Model-Driven Analytics in SDN Networks

Mouli Chandramouli
Cisco Systems, Bangalore, India
moulchan@cisco.com

Alexander Clemm[1]
ludwig@clemm.org

***Abstract*— Analytics of network telemetry data is useful for addressing many important network operational problems. While Big Data techniques have been pushing scale boundaries for processing data ever further, in many cases the real bottleneck for analytics is the acquisition, i.e. the ability to generate and export the data on which analytics depends. To address this issue, we have earlier introduced DNA, a framework for Distributed Network Analytics that pushes analytics processing into the network and dynamically sets up data sources as needed. One of the challenges of such a framework concerns providing users with simple ways to articulate network analytics queries and instruct the network which data to generate and provide. We have addressed this issue using a model-driven approach that is presented in this paper. Using YANG as a way to model network analytics tasks, our system lets users articulate network analytics tasks at a very high level of abstraction that is subsequently broken down by the framework into lower-level analytics tasks which are deployed across the network.**

Keywords—Network Analytics, Service Assurance, SDN, YANG

I. INTRODUCTION

Network devices generate a rich set of telemetry data, such as flow records, MIB data (such as interface counters), service-level measurements, and system event (syslog) records. Network telemetry data facilitates many operational tasks, from monitoring networks and services for anomalies, trends, and changes, to diagnosing subtle causes for intermittent faults and performance degradations. It therefore comes as no surprise that network service providers have significant interest in acquiring and analyzing network telemetry data to obtain near-time visibility of the network health.

A common approach to network analytics involves centralized, possibly cloud-based systems. With this approach, data is collected from the network in a central place and subsequently analyzed. Networks and networking services can easily involve tens of thousands of devices, resulting in substantial amounts of data. Much of the focus has been accordingly on scaling the processing of that data, leading to the development of systems and algorithms commonly associated with "Big Data" such MapReduce, Hadoop, Hive, or Pig.

There are several hidden assumptions with a centralized network analytics approach. Most importantly, its effectiveness depends on the ability of the network to provide the required raw data in the first place. However, this can turn out to be a formidable task, as generating and exporting that data is resource-intensive and subject to bandwidth constraints. The factor that dominates performance and scale lies in many cases in the generation and export of the required data records, not analytics processing itself. A practical bottleneck lies furthermore in the ongoing system management tasks that are involved with setting up networking devices for analytics tasks as needed and keeping up with dynamic network changes.

To address those challenges, we have introduced a novel framework for distributed analysis of network telemetry data, Distributed Network Analytics (DNA) [5]. DNA consists of DNA Agent, an embedded analytics application that is deployed with networking devices, and DNA Controller, a Software Defined Networking (SDN) control application [12][9] that orchestrates analytics tasks across the network. Contrary to traditional analytics solutions that bring data to the processing, DNA brings processing to the data and in the process also takes care of system management tasks such as dynamically managing telemetry sources as needed.

One aspect that needs to be addressed for a network analytics framework like DNA, or for any network analytics application, concerns the articulation of network analytics queries. Our requirements included the ability to let users articulate new analytics queries in simple fashion that can be accomplished by a network administrator without requiring analytics code development skills. It also included the need to deal with heterogeneity of network devices. Different devices may expose the same data in different ways, for example involving different show commands or MIB or YANG objects. Some data may be supported by one device but not another, such as service level measurements To the user, network device heterogeneity differences should be hidden as much as possible and allow the user to articulate network analytics tasks holistically across the network, instead of needing to deal with the network one device or one type of device at a time. This means that users need to be able to apply operations across the network which are understood and supported by as many devices as possible, ideally all of them. Short of that, users would have to revert to manage networks one device at a time. Alternatively, SDN controllers would need to degenerate into legacy network management systems that require development of device-specific adapters for each device, resulting in systems that are unwieldy, slow, and costly to sustain.

