DNA: An SDN Framework for Distributed Network Analytics

Alexander Clemm, Mouli Chandramouli, Sailesh Krishnamurthy[1]
Cisco Systems, Inc.
San Jose, California and Bangalore, India
{alex|moulchan}@cisco.com, sailesh@gmail.com

***Abstract*— Analytics of network telemetry data helps address many important operational problems. Traditional Big Data approaches run into limitations even as they push scale boundaries for processing data further. One reason for this is the fact that in many cases, the bottleneck for analytics is not analytics processing itself but the generation and export of the data on which analytics depends. The amount of data that can be reasonably collected from the network runs into inherent limitations due to bandwidth and processing constraints in the network itself. In addition, management tasks related to determining and configuring which data to generate lead to significant deployment challenges.**

In order to address these issues, we propose a novel distributed solution to network analytics. Analytics processing is performed at the source of the data by specialized agents embedded within network devices, which also dynamically set up and reconfigure telemetry data sources as required by an analytics task. An SDN controller application orchestrates network analytics tasks across the network to allow users to interact with the network as a whole instead of individual devices one at a time. The solution has been implemented as a proof-of-concept, called DNA (Distributed Network Analytics).

I. INTRODUCTION

Networks offer rich sets of network telemetry data, such as flow records, interface statistics and MIB data, IPSLA performance measurements, or system event (syslog) records. Analyzing this data can yield important insights about a network, such as traffic patterns and how the the network is used, about network intrusions and attacks on the network or its users that may be in progress, about instabilities that may lead to degradations or fluctuations in service quality, about anomalies that may be indicative about impending failures, and much more. This means that analytics can help operators of networks and networking services solve many operational problems, and vice-versa, networks are an interesting application area for analytics.

Traditional analytics involves hauling large amounts of data to a backend, where that data is subsequently processed and analyzed. In general, the larger the volume, the better, as more data may yield more insights than less data. Much of the research has therefore focused on making sure processing scales well and can keep up with this volume; important results with impressive scaling properties include Map Reduce algorithms [1], Hadoop [2], Hive [3], and more. Applying analytics to the networking domain is especially challenging, as the amount of data to be analyzed grows with the number of nodes in the network, the number of connected endpoints, and the number of flows in and the amount of traffic in the network, all growing at much faster rates than Moore's law.

In addition to the ability to process the collected data, the effectiveness of analytics does depend on the availability of a large set of raw data in the first place. Generating network telemetry data is expensive and limited by resources in the network. For example, interface statistics provided by a Management Information Base (MIB) can perhaps be polled or read out from the device once a minute, but not every second, let alone millisecond, although such data would be interesting for analytics involving fast-changing statistics and health data. Similarly, network measurements using IPSLA [4] can only involve a limited number of probes and amount of test traffic at any one time. Syslog messages at very low severity levels (e.g. debug) are typically turned off in production networks. Netflow is rarely fully turned on but typically sampled. All these remedial measures are taken in order to conserve system resources at network devices as well as bandwidth resources needed to export that data, which can be a particularly critical consideration in deployment scenarios involving WAN links. While network devices are generally capable to deliver any conceivable piece of network telemetry data, they cannot deliver all at the same time. As a result, only a subset of all potential data is generated at any one point in time.

This means that even when a network analytics application is able to ingest and process network telemetry data at very large scale using "Big Data" techniques, it is inherently limited by the fact that not all the data that might be useful for the analysis may be available. This is an important limitation that may not be evident at first. This issue is compounded by the inherent wastefulness of many Big Data applications, in which a lot of the generated data ends up being effectively thrown away, while in hindsight other data that might have been generated in its place would have been more valuable. In many cases, such as real-time network monitoring, users and applications are only interested in outliers and anomalies, to identify what is out of the ordinary and the hot spots in the network that warrant more detailed attention. In other cases, such as for real-time network reports, it are really only summarizations and aggregations of that data that are of interest (such as histograms and distributions over time), not individual data items themselves.

[1] Sailesh Krishnamurthy has left Cisco and is now at Amazon

Finally, there is an issue of very practical nature that is often considered a matter of mundane network administration and therefore ignored, although it turns out to be in practice a big hurdle to deployments of network analytics at scale. This concerns the fact that the network needs to be set up for export and collection of the data that is to be subjected to analytics. Not only do targets for exported data need to be configured, but also measurement probes set up and thresholds and sampling rates determined. Any solution ignoring those issues or leaving them as "exercise for the users" is fundamentally incomplete.

The limitation of Big Data in the context of Network Analytics ultimately stems from the fact that it addresses only the processing of analytics data itself, not the entire analytics lifecycle as a whole. That lifecycle (depicted in Figure 1) also includes the generation, exporting, and collection of data, as well as potentially acting on it (such as dynamically reconfiguring which data to generate and collect). It can also include preprocessing data at the source, for example (in the case of SNMP) applying thresholds via RMON or defining custom events and performing simple computations through event and expression MIB. Of course, such capabilities need to be configured as well, not just on a device but across the entire network. It is this entire network analytics lifecycle that is addressed by Distributed Network Analytics (DNA).

