Resource monitoring in a Network Embedded Cloud: An extension to OSPF-TE

Amir Roozbeh
and Azimeh Sefidcon
Ericsson Research, Packet Technology
Stockholm, Sweden
Email: {amir.roozbeh,azimeh.sefidcon}@ericsson.com

Gerald Q. Maguire Jr.
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden
Email: maguire@kth.se

***Abstract*—A "network embedded cloud", also known as a "carrier cloud", is a distributed cloud architecture where the computing resources are embedded in, and distributed across the carrier's network. This idea is based on a transformation of carrier-grade networks toward a platform for cloud services and the emergence of cloud computing in a telecommunication environment. In this model the network's resources are simply another set of cloud resources. This changes the resource management problem as compared to resource management inside a data center. In order to meet applications' requirements regarding available bandwidth and computing resources, one needs to consider link conditions and traffic engineering when allocating resources in a network embedded cloud. This paper presents a resource monitoring solution for a network embedded cloud by proposing an extension to the Open Shortest Path First-Traffic Engineering (OSPF-TE) protocols. Modeling, emulation, and analysis show the proposed solution can provide the required data to a cloud management system by sending information about virtual resources in the form of a new opaque link-state advertisement, named cloud LSA. Each embedded data center injects less than 40 bytes per second of additional traffic into the network when sending cloud LSAs at most every 5 seconds.**

Keywords: network embedded cloud, carrier cloud, Cloud LSA, opaque LSA, link-state update policy, OSPF-TE, resource monitoring, network-aware, cloud management system, extended TED, data center model

I. Introduction

Cloud computing provides scalable on demand resources over the network, thus shifting the computing infrastructure (both hardware and software resources) into a virtualized environment hosted by cloud providers. This virtualized infrastructure could be centralized or distributed. A centralized cloud is usually realized as a huge data center, while a distributed cloud consists of multiple geographically dispersed medium or small size data centers [1]. Typically, cloud providers do not own the network(s) that their cloud services travel over, but rather these are owned by one or more carriers. As a result, the cloud provider does not have control over the quality of experience (QoE) of the end user beyond the network within their data center(s). Therefore, if the network is a bottleneck the cloud provider cannot guarantee end-to-end performance.

A "network embedded cloud" is a distributed cloud architecture in which the network providers add computing and storage resources to their existing network infrastructure in order to offer cloud services. In this approach, the cloud provider controls both the computing resources and the network infrastructure, hence the network's resources are simply another set of cloud resources. As a result, all of the computing and network resources are viewed as being elastic and allocated based upon demand. A network embedded cloud provider can offer cloud services based upon a large number of small data centers, which are embedded in their carrier network, with more control over QoE [2].

A cloud network typically has two parts: "computing and storage resources" and a "cloud management system" (CMS). The CMS aims to provide secure, optimal, and scalable management of the virtualized resources. To achieve efficient management (e.g., resource allocation, resource placement, resource offering, and resource treatment [3]), the CMS needs information about the status of the cloud's resources. In a network embedded cloud the network assets are parts of the cloud architecture, thus efficient resource management by the CMS requires network awareness. The CMS relies on a resource monitoring system to provide information about the available resources at each of the distributed data centers (e.g., CPU, RAM, storage, and/or different classes of virtual machines) ***and*** information about the carrier networks (e.g., network topology and link attributes). In this paper, we present an extension to OSPF-TE, which we will refer to as Cloud-OSPF, to provide the required information to a network embedded cloud's CMS. OSPF-TE was extended by proposing a new type of traffic engineering LSA, named Cloud LSA, to convey virtualization and other resource status information to all the nodes in the network embedded cloud.

The reminder of this paper is organized as follows: Section II presents an overview of the OSPF protocol and its traffic engineering extension, OSPF-TE. Section III describes different link-state update policies. Section IV gives a detailed description of the proposed Cloud-OSPF and its sub modules, the data center resource utilization model, the effect of different update policies on this solution, and the structure of the proposed Cloud LSA. In Section V an analysis is given based on a worse case expected traffic load generated by the proposed solution. Additionally, an evaluation is made based on the traffic load generated by an implementation of Cloud LSAs. Finally, we conclude the paper in Section VII.

II. OSPF Overview

Open Shortest Path First (OSPF) [4] is a Link State (LS) routing protocol. In an OSPF network, routing information (e.g., the set of usable interfaces and neighbors) will be encapsulated in a unit of data referred to as a link-state advertisement (LSA). The participating OSPF routers in an autonomous system will each keep the information received via these LSAs in a database, which is referring to as a link state database (LSDB). The OSPF protocol runs directly over the Internet Protocol (IP) and has five different types of messages.