We addressed these requirements by making analytics tasks model-driven. Categories of analytics tasks are modeled as *analytics templates* that can be customized by users. The translation of templates to actual analytics processing is

[1] Alexander Clemm is with Huawei Technologies, Santa Clara, California/USA. However, the paper is based on efforts while he was still at Cisco

maintained in a set of *bindings*, specifically *maplet bindings* that define the translation for processing to be performed in the device. The prerequisites that a system needs to support in support of a query are specified through *capabilities*. Like a policy, the same capability may be rendered differently by different devices to shield users and controller from device differences. In our system, analytics templates, template bindings, and capabilities can all be specified via data models, in addition to the data itself that is being subjected to analytics.

We chose YANG [3] as data modeling language to facilitate analytics configuration through Netconf [8] or Restconf [2] and ease of integration with SDN frameworks. At the same time, data that is subjected to analysis does not necessarily have itself to be described via YANG but can be incorporated by proper references. In fact, the vast majority of data sources supported by DNA concern non-YANG defined data, such as MIB objects of information elements in flow records.

In this paper, we will present our model-driven approach to distributed network analytics in greater detail. Section 2 will provide some additional background on DNA. Section 3 gives an overview of the analytics data model and its various components. Section 4 describes the YANG data tree that is basis for our implementation. Section 5 gives an outlook and provides conclusions. Please note that this paper should not be construed as providing any indication regarding product direction and no such inferences should be made.

II. BACKGROUND

A. YANG and Netconf

YANG is a standards-based, extensible data modeling language to model network device configuration and operational data. That data provides contents used by Netconf operations, remote procedure calls (RPCs), and server event notifications. Among the goals of YANG are a highly readable data model that supports the definition of data hierarchies, facilitates the validation of configuration data before it is applied, and promotes model reuse. The data that is described by a YANG model data model is conceptually contained in a data store and can be instantiated as XML or JSON.

YANG was originally defined as the basis to provide interoperable data in conjunction with Netconf. However, more recently YANG has been used also as the basis for model-driven controller architectures and SDKs that generate APIs from model definitions, notably in the case of Open Daylight [11], an open source SDN controller platform of the Linux Foundation.

B. Distributed Network Analytics - DNA

Distributed Network Analytics is a distributed solution framework that provides a network analytics on the basis of network telemetry data, such as flow records, device statistics represented via MIB objects, or IPSLA service level measurements [4]. DNA is specifically targeted at operational use cases that have near-real time characteristics, such as monitoring a network for changes, trends, and anomalies.

The DNA architecture consists of two components, as depicted in Figure 1:

- A DNA Controller, an SDN application running on top of an SDN Controller Framework such as Open Daylight, that orchestrates network analytics tasks across the network, collates the results reported from DNA Agents, and provides a single point of entry for users of the Network Analytics Service.
- A DNA Agent, an embedded application running on each network element, configures underlying telemetry data sources and performs analytics on the resulting telemetry data streams.

The DNA solution is at its core a network-embedded management application [7] and offers important advantages over centralized analytics solutions. Not only is the amount of required off-box processing reduced, but CPU and bandwidth within the network are conserved as well: additional cycles for analytics performed in the device are offset by the avoidance of cycles that would otherwise be required to generate, format, and transmit data that is not required for the actual task at hand. Likewise, management of analytics tasks themselves is greatly facilitated. Perhaps most importantly of all, DNA Agent configures the data sources as needed. This allows to dynamically adapt what data is generated when it is needed for a specific analytics task, which is important because in many cases there are practical limits to the amount of telemetry data that a device can generate, for example with regards to service level measurements involving synthetic traffic or very high sampling rates of interface statistics.