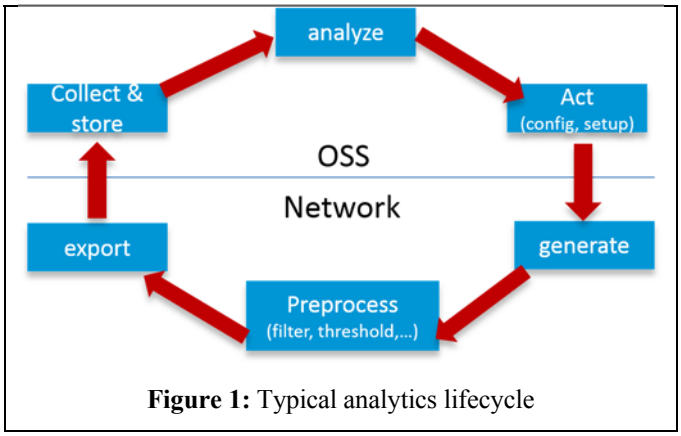

Figure 1: Typical analytics lifecycle

Instead of relying on data to be first generated and collected from many locations to be made centrally available for processing, analytics are performed right at the source. Analytics of data that can be performed within the scope of an individual networking device is conducted by a DNA Agent, a software function embedded in the network device right where the data is generated. Also, data sources are set up and dynamically adjusted to generate exactly the data that is needed to support the analytics tasks that are to be performed. Instead of large volumes of raw data, devices export much smaller volumes of condensed information, or ***Smart Data***. The capability to perform analytics tasks in distributed fashion across the network is complemented by the ability to administer and manage those tasks in centralized fashion, allowing users and applications to interact with the network in holistic fashion as opposed to one device at a time. This is accomplished using a DNA Controller, a Software Defined Networking (SDN) control application.

By performing analytics in the network, not only is the amount of required off-box processing reduced. Given the effectiveness of Big Data technology, this constitutes only a side benefit and is not one of DNA's primary drivers. More importantly, the required on-box processing within the network can be reduced as well. This appears counter-intuitive at first, as clearly analytics processing at the source requires CPU and bandwidth. Processing power on network elements is particularly precious as it comes at a much higher premium than additional processing power in a data center. However, these additional cycles offset by the avoidance of cycles that would otherwise be required to generate, format, and transmit data that is now no longer required, and by the saved opportunity cost of potentially not getting the most useful data for the job.

Some more advanced analytics, specifically cross-device analytics that correlates data across devices, as well as applications that mine historical logs, may still need to be performed off-box. Smart Data is not intended to replace Big Data, but to complement it smartly and allow it to operate on data that is richer, more impactful, and more efficiently obtained than data than would otherwise be at its disposal.

The remainder of this paper is structured as follows: Section 2 presents some of the use cases to which DNA is applied. While DNA is a general-purpose framework and not limited to those specific scenarios, the use cases have been used to validate DNA's design and provide further insights into its design and associated problem statement. Section 3 provides an overview of the overall DNA architecture. It describes its main building blocks, the DNA Agent that resides inside the network elements and the DNA Controller which orchestrates the deployment and execution of analytics tasks across the network. The section includes a description of the Network Analytics Service that users and applications use to interact with DNA. Section 4 discusses DNA's proof-of-concept implementation. (This is only a proof-of-concept and no inferences about Cisco product strategy or direction should be made.) Section 5 provides a performance analysis. Section 6 discusses related work. Section 7, finally, concludes the paper and provides an outlook of future work.

II. Use Cases

The following are two examples of DNA applications that illustrate the types of problems that distributed network analytics can address. It should be noted that DNA is not restricted to those particular use cases.

A. Where is my IP traffic

Consider the scenario where a network administrator would like to know which network devices can see traffic with particular properties at this particular moment, for example, traffic from or to a specific IP address that is deemed suspicious. A naïve way of addressing this scenario would involve turning on Netflow across the network, collecting vast amount of flow records, then analyzing and filtering these at the backend to identify those devices that exported a flow record containing a field with the IP address in question. However, doing so results in an unacceptably high load on the network. One mitigation technique might involve reverting to

sampled Netflow. However, in this case this is unacceptable, as small flows with that IP address might not be detected and hence some networking devices might be missed.

Using DNA, an application can instead issue a single request to turn on Netflow across the network, or a specified portion of it, accompanied with a filter for the IP address in question. A targeted request, this can be limited to a very brief duration, with flow expiration timers set accordingly and subsequent automatic task cleanup without requiring further intervention on the side of the user. Only those devices at which corresponding flow records are observed report them to the user. Rather than having to sift through a vast number of records to find the needles in the haystack, the application can immediately focus on performing the next steps of the analysis.

B. *Network brownout*

Another scenario involves brownout scenarios that may build gradually over time without being noticeable by end users, only to result in sudden rapid deterioration of service levels. An indicator for such scenarios includes packet drop rates across interfaces. Other indicators include link utilization, distribution of traffic mixes (e.g. proportion of high priority application traffic), service levels such as jitter and round-trip delay to a given destination, or combinations thereof. Of interest here may not be so much the absolute rate, as in a practical setting it is often not possible to articulate a specific critical threshold. Instead, what is of interest is the trend of these rates over time, and whether the trend is sustained or may even be accelerating. Basically, an administrator will want to know what is changing, even if changes are subtle and build up slowly over time.