Eleven distinct types of LSAs are currently registered for the OSPF protocol [5]. LSA types 9, 10, and 11 are opaque link-state advertisements (opaque LSAs) [6]. In fact, an Opaque LSA can piggyback any information and floods this information within an OSPF network. The information contained in opaque LSAs may be used by OSPF protocols or any other application in the OSPF domain that can utilized OSPF as a transport protocol.

The traffic engineering LSA (TE LSA) is used for traffic engineering purposes. The TE LSA starts with a common LSA header with LSA type 10 and opaque type 1. The payload portion of the TE LSA carries information in the format of one or more nested triplets: type/length/value (TLV).

OSPF-TE [7] is an extension to the OSPF protocol to add traffic engineering capabilities. OSPF-TE uses the TE LSA, which has an area flooding scope, to carry TE information. This LSA identifies the originating router, this router's neighbors, and TE parameters. The traffic engineering database stores some additional link attributes (e.g., bandwidth and administrative constraints) in addition to the information that is stored in the LSDB.

Consider an OSPF network with a full mesh topology which contains "N" routers. If a router needs to send a LSA, it sends $(N-1)$ LSAs to the routers in the area. In response, each router other than the LSA originator sends $(N-2)$ LSAs to all the routers except for itself and the LSA originator. As a result, the total number of the LSAs transmitted in an OSPF network due to the injection of one LSA can be calculated as:

$$RedundantLSAs = (N-1)+(N-1)(N-2) = (N-1)^2$$

From this we can see that in a fully-connected mesh OSPF network with "N" routers each LSA requires on the order of N^2 redundant messages to be flooded. This issue, known as the "LSA N-squared problem", can lead to scalability problems since the value of N could be very large [8]. Several techniques have been proposed to reduce the number of LSAs, including an area approach, a spanning tree network architecture, and a configuration-information approach.

III. Link-state update policy

A link-state update policy determines when link-state updates should be distributed. As routers and third party applications may use link-state information when making decisions, having up-to-date information is necessary. Several link state update policies (including periodic, immediate, threshold-based, and class-based update policies) have been proposed in literature [9]–[11].

An immediate update policy floods LSA updates as soon as new information is available. This provide maximum accuracy for receiver, but this policy generates a lot of traffic when the frequency of resources changes are high.

A periodic update policy generates new LSAs based upon a predetermined period T, thus after a period T the router floods LSAs into the network regardless of whether or not it has new information. Thus, in this update policy the number of updates can be calculated beforehand. In this update policy, the fixed period T is the key factor controlling the sending of updates.

A threshold-based update policy triggers new link-state updates when the available resources changes by a predefined threshold value [12]. There are two categories for threshold-based policies: "absolute threshold-based" and "relative threshold-based" update policies. In an absolute threshold-based policy, updates are disseminated when the change between the current and the previously advertised value of a resource exceeds a certain threshold value, while in a relative threshold-based policy updates are disseminated when the relative change between the current and the previously advertised value of a resource exceeds a certain threshold value as described in the equation below.

$$\frac{|R_n - R_p|}{R_p} \geq thr$$

Where thr denotes the threshold value, R_n represents the current value of a resource, and R_p denotes the previously flooded value. With respect to this policy, the threshold value is the key factor controlling the sending of update.

A class-based update policy divides the resource into classes and updates are flooded whenever the available resources move to a different class from the current class. When the maximum capacity of the resource is C, classes can be described as:

$$\Big[0, \beta C\Big), \Big[\beta C, (f+1)\beta C\Big), \Big[(f+1)\beta C, (f^2+f+1)\beta C\Big), \ldots$$

Where β denotes a base class factor ($\beta < 1$), and f denotes a class growth factor. The class's size could be fixed (f = 1) or exponential ($f > 1$). This update policy can generate lots of unnecessary updates when the available resources fluctuate around a class boundary. Hysteresis can be used to prevent excessive unnecessary updates. In this method, a new update occurs only when the available capacity passes the class boundary ***and*** the magnitude of the changes are significant (e.g., the available capacity passes the middle of adjacent classes) [13]. The base factor and growth factor are the key factors controlling the sending of updates in a class-based update policy.

IV. CLOUD-OSPF

As described earlier, the network embedded cloud's CMS requires information not only about reserve-able capacity in each distributed data center, but also about network conditions including the topology and link characteristics. To provide this information we propose an extension to the OSPF-TE called "Cloud-OSPF". As shown in Fig. 1 an extended OSPF-TE router is installed at each distributed data center to provide the required processing and network information for a network embedded cloud's CMS. OSPF provides information related to the network's topology, OSPF-TE provides information regarding the link characteristics and TE metrics, and Cloud-OSPF provides information about each data center's resources.

