Realtime MicroCloud-based Flow Aggregation for Fixed and Mobile Networks

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

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 often using protocols such as NetFlow and IPFIX. As each probe monitors a part of the traffic, the data collector has the responsibility of merging data coming from all the probes and correlate information. This task adds extra load on collectors and prevents traffic information to be available until it has been correlated, thus preventing (near) realtime traffic monitoring.**

This paper describes how the microcloud architecture can be used to provide real-time traffic monitoring and correlation on large distributed environments where monitoring traffic is analyzed by several probes that collectively concur to the monitoring task. This work has been successfully validated on using this architecture for monitoring the .it DNS ccTLD and a large 3G mobile network with million of users.

***Index Terms*—Real-time Distributed traffic monitoring, NetFlow/IPFIX, DNS, Mobile Networks.**

I. Introduction

In NetFlow [1] and IPFIX, 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 that runs on a centralized management station and is responsible to filter, aggregate and eventually dump flows in a persistent database.

Flow-enabled network probes are commonly available in existing network infrastructures. In fact, major vendors include some form of flow-based network monitoring capabilities in devices such as routers and switches, although quite often such embedded implementations have several limitations both in terms of performance and analysis capabilities. If traffic analysis functionalities are not provided by the existing network infrastructure, it is possible to augment it with additional software-based probes [3, 4]. These probes are deployed on standard PCs that receive a copy of the network traffic to be analyzed by means of a span port or a network taps. Software-based probes have drastically changed the way of monitoring network using a passive approach. In practice, their flexibility has allowed moving the network monitoring from a network-centric to a service- and user-centric task. 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.

Service-oriented probes have also promoted the *push* paradigm, which is the model behind the flow-record monitoring, to its limits, especially in the context of large networks. In fact, we believe that by broadening the scope of flow monitoring to applications and services, software-probes have only made the limitations of current flow-based network monitoring architectures more visible. In a strict *push* model, it is the probe to decide when to export a specific flow record depending on the traffic condition and on the flow status. The collector(s) passively listens for flow records coming from one or multiple probes and process them without any interaction with the sources, whereas the analysis performed by each network probe is totally independent from the rest of the network monitoring infrastructures. The main problem is that the centralized management station can only have a *deferred* view of the network: the observed delay is proportional to the lifetime of each flow and not acceptable to perform real-time network monitoring. The delay 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). Similar limitations are also encountered when dealing with encapsulations. The lifetime of each flow can be arbitrarily reduced to decrease the delay, but this comes at a cost of higher flow rates observed at the collector side.

In this paper, we propose a network monitoring architecture that combines the push-model and a *publish-subscribe* mechanism to overcome these limitations. The architecture introduces a distributed knowledge database that i) is accessible by every network probe and network collector, ii) keeps timely sensitive information, or events, for a configurable amount of time. This knowledge database is implemented as a cache that can be eventually distributed across network nodes to make the system both scalable and resilient. We main paper contributions include:

- We show that the push model presents serious

limitations when used in the context of service oriented network monitoring in large and complex networks.
- We identify use cases where the current flow-based monitoring infrastructures are unsuitable to implement real-time monitoring tasks.
- We highlight that similar problems are also encountered when the divide-and-conquer processing paradigm is used to exploit modern multi-core architectures.
- We propose a novel architecture based on modern key-value stores to solve the aforementioned issues.

The rest of the paper is structured as follow. Section 2 describes the background and the motivation of our work. Section 3 describes the proposed architecture, which is evaluated in Section 4 against two real network monitoring scenarios. Section 5 highlights some open issues, future work items and extensions for the measurement architecture described on this paper. Finally, Section 6 concludes the paper.

II. Background and Motivation

Software-based probes have been originally preferred to embedded probes to avoid stressing the existing network infrastructure with additional load and to have a clean separation between production networks and the network monitoring infrastructures responsible to analyze their traffic. However, software probes are becoming more and more attractive than probes embedded in switches and routers for at least three reasons:

