MicroCloud-based Network Traffic Monitoring

Luca Deri
ntop / IIT-CNR
Pisa, Italy
deri@ntop.org, luca.deri@iit.cnr.it

Francesco Fusco
ETH
Zürich, Switzerland
fusco@tik.ee.ethz.ch

Abstract— Monitoring of large distributed networks requires the deployment of several probes at different network locations where traffic to be analyzed is flowing. Each probe analyzes the traffic and sends the monitoring data toward a centralized management station. This semi-centralized architecture based on the push model is extensively adopted to analyze large distributed networks. However, this architecture presents serious limitations when used to provide real-time traffic monitoring and correlation capabilities across all probes.

This paper describes a novel architecture that addresses the problem of real-time traffic correlation and alerting, by exploiting modern cloud infrastructures. In particular, we propose the adoption of a small-sized cloud to provide a consistent data space that is: i) shared among distinct probes to selectively store monitoring data, and, ii) accessible by external applications to retrieve selected information. We validate our architecture on large distributed networks in the context of DNS traffic monitoring.

***Index Terms*—Distributed traffic monitoring, cloud computing.**

I. INTRODUCTION

NetFlow [1] and IPFIX are the de-facto standard technologies used to monitor network traffic in a passive manner. In the context of network flow monitoring, a flow [2] is defined as a set of IP packets passing through an observation point during a certain time interval, sharing common properties including, but not limited to, ingress/egress interface, protocol, source/destination IP addresses and ports. Flows have a specified lifetime that begins when the first flow packet is received at the observation point, and ends due to timeout or maximum duration. A flow-enabled network probe is deployed in a vantage point to aggregate packets into flows and to produce flow records, which carry statistics about each analyzed network flow. Flow records corresponding to expired network flows are exported by the probe toward a flow collector using a standardized flow export protocol such as NetFlow or IPFIX. The flow export protocol defines the flow-record encoding format as well as the transport protocol (e.g., UDP or SCTP). The flow collector is an application executed on a centralized management station that filters, aggregates and eventually dumps the received flow-records in a persistent database. The standard flow-monitoring paradigm is based on a strict *push model*. The collector passively listens for flow records originated by one or more probes and processes them without any interaction with the sources. Each network probe is totally independent from the rest of the network monitoring infrastructure.

Existing network infrastructures come with built-in flow-monitoring functionalities. In fact, network devices such as switches and routers commercialized by major industry players come with embedded network probes providing some form of flow-based monitoring capabilities. These embedded implementations, which usually provide limited performance and analysis capabilities, have made flow-monitoring the de-facto monitoring paradigm.

Standalone software-based network probes [3], [4] have been introduced to overcome the limitations of embedded probes. They are more flexible and easier to extend than embedded probes and can be easily deployed on standard servers that receive a copy of the network traffic to be analyzed through traffic mirroring ports or network tap devices. In practice, software-based probes have drastically changed the way of monitoring network using a passive approach. In fact, software-based probes have extended the concept of flow record, which is usually limited to packet header fields onto embedded implementations, to the application domain making possible to analyze networking services, such as DNS, VoIP and the web from the application layer point of view. This means that software-based probes have allowed moving the network monitoring task from network-centric to a service and user-centric task.

By broadening the scope of flow monitoring to applications and services, software-probes have made the limitations of the push paradigm more pronounced. The main problem of a strict push model in the context of flow-monitoring is that i) the centralized management station can only have a deferred view of the network, because the probes autonomously decide when to emit flow-records, and, ii) network probes analyze the traffic independently from each other without having the possibility to share information. These limitations prevents timely correlations between network flows originated from distinct network probes and belonging to a single application-layer session to be performed (e.g., correlate signaling and audio traffic in a VoIP session).

In this paper, we highlight the severe limitations that a strict push-model introduces in the context of service oriented network monitoring in large and high-speed networks and we describe practical network monitoring use-cases where the model manifest its limitations. Then, we describe a distributed network monitoring architecture that overcomes these limitations by augmenting the push-model with an *publish-subscribe* mechanism. Our architecture is, in fact, based on a distributed knowledge databased that i) is accessible by every network probe and network collector, and, ii) keeps timely sensitive information for a configurable amount of time. The knowledge database, implemented using modern key-value stores, can be eventually distributed across network nodes to make the system both scalable and resilient.

