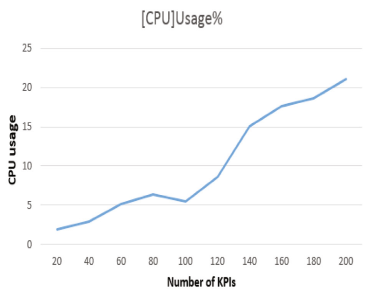
Data Hub was installed on a separate virtual machine with 4GB of RAM and 2400 MHz dual core processor. KPIs from base stations could be requested by operator/consumer in which he is interested. Based on the request Data Hubs subscribe to Kafka topics. Current network Performance data which we had was sufficient to calculate 220 KPIs.

We thus created one consumer which subscribed to different number of KPIs for different Data Fetchers. The number of KPIs requested for different Data Fetchers increased linearly from 20 KPIs to 220 KPIs. This consumer also subscribed for resource metrics of each Data Fetchers. Resource metrics were reported on each Data Fetchers using Linux utility Collectl [1]. Once these Resource consumption metrics were generated on Data Fetcher, they were streamed to the consumer using our system. Hence we got Resource consumption metrics of each Data Fetchers at a single place to analyze Data Fetcher performance.

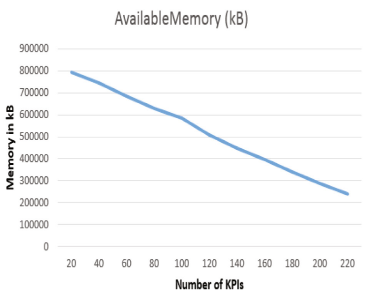
Raw real network performance data were replayed in each Data Fetcher for 24 h, while each Data Fetcher was calculating different number of KPIs from this data. Resource metrics of all Data Fetchers were analyzed for this period and statistical data was extracted from it.

This experiment gave us interesting performance insights of our concept. We analyzed Resource metrics of Data Fetchers such as CPU utilization, Memory Utilization and network stats for all Data Fetchers. CPU consumption increased almost linearly with number of KPIs subscribed. We got mean and median CPU usage figure of about 22% and 21.0046% respectively for Data Fetcher with all 220 KPIs subscribed.

There was relatively higher CPU consumption at all Data Fetchers when streaming was started but that could be regarded as a startup overhead as CPU utilization settled back from that peak value. Free, Buffered and Cached memory for each Data Fetchers were analyzed and provided similar findings that free memory was decreasing with amount of KPIs subscribed. Plots in Fig. 4 shows variation of CPU Utilization and Free Memory with number of KPIs subscribed.



(a) CPU Utilization VS Number of KPIs



(b) Available Memory VS Number of single KPI subscription topics

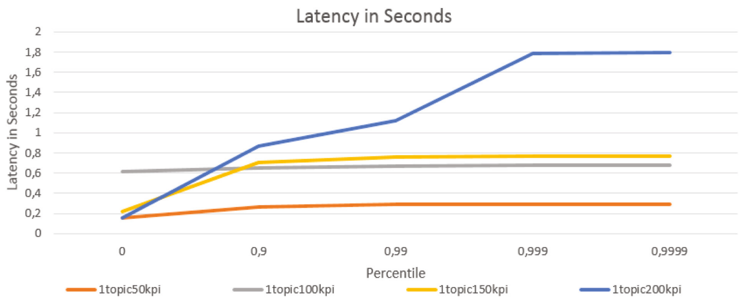
Fig. 4. CPU utilization and free memory distribution of a Data Fetcher with the number of single KPI subscription topics.

High frequency performance reporting is necessary in optimizing the future mobile networks. Low end-to-end delay was considered a vital part of the implementation.

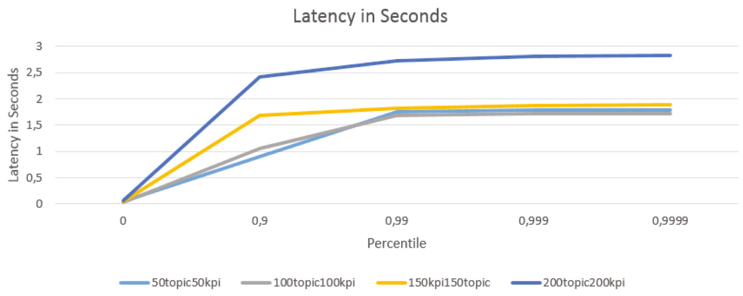
We calculated latency of our proposed solution and studied its variation with the number of Kafka topics. We wanted to find out what kind of topic set-up would be optimal to implement a data stream. For measuring latency we installed Data Hub on same machine in which Data Fetcher was installed. This was done in order to get precise latency figures. Topics subscriptions to same machine's Data Fetcher were made. Time difference between publishing of data at Data Fetchers and receiving of this data which has traversed entire network was measured. Single topics which subscribed for multiple KPIs and individual topic per individual KPI were created. Latency figures for each topics were then measured separately.