- **Most embedded probes are only capable to analyze the network traffic up to the packet header.** Although some years ago this was a common practice, today most of the emerging companies offer products that through DPI (Deep Packet Inspection) [6] are able to characterize the application protocol, trigger immediate flow export rather to wait the flow to expire when conditions are met (e.g. a used connected to the network), and thus report it on exported flows. This is a mandatory feature for accounting network traffic, as relying on TCP/UDP ports for detecting the application protocol is not dependable anymore. Throughout this paper we use the term DPI to characterize the analysis of packet payload for the purpose of detecting the application protocol and decoding specific protocol messages.
- **Most embedded probes are not able to analyze 10 Gbit traffic carried on network backbones without relying on sampling techniques.** Sampling can happen both at packet and flow level [5]. When using packet sampling, the probe receives fewer packets and thus the load on the probe is reduced but not the number of computed flows with little relief on the collector. When using flow sampling, the probe analyzes all incoming packets but exports only a subset of the flows thus reducing the load on the collector. In both cases, sampling leads to inaccurate traffic analysis and accounting and thus it is used very seldom by network operators. Furthermore the use of sampling makes DPI and protocol analysis hard to implement, as not all protocols can be detected when sampling is used.
- **Most embedded probes only support limited encapsulations.** Encapsulated traffic is becoming pervasive due to the use of protocols such as GRE (Generic Route Encapsulation), Mobile IP, PPP (Point-to-Point) and GTP (GPRS Tunneling Protocol) [9]. Most probes limit their scope to VLAN and MPLS tagging, which is not sufficient to enable the analysis of user traffic in modern backbones. It is worth noting that supporting encapsulations does not just mean that the probe is able to recognize additional packet headers, but also that the probe is able to merge up/down-stream tunnels as in the case of GTP.

Flow correlation is usually performed at the collector side, where all the flows are received and typically stored in a database. The correlation process is both time and resource consuming: for each flow to be added into the flow database a correlation query needs to be performed. The result is that the load on the database is increased as well the collection latency and throughput. Flow based technologies, such as NetFlow/IPFIX are not suitable for near real-time traffic monitoring because flow records have a lifetime that can range from tenth of seconds to a few minutes. This criticism is the reason why in the past couple of years products such as Riverbed Cascade Pilot have tried to overcome the lack of real-time monitoring by creating custom probes not based on the flow paradigm. In our opinion this criticism is correct, but at the same time we believe that there is a value in supporting the NetFlow/IPFIX.

In summary modern flow-based traffic monitoring requires:

- Ability to monitor traffic at 10 Gbit speed while supporting the most popular traffic encapsulations, including mobile network traffic.
- Ability to partition traffic monitoring both across systems and available processor cores, while being able to correlate data produced by the various probes.
- Support of distributed traffic monitoring where multiple network probes, each analyzing a portion of the overall traffic, cooperate to the same global monitoring goal.
- Inspection of packet payload for identifying the application protocol and characterizing the traffic with metadata (e.g. compute the HTTP return code) information that enables monitoring of real user experience and detailed reporting of errors.
- Real-time correlation of traffic seen by distinct probes for associating users with flows (e.g. user X has issued a,b,c HTTP requests), and cross-flows correlation (e.g. a RTP flow with its corresponding SIP signaling session).
- Creation of a near-realtime knowledge base containing the main metrics of monitored hosts and protocols,(e.g. bytes and packets), so that it is possible to see what is happening on the network on near-realtime without having to wait flow expiry.

These requirements have been the motivation for this work, as we have not found in the open-source community or in commercial products, a traffic probe able to support all these features while being:

- Moderate in usage of computing resources so that it can run on both small embedded devices and high-end servers.
- Open source, which we believe is a value, in particular when monitoring telecommunication networks that are still mostly based on proprietary software.
- Extensible by end-users by means of plugins for creating dissectors for new protocols.
- Able to monitor 10 Gbit networks on commodity hardware without using any custom network adapter.

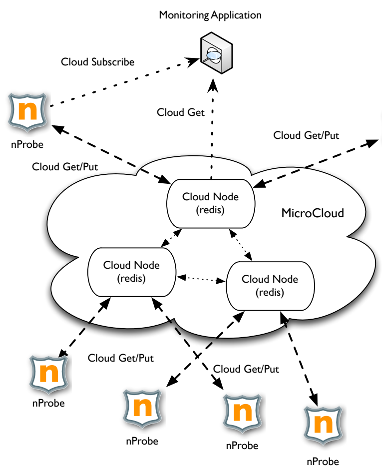
The core idea behind this work is the need to create a distributed and constantly updated knowledge database for network monitoring, made of a small (in size and number of nodes) cloud, that we call *microCloud*. Each probe accessing the cloud has an active role in enriching it with the monitoring data it analyzes, and at the same time fetches from the cloud the information necessary for correlating flows together, and associating users their traffic often known as subscriber awareness. 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.