Using DNA, an application can deploy a corresponding analytics query across the network. This query continuously computes the current packet drop rate (or other parameter of interest) from successive packet drop counter readings and feeds these readings into a trend analyzer that computes simple trend measures, such as the number *n* of periods that showed an increase over the previous period over a rolling time scale of the past *m* periods. The device reports query results, complete with the then-current interface statistics, once the trend exceeds a certain threshold, or once the trend exceeds (for example) the top percentile in historical durations for which such trends are sustained (which is also computed continuously), or if a trend-of-trends is observed (feeding the output of the first trend measure into a second trend aggregator).

III. DNA Architecture

A high-level overview of the DNA Architecture is depicted in Figure 2. The architecture consists of two components:

- A DNA Controller, an SDN application that orchestrates network analytics tasks across the network, collates 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 as needed and performs analytics on the resulting telemetry data streams.

A. *DNA Controller*

The DNA Controller provides users and application with a Network Analytics Service. This service allows users to request the execution of Network Analytics Tasks. A Network Analytics Task includes the following aspects:

- The analytics task to be performed across the network. This includes the definition of the source data that is required, and the aggregation and queries that are to be performed on that data.
- The scope of the network where the analytics task is to be conducted. Some analytics tasks might involve the entire network, whereas others may be conducted only on parts of the network, for example a subdomain of the network or on devices playing a certain role in the network.
- The schedule of the analytics task, specifically for ongoing tasks that are expected to push results whenever new analytics query matches are identified, not in one-time pulling of data.

The DNA Controller encompasses several functions, as described in the following subsections.

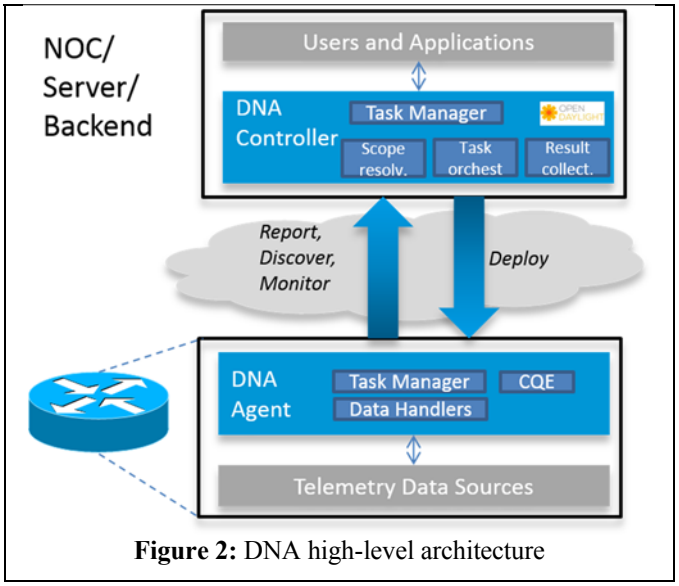

Figure 2: DNA high-level architecture

1) *Analytics Task Decomposition*

An Analytics Task Decomposer decomposes the task request that is received into a set of *maplets* that need to be deployed across DNA Agents in the network. A maplet specifies an analytics task to be performed by a DNA Agent on a device. It specifies required source data and, by extension, any needed configuration of data producers (determined and performed by logic on the DNA agent), as well as the analytics query and aggregation that is to be performed on the device.

In many cases and in our initial implementation, the analytics task translates directly into a maplet, with the same maplet being deployed at each DNA agent. Going forward, it is conceivable that advanced tasks translate into a set of different maplets, differentiated by network device, taking additional context into account: for example, place in the network,

capability of the network device, or topological relationships such as upstream or downstream on a given path.

It should be noted that maplet differentiation is not intended to accommodate different device variations implementing certain data sources. DNA Agents are expected to implement a consistent interface and, when a data field is supported, adhere to a consistent format; it is up to the DNA Agent implementation to hide any device-specific variations and render the data as needed.

2) Network Scope Resolution

A Network Scope Resolver determines the DNA Agents at which to deploy the maplets. The Network Scope is specified by the initiator of the network analytics task; it can be considered a specific type of policy specifying the criteria that a network element must meet to participate in the analytics task. Simple examples of network scope include lists of devices by wild carded device names, by IP addresses, subnets, ranges, by device tag or property (e.g. device type), more complex scopes involve PINs (place in the network), device roles, and topological relationships (such as neighbors in a physical topology no more than two hops removed). The Network Scope Resolver operates on network inventory and topology data that is supplied by the underlying SDN controller infrastructure.

3) Network Task Deployment and Monitoring

Once the network scope is resolved, tasks are deployed across the network. In addition to the deployment itself, an important function of the DNA Controller is to monitor the status of the task and keeping a tab on the responses that are received. Deploying a maplet to a DNA Agent will result in an acknowledgment whether the task "took". (This is separate from the actual reporting of analytics results.) In a large network, various error conditions are to be expected. For example, not every DNA Agent may support the required capabilities and some DNA Agents may be overloaded or in a degraded state and therefore not able to perform the requested tasks. DNA does not pursue transactionality, meaning that there is no requirement for the same analytics task to either "take" at every DNA Agent or not at all. Instead, the DNA Controller keeps a tab on responses that are received and maintains an overall health status of the network analytics task as a whole. It is up to applications to decide when a network analytics task is so degraded that it should be terminated.

