Magnet: Real-Time Trace Stream Analytics Framework for 5G Operations Support Systems

Sebastian Robitzsch, Faisal Zaman, Sven van der Meer, John Keeney, and Gabriel-Miro Muntean

ABSTRACT

The era of petabyte data has arrived as the digital big data universe continues its expansion toward exascale with massive volumes of data generated by diverse distributed sources. The size of big data makes it very difficult to gain insight into the meaning of data. In industrial applications, in order to explore both the meaning of data and the complex relationship between data components, big data needs to be processed and reduced enabling further deeper analysis in a timely manner. In this article an integrated data analytics framework is presented designed to extract the set of instances exhibiting statistical dependency from the massive volume of data in a pre-defined quasi real-time manner. The parallel computing model of MapReduce is enhanced to realize Magnet. The solution presented in this article is applicable to the telecommunications market where it optimizes next-generation network management systems for heterogeneous radio access technologies.

INTRODUCTION

The continuous increase in the number of mobile devices and users on one hand and advent of ubiquitous communication technologies, development of innovative networking applications and user demand for high quality rich media content on the other hand, are behind the latest massive amounts of data generated and exchanged, whose volume continues to grow at an exponential pace. The astounding increase in the volume of data generated is shown for instance by the fact that the amount of information created from the dawn of human civilization to 2003 (i.e., 5 exabytes) is now generated in just a couple of days. Additionally, the data digital universe has been experiencing a two-fold expansion every two years since 2012 so that the annual global IP traffic is expected to exceed the zettabyte (1000 exabytes) threshold by the end of 2016, and reach 1.6 zettabytes per year by 2018 according to forecasts by leading industry forums. This huge amount of information (lately labelled as the “data explosion,” derived from “big data”) leverages myriad opportunities for its analysis and use. However, such exploration poses significant challenges, mostly as this massive volume of data, represented by heterogeneous and diverse dimensionality data components, is too big for the processing capacity of the traditional analytical tools. So far, in order to be able to cope with this influx of big data, business enterprises constantly scale-up the analytic performance by employing incremental upgrades to existing solutions. However, these solutions have severe limitations, especially in terms of handling streams of big data transferred over large bandwidth networks and originating from multiple sources and therefore proposal of novel approaches is required.

Such big data is usually bursty in nature and can be handled by spreading the incoming data into multiple windows; this will also facilitate distributing the data processing. Alternatively, based on the data incoming rate, resources can be allocated dynamically [1]. Variability in the data is also caused by larger proportion of irrelevant, redundant and noisy information coming from various sources. This in turn makes very difficult the extraction of meaningful knowledge from the data within a limited time. Simple lightweight data projection techniques can serve the purpose of reducing the data dimension, but is capable of handling only limited volumes of data. Parallelizing the projection can solve the problem, but there is a possibility of higher approximation error. Event-based stream processing (ESP) is advantageous for reducing the number of events (which are distinct data instances labelled as such by the operators of ESP) with considerable low latency, but the scalability is constrained [2].

This article addresses these challenges by proposing Magnet, an integrated scalable data analytics framework designed to extract the set of instances exhibiting statistical dependency from large amounts of data in a given time period. Magnet consists of two major components: a load balancer and a data reducer. The load balancer dynamically balances the load of the incoming data via an arrival rate-based adaptive window solution. Once the data is loaded, the data reducer approximates the data by applying an on-the-fly parallel random projection technique and finding correlated instances. Correlated instances do not necessarily imply a pattern of instances, but make the task of pattern discovery more precise. The data reducer is designed to work in conjunction with MapReduce (MR) [3], a ubiquitous parallel computing paradigm to process large data. Basically, MR performs its tasks in batches with high latency, while Magnet’s low latency is the result of using micro-batches of the input data and pipelining the projection and correlation computation jobs.

Our research is focused specifically on the telecom industry as it has dealt with large
amounts network data for decades and is preparing to cope with big data, mostly due to the unprecedented rise in network control information generated by the next generation mobile networks. The purpose of Magnet is to serve specific business needs, making big telecom network data a “service” rather than a “technique” by integrating the analytical results to pattern discovery to enable predictions of the network scenarios and respective solutions. Magnet’s performance is tested on artificially generated telecom network trace data. The input consists of an online stream generated by the in-house developed emulator OpenMSC (available at www.github.com/serona-line/OpenMSC). The contribution of this work can be summarized as follows:

- Developed a parallel algorithm for scaled and refined approximation of data.
- Developed an analytical framework, Magnet, that is capable of handling streams of data.
- Evaluated Magnet on abstracting the symptomatic events leading to cell congestion from an artificially simulated trace file.