II. Background and Motivation

Flow correlation is the process of clustering semantically related flow records. For example, correlation is required to associate flow-records corresponding to the two traffic directions (i.e., IP source-to-destination and destination-to-source) that are often analyzed by distinct network-probes due to asymmetric IP routing, MPLS, layer 2/3 VPNs, or traffic balancers. In a flow-monitoring architecture based on standard NetFlow/IPFIX technologies, flow correlations can only be performed by the collector and are usually implemented as queries over the database back-end storing the entire flow-record stream. This way of performing correlations presents two main shortcomings: i) the storage backend is constantly taxed with periodic queries, and, ii) the correlations can only be computed when the flow-records are received by the collector making the architecture unsuitable for real-time monitoring.

Software-based probes running of commodity hardware have replaced embedded network probes primarily because they provide the flexibility and extensibility required to analyze modern network traffic flowing on 10 Gbit links using a service and user-centric monitoring approach. In fact, thanks to recent academic and industrial research [5], [6], [7] they are able to process packets at line-rate in 10 Gbit links and even to perform traffic analyses up to the application protocol by means of Deep Packet Inspection (DPI) [8]. Additionally and in contrast with probes embedded into existing network devices, software-based probes provide the extensibility required to support emerging encapsulation protocols such as Generic Route Encapsulation (GRE), and Mobile IP. These encapsulations techniques have become widely used in 10 Gbit links as they are commonly used as backbone links to consolidate traffic originated by heterogeneous networks.

In summary, software based probes made current flow-monitoring architecture more flexible, capable of supporting a large number of encapsulation protocols and of performing complex application level traffic analyses. However, the replacement of embedded probes with standalone probes do not enable real-time analysis and low latency correlations, which are extremely important especially when analyzing networked services from the application protocol point of view. Flow-correlation is required, for example, in the context of Voice over IP (VoIP) traffic analysis, where flows-records corresponding the signaling traffic have to be correlated with the corresponding media flows to associate call quality with VoIP peers. Similar correlation is desirable on networks where users are identified using protocols such as Radius (RFC 2865), so that flows can be automatically associated with users that authenticated to the network. DNS and SNMP flow analysis also requires the probe to handle the sub-flow identifier such as DNS transaction identifiers and SNMP request identifiers.

In this work we propose a novel architecture that unlike the standard flow-monitoring architecture can enable low-latency aggregations and real-time monitoring. At the core of our architecture there is a distributed knowledge-database called MicroCloud that contains updated network monitoring information. Our architecture is based on a collaborative model: each probe enrich the database with up-to-date monitoring information that is quickly available to the other architecture components to correlate flows together and to associate traffic with the originating user. This architecture overcomes a limitation of many monitoring tools that are IP/MAC address centric instead of user-centric: users think in terms of services and identities, whereas IP and MAC addresses are intermediate low-level information used by computer to communicate.

III. MicroCloud Monitoring Architecture

Our MicroCloud-based network monitoring architecture is depicted in Figure 1. It is composed by network probes based on nProbe [3], an open-source NetFlow/IPFIX probe written by the authors and by a MicroCloud consisting of one or more nodes. Each MicroCloud node, which is part of a distributed knowledge database, runs an instance of a key-value store. Network probes can emit flow records related to the analyzed traffic toward a centralized management station or update the knowledge database by setting keys in the cloud nodes. Monitoring applications can access the monitoring information available both in probes and cloud nodes using an unified interface, which allows to subscribe to specific events or to retrieve information using a polling mechanism.