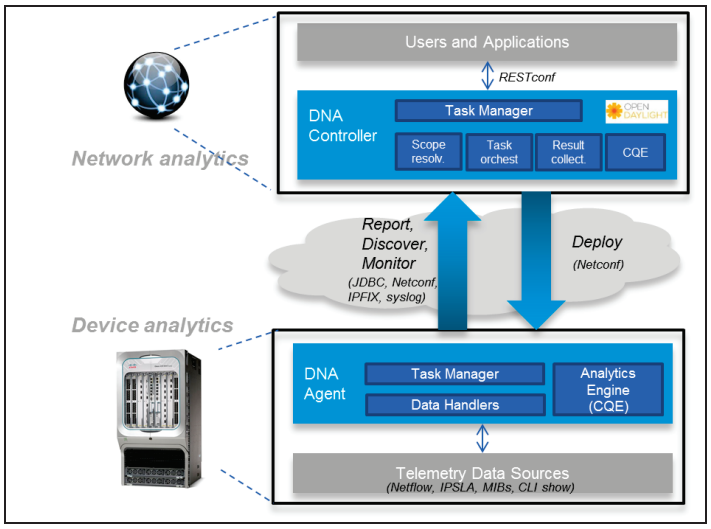

Figure 1: DNA Architecture

III. ANALYTICS MODEL OVERVIEW

In this section, a data model for model-driven analytics in DNA is presented. Our analytics data model applies at two levels, network controller and network device. At the network controller level, the model serves as the basis of interaction between the network analytics service provided by the controller and the end user for articulating the analytic queries. At the network device level, the model serves as the basis of

interactions between controller and device. As a result, there are actually two models – a network analytics model, and a device analytics model. Because a network analytics task is ultimately broken down into a set of device analytics tasks, the models are clearly interrelated, but they are not the same and need to address different requirements.

We first provide a summary of the requirements that the model had to address. Subsequently we provide an overview of the model components and explain how the requirements are addressed by the model.

A. Model Requirements

DNA needs to meet a number of important requirements, which the data model has to facilitate. Most importantly, it has to allow to apply analytics to different sources and types of network telemetry data. This includes data that is defined in YANG itself and exposed (for example) via a Netconf or Restconf server. However, it also needs to cover data that is available only via different formats and interfaces, including MIB objects, CLI show command output, Netflow and IPFIX records, and IPSLA service level measurements. It also needs to be expansible to cover other interfaces in the future, such as syslog messages, IoT sensor data streams, and packet captures.

At the network controller level, the model needs to be very easy to use and facilitate simple interactions between users (or client applications) and the system. Specifically, the model has to be able to encapsulate predefined analytics use cases and allow for their customization, so that users can adapt the model to their specific situation with minimal programming.

While the model needs to shield users from the need to specify detailed analytics logic, it has allowed for the dynamic introduction of new types of analytics tasks and queries by experienced users, such as consultants of a services organization or experienced network operations personnel engaging in DevOps methodology [1].

At the device level, DNA Agents need to accommodate a wide range of telemetry sources, including but not limited to MIBs, CLI show output, Netflow records, active service level measurements such as provided through IPSLA, as well as YANG-defined objects themselves. In addition, DNA Agents must be able to support a variety of analytics engines as well as result exports, requiring analytics queries and result data to be described in ways to support different formats.

B. Model Components

The data model at the controller consists of the components that are depicted in Figure 2. Broadly speaking, the model distinguishes between aspects that define how network analytics tasks can be configured, and aspects that represent instantiated analytics tasks during runtime. Each of those aspects is further defined in the following sections. How those components are represented using YANG is described in Section IV.

a) Network Analytics Task Template

A *network analytics task template*, or simply *analytics Template,* is a pre-canned analytics query with predefined semantics, which does allow for some degree of customization by a user. The idea is to have the DNA Controller provide a library of predefined Analytics Templates that users can choose from, and allow the users to make certain customizations to those queries when they request an analytics task. This enables network operators to articulate new network analytics tasks in simple fashion without requiring specialized analytics software development skills or complex testing of analytics logic. Customizations can affect aspects such as which data items to subject to a query or which of a set of aggregators to apply to a set of data items, but they do not affect the logic flow per se (which would require programming).

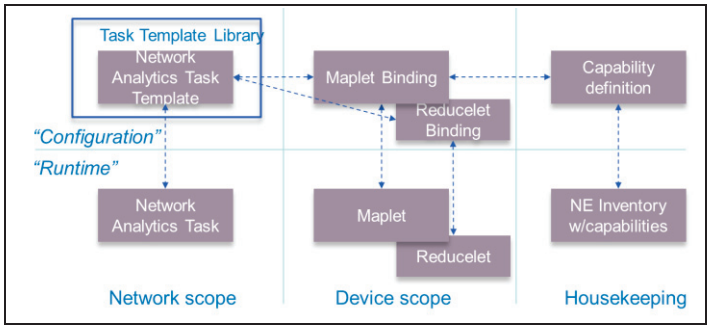

Figure 2: Network Analytics Model Components

b) Network Analytics Task

A *network analytics task* represents a task requested by a user. It instantiates an analytics template with a set of parameters as applicable. It also includes scheduling information (as tasks will typically be running for longer durations of time) and a *network scope*, i.e. a policy regarding which devices in the network will be subjected to the task, such as an enumerated set of devices in a list, all devices of a certain type, or all devices tagged with a certain property.