4) Results Collection

DNA Agents report results of analytics queries on a continuous basis. Whenever a match for a query is detected, or an aggregate of data becomes available, the DNA Agent will asynchronously push a result record that includes a reference to the task that the record belongs to.

The DNA Controller collects and collates the results. At this point, results are simply passed on to the user. However, it is conceivable to perform cross-device analytics at the controller and aggregate results that are received from multiple DNA Agents further. For this purpose, DNA supports the concept of an optional *reducelet*, which can be included in the definition of advanced analytics tasks that consolidate multiple result streams.

5) Other Considerations

DNA Agents do not allow analytics tasks to run indefinitely. The DNA Controller may periodically "refresh" network analytics tasks if needed required, but unless they are extended, tasks will eventually automatically expire, resulting in cleanup of the task and emitting a corresponding notification to the controller. This facilitates "garbage collection" and avoids the situation in which orphaned analytics tasks consume processing resources in the network.

It is conceivable that changes in the network, such as addition of new devices or changes in network topology, occur while network analytics tasks are running. In this case, it is conceivable to have the DNA controller assess whether to deploy a maplet to a new device (for example) whenever the controller detects a network change. In the case of our system, changes in the network are simply picked up when the next analytics task refresh occurs.

Articulation of analytics queries involves a variation of SQL that can be used for continuous stream queries, performed over the stream of data produced by network telemetry data sources. In order to facilitate the composition of new queries, a concept of Analytics Task Templates is supported, containing pre-canned customizable analytics tasks, supplied for example by network administrators. An application or user can articulate an analytics task simply by referring to an existing template and populating its parameters. The DNA Controller then applies a predefined query mapping associated with the template for the analytics task at hand.

B. DNA Agent

The DNA Agent is an application that is embedded in a network device. It conducts the analytics tasks local to the device as specified by the maplets, analyzing streams of local telemetry data and pushing results back to the DNA Controller. It also configures the underlying telemetry data sources as needed, for example, setting up network measurements or configuring local statistics collection, to provide the raw data that is needed as a basis for that task. The DNA Agent encompasses several functions, as described in the following.

1) Task Handling

A Task Handler performs most of the DNA Agent's housekeeping and orchestration functions.

When the DNA Agent receives a maplet from a DNA Controller, the Task Handler verifies that the DNA Agent has the needed capabilities and sufficient processing capacity so service the maplet. Subsequently, the data handlers are instructed to set up the data sources as needed and the query is passed to the processing engine.

Maplets are not allowed to be in effect indefinitely but are associated with an expiration time. Upon expiration, cleanup operations are performed to remove queries and cease the generation of data that is no longer needed. While expiration of maplets can be periodically extended, this mechanism avoids situations in which "orphaned" analytics tasks run forever, i.e. tasks that end users have lost interest in but that continue to consume resources in the network.

2) Data Handling

Data handlers configure telemetry data sources inside the device in order to generate a stream of data that can be fed to the DNA Agent's query and aggregation engine. One example of such a data source is Netflow, which produces a stream of flow records. Another example: MIB objects, accessed in our case through DCM, a device feature that produces streams of MIB data, with MIB snapshots taken either periodically or under certain conditions [5]. A third example concerns measurements of service levels, as provided through IPSLA. In each case, data is not simply provided by default, but configuration is required so that data is generated. In some cases, this data generation can be resource intensive, which is why DNA's ability to turn on that data generation only when it is needed is so important.

Once the data source has been set up, telemetry data is pushed to the data handler, which massages it into a data stream that can be processed by the embedded analytics engine (next section). The data handler also provides harmonization of data if needed, performing an additional rendering step that transforms a local representation of data into a representation that a maplet refers to. As a convenience feature, named data items can be grouped into "capabilities", collections of data items that are frequently used together for analytics purposes. DNA Agents advertise which capabilities they support, and controllers and templates can indicate which capabilities are required for specific analytics tasks. However, to avoid the need to define expansive sets of new data models for analytics purposes, analytics tasks can refer to data also in other ways, e.g. to MIB objects or Netflow Information Elements.

As a general design principle, if data is natively represented in different ways by different devices, it is up to the DNA Agent to ensure the data is rendered in a way that can be consumed by the DNA Controller, instead of the DNA Controller needing to perform special treatment depending on the specific devices that host the DNA Agents. This is enabled by allowing analytics tasks to refer to source data as named data items, and have the data handler perform any rendering from a local to a common representation. This decision is designed to keep DNA nimble and avoid the situation in which the DNA Controller over time deteriorates into a "fat" management system that becomes costly and slow to maintain.

3) Query and Aggregation Processing

Analytics logic and query processing leverages a network-embedded version of CQE [6], an advanced stream processing engine which is also used by off-box analytics applications. The data records from a data producer are framed as rows in a database table. Each data record produced constitutes a separate row in the table. Fields in the records are considered different table columns. Records from different data producers or different types of records by the same data producer are considered as separate conceptual tables.

Processing includes the ability to aggregate raw data according to statistical baselining functions such as percentiles or top-n, histograms, to conduct basic trend analysis, to apply filters and comparators, and to, more generally, apply queries on telemetry data, both across windows in time ("continuous") and across individual data items. Aggregation tasks are expressed as CQE SQL queries that are applied against these tables. CQE does allow for continuous queries by associating records with a time window and continuously updating query results as new rows are added to the query (or old rows are "aged out"). As new query matches are found or updates to aggregates are computed, the corresponding results are pushed to the DNA Controller.