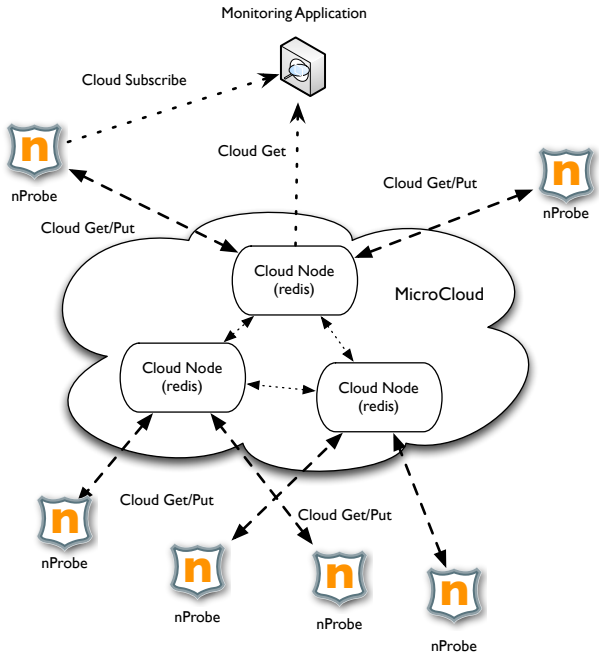

Fig. 1. MicroCloud Architecture

A. The network probes

The core of the monitoring infrastructure is based on nProbe, which is an advanced network probe capable of analyzing traffic, classifying it into flows, and emitting the corresponding flow-records according to the user-specified template, i.e. nProbe supports flexible NetFlow in the Cisco parlance. nProbe leverages a Deep Packet Inspection (DPI) library named *nDPI* (http://www.ntop.org/products/ndpi/) to recognize more than 140 application layer protocols including Skype, BitTorrent, WebEx, Twitter and Facebook. Additionally, we have made nProbe capable of dissecting traffic

encapsulated in commonly used encapsulation protocols. nProbe is extensible by means of plugins. Nowadays, nProbe comes with plugins for analyzing email traffic (SMTP, IMAP, and POP3), database traffic (MySQL and Oracle) and HTTP traffic. Plugins extract from network flows application-specific metadata such as URL and return code in the case of the HTTP plugin, with the purpose of providing a rich monitoring experience. For example, nProbe can report the request service time, i.e. the application delay and network latency for each URL so that network administrators do not have aggregate performance values, but rather fine-grained information extremely useful to pinpoint specific performance issues.

However, by using a standardized monitoring architecture based on NetFlow/IPFIX, the rich set of traffic monitoring information that nProbe can derive by dissecting live traffic can only be visible to network administrator when flow records expires. For example, in the context of HTTP traffic monitoring, an URL would be available only when the corresponding network connection is terminated, or when a maximum timeout has elapsed.

B. MicroCloud Nodes

MicroCloud nodes are database nodes where network probes can store time sensitive monitoring information for a configurable amount of time. MicroCloud nodes have to sustain high insertion rates and yet provide low response times when queried from monitoring applications. Modern databases implementing the key-value store paradigm represent a good option to achieve the goal as they are known to provide higher insertion rates than commonly used relational databases trading query expressivity for performance. Data stored in the cloud is uniquely identified by a key. Network probes update the knowledge database by inserting key-value pairs. After evaluating possible key-value store alternatives we decided to built our architecture on top of the key-value store called *Redis* (http://www.redis.io) for the following reasons: i) it provide persistence, so that data is preserved across restarts as on standard databases, ii) it natively supports the publish/ subscribe paradigm, so that cloud participants can subscribe to have information about relevant monitoring events, iii) it can be accessed with many languages and uses a simple client-server protocol. Monitoring applications can access the this information by querying for specific keys. In Redis, the list of available keys can be retrieved using patterns. For example, the command KEYS ip.192.168.* can be used to retrieve a list of keys starting with the specified prefix. To exploit this functionality we organize the information hierarchically (i.e., as multiple trees). Within each MicroCloud node there are many distinct groups (or trees):

- MAC, IP, VLAN and application protocols group. On these groups the keys are unique by semantic (e.g. an IP is unique). In case the same IP is seen multiple times (e.g. the same IP on two different VLANs) the value associated to that key holds the information (e.g. <vlanA>.bytesSent, <vlanB>.bytesRcvd and so on).
- nProbe plugin-related groups. Each plugin that saves data into the cache can do it both enriching the above group (e.g. and host X sends a DNS request, the DNS plugins increments the value of dns.queries attribute belonging to host X), and creating specific hashes. Please note that all plugins contribute to enrich the cache by setting the information they learn from traffic such as the operating system type and version (e.g., the HTTP plugin can extract the information by parsing the user-agent field).