The principle of how Magnet works and where the MP design has been improved is described below. This is followed by describing the realization of Magnet in the E-Stream architecture section. The evaluation of the integrated Magnet solution is presented below followed by a discussion on how Magnet fits in the context of next generation operations support systems (OSSs). The work is concluded in the last section.

**The Magnet Algorithm**

The success of Magnet relies on its ability to scale the processing of massive data volumes, approximate the data with high accuracy and introduce a limited time delay. In Magnet, parallelizing the process of approximation and correlation computation is the source of scalability; replicating the parallelization process incurs minimal approximation error and adaptive slicing of the stream integrated with pipelining the processes facilitate stream processing.

The workflow of Magnet is presented in Fig. 1 and consists of three stages: dynamic load balancing, refined data approximation, and correlation finding. These three stages ensure low latency and low error, which are the pivotal features of a stream processing model [4]. In summary, Magnet is capable of:

- Rate-based diffusion of stream: Magnet splits the stream based on the arrival rate and spreads the split stream among smaller slices.
- Fault tolerant processing: Magnet replicates each slice of the split stream to maintain better approximation accuracy.
- Incremental processing: Magnet supports pipeline parallelism of the tasks.

Conventional data approximation techniques incur a significant number of distortions while projecting data points onto lower dimensions. In Magnet the data approximation (DA) was realized using MapReduce, where data partitioning and re-processing of each partition has been optimized, as described with further detail in [5]. This optimization involves an extension of the approximation process by processing each partition multiple times according to statistical theory. Consequently, the approximated results will be more accurate, which is why the chosen MapReduce approach can be described as enhanced. In Magnet, DA approximates a micro-batch of streams and starts correlation finding on that micro-batch rather than waiting for the whole stream-batch to be processed. Sequential processing is at the core of this hybridization of batch processing and real-time processing. This design fully serves our objective of maintaining low latency and high throughput processing. Spark-based stream processing is similar to the proposed approach, where the stream is split into discretized small batches but processed in-memory, which allows a more interactive and faster processing of batches by design. However, at the time E-Stream started, MapReduce still provided a much more stable environment compared to Spark, which was important toward a future integration into Ericsson’s xStream system [6]. Thus, MapReduce was favored over Spark.

**Dynamic Load Balancing**

The concept of dynamic load balancing is based on controlled processing of the incoming data stream. This entails feeding the processors with a manageable data volume, while the I/O is confronted with the event storm. In this mechanism the incoming data volume is controlled adaptively with the data arrival rate. In Magnet the data arrival rate (also defined as the stream burst rate) is estimated first, and based on that the window length is derived which is the volume of the incoming data stream, leading to seamless interaction between the streaming rate and the capacity of the processors (i.e., buffer size).

---

**FIGURE 1. Internal workflow of Magnet.**

Unprocessed stream

Jumping window

of size

\( N \) sec = \( V \) b/sec

Stream rate = \( V \) b/sec

Slice 1

Slice 2

Slice 3

Dynamic load balancer

DA 1

DA 2

DA 3

DA k

CF 1

CF 2

CF 3

CF k

Refined data approximator

Correlation finder

Correlated events

Slice 1
Refined Data Approximation

The purpose of refined data approximation (RDA) is to increase the efficiency of load balancing by further reducing the events, while maintaining lower estimation errors. RDA proceeds on-the-fly and therefore has no storage requirements. The theoretical basis of RDA is underpinned by the seminal Johnson-Lindenstrauss (JL) lemma [7].

In event-based streams, distinct events exist as inputs rather than continuous values and this requires a careful re-configuration of the approximator. Random hash-based indexing is utilized to project events to lower dimensions. To maintain the on-the-fly processing of the input events we simply recomputed the entries by Gaussian random hashing. Following the JL lemma and the law of large numbers, we repeat the on-the-fly approximation several times and parallelize the whole procedure to scale the processing performance. The underlying idea is to partition the data and run the data approximation several times on each partition; in this way we are replicating the projection several times to refine the approximation.