C. Interfaces

There are two sets of interfaces used by DNA: An interface provided by the DNA Controller to the user of the DNA service, and the interface provided by the DNA Agent that DNA Controller and DNA Agent use to communicate.

We decided to use open data-model driven interface design both for the controller and for the agent. At its basis is a set of YANG modules [8] that have been defined to define the configuration of analytics tasks both at the network and at the device level. The configuration of a network analytics task includes:

- A list of data items and a stream of corresponding data records, parametrizing items such as the frequency or policy with which records are being generated or the required accuracy of items if involving performance measurements. Data items themselves do not necessarily have to be YANG-defined; the model allows to also refer to MIB objects, flow records and their information elements, even named CLI show output fields.
- An analytics query that operates on this data stream. A choice is given between specifying a continuous query that is directly processed by CQE, or referencing a template of a "precanned" query and providing a list of template parameter assignments. (The template is subsequently expanded by the controller into a corresponding CQE query.)
- A network scope that defines a policy which network elements are subjected to the task.
- A schedule, specifying start and end time or duration.
- The preferred export format.

The YANG module of the agent mirrors the model of the controller and uses many of the same groupings but does not include the network scope which is resolved by the controller. In each case, also monitoring support is provided that allows to retrieve the list of current tasks and their status.

Results of the analytics task are exported from the DNA Agent to the DNA Controller using a choice of interfaces, including a JDBC-based interface that writes analytics results directly to the controller, IPFIX [13], in which case the configuration of the task includes a definition of the IPFIX template used to export analytics results, or syslog.

IV. Implementation and Proof of Concept

We have implemented a Proof of Concept of DNA, operating on a virtualized testbed of Cisco networking devices.

The DNA Controller was implemented as an application on top of Open Daylight [7]. Our implementation took extensive

advantage of the model-driven toolchains provided by Open Daylight. From the YANG data model, we generated a ReST-like interface per the RESTconf spec [9]. This interface is used by external applications to interact with DNA. In addition, the DNA controller application relies on Open Daylight for inventory and topology information. At this point, automatic network discovery is not yet supported and needs to be administratively populated at the controller.

Because it is readily supported by Open Daylight, Netconf [10] was chosen as communication mechanism between DNA Controller and DNA Agent to orchestrate the deployment of analytics tasks and monitor their status. The YANG model was subsequently used to generate the internal controller APIs used by the DNA Controller application to communicate with the DNA Agents. As mentioned, the analytics results themselves are exported using JDBC (default), or alternatively (in particular when an export destination other than the controller is chosen) IPFIX or syslog.

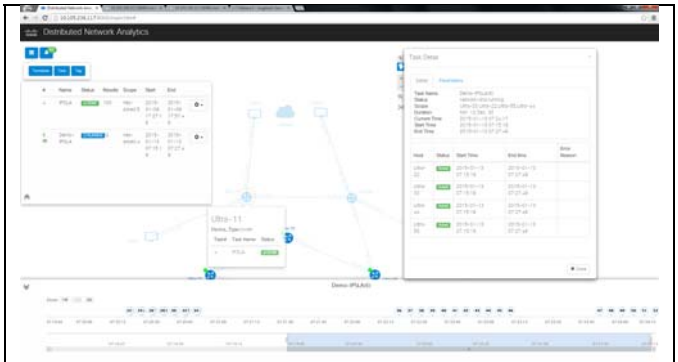

Figure 3: Implementation GUI

A GUI (depicted in Figure 3) has been implemented on top of the DNA Controller to allow network administrators to interact with the Network Analytics Service provided by DNA, and to visualize network analytics results. Analytics queries can be specified either by entering an analytics query directly, or by picking from a list of templates of "precanned" queries and customizing aspects such as which data items to apply the query to (e.g. link utilization? packet drop rates? round trip jitter to a server?), which aggregators to use (top-n? trending? percentiles?), and which thresholding, if any, to apply.

The DNA Agent is implemented as a container-based application, running inside an LXC Linux container [11] hosted on the network element. Configuration of data producers within the network device is performed through onePK [12]. Our footprint currently stands at around 120MB. At this point, support for Netflow, SNMP MIBs, and IPSLA is provided. As mentioned earlier, exporting of results from the DNA Agent to the DNA Controller occurs through a JDBC-based push protocol or IPFIX export.

V. Performance Analysis

In this section, a performance analysis is presented to illustrate the efficiency of the proposed Distributed Network Analytics framework in comparison to the traditional centralized analytics framework. It may be useful to illustrate those benefits for specific use cases such as Where is my IP and Network Brownouts. Two metrics are considered to compare the two approaches, the volume of data, and CPU overhead.

A. Volume of data

A well-known and most-often used solution for the Where is my IP usecase, is the traditional centralized analytics approach. Netflow needs to be configured on all routers in the network. From all routers in the network, Netflow records are then exported to a Netflow collector, where they are post-processed and analyzed using Big Data techniques.