We noticed that when a stream is started the initial latency will be high and is an outlier, but it then settles down to a "steady state" where latency figures mostly stays in the same range. Since these performance management data streams would be long lived, we are more interested in these steady state latency figures. Hence we removed initial reading, which was an outlier, from our latency records. Latency was higher with one KPI per topic scenarios. Largest recorded steady state latency in the experiment was with scenario when 200 KPIs were subscribed, with each KPI having a separate topic. In this case latency at 99.99 percentile was 2.8274 s while the median latency was 0.932 s. Average Round Trip Time (RTT) between Data Switch and Data Fetcher using ping was measured at 3.665 ms. Figure 5 shows percentile graph for variation of latency figures with different topics in the steady state.

Overall the end-to-end delay of the implementation appeared low and suitable for near-real time reporting.



(a) Latency distribution for single topic with multiple KPIs



(b) Latency distribution for one KPI per topic

Fig. 5. Latency distribution with different number of topics. In the single topic case, all calculated KPI values are sent in the same stream. In the KPI per topic case multiple separate streams are used.

5 Discussion and Future Directions

In this section we will discuss briefly the related work and existing models in the telecommunication management systems and achievements of our implemented system against them. We also discuss about some future directions for our work.

In their paper [16] Simões et al. compared five different telecommunication management system models ranging from mobile agents model to static centralized model based on SNMP [8]. This paper explored benefits of mobility, locality and distribution in different models. Models which were explored were the Static Centralized Model, the Migratory Model, the Master/Worker Model, the Static Delegated Model and the Migratory Delegated Model. The performance of these models were estimated and measured as the time needed to complete an operation. Tests were carried out with different network condition parameters. Network traffic was also measured. The steady state behavior and setup cost are compared separately. Simões et al. [16] concluded that for performance, as measured by time it takes to perform the management operation, distribution provides better results than locality when bandwidth is abundant. The gains of locality became more apparent when bandwidth was limited.

Current LTE architecture have already incorporated concepts of indoor small cells, HeNB, etc. [14] which have scaled up complexities for a cellular operator as they need to monitor performance of these exponentially multiplying network elements. Providing abundant bandwidth for North Bound (NB) interface or management interface for such new base stations would be an expensive task. So Operators would rather prefer limited bandwidth for NB interface. This means gains of locality would be considerable in that scenario as discussed in [16].

Network traffic can be reduced in uplink direction by processing and extracting out information needed from raw performance data. We did an analysis of this compression due to extraction of information at edge. Let's consider a single OSS application interested in performance management data of a base station, the data required to send in uplink from that base station can be given by expression

$$T_c = h + C \times i \tag{3}$$

where h is the overhead introduced by the communication protocol, such as headers, C is the number of raw counters needed to be transferred at once and i is the size of a counter value. For the proposed system this case could be expressed as

$$T_d = h + K \times i \tag{4}$$

where K refers to the number of computed KPI values. It can be assumed that K $\leq$ C, because a KPI is a extracted from one or more counters. This is the semantic compression provided by the system. Reduction in traffic volume depends on the complexity of the requested KPIs. A KPI can be a singular counter value so it does not compress in that case, while more complex KPIs can consist of dozens of counters and then the volume that needs to be transferred may be reduced to a tenth of the original size.

Network traffic is reduced in the downlink direction by the use of publish/subscribe pattern which omits the periodical requests of polling pattern and thus saves bandwidth in long run. The initial stream start request can be larger in size than a simple SNMP request, but it needs to be issued only once for a particular type of stream. For the centralized system the traffic generated would be for a one application

$$T_c = \sum (h + C \times i) \tag{5}$$

where C is the number of counters fetched, and i is the size needed to represent a request of a counter. Summation represents that the request needs to be sent before each response. Expression

$$T_d = (h + K \times i) \tag{6}$$

gives the required bandwidth in the decentralized case. Here K $\times$ i represents the size needed to request K KPIs of size i. The size i depends on the methods selected for storing and/or transferring KPI formulas. In case multiple applications are assumed, if they request same data, overlapping topics provide further gains for the publish/subscribe model.