Data Intensive Correlation Finder

The correlation finder focuses on identifying any correlation between the data instances in order to capture the dependency relation between the instances. This correlative structure between the events can be detected by analyzing the eigen-values of large dimensional random matrices, and as a by-product the events with poor spectral condition can be filtered out. These events can be ignored as noisy events. This technique is more data intensive in the sense that it analyzes the eigen-space (spectrum) of the covariance matrix of the observed event set and identifies eigen-states coming from random noise using the known eigenvalue distribution of random matrices. This results in a decomposition of the covariance matrix: a part containing useful information from events with potential correlative structure and another part capturing the random noise. We designed the correlation finder to work with RDA in a pipeline parallel framework. In this framework the correlation finder starts processing the projected data from RDA before all the data is projected. A more detailed description and evaluation of the correlation finder can be found in [5].

Realization of Enhanced MapReduce

Magnet aims to constitute a modular system capable of discovering network events patterns by analyzing the telecom trace data and predicting network incidents from the event patterns to provide corrective actions. Accurate discovery of event associations requires designing a more intelligent data collection mechanism to extract useful information only. For responding to network incidents pre-actively, this information should be forwarded to the next working modules within a minimal time delay. In E-Stream, Magnet provides the functionality of smart and scalable data collection.

MapReduce (MR) is a special data-based programming model that has been established as a standard practice for processing large amounts of data in parallel. It includes two major phases: map, in which all the input data is transformed by a single argument in parallel, and reduce, in which all the transformed input data is grouped based on multiple arguments. The niche of MR usage in parallelization and scalability is the consequence of statelessness of the mapping algorithms (mappers) and independent processing of different reducing solutions (reducers). The executions of the tasks take place sequentially, which limits the scope of MR in terms of processing continuous large scale data (stream). If the data going through MR is kept reasonably small and processing is done incrementally, MR handles streaming data with reduced delay response. Following this observation, the basic workflow of MR was enhanced to enable Magnet processing of event streams by micro splitting the trace stream as described next.

Micro-Splitting of the Trace Stream

Stream micro-splitting starts with slicing the incoming data stream into small batches that are computed based on buffer size (processing power) and then the data collected in each batch is submitted to the MR. These small batches are part of the jumping windows in which the results of the batches are accumulated. We consider the deployment of jumping windows instead of the sliding windows as the latter requires more processing power to accumulate the outputs. The length of the jumping window is controlled dynamically according to the stream burst rate, and the size of the slices depends on the minimal data volume that should be handled before sending it to the reducer (which actually performs the processing). The advantage of such micro splitting of streams is that after receiving a specific time slice from every mapper, the reducer starts the combination process and merges the result with the previously merged results.

Incremental Processing

One of the sources of high latency in MR is that the commencement of one process needs to wait for the completion of the process started earlier. The enhanced MR includes a modification of the basic MR functionality to perform parallel process execution. Micro splitting the stream facilitates pipelined parallelism between the components of Magnet; this procedure can also be defined as incremental processing. In this improved solution, the reducer does not need to wait until the map phase finishes the task. The reducer needs to compute the aggregated slice value only after receiving the data corresponding to the same slice from all mappers. After this is performed, it
calls the user-defined merge() function to merge the slice results with the jumping window results.

**Integrating Magnet in E-Stream**

E-Stream aims to constitute a modular system capable of discovering network events patterns by analyzing the telecom trace data and predicting network incidents from the event patterns to provide corrective actions. Accurate discovery of event associations requires the design of a more intelligent data collection mechanism to extract useful information only. For responding to network incidents pro-actively, this information should be forwarded to the next working modules within minimal time delay. In the E-Stream context, Magnet provides the functionality of smart and scalable data collection, processing and pattern identification of any root-cause relationship possibly available in the trace stream. The presented framework follows the E-Stream system design, as described in [8, 9]. E-Stream defined the five modules:

- Dimension reduction module (DRM)
- Episode discovery module (EDM)
- Episode classification module (ECM)
- Pattern matching module (PMM)
- Recommender system module (RSM)

Magnet combines DRM, EDM, ECM and PMM in a single solution that can be directly mapped to the architecture defined in E-Stream. As depicted in Fig. 2, the trace stream is first minimized in spatial size to reduce the computational complexity of discovering episodes (sequence of EventIDs that indicate potential patterns of interest) in the next steps that are then further classified according to their correlation.

The illustrated pattern model library in Fig. 3 represents the linking storage element between the episode discovery and the future pattern model matching mechanisms akin to a commonly known relational database implementation.