The cumulative effort can be estimated by the amount of data that needs to be generated, exported, and stored. To estimate the data volume, the parameters to consider are: the interface speed at the router, the number of active flows on the interface, and finally the Netflow packet sampling rate. Of course, the number of records exported depends on the number of flows, not the interface speed per se. It is conceivable that there can be a small number of elephant flows, or a large number of mice flows, or some mixture. In addition, it has been observed that the volume of flow records also depends on other factors such as flow expiration timers. However, in practice, a correlation between interface bandwidth and number of flows is often observed. A conservative rule of thumb [16] suggests that flow record volume amounts to roughly 2 % of the interface bandwidth. In general, the flow volume can be defined as a function of (# flow records, record size). The flow record size is a function of (# of information elements in a record). Typically there are roughly 20 IEs with 4 octets each = 80 octets/ record. The number of flow records in an hour can be estimated as (average interface utilization * interface speed over an hour) / (packetsize*packets per flow).

Let us consider a 1 Gbps interface, link utilization of 40 %, typical for a traffic engineered network. Let us assume that the average flow size of 100 packets with each packet having 500 bytes on average, i.e. 400 kb/flow. For a 1 Gbps interface with 40 % utilization, the average traffic for the interface can be 400 Mbps. At 400 kb/flow, this means roughly 1000 flow records per second on the interface. 1000 records/second * 3600 seconds * 80 octets/ record = 288 MB of data to be exported from a single interface on a router in an hour. The total volume of data exported from all routers to the central collector is linear to the number of active flows and the number of nodes. The amount of processing to search for particular records meeting a specific match criteria is proportional to the amount of data stored, as each record needs to be inspected.

In contrast, with DNA, Netflow data generation at the network device is configured only for the duration of the task and stopped upon completion of the analytic task. Netflow metering remains the same during the duration of the analytic task, however, flow data is merely passed to the DNA Agent contained in the same router, i.e. from one software process to another. A result record is exported whenever there is a match for the filter condition for the expired flows in the Netflow cache; only a small fraction of flow records will match that critera. If a short flow expiration timer of 5 seconds is configured for Netflow, then in the worst-case scenario of one match each window, 12 results per minute are exported. Assuming an export record size of 80 octets, that amounts to

960 bytes per minute or less than 60 KB per hour per interface, in contrast to 288 MB. If the task is not run continuously or the worst-case scenario does not apply, the volume of results data is reduced even further.

While the analysis presented is based on the average flow size distribution and deterministic values for some parameters such as flow-expiration timers, and such factors can vary greatly in practice, the takeaway is that the amount of data to be exported is reduced by several (3-5) orders of magnitude. Further reductions occur if tasks are not run continuously but are applied only "on demand" for short durations.

Similar conclusions can be derived for the Network Brownout use case. As explained earlier, it is preferable to identify trends of link utilization, packet drop rates, and end-to-end delay before actual network congestion occurs. A key data source for this class of use cases are MIB objects, e.g. from the IF-MIB. End-to-end measurements further involve IPSLA. With the centralized analytics approach, MIB objects are periodically and repeatedly polled. Continuous SNMP polling causes excessive CPU on the router, limiting the granularity at which measurements that can be achieved. An SNMP PDU to poll five MIB objects, including varbinds and headers, but not including IP and UDP overhead, can be estimated to require roughly 120 octets per request and response. Assuming just 20 interfaces to poll once a minute results in 4800 octets of SNMP traffic per minute, or 288 Kbytes / hour.

With a DNA approach, results are exported only when a match for the trend (as defined by the user) is detected. This may occur perhaps only once per 1000 or 10000 polling cycles, correspondingly reducing the amount of data to be exported by several orders of magnitude. (Potentially exported data is reduced even further if only a summary of the trend is reported, not all objects that were used to derive at that conclusion.)

B. CPU

In terms of CPU, there are three contributing components: the CPU of the Netflow metering process, the CPU of the Netflow export process, and the CPU for the DNA analytics.

We have performed some initial measurements whose results are depicted in Figure 4. We ran two series of experiments. In one case, we deployed a single query. In the other case, we deployed five separate queries, all operating on the same data set. We then varied the number of flows, resulting in a different number of flow records being generated. What we found was that an increase in the number of flows to be processed, not surprisingly, resulted in a linear increase in CPU consumption. However, we found that the increase in the number of queries resulted only in a slight increase of CPU.

We conclude that the analytics processing performed on the device is dominated by the number of records being generated; the analytics tasks themselves result in only negligible additional load, further offset by the savings in CPU needed for the export process which was not considered here.

Not considered in our analysis is the fact that the total time to obtain the result of the analytic task is also greatly reduced. The latency to copy the Netflow data to the container is much smaller than exporting a record to a collector and the result data to be exported to northbound application is much smaller, thus the latency for export is also reduced. As a net result of the above-mentioned factors, it is possible to obtain an almost near real-time view of the transient data from the network.