c) Maplet (and Reducelet)

A *maplet* represents the task that is deployed and run at a device. The maplet defines the analytics query, data streams that are to be subjected to the query, as well as additional information needed to configure sources of the data streams. In addition, the maplet can include scheduling information.

The information contained in the maplet needs to be sufficient to render the data stream. For example, if the data stream involves flow records, the stream configuration needs to include parameters such as flow expiration timers. Similarly, if the data stream involves IPSLA service level measurements, the stream configuration needs to include information that characterizes the test traffic, such as the number of test packets and probing intervals.

A *reducelet* defines an analytics query that is used to correlate result streams sent by DNA Agents to the DNA Controller. In many cases, the reducelet will in effect be an identity query. For example, when the purpose of a network analytics task is to monitor certain conditions inside network elements which when met are reported back to the user, no further analysis is required at the controller. However, certain network analytics tasks may involve comparing results, such as

when the severity of conditions needs to be ranked across devices.

d) Maplet (and Reducelet) bindings

To run an actual Analytics Task that instantiates an Analytics Template, the query that is implied as part of an analytic template needs to be translated to an actual analytics query respectively device analytics task that can be deployed to the device. This means that for a given Analytics Template, rules for how to generate a corresponding query (that can be fed to an analytics engine) need to be defined which can be interpreted at runtime. In addition, rules need to be defined for how the parameters with which the user can customize the query map into that query. We refer to the corresponding mapping rules of an Analytics Template as a *binding*.

In addition to the mapping rules and the Analytics Template that it supports, a binding may also contain a list of prerequisite capabilities that need to be supported by DNA Agents for the generated maplets to be run. An example of a prerequisite capability is the support for a specific analytics engine or for a specific type of data source. It is therefore conceivable that several bindings for the same Analytics Template exist, providing alternative mappings dependent on supported capabilities. For example, low footprint devices might support only basic analytics processing capabilities, whereas higher footprint devices might support advanced analytics processing capabilities that could be optimized in different ways. Similarly, with evolution of technology landscape, there may be new analytics engines to take advantage of in the future. Instead of defining Analytics Templates with a specific engine in mind, the ability to define separate bindings allows to separate the concerns of how to present an analytics task to the user, from how to map it.

Bindings are maintained and interpreted at the controller to render a maplet that is then deployed at the DNA Agent. An alternative design would have been to maintain bindings at the DNA agent and render the maplets locally there. In that case, DNA Agents would need to become Analytics Template aware and allow to dynamically deploy bindings to DNA Agents, either as part of the query or through a separate control flowThe DNA Controller would in that case not deploy analytics tasks containing analytics queries to the DNA Agent, but analytics tasks requests would simply contain the instantiated template.

e) Capability Definitions and NE Inventory

Networks are heterogeneous in nature. Analytics Tasks will therefore have to be able to run across a multitude of devices. Those devices may differ in terms of what telemetry data sources they support. Some telemetry sources may not be supported everywhere – for example, IPSLA will be supported on some but not all devices. Some devices support Netflow v9 or IPFIX, others support only Netflow v5. Even more importantly, the data offered across the same type of telemetry source will differ. Some MIBs may be supported on some devices but not on others. Sometimes, the same data is supported on different devices, but resides in different MIBs.

The value of DNA lies in no small extent in its ability to support operational scale in a diversity of platforms. Users can articulate a single query and the query should be deployed across the network, regardless of which or how many devices need to participate in the query. Differences between devices affect the ability to deploy an analytics task across the network, inconveniencing users and potentially reducing DNA's effectiveness. Where a single analytics request might have sufficed, multiple analytics requests would now be required, one for each set of devices that supports a given functionality. In addition, multiple bindings or even Analytics Templates need to be maintained.