**Evaluation**

**Emulation Set-up**

In order to test and evaluate Magnet, two high end 24 core servers were utilized with 128 GB RAM each running Ubuntu server 12.04. In order to guarantee a scalable environment where no single module is permitted to utilize all computational resources available on the physical machine, DRM, ECM, PMM and RSM were set up in a virtualized environment using kernel virtual machines (KVMs).

The integrated E-Stream test-bed is only as good as the environment in which it is presented. Without a real continuous stream of trace data, the implemented modules would not be able to be tested against their scalability and the real-time requirements. That is why OpenMSC was developed, which is a highly configurable C-based emulator that generates a stream of integer numbers, denoted as EventIDs, that represent the control plane communication data of a telecommunication network. OpenMSC allows specifying a single success use case, e.g., the set-up of a call from a mobile phone, and an arbitrary number of failure use cases (see the Appendix for a detailed message sequence chart). Each use case is referred to as a communication description with communication descriptors being represented by integer numbers. More precisely, a descriptor denotes a single primitive exchange between a source and a destination NE and is part of an entire use case. Multiple communication descriptors are referred to as a communication description (the entire use-case). In comparison to discrete or continuous event simulators that produce a sequence of events using an internal simulator clock to model the system, OpenMSC generates each EventID based on the actual system time. All communication descriptors are specified in an MSC file that follows the same terminology as the mscgen tool (available at http://www.mcternan.me.uk/mscgen) to plot MSCs. The concept of OpenMSC is based on the assumption that each user equipment (UE) in the network follows the same control plane procedures, as specified in advance through a MSC configuration file. Additionally, each UE repeats this action within a given time-frame. OpenMSC lets the researcher set parameters such as number of UEs and number of base stations (BSs) in the network. As mentioned, a particular communication descriptor is represented as a single numeric EventID due to the requirement of most data mining algorithms that are able to work with numbers only. Therefore, OpenMSC translates each communication descriptor entity into a single unique integer representation, then concatenates these representations in a standardized way, as shown in Table 1.

Each communication descriptor has a source NE and a destination NE that exchange information using primitives that are standardized by 3GPP. As primitive names can be the same across multiple protocols, OpenMSC distinguishes between the protocol type and primitive name. Each primitive has various information elements (IEs) that hold particular values. The total length of the generated EventID is 19 due to the maximal length of the data type unsigned long long in all modern programming languages. The source and destination NEs are represented by a five digit long integer values. The integer representations are being allocated on an iterative basis where an
unknown NEs will receive a number that is incremented by one, when compared to its predecessor NE. The only exceptions are UEs and BSs, as they are treated differently by OpenMSC. In the case of a BS, OpenMSC calculates a BS EventID \( ID_{bs} \) by:

\[
ID_{bs} = 100 \cdot BS(m)
\]  

with \( m \in Z \) and \( m > 0 \). If the total number of BSs, \( n_{bs} \), equals 50, the five digit long numerical representation of the second BS \( (m = 2) \) is 00200. The numerical UE representation is calculated as follows:

\[
ID_{ue} = 100 \cdot BS(m) + UE(n)
\]  

with \( m = n \in Z \) and \( m > 0 \) for instance, let \( m = 2 \) and \( n = 45 \), the corresponding UE identifier \( ID_{ue} \) would be 00245. All other NEs receive a unique integer in the range of 1 through 99. This ensures that the five digit long numerical representation of a NE is always unique.

The only limitation to generating the EventID as described above is the total number of NEs, primitives and IEs that can be used in the entire emulation while ensuring a unique integer representation for every piece of information. However, with 999 BSs, 99 UEs at each BS, and a further 99 NEs (not BS or UE), it is ensured that OpenMSC still provides enough flexibility to generate data-streams emulating rather large networks.

The MSC used for this example is shown in the Appendix, which comprises the two communication descriptions, Success and Failure, which consist of nine and two communication descriptors, respectively. There are three NEs chosen, and two protocol types, i.e., radio resource protocol (RRC) and S1 application protocol (S1AP). The configuration used in OpenMSC to generate the data stream in the testbed is also given in the Appendix. When providing the two input files, as given in the Appendix, OpenMSC generates an EventID rate per second of approximately 210. The stream is filled up with random EventIDs representing noise; the generation of noise EventIDs follows a normal distribution.