Integration of the proposed concept into a real networks managements system is an interesting future research topic, which may propose needs for additional open standards. One promising future standard seems to be the Mobile Edge Computing standard. With Mobile Edge Computing (MEC) [2] being standardized we believe our Data Fetchers could be installed on MEC platforms connected to the base stations. This computation of information would not be computationally heavy for MEC platform as demonstrated in Sect. 5. The proposed approach could help operator to save precious bandwidth resources. Moreover reducing performance traffic at network edge by computing out and sending only information required would also decrease complexities which could otherwise arise in a centralized system which would then have to handle enormous volumes of raw performance data of entire network and then extract information required by any OSS application.

There are various other future directions to evaluate performance of proposed solution and tune it up according to telecommunication guidelines. For example, we are planning to evaluate our system performance with up to thousands of scattered Data Fetchers to find out its problems and limitations.

6 Conclusion

Ongoing digital evolution demands for scaling up of cellular network infrastructure with new network devices and technologies. This will complicate existing centralized monitoring of network performance due to increase in volume of M-Plane data. In this paper we discussed and evaluated a solution based upon local processing of required information from raw performance data at the network edge, near base stations. This information is then streamed on demand using publish-subscribe scheme across the network through our solution. Proposed solution helps in reducing M-Plane traffic both in uplink and downlink directions and reduces complexities of centralized processing of data. Current Evaluation of proposed solution gave encouraging results for its performance and reduction of M-Plane data.

References

1. Collectl. http://collectl.sourceforge.net/
2. ETSI-Mobile Edge Computing. http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing
3. NGMN use cases related to self organising network. https://www.ngmn.org/uploads/media/NGMN_Use_Cases_related_to_Self_Organising_Network_Overall_Description.pdf
4. 3GPP. Telecommunication management; Key Performance Indicators (KPI) for Evolved Universal Terrestrial Radio Access Network (E-UTRAN): Definitions. TS 32.450, 3rd Generation Partnership Project (3GPP), August 2008
5. 3GPP. Telecommunication management; Self-Organizing Networks (SON) Policy Network Resource Model (NRM) Integration Reference Point (IRP); Requirements (3Gpp. TS 32.521 version 9.0.0 Release 9). TS 32.521, 3rd Generation Partnership Project (3GPP), April 2010
6. Kafka, A. June 2016. http://kafka.apache.org/documentation.html,
7. Baldi, M., Picco., G.P.: Evaluating the tradeoffs of mobile code design paradigms in network management applications. In: Proceedings of the International Conference on Software Engineering, pp. 146–155. IEEE (1998)
8. Case, J., Fedor, M., Schoffstall, M., Davin, J.: RFC 1157: Simple network management protocol (SNMP) (1990)
9. Clemm, A., Wolter, R.: Network-Embedded Management and Applications: Understanding Programmable Networking Infrastructure. Springer, Heidelberg (2013)
10. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Comput. Surv. (CSUR) **35**(2), 114–131 (2003)
11. Goodhope, K., Koshy, J., Kreps, J., Narkhede, N., Park, R., Rao, J., Ye, V.Y.: Building linkedin's real-time activity data pipeline. IEEE Data Eng. Bull. **35**(2), 33–45 (2012)
12. Hertel, G.: Operation and maintenance. In: Hillebrand, F. (ed.) GSM and UMTS The Creation of Global Mobile Communication, pp. 445–456. Wiley, Chichester (2002)
13. Hämäläinen, S., Sanneck, H.: LTE Self-Organising Networks (SON): Network Management Automation for Operational Efficiency. Wiley, New York (2011)
14. Nossenson, R.: Long-term evolution network architecture. In: IEEE International Conference on Microwaves, Communications, Antennas and Electronics Systems, COMCAS, pp. 1–4. IEEE (2009)
15. Olsson, M., Mulligan, C.: EPC and 4G Packet Networks: Driving the Mobile Broadband Revolution. Academic Press, Amsterdam (2012)
16. Simões, P., Rodrigues, J., Silva, L., Boavida, F.: Distributed retrieval of management information: is it about mobility, locality or distribution? In: Network Operations and Management Symposium, NOMS. IEEE/IFIP, pp. 79–94. IEEE (2002)
17. TeleManagement Forum. Telecom operations map. Approved version 2.1. GB910. TeleManagement Forum, March 2000
18. Trossen, D., Reed, M.J., Riihijärvi, J., Georgiades, M., Fotiou, N., Xylomenos, G.: IP over ICN-the better IP? In: European Conference on Networks and Communications (EuCNC), pp. 413–417. IEEE (2015)