Homogenization of manageability interfaces goes beyond the scope of DNA. However, DNA can mitigate the impact of heterogeneity. This is achieved through the concept of *capabilities*.

A *capability* refers to a set of functionality with pre-established meaning that is supported by a DNA Agent. Capabilities can refer to specific data sources (e.g., IPSLA, Netflow v5, Netflow v9, SNMP MIBs), to specific sets of data (e.g., interface-stats, bgp-stats), or even to analytics capabilities (e.g. Spark-Streaming, Storm). DNA Agents are aware of the capabilities they support and announce those capabilities to the Controller.

The functionality that a capability refers to is in many cases pre-established. For example, there is a predefined list of capabilities that refers to data sources and analytics capabilities. However, in the case of capabilities that refer to sets of data supported by a device, new capability definitions can be dynamically added. For this purpose, new capability definitions can be introduced that include a list of *named data items* that specify particular data items and their data types that can be subjected to analytics. The rendering of named data items to specific data sources is up to the DNA Agent that supports the capability. For example, a capability "Interface_Stats" might include a set of named data items such as "inOctets" and "inDiscards", which a DNA Agent might map to a set of MIB objects as defined in RFC 2863 [10]. This is a pure convenience function that allows users and controllers to reference frequently analyzed data items by a common name. Alternatively, data items can be referenced by identifiers as implied by the data source.

DNA Agents are aware of the capabilities that they support. Information about supported capabilities is maintained as part of the DNA Controller's network inventory. This way, when an Analytics Task is requested, the DNA Controller can select matching bindings and validate whether prerequisite capabilities are supported by DNA Agents that are within the task's scope, as well as handle interactions with users when they are not and a requested analytics task will be degraded accordingly.

C. Model usage

Figure 3 depicts how the model components are used to collectively drive the deployment of an analytics task across the network. First, a user defines a network analytics task by selecting a template from the controller's template library. The user customizes the task with parameters such as which data

items to subject to the task (for example, interface or BGP or ACL statistics), which one of a set of trending functions to select, or which percentile value to compare a dynamic threshold against. Subsequently, the controller needs to generate a set of maplets to deploy to the network devices within network scope. For this purpose, the controller checks the template bindings. Where more than one template binding exists, the controller selects one, for example based on the capabilities that are supported by the device in question. Subsequently, maplets are deployed.

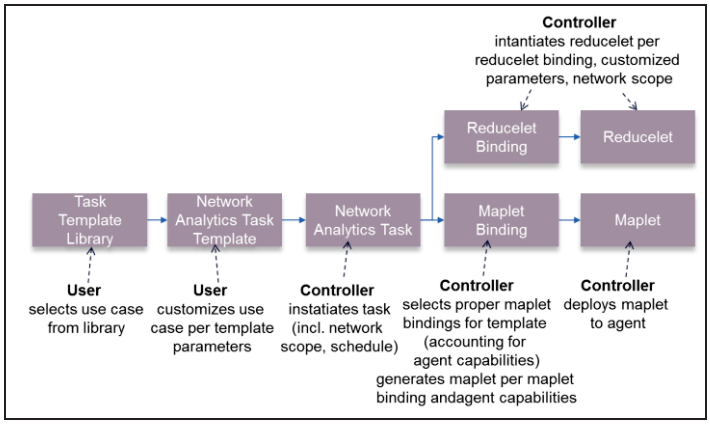

Figure 3: Use of the model to drive an analytics task

IV. YANG DATA MODEL

The following section provides an overview of how the model is mapped into YANG, and why YANG was chosen as data definition language in the first place.

A. *The Rationale for YANG*

YANG was chosen as data definition language for a number of reasons that include the following:

- YANG is readily supported by controller frameworks, such as Open Daylight (foundation for DNA Controller) or Network Service Orchestrator [14], as well as agent frameworks, such as YumaPro [15] or Cisco's ConfD. This facilitates implementation on both agent and controller.
- YANG is readily supported by Netconf and Restconf, making the resulting analytics model easily accessible for applications
- With proper structuring of the model, YANG allows for easy extensions using YANG augmentations without needing to churn the base model.
- Model definition using YANG allows for easy tie-in of the analytics model with data source configuration, which may be configurable via a YANG-supported interface already
- As YANG is gaining traction with networking vendors, the number of data models specified using YANG is exploding, allowing for easy tie-in with model-driven analytics that is YANG based. By the same token, it allows for easy tie in with MIB data (still an important category of telemetry data), which can be easily transposed into YANG [13].
- It facilitates an integration strategy with YANG-Push that allows applications to subscribe to YANG data [6], for example in cases where subscribed data should also be subjected to analytics.