The microCloud however is not just a information correlation technology but rather a short-term database where data is stored and used on a collaborative fashion. Every component is responsible for enriching it by adding the information it sees, and it can exploit the cloud for accessing in real-time to data that would simplify its task. For instance by extending this principle at security devices such as IPS/IDS it would be possible to propagate security-related information in real-time across all cloud members and thus execute specific actions. For instance when an IPS detects that IP a.b.c.d is sending suspicious traffic, it might mitigate it and report this information to the cloud where a packet-to-disk application might be listening and dumping to disk packets of such suspicious IP. In a way the microCloud is a collaborative real-time mechanism that promotes collaboration and information sharing across its members. Within the scope of this paper, we limit our analysis to traffic monitoring but the concept we present is very general and applicable to networking in general.

III. MicroCloud Monitoring Architecture

The microCloud is made of one or more nodes, where each node is based on a key-value database named Redis [10] we selected because:

- The Redis database engine is very fast. A single database can serve about 100k requests/sec from multiple clients. A single client can perform over 50k key/value set/sec.
- Unlike memcached, Redis is persistent, so that data is preserved across restarts as on standard databases.
- Redis supports (limited) clustering, data replication and migration. Unlike other databases where multi-node deployments are compulsory, in Redis this is an optional feature that can be considered only for large systems. This makes Redis an ideal solution for both small embedded monitoring probes and large, multi-CPU systems.
- The communication protocol between a Redis client and the server is very simple so that we can implement it also on network probes as explained later in this section.
- Redis supports publish/subscribe, so that we can propagate relevant monitoring data (e.g. a user dis/connected to the network) to all cloud participants that want to be informed about specific events by leveraging on this information distribution mechanism.
- Redis is open-source, its code is small in size, well written and documented, and supported by a large community. Unlike similar solutions such as Hadoop that is Java-only, Redis supports clients written in most programming languages significantly easing its usage also on existing monitoring environments.



1. MicroCloud Architecture

Each monitoring probe is based on nProbe [7], an open-source NetFlow/IPFIX probe written by the authors. The nProbe core is responsible for analyzing traffic, classifying it into flows, and emitting them according to the user-specified template, i.e. nProbe supports "flexible netflow" in the Cisco parlance. The core of nProbe implements packet header parsing (it supports all the popular encapsulations), and a flow cache that stores traffic information. nProbe implements DPI natively as it leverages the nDPI framework (http://www.ntop.org/products/ndpi/), an open-source DPI framework based on a fork of a open-source framework named OpenDPI which is no longer available. nDPI recognizes more than 160 protocols including Skype, BitTorrent, WebEx, Twitter and Facebook. Thanks to nDPI, nProbe can detect the application protocol (i.e., protocol detection is not based on ports) and thus export this information on flows. nProbe is also extensible by means of plugins and provide users with dissection plugins for several popular protocols including HTTP(S), GTP-C v0/v1/v2, Database (MySQL, Oracle), Email (SMTP, IMAP, and POP3), VoIP (SIP and RTP), DNS. Plugins extract from network flows domain-specific metadata (e.g. URL, and return code in HTTP, SQL query on databases plugins) with the purpose of providing a rich monitoring experience and report errors along with its context. For instance nProbe can report the request service time (application delay) and network latency (network delay) for each URL so that network administrators do not have aggregate performance values, but fine-grained information that can be used to pinpoint specific performance issues. Tracking protocols such as radius and DNS cannot be done by analyzing the 5-tuples only. To address these cases, nProbe supports something we called sub-flow identifier that uniquely identifies a communication inside a flow (e.g. the transactionId on DNS, or packetIdentifier on Radius). Tracking GTP-U (i.e. the traffic of mobile users on a 3G/LTE network) traffic is even more complicated as the signaling traffic GTP-C (i.e. , GTP-C) specifies the tunnels identifiers (at least four) used to determine where the GTP-U traffic for a specific mobile user will be flowing. As explained below, this tunnel information needs to be persistently stored into the cloud as:

- Depending on the network topology and user roaming inside the mobile network the traffic of a specific user can be observed in distinct vantage points, and therefore, probes need to have access to tunnel information corresponding to each observed user. Therefore, this information has to be stored on the cloud.
- This information can last very long time and thus it has to be persistent across probes restart.