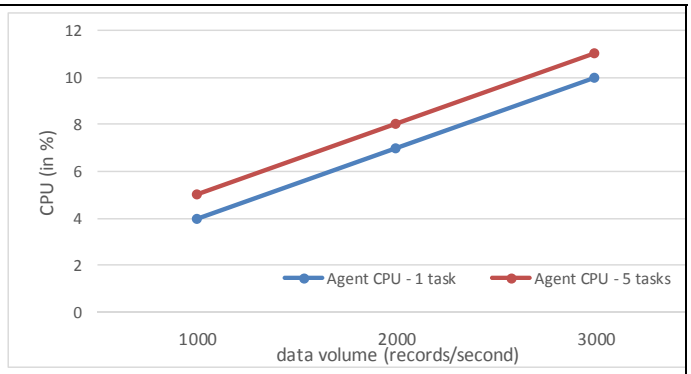

Figure 4: CPU vs. volume of data vs. number of tasks

VI. Related Work

Historically, analytics technology was consumed in the form of OLAP (Online Analytics Processing) systems. These Decision Support Systems (DSS) included massively parallel (MPP) relational databases as well as multi-dimensional cube-based databases – both primarily motivated by Business Intelligence use cases. The techniques for parallel query execution in shared-nothing MPP systems are well-understood and described in early papers about Gamma [20], Bubba [22], Volcano [21] etc. Subsequently, these features became standard in most commercial relational systems and were extremely successful in driving the database industry forward. For all practical purposes, however, these ideas were only available to users of these systems.

Once the Internet took off, there was a pressing need for the underlying MPP data processing techniques to be exposed to non-relational and non-SQL developers. Furthermore, the sheer scale of data to be processed exceeded the limits of what could be achieved with traditional MPP systems. These requirements were felt urgently at Google leading to the implementation of highly-scalable data infrastructure such as GFS [28] and MapReduce [1] and their open source equivalents (Hadoop [2] and HDFS). Unlike the BI use cases that were the motivation of the previous generation analytics systems, these Internet-scale systems were typically used to solve large and complex data problems dealing with massive graphs, clickstreams etc in a batch processing fashion. The initial programming model for these systems was the MapReduce (MR) framework. Over time, however, the value of more declarative approaches became evident, leading to various SQL and non-SQL alternatives, each generating MR jobs. These alternatives include Hive [18], [3], Pig [19] , Spark [24], Shark [27] etc.

While newer analytics systems were much more scalable than older systems and generally better at interactive queries, they were still primarily used at batch-mode analysis. Such systems were great at crunching massive data sets but not ideal to satisfy the low-latency requirements of operational analytics systems that operated over high velocity streams of log data. Some examples of streaming database systems include early

research projects like TelegraphCQ [29], STREAM [17] etc. as well as commercial systems like Truviso [25], StreamBase etc.

While the batch processing and streaming systems evolved separately, the batch systems are offering more real-time functionality by (a) being more interactive for traditional queries, and (b) attacking the streaming space by pursuing a micro-batch approach. Examples of the former include systems like Impala and Presto that have built traditional SQL execution engines on top of data from HDFS as well as the more interesting Spark system which was built around the idea of a distributed memory abstraction called "Resilient Distributed Datasets" (RDD). RDDs essentially cache intermediate state produced by MPP-style computation in memory while still providing fault-tolerance and thereby enable highly interactive and iterative applications. This RDD abstraction also enabled the development of Spark Streaming [23, 24] that uses the notion of a "discretized stream" that can provide strong consistency and fault recovery while still enabling certain kinds of streaming queries.

DNA explores a very different space of the analytics landscape by focusing on the following key aspects, none of which is covered by the classic systems and earlier research described above:

1. **Focusing on network telemetry data -** While other systems certainly build data integration software to ingest incoming data, they tend to ignore the very real challenges of extracting data from network elements, which can shift bottlenecks from analytics processing itself to the generation of raw data. Networks can produce an enormous volume of data. However, indiscriminate turning on of all probes is not feasible. Approaches are needed that set up sources in a targeted fashion as required by the actual tasks.
2. **Dynamic configuration and adaptation of data sources as integral part of analytics setup -** Traditional systems make an inherent assumption that data streams are a given and come "for free", and that any associated system administration tasks can easily be delegated, focusing solely on query processing instead. Our approach is unique in that it integrates turning up, configuring, and adapting data sources as needed to support queries.
3. **Operating over a wide-area network -** Data systems tend to ignore the costs associated with moving data across a WAN environment. Some earlier work on federated databases, such as the Mariposa [29] and Garlic [30] systems, did consider network costs in distributed query optimization but not in a streaming fashion that is critical for operational analytics.
4. **Pushing processing into network elements** - Network elements like switches and routers now have the hardware capacity to sustain data processing at the edge as well as improved software architecture that can enable integrating application software (such as a continuous query processor) using container isolation. This new capability allows us to exploit the network element in pre-processing data extracted by the probe and dramatically lowering the impact on expensive transit links for the Wide Area Network, which were developed and maintained at considerable expense in order to sustain user and not analytics traffic.
5. **Controlling and programming tasks at a network level using SDN (**Software-Defined Networking) - Finally a big value of the DNA system consists of providing a higher order analytics service that is able to operate on the network as a whole instead of individual devices. The key advances in SDN has enabled software controllers that in turn enable network-wide programmability of an "analytics plane" that governs data probes and processing at network elements just like the "control plane" that governs data communication.

In the network management space, there is a tradition of research dealing with delegating simple tasks into the network and distributing corresponding processing. An overview and collection of important work is given in [14]. In the analytics space, one important effort is SCRIPT, a system for decentralized IP flow collection and analysis [15]. It exploits understanding of flow semantics to provide analytics that correlate records about the same flows observed on different routers, without limiting itself to records observed on a single node. It does so using a peer-to-peer overlay that is established between different routers and managed by a central controller, whereas in DNA's case nodes do not need to be aware of one another and only communicate with a controller. SCRIPT is focused specifically on flow analysis and as such processes only flow records. It does not incorporate other telemetry data sources, such as active measurements or interface statistics.