The numerical UE representation is calculated as follows:

\[
ID_{ue} = 100 \cdot BS(m) + UE(n)
\]  

with \( m = n = 45 \), the corresponding UE identifier \( ID_{ue} \) would be 00245. All other NEs receive a unique integer in the range of 1 through 99. This ensures that the five digit long numerical representation of a NE is always unique.

The only limitation to generating the EventID as described above is the total number of NEs, primitives and IEs that can be used in the entire emulation while ensuring a unique integer representation for every piece of information. However, with 999 BSs, 99 UEs at each BS, and a further 99 NEs (not BS or UE), it is ensured that OpenMSC still provides enough flexibility to generate data-streams emulating rather large networks.

The MSC used for this example is shown in the Appendix, which comprises the two communication descriptions, Success and Failure, which consist of nine and two communication descriptors, respectively. There are three NEs chosen, and two protocol types, i.e., radio resource protocol (RRC) and S1 application protocol (S1AP). The configuration used in OpenMSC to generate the data stream in the testbed is also given in the Appendix. When providing the two input files, as given in the Appendix, OpenMSC generates an EventID rate per second of approximately 210. The stream is filled up with random EventIDs representing noise; the generation of noise EventIDs follows a normal distribution.

unknown NEs will receive a number that is incremented by one, when compared to its predecessor NE. The only exceptions are UEs and BSs, as they are treated differently by OpenMSC. In the case of a BS, OpenMSC calculates a BS EventID \( ID_{bs} \) by:

\[
ID_{bs} = 100 \cdot BS(m)
\]  

with \( m \in Z \) and \( m > 0 \). If the total number of BSs, \( n_{bs} \), equals 50, the five digit long numerical representation of the second BS \( (m = 2) \) is 00200. The numerical UE representation is calculated as follows:

\[
ID_{ue} = 100 \cdot BS(m) + UE(n)
\]  

with \( m = n \in Z \) and \( m > 0 \). For instance, let \( m = 2 \) and \( n = 45 \), the corresponding UE identifier \( ID_{ue} \) would be 00245. All other NEs receive a unique integer in the range of 1 through 99. This ensures that the five digit long numerical representation of a NE is always unique.

The only limitation to generating the EventID as described above is the total number of NEs, primitives and IEs that can be used in the entire emulation while ensuring a unique integer representation for every piece of information. However, with 999 BSs, 99 UEs at each BS, and a further 99 NEs (not BS or UE), it is ensured that OpenMSC still provides enough flexibility to generate data-streams emulating rather large networks.

The MSC used for this example is shown in the Appendix, which comprises the two communication descriptions, Success and Failure, which consist of nine and two communication descriptors, respectively. There are three NEs chosen, and two protocol types, i.e., radio resource protocol (RRC) and S1 application protocol (S1AP). The configuration used in OpenMSC to generate the data stream in the testbed is also given in the Appendix. When providing the two input files, as given in the Appendix, OpenMSC generates an EventID rate per second of approximately 210. The stream is filled up with random EventIDs representing noise; the generation of noise EventIDs follows a normal distribution.
real-time classification of discovered episodes in the previous step is presented here. The classification of all discovered episodes of all lengths utilizes the graph theory theorem of building an acyclic unidirectional tree of episodes. The top level episodes (pattern models) are the top of the pyramids in Fig. 5a, while the next level of episodes (partial patterns) of the acyclic graph are displayed below and connected with a black line.

The discovered pattern model tree is then populated in the pattern model library, which is the input to the pattern matching task. It can be reported that when scaling up the stream rate, the processing of thousands of episodes in order to build the pattern tree was always finished in real-time utilizing. It can be concluded that all pattern models and their corresponding partial patterns were discovered by Magnet.

**Pattern Matching**

This section presents the last step of the Magnet framework: the matching of the discovered pattern models in the online trace stream. The objective is to match and predict the pattern models from the reduced event streams on the basis of the pattern model information provided by the pattern model library. The matching approach is evaluated using traces of events drawn from OpenMSC; the E-Stream emulator generates a continuous stream of mobile telecom control plane communication events. For visualization purposes, the matching function is designed to show matched and predicted pattern models only together with their corresponding first-level partial patterns. The prediction itself is based on the assumption that if a non-top level pattern model has been found, it must have been actually there and was probably only lost due to the slicing of the stream Magnet must undertake. The visualizer, depicted in Fig. 5.b, illustrates orange squares for matched pattern models, pale blue squares for predicted pattern models, dark blue squares for pattern models available in the pattern model library and dark blue circles for Level 1 pattern models.