B. *Module Structure*

The DNA Yang module tree diagram is depicted below. In the interest of brevity, simplifications have been applied and only some key parts are shown.

```
module: task-templates
   +--rw task-templates
      +--rw task-template* [tt-name]
         +--rw tt-name                string
         +--rw stream* [stream-name]
         |  +--rw stream-name    string
         |  +--rw field* [field-name]
         |     +--rw field-name    string
         |     +--rw type?         datatype
         +--rw task-parameters* [param-name]
         |  +--rw param-name    string
         |  +--rw type?         datatype
         +--rw result-stream
            +--rw field* [field-name]
               +--rw field-name    string
               +--rw type?         Datatype
```

Figure 4: YANG tree for task templates

Task templates specify merely the structure of data streams to be processed as well as the result stream. Task parameters include all additional parametrization, such as which input data to populate particular fields with or which aggregation function to apply.

```
module: bindings
   +--rw bindings
      +--rw maplet-binding* [tt-ref mb-name]
      |  +--rw tt-ref                tt:template-ref
      |  +--rw mb-name               string
      |  +--rw required-capability*  capability
      |  +--rw stream-binding* [stream]
      |  |  +--rw stream          tt:stream-ref
      |  |  +--rw field-binding* [field]
      |  |  |  +--rw field      tt:field-ref
      |  |  |  +--rw binding?   string
      |  |  +--rw source-config* [source]
      |  |     +--rw source          tt:source-type
      |  |     +--rw source-config
      |  +--rw (analytics-binding)?
      |     +--:(csa)
      |     |  +--rw csa-binding
      |     +--:(spark-streaming)
      |        +--rw spark-binding
      +--rw reducelet-binding* [tt-ref rb-name]
         +--rw tt-ref           tt:template-ref
         +--rw rb-name          string
         +--rw (analytics-binding)?
            +--:(csa)
            |  +--rw csa-binding
            +--:(spark-streaming)
               +--rw spark-binding
```

Figure 5: YANG tree for bindings

Bindings include the rules for how to generate a query for a task template, based on parameters supplied by the user for which parameter substitution is applied. The specific rules depend on the target analytics engine, subsumed e.g. under "csa-binding" and "spark-binding".

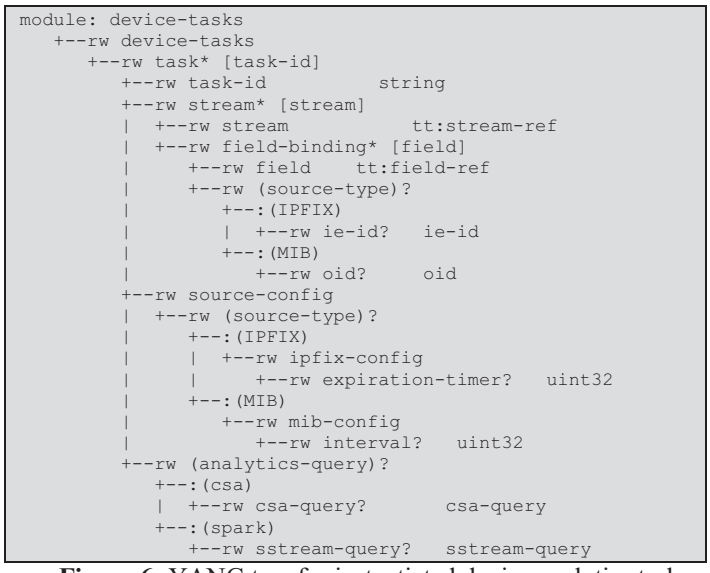

Figure 6: YANG tree for instantiated device analytics tasks

Device tasks finally include the instantiated analytics query, as well as the precise source configuration to populate the data streams. Data source specific parameters are defined via augmentations in additional modules. For example, to add support for IPSLA, IPSLA specific configuration parameters such as type of probe, number of packets, probe intervals will be defined in a new "case" statement in a module that augments the "source-config" choice.