VII. Conclusion

DNA demonstrates how SDN Controllers and programmable, intelligent features embedded inside the network can complement each other to collectively provide significant value to users that could not be provided by either alone. By delegating analytics processing to network devices, DNA not only allows back-end applications to scale better, but to provide analytics in better quality than would otherwise be possible. Perhaps more important than distributing analytics processing itself is the fact that DNA is also able to dynamically set up and adjust network telemetry data sources and queries as required by the analytics task, shielding users from secondary management aspects such as how to set up those sources and managing the deployment of corresponding tasks across the network. We have found that the amount of networking resources consumed by analytics is dominated by the number of generated telemetry data records, while analytics processing itself is fairly negligible, which makes embedding analytics inside network devices even more feasible.

Acknowledgment

DNA is the result of the concerted effort of a larger team. For their contributions and support, we would like to acknowledge Bhaskar Bhar, Fabrizio Corno, Nitish Gupta, Robert Lerche, Billy Liu, Kim Macpherson, Jagadeesh Maiya, Chris Metz, Ahwin Pankaj, Manjunath Patil, Ganesan Rajam, Lukas Sedlak, Shashidar Srinivasa, Anbalagan V, Kiran Vishnubhatla, Jacob Zhang, Joe Zhang, and Yifan Zhang.

REFERENCES

[1] J. Dean, S. Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. 6th Symposium on Operating System Design and Implementation (OSDI'04), San Francisco, CA, December 2004.

[2] Hadoop. http://hadoop.apache.org (accessed on 3/7/2014).

[3] Hive. http://hive.apache.org (accessed on 3/7/2014).

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

[5] Cisco Data Collection Manager. http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/bsdcm/configuration/15-e/bsdcm-15-e-book.pdf (accessed on 3/7/2014).

[6] S. Krishnamurthy, M. Franklin, J. Davis, D. Farina, P. Golovko, A. Li, N. Thombre: Continuous Analytics over Discontinuous Streams. ACM SIGMOD 2010.

[7] Open Daylight. http://www.opendaylight.org (accessed on 9/28/2014).

[8] M. Bjorklund: YANG – A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, IETF, October 2010.

[9] A. Bierman, M. Bjorklund, K. Watsen, R. Fernando: RESTCONF Protocol. draft-ietf-netconf-restconf-04, IETF, January 2015 (work in progress).

[10] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman: Network Configuration Protocol (NETCONF). RFC 6241, IETF, June 2011.

[11] Linux Containers. http://www.linuxcontainers.org (accessed on 9/28/2014)

[12] Cisco's One Platform Kit (onePK). http://www.cisco.com/c/en/us/products/ios-nx-os-software/onepk.html (accessed on 9/28/2014)

[13] B. Claise: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. RFC 5101, IETF, January 2008.

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

[15] B. Stiller, C. Morariu, P. Racz: Scalable and Robust Decentralized IP Traffic Flow Collection and Analysis (SCRIPT). In [14].

[16] E. Boschi, L. Mark,cJ. Quittek, M. Stiemerling, P. Aitken: IPFIX Implementation Guidelines, RFC 5153, 2008.

[17] A. Arasu, B. Babcock: STREAM: The Stanford Stream Data Manager. *IEEE Data Eng. Bull. , 26* (1), 19-26, 2003.

[18] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. S. Sarma, R. Murthy, H. Liu: Data warehousing and analytics infrastructure at facebook. *SIGMOD Conference*, (pp. 1013-1020), 2010.

[19] C. Olston , B. Reed , U. Srivastava , R. Kumar , A. Tomkins, Pig latin: a not-so-foreign language for data processing. *SIGMOD Conference,* 2008.

[20] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H. Hsiao, R. Rasmussen: The Gamma Database Machine Project. *IEEE Trans. Knowl. Data Eng. , 2* (1), 44-62, 1990.

[21] Graefe, G.: Volcano - An Extensible and Parallel Query Evaluation System. *IEEE Trans. Knowl. Data Eng. , 6* (1), 120-135, 1994.

[22] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, P. Valduriez: Prototyping Bubba, A Highly Parallel Database System. *IEEE Trans. Knowl. Data Eng. , 2* (1), 4-24, 1990.

[23] M. Zaharia, T. Das, H. Li, S. Shenker, I. Stoica: Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters,. *HotCloud,* 2012.

[24] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, I. Stoica: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. *NSDI* 2012.

[25] M. J. Franklin, S. Krishnamurthy, N. Conway, A. Li, A. Russakovsky, N. Thombre: Continuous Analytics: Rethinking Query Processing in a Network-Effect World, CIDR, 2009.

[26] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, A. Yu: Mariposa: A Wide-Area Distributed Database System. *VLDB Journal , 5* (1), 48-63, 1996.

[27] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, I. Stoica: Shark: SQL and Rich Analytics at Scale. *SIGMOD Conference* 2013.

[28] S. Ghemawat, H. Gobioff, S. Leung :The Google file system. *SOSP*, (pp. 29-43), 2003.

[29] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah : TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. *CIDR Conference,* 2003.

[30] V. Josifovski, P. Schwarz, L. Haas, E. Lin: Garlic: a new flavor of federated query processing for DB2. *SIGMOD Conference,* 2002.