**Discussion**

This last section aims at putting the conducted research into perspective, addressing questions around applicability, scalability and ease of integration into existing OSSs. For this discussion the main objective of Magnet is of significant importance, i.e., the system should be agnostic to the use case in which it should discover root-cause relationships of existing and potential misbehaviors. With this objective in mind, Magnet (the final solution of the E-Stream project) is a fundamental step forward, as part of Ericsson’s global R&D initiative in the area of management of next generation OSSs, being ahead of the state of the art in this space. Magnet, as result of a collaborative Dublin City University-Ericsson R&D effort, complements Ericsson’s industrial-driven xStream [6] approach, which is currently embedded in a new enterprise network management product. While the existing Magnet system as a research prototype might run into performance issues, Ericsson has viable software that can use and deploy the Magnet ideas in a commercial, scalable and high-performance environment. The related Ericsson works [6, 10, 11] focus on real-time evaluation of large-scale data streams demonstrating that Magnet’s objectives of designing a solution that is agnostic to the actual stream’s content while persistently discovering root-causes is feasible. To this extent, the choice of MapReduce as the agnostic stream processing platform on which Magnet has been realized can be explained by a rather pragmatic software engineering viewpoint which prefers platform stability and integrability as long as the overall objectives can be met.

**Conclusion**

This article introduces an innovative integrated scalable data analytics framework Magnet proposed for extracting the correlative structure between events (data instances). One of the key functionalities of Magnet is to reduce data through a refined approximation process that is based on randomization and rough preservation of the statistical relationship between data instances. The approximation process is applied to multiple copies of partitioned data in order to support both increased scalability and high accuracy of the data approximation process. This also results in faster correlation computation. Further scalability is introduced by means of pipelining the approximation and correlation computation processes. The Magnet framework is realized in the MapReduce model proposed for handling massive amounts of data. Fundamental enhancements are proposed for MapReduce to handle pipelining of the two processes and process con-

---

**FIGURE 5.** a) Pattern tree of classified episodes; b) visualisation of the matched and predicted pattern models in the data stream.

---

From an application point of view Magnet enhances the effectiveness of the algorithms looking for patterns in data by providing only the correlated instances. The other aspect of Magnet is that it can also be implemented in any stream computing model for building a real-time stream analytics system.
Continuous flows of big data (streams). Magnet evaluation was performed and the experimental results show that the framework is capable of reducing a significant portion of the large input data stream and at the same time keeping the data fragments with potential correlative structure. Removing the bulk of the events through randomization and then keeping the set of events linked through statistical or temporal dependency, has enabled the reduction at such scale. The processing time of the framework is highly encouraging and recommends its utilization as a continuous analytics system. From an application point of view Magnet enhances the effectiveness of the algorithms looking for patterns in data by providing only the correlated instances. The other aspect of Magnet is that it can also be implemented in any stream computing model for building a real-time stream analytics system.

**Appendix**

For completeness of this publication, the authors provide the configuration file of OpenMSC so that readers interested in following up can benchmark their solution against E-Stream:

```json
openmscConfig: {
  numOfBss = 1;
  numOfUserPerBss = 10;
  ueActivity-Dist = "exponential";
  ueActivity-Dist-Lambda = 0.2;
  cdOverlap = false;
  informationElements = ( {
    ieName = "SIRErrorValue";
    ieDist = "gaussian";
    ieDistMu = "80.0";
    ieDistSigma = "5.0";
  },
    { ieName = "ErrorCode";
    ieDist = "constant";
    ieDistValue = "1";
  });
  noise = {
    uncorrelated = ( {
      distOccurrence = "uniform_real";
      distOccurrenceMin = "0.001";
      distOccurrenceMax = "0.01";
      eventIdRangeMin = "1";
      eventIdRangeMax = "500";
    });
  };
}
```

Furthermore, the MSC used as the second required input file for OpenMSC is also provided in Fig. 6.