V. Experiences, Conclusions, Future Work

We have applied the model to a variety of use cases and have received overwhelmingly positive feedback by network providers deploying Distributed Network Analytics in their networks. One use case involves a monitoring scenario, in which device analytics tasks involve computing a baseline for the normal operating range of key statistics and performance indicators. Subsequently, the current state of those statistics and performance indicators is compared against the baseline. When the top of the baseline is approached, for example the top percentile is breached, an analytics match is found and a result sent, which is in effect treated as a special type of threshold crossing alert. Other use cases are similar, involving for example analysis of trends and reports when sustained trends are observed, even when nowhere near extremes.

In general, we have found that the system is used to perform overwhelmingly fairly simple analytics. While the Agent includes a full-fledged stream processing engine, most of the use cases involve simple aggregations and time series analysis. The most important requirements of users are the ability to do away with polling, be able to cut down on the volume of analytics data, and simplify the configuration of data generation tasks across the network; beyond that, the 90/10 rule applies (over 90% of analytics use cases exercise less than 10% of analytics processing features).

We have found that the model addresses our requirements to conduct network analytics very well. YANG has proven adequate to represent our analytics model; the ease with which it can be integrated into SDN and networking environments and their associated control interfaces has made it a smart choice.

That said, there are a number of possible extensions that have not yet been addressed, which would bring the power of the solution concept to yet another level. One concerns the ability to close control loops and specify actions to take in response to analytics conclusions. Such an ability will be key to true automation and greater network intelligence. Of course, challenges abound, from security to the problem of how to deal with heterogeneous devices lacking a common network programming model. A second extension concerns the ability to stage different phases of analytics to spin off additional analytics tasks when warranted by conditions. For example, upon observation of a baseline violation, additional supporting evidence might be collected and secondary forensics analytics tasks be launched. This type of capability will be key for smart analytics tasks that dynamically adapt processing and are able to zoom into underlying data sources precisely when needed during any given analytics task.

References

[1] L. Bass, I. Weber, L. Zhu: "DevOps – A Software Architect's Perspective", Addison Wesley, 2015.

[2] A. Bierman, M. Bjorklund, K. Watsen: "RESTCONF Protocol", draft-ietf-netconf-restconf-17, IETF, September 2016 (work in progress).

[3] M. Bjorklund, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 7950, August 2016.

[4] M. Chibha, A. Clemm, S. Medley, J. Salowey, S. Thombare, E. Yedavalli: "Cisco Service-Level Assurance Protocol", RFC 6812, IETF, January 2013.

[5] A. Clemm, M. Chandramouli, S. Krishnamurthy, "DNA: An SDN Framework for Distributed Network Analytics, IEEE/IFIP International Symposium on Integrated Network Management (IM 2015), Ottawa, Canada, May 2015.

[6] A. Clemm, A. Gonzalez Prieto, A. Tripathy, E. Nilsen-Nygaard: "Subscribing to YANG datastore push updates", draft-ietf-netconf-yang-push-03, IETF, June 2016.

[7] A. Clemm, R. Wolter (eds.): "Network-Embedded Management and Applications", Springer, New York 2013.

[8] R. Enns, Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[9] D. Kreutz et al., "Software-Defined Networking: A Comprehensive Survey," Proceedings of the IEEE Vol 103 No 1, January 2015.

[10] K. McCloghrie, F. Kastenholz: "The Interfaces Group MIB", RFC 2863, IETF, June 2000.

[11] J. Medved, A. Tkacik, R. Varga, K. Gray: "OpenDaylight: Towards a Model-Driven SDN Controller architecture", IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2014.

[12] T. Nadeau and K. Gray: "SDN: Software Defined Networks", O'Reilly Media, 2013.

[13] J. Schoenwaelder: "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, IETF, July 2012.

[14] "Cisco Network Services Orchestrator Enabled by Tail-f". Accessible at http://www.cisco.com/go/nso (last accessed 29 September 2016).

[15] YumaPro. https://www.yumaworks.com/yumapro-sdk/