Each probe enriches the cloud by:

- Adding/removing user mappings whenever a user (dis)connects to the network (e.g., using Radius or GTP).
- 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.
- When a flow expires, the probe updates the database entries including (but not limited to) flow peers, ports and protocols with the flow information.

The data types. The cloud contains two type of information: *persistent* (e.g. user to IP mapping) and *volatile* (e.g. host X traffic). The persistent information is never flushed from the cloud unless a probe explicitly requests that. There are however cases where the information should stay in the cloud for long time frames (e.g. a 3G modem controlling river water level is powered when installed and turned down when replaced, so its registration stays on the cloud for the unit lifetime that can even be years) so it is important that the cloud information is maintained for long time and not automatically flushed by timeout. In contrast, volatile data such as host traffic can have a retention period after which if the data is not updated, it is automatically removed from the cache. This harvesting mechanism, implemented through the Redis TTL (time-to-live) command, is necessary to purge from the cache stale information that is no longer required.

The data model. Data stored in the cloud is uniquely identified via a key. Users familiar with the SQL language, will probably be disappointed by the lack of the WHERE clause (e.g. SELECT X where Y). However, Redis allows keys matching given pattern to be listed (e.g. KEYS ip.192.168.*). This is not a real limitation, as probes do not need to glance through the cache data but they rather update information uniquely identified with a unique key (e.g. IP and VLAN). Inside the cache, the information is organized hierarchically in several groups:

- MAC, IP, VLAN and application protocols group. On these groups the keys are unique by definition (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. <vlan A>.bytesSent, <vlan B>.bytesRcvd and so on). This is because for each key we do not associate a single value but rather a hash (or list/set) containing several unique fields.
- 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. For instance the GTP plugin creates a specific dictionary that contains the GTP tunnel information for a specific user, whereas the radius plugin adds the dictionary field "username"=<user id> to the IP address present on the IP group. 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).

Promptly updates are mandatory on real-time monitoring. Therefore, as soon as a probe has to report important information (e.g., a user authenticated with radius), this data is immediately placed on the cloud without waiting for the corresponding flow to be expired. As each probe can set multiple keys per each flow, nProbe opens two communication channels with a Redis 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 one 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 as the requested information is necessary immediately. In contrast, the asynchronous channel carries requests that do not need to be performed in real-time so that they can be grouped together and executed in batches to reduce the network communications.

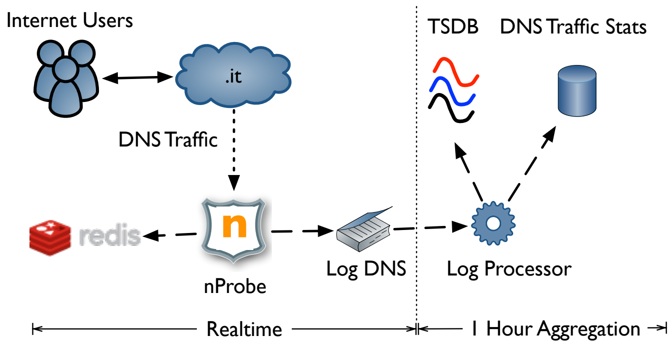
Although communications with a Redis node are very efficient, nProbe cannot update the cache for each received packet, as this would be too costly. Therefore, the microCloud can only offer an alternative view of the traffic, which is near-realtime. However, we have implemented in nProbe a subset of the Redis protocol. In this way, in addition of being a cloud client, nProbe can also act as a Redis node that can be queried by client applications for reading in real-time global traffic counters (e.g. number of bytes/packets) and for accessing the flow cache. In the future, we plan to implement a REST interface that provides the same information to web-based applications (e.g. an ajax-based realtime graph) so that they can access nProbe data natively without having to use the Redis protocol. The separation between network probes and knowledge base implemented by the proposed cloud-based architecture offers new interesting possibilities. Our architecture is in fact an open architecture that allows third-party network monitoring applications to be easily developed. Such applications can be coded using one of the Redis client bindings, including scripting languages such as Perl and Python. In this way, monitoring applications can be highly modular and implemented by software engineers without a deep background networking background. It is in fact possible to code simple scripts solving very specific requirements such as emit alerts whenever a user has accessed specific URLs, and periodic dump of selected traffic metrics onto time series (e.g. RRD files). It is worth to remark once more that the microCloud should not be perceived as a persistent database but rather as a data cache where all the network knowledge is stored and to which multiple agents can have access for enriching it or reading data from. The Redis security and authentication mechanisms prevent that unprivileged clients can delete or corrupt the cache, and restrict data access according to the specified policies.