Keys can be persistent or volatile. Persistent keys are never flushed from the cloud unless explicitly requested by a probe. These keys can for example be used to store mappings from users to IP addresses. In contrast, volatile keys, which are used for example to store traffic counters for a specific host are automatically removed from the database if they are not updated for a configurable amount of time.

C. Updates and Queries to the MicroCloud

Promptly updates are mandatory to enable real-time monitoring. Therefore, each probe update knowledge databases without waiting for the corresponding flow to be expired when time sensitive information has to be made available. More in detail each probe updates the MicroCloud when:

- A user identified using radius (dis)connects to the network. The cloud is immediately updated when a user is discovered. In contrast, when an user has to be deleted the corresponding entries are only marked for deferred deletion based on a timeout. In this way, flow records still present in the flow cache can still be matched with the user information.
- Depending on the probe configuration, when a flow expires the probe can rely on the cloud to discover the user associated with such a flow. In case of sequential flows (e.g. multiple requests on the same HTTP 1.1 connection), the user is computed on the first flow and cached until the end of the connection.
- A flow expires, to update the database entries including flow peers, ports and protocols with the flow information. For instance when a HTTP flow from host X to Y expires, the probe updates the cloud counters corresponding to host peers by incrementing the corresponding bytes and packets.

  As each probe can set multiple keys per each flow, nProbe opens two communication channels with a MicroCloud Node instance. The first channel is synchronous and used to get information from the cloud (e.g. read the user name associated to a given IP), whereas the second is asynchronous and used to set or delete data from the cloud. We decide to use two distinct channels to accommodate for distinct requirements: low latency and high throughput. The synchronous channel is used by the probe to get replies with low latency for the information that has to be immediately received. In contrast, the asynchronous channel is used to enqueue requests that do not need to be performed in real-time. By grouping together requests and by executing them in batches network bandwidth can be saved.

Flow information that needs to be constantly updated for each analyzed packet, such as the number of bytes, is never stored in the MicroCloud nodes. In fact, even if the communications with MicroCloud nodes are very efficient,

updating a node for each received packet is not practical as it would cause significant network traffic and would increase the load on the nodes. To make this information available sooner to monitoring applications, we have implemented into nProbe a subset of the Redis protocol. In this way, a network probe can be queried by monitoring applications for accessing the information stored within its flow cache using the same interface that can be used to query MicroCloud nodes.

The separation between network probes and knowledge database implemented by the proposed architecture offers new interesting possibilities. Our architecture is in fact open, as it allows third-party network monitoring applications to be easily developed. Such applications can be coded using one of the languages supported by the Redis client, including scripting languages such as Perl and Python. In this way, highly modular applications can implemented by software engineers without a deep networking background. It is in fact possible to code simple scripts solving very specific requirements such as emitting alerts whenever a user has accessed specific URLs, and creating time series of selected traffic metrics.

It is worth to remark that the MicroCloud does not have to be perceived as a persistent database but rather as a distributed and up-to-date cache constantly enriched by multiple agents. We believe that the knowledge cache enables applications that go beyond network monitoring.

IV. MicroCloud Validation

The MicroCloud-based architecture is currently deployed in a large production network to perform DNS traffic monitoring. In particular, nProbe is at the core of a DNS traffic monitoring system that analyzes the italian (.it) DNS registration service since a couple of years. The .it ccTLD relies on seven DNS nodes some of which using anycast addressing. Figure 2 depicts the architecture of a typical DNS monitoring node. In our setting, a typical monitoring node handles more than 60 million queries per day with peaks of a few thousand queries/ sec. Inside the .it DNS network, the MicroCloud is currently deployed on three national DNS nodes, and soon it will be extended to the rest of the nodes.