**Acknowledgment**

The authors would like to thank (in no particular order) Zhuo Wu, Anderson Simicsuka, Gabriel Hogan, Dr. Zhiguo Qu, Dr. Zhenhui Yuan, and Dr. Conor McArdle for their contributions. The support of the Enterprise Ireland Innovation Partnership with Ericsson is gratefully acknowledged.
References

Biographies
Sebastian Robitsch has been a postdoctoral researcher at Dublin City University, Ireland, for two years, leading E-Stream, a nationally funded collaborative project with Ericsson Ireland in the area of real-time data mining OSS solutions in mobile networks. In the past he has been with T-Systems, Germany; Fraunhofer FOKUS, Germany; and Nokia Research Centre, Finland, working on research issues ranging from interference and self-configuration techniques in 802.11-based multi-antenna mesh networks, heterogeneous radio access networks to system architecture design for trace analytics and recommender systems for next-generation OSSs. His recent research efforts focus on the softwarization of network functions following existing SDN and NFV paradigms in order to allow a decoupling of infrastructure, service and content providers for a more versatile communication network. How to bring SDN and NFV to the monitoring task of large-scale networks has been one of his more recent proposal efforts. He received his Ph.D. from University College Dublin, Ireland, in 2013, and an M.Sc. equivalent (Dipl.-Ing. (FH)) from the University of Applied Sciences Merseburg, Germany. Currently, he is with InterDigital Europe working on proof of concept realizations of 5G next generation networks funded by the Horizon 2020 scheme.

Faisaz Zaman is an associate manager at Accenture. Previously, he was with Adaptive Mobile Security and a post-doctoral researcher with the Performance Engineering Laboratory and Network Innovations Centre, Rince Institute, Dublin City University, Ireland. He received his Ph.D. in information science from Kyushu Institute of Technology in 2011. In his previous tenure as a post-doctoral researcher at the Kyushu Institute of Technology, he analyzed time series data for weather forecasting and micro-array data for gene classification. He also worked as a statistical programmer at Shaft Consultancy Ltd., leading teams to analyze medical trial data. He is program committee member of several data mining conferences. He has published 30 articles, conference proceedings, books, book chapters, conference papers, and technical reports. He has experience in supervising Ph.D. and M.Sc. level students.

Sven van der Meer received his Ph.D. in 2002 from Technical University Berlin. He joined Ericsson in 2011, where he is currently a master engineer leading a team that will enhance the capabilities of Ericsson’s OSS products. Most of his current time is dedicated to designing and building advanced policy and predictive analytics systems. In the past, he has worked with Fraunhofer FOKUS (Berlin, Germany), Technical University Berlin (Germany) and the Telecommunication Software and Systems Group (TSSG, Ireland), leading teams and projects, consulting partners and customers, and teaching on the university level. He is actively involved in the IEEE CNOM community as a standing member of programme committees (IM, NOMS, CNMS, and APNOMS among others) and has helped to create and organize successful workshop series (MACIE, MUCS, and ManFed. Com among others). He has also contributed to standardization organizations, namely the OMG and the TM Forum. He has published more than 100 articles, conference proceedings, books, book chapters, conference papers, and technical reports. He has supervised and evaluated six Ph.D. candidates and more than 30 M.Sc. students.

John Keeney is a senior researcher at LM Ericsson Ireland, working in the Network Management Laboratory in Ericsson’s Software Research Campus in Athlone. His research focus is on monitoring and managing complex systems, especially telecom systems, with a particular focus on knowledge extraction, event stream processing, and performance analysis. His work at Ericsson centers on online analytics and optimization of radio access network performance to inform the next-generation operation system support (OSS) concepts for Ericsson’s OSS product unit.

Gabriel-Miro Muntean received the Ph.D. degree from Dublin City University, Dublin, Ireland for research in the area of quality-oriented adaptive multimedia streaming in 2003. He is an associate professor with the School of Electronic Engineering at Dublin City University (DCU), Dublin, Ireland and co-director of the DCU Performance Engineering Laboratory. He was principal investigator at E-Stream, a collaborative project with Ericsson Ireland in the area of real-time data mining OSS solutions in mobile networks. His research interests include quality-oriented and performance-related issues of adaptive multimedia delivery, performance of wired and wireless communications, energy-aware networking, and personalized technology-enhanced learning. He has published over 300 papers in prestigious international journals and conferences, authored three books and 18 book chapters, and edited six other books. He is an associate editor of IEEE Transactions on Broadcasting, an editor of IEEE Communications Surveys and Tutorials, and a reviewer for other important international journals, conferences, and funding agencies. He is a senior member of IEEE and the IEEE Broadcast Technology Society and coordinator of the EU Horizon 2020 project NEWTON (http://newtonproject.eu).