IV. MicroCloud Validation

The microCloud has been validated in two different production environments where it is running since a few months. The goal is to verify the usage of the microCloud in real life, and check if its usage has negative dependencies on the probes in terms of increased load or packet drops.

A. DNS Traffic Monitoring

nProbe is used since a couple of years as the cornerstone of the DNS traffic monitoring system used to monitor the .it DNS registration service.



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

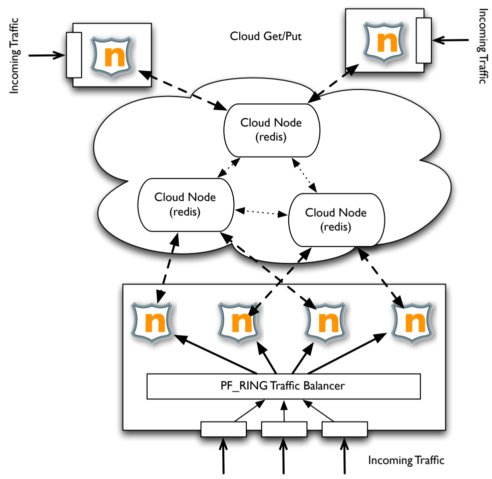
The .it ccTLD relies on seven DNS nodes some of which using anycast addressing. The following figure depicts the architecture of a typical DNS monitoring node. The existing monitoring system [11] was relying on DNS flow traces generated by nProbe in realtime. As aggregation is a computationally expensive activity, it was performed once a hour leveraging on traces produced on the past hour. The drawback of this solution is that it is not possible to see what is happening in realtime, and also that during the analysis, which lasts for more than 15 minutes, the system load was high enough to lead to packet drops on the probe and reduced response time to the web monitoring console. The new microCloud architecture has overcome all these limitations by allowing us to have a realtime view of the DNS system, while enabling the creation of simple realtime applications. For instance 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 A typical monitoring node handles more than 60 million queries/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 remaining nodes. Each monitoring node is physically located inside the core network of IXPs (Internet eXchange Points). Due to colocation contracts, the Internet connection cost of such nodes is flat for traffic from/to the Internet but it's on volume for traffic from/to each node to the .it DNS network. For this reason we have decided not to have a single microCloud with nodes speaking each other, and thus each node has its own local cloud; this would preserve the Internet bill with the disadvantage of not having a single distributed cloud. Whenever we produce live reports or dump aggregate statistics across all nodes, we query all the monitoring clouds from a central point in order to produce an aggregated view of the .it DNS traffic.

In terms of performance, the batch Redis update system implemented in nProbe guarantees that Redis updates do not slow down DNS processing. Considering all data caching, update performed in batches and DNS protocol handling, the latency between a DNS response received by the probe and the Redis database updated is around one second. We believe that this is a great result as it enables us to have a near real-time view of the networks with a simple architecture. For real-time traffic view, it is possible to query directly each nProbe via the redis protocol whose a subset has been implemented into the probe as previously explained.

B. 3G/LTE Traffic Monitoring

The microCloud is successfully used to monitor 2G/3G traffic of the Bulgarian Telecom (VIVACOM) mobile network. Each monitoring node receives a copy of the traffic as seen on the Gn interface, which is where GTP-encapsulated mobile subscriber traffic is flowing [9].