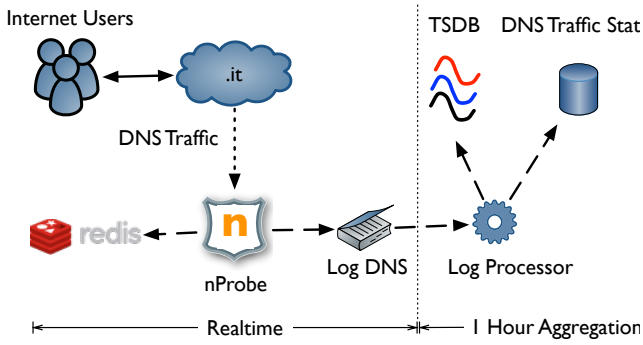

Fig. 2. MicroCloud DNS Monitoring System for .it: Node Architecture

Before the deployment of our MicroCloud-based system, DNS monitoring was performed by analyzing DNS flow traces generated by nProbe in real-time [9]. As performing aggregations is a computationally expensive task, aggregations were only performed once a hour over hourly DNS traces. This approach not only was unable to provide monitoring information in real-time, but was also causing packet losses during the analysis due to the high system load. Our MicroCloud architecture has solved the problem. In fact, thanks to the batch-based update system implemented into nProbe, the updates to the MicroCloud node do not slow down the processing of the DNS traffic. In addition, it is finally possible to analyze the DNS traffic in near real-time, as the latency between a DNS response received by the probe and the Redis database updated is around one second. This is an impressive achievement compared to the one hour delay we had in our previous monitoring architecture.

Additionally, the MicroCloud architecture had enabled the quick implementation of simple real-time monitoring applications. For example, we can now monitor DNS queries made by suspicious IP addresses that have been reported by CENTR, the council of TLD domain registries, in real-time.

V. Final Remarks

This paper has presented a novel architecture that enable the implementation of real-time traffic correlation and monitoring, as well distributed alerting. Each monitoring node communicates with a small-sized cloud that acts as a distributed consistent memory cache where monitoring information is maintained. Traffic probes enrich the cloud by storing into it information about hosts, protocols, and user-to-IP mapping. The information is easily accessible for both network probes and monitoring applications, which can be implemented with little effort using scripting languages. Although this paper focuses on traffic monitoring, the concept of the microCloud has a broader scope as it can be applied also to other areas of networking including management and security.

References

[1] B. Claise, “Cisco Systems NetFlow Services Export Version 9.” RFC 3954 (Informational), Oct. 2004.

[2] [2] B. Claise, “Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information.” RFC 5101 (Proposed Standard), Jan. 2008.

[3] L. Deri, “nProbe: an Open Source NetFlow probe for Gigabit Networks,” in *In Proc. of Terena TNC 2003*, 2003.

[4] P. Lucente, “pmacct: steps forward interface counters,” tech. rep., 2008.

[5] ntop.org, “Benchmarking PF_RING DNA” http://www.ntop.org/pfring/benchmarking-pfring-dna/, 2012.

[6] L. Rizzo, “Netmap: a novel framework for fast packet I/O,” in *Proc. of the 2012 USENIX Annual Technical Conference*, 2012.

[7] Intel, “Data Plane Packet Processing on Embedded Intel Architecture Platforms.” http://download.intel.com/design/intarch/papers/322516.pdf, 2009.

[8] M. Becchi, M. A. Franklin, and P. Crowley, “A workload for evaluating deep packet inspection architectures,” in *4th International Symposium on Workload Characterization (IISWC 2008)*, pp. 79–89, 2008.

[9] L. Deri, L. L. Trombacchi, M. Martinelli, and D. Vannozzi, “A dis- tributed dns traffic monitoring system,” in *Proc. of th 8th Int. Conference of Wireless Communications and Mobile Computing*, pp. 30–35, 2012.