3. MicroCloud-based 2G/3G Monitoring System: Node Architecture

As traffic can be received on multiple ingress interfaces due to network topology or due to the adoption of network taps, it is first necessary to merge traffic together. This task is carried on by the PF_RING [8], an open-source Linux kernel module that has been originally designed to accelerate packet capture and that nowadays provides several packet-balancing facilities including packet clustering. Thanks to packet clustering, incoming packets belonging to GTP tunnels of the same mobile users are merged and sent to the same nProbe instance. In this way, up- and down-stream directions of the same flow are monitored by the same probe. Similarly, GTP-C traffic is shared across all probes in order not to overload a single probe with signaling and to avoid that in the unlucky case of probe crash, all the signaling traffic analysis is stopped. The GTP encapsulation is handled by the nProbe core, whereas we have developed a new plugin for handling GTP-C that is responsible to propagate subscriber information into the cloud. Such microCloud maintains information about GTP tunnels, and subscriber awareness by mapping users to traffic. nProbe has been enhanced to export this information via NetFlow/IPFIX using a new information element named FLOW_USER_NAME. A similar feature is supported by nProbe for radius traffic. This means that for all flows, nProbe searches the traffic-to-user mapping onto the microCloud.

The node architecture deployed on Figure 3 is currently monitoring VIVACOM's 2G and 3G networks (in additional to a 4G/LTE testbed) and thanks to PF_RING balancing it has been possible to balance the traffic across cores and thus handle the input traffic rate without dropping packets. Due to the distributed network topology, multiple servers are used to handle traffic that is monitored on different locations. However the cloud is unique as all nodes refer to the same cloud, and it is not partitioned across locations. This is important as on mobile networks, users move inside the network and thus a give user can be seen at different monitoring points depending on its current physical position onto the network and type of handheld used (e.g. 2G vs. 3G). The microCloud is an ideal solution for taking into account these cases as regardless of the monitoring point, it is always possible to locate the correlation information if known to the cloud. Another advantage of this architecture is that at any time it is possible to know the active users on the network, their GTP tunnels (this information is necessary for intercepting user traffic for instance for troubleshooting issues) and traffic type/protocols in realtime. Using a conventional database-based architecture, all this information would have been available only after aggregation and thus not in real-time as it happens with the microCloud.

V. Open Issue and Future Work

Although operational and used in production networks, the microCloud concept is still an on-going work under active development. We acknowledge that Redis is the most suitable open-source infrastructure available, but we are aware that there is still significant work to be done to extend the cloud infrastructure with additional functionalities. For instance, we would like to introduce into the cloud a long-term storage system that allows us to maintain a historical view of key metrics and counters. We are also trying to address some limitations of the current Redis design. One feature that we would like to introduce in Redis is the support for hooks to notify applications when specific events happen (e.g. a key is purged from memory due to its TTL).

On the networking side, we are introducing the support for microCloud to all network components we are developing (i.e., traffic load balancers, application firewalls, and new specialized nProbe plugins). The idea is that each network component should contribute to the microCloud by exporting into it traffic counters, configuration information and any metadata information that can be useful for the purpose of network traffic monitoring and network awareness.

VI. Final Remarks

This paper has presented a novel architecture that implements 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. Overcoming the limitation of database-based monitoring systems, this architecture guarantees traffic counters consistency even if multiple probes monitor a portion of the same traffic. Furthermore it enables the creation of simple real-time monitoring applications that can use the data stored on the microCloud to accomplishing management functions that until now would have been implemented as monolithic and hard to maintain traffic monitoring applications. 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.

Acknowledgment

Our thanks to Italian Internet domain Registry that has greatly supported this research work, and VIVACOM Bulgaria Telecom that has provided us country-wide infrastructure for deploying the software described on this paper.

References

1. B. Claise, Cisco Systems NetFlow Services Export Version 9, RFC 3954, October 2004.
2. B. Claise, Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information, RFC 5101, January 2008.
3. L. Deri, nProbe: an Open Source NetFlow Probe for Gigabit Networks, Proc. of Terena Network Conference, 2003.
4. L.Deri at al., A Distributed DNS Traffic Monitoring System, Proceedings of TRAC 2012 workshop, August 2012.
5. T. Zseby and others, Sampling and Filtering Techniques for IP Packet Selection, RFC 5475, March 2009.
6. M. Becchi, M. Franklin, and P. Crowley, A workload for evaluating deep packet inspection architectures, Proceedings of IISWC 2008, September 2008.
7. B. Newport, Evolving the Key/Value Programming Model to a Higher Level, Proceeding of Qcon Conference, 2009.
8. F. Fusco and L. Deri, High Speed Network Traffic Analysis with Commodity Multi-core Systems, Proceedings of IMC 2010, 2010.
9. 3GPP, General Packet Radio Service (GPRS); Service Description, Stage 2, Technical Specification 3GPP SP-56, V11.2.0, 2012.