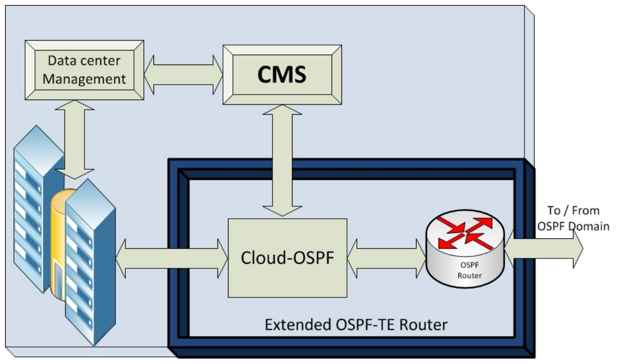

Fig. 1. Proposed solution architecture.

A. Cloud-OSPF implementation

Our implementation of Cloud-OSPF consists of two separate modules (as shown in Fig. 2): Cloud-OSPF-Receiver and Cloud-OSPF-Sender. Utilizing two separate modules offers flexibility, as either the Cloud-OSPF-Receiver or Cloud-OSPF-Sender can be shut down independently, thus reducing the router's processing load. The Cloud-OSPF-Sender is responsible for reading and examining the cloud resource database, which contains information about the resources of an embedded data center. The data center is responsible for providing these details to the Cloud-OSPF-Sender. The Cloud-OSPF-Sender utilizes the OSPF router to flood the cloud resource database data into the OSPF domain (which is also the network embedded cloud domain) as a new type of opaque LSA (i.e., "Cloud LSA"), when required by the selected update policy. The Cloud-OSPF-Receiver is responsible for analyzing received Cloud LSAs that were generated by other instances of the Cloud-OSPF-Sender module within the OSPF area and keeping the extended traffic engineering database (TED) up to date. This extended TED contains information about all the embedded data centers plus the network's topology and link attributes. The CMS of the network embedded cloud can query this extended TED to obtain the information necessary to make an appropriate resource application decision.

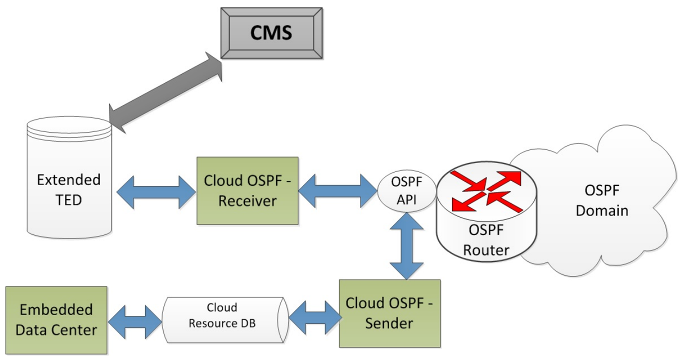

Fig. 2. Proposed solution design.

The Cloud-OSPF implementation was coded in the C programming language and is running on linux with a 3.2.0-33-generic kernel and the databases were built upon SQLite [14]. All the interfaces necessary to originate and capture the LSAs were created using the open source Quagga routing suite [15] OSPF API [16].

B. Cloud LSA

The proposed Cloud LSA is an opaque LSA which was designed to convey information about embedded data center resources to the CMS. This new LSA is as an enhancement of the OSPF-TE protocol as it makes use of the TE LSAs, which are opaque LSAs with an area wide flooding scope (LS type 10). There are several ways that we could store information about a data center's resources in the payload portion of the Cloud LSA, for example as a new type of TE TLV or by extending the existing TE TLV [17]. Selecting the best alternative for the Cloud LSA structure and the encoding of the resource information requires further research. Here we will utilize the Cloud LSA when using a sub-TLV for the Node attribute TLV (TLV type 5) with a proposed sub-TLV type 3. This Cloud LSA contains information about the data center's resources (specifically available CPU, RAM, storage, and the data center's location) which combined requires 44 bytes of data. The proposed Cloud LSA format is shown in Fig. 3.

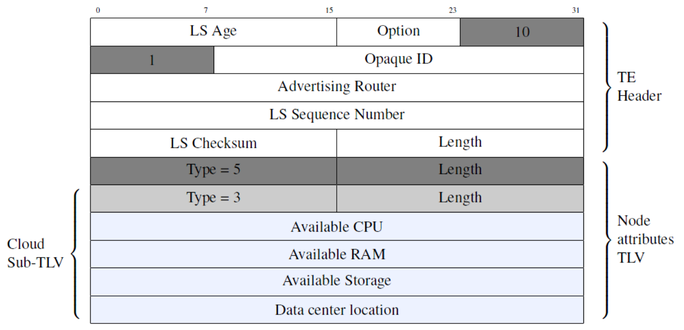

Fig. 3. Cloud LSA structure.

C. Data center resource utilization modeling

Unfortunately, neither cloud providers nor the literature in the field provide detailed information about how resource status changes as a function of time. As a result, we have utilized a simple data center resource utilization model to simulate an embedded data center's activity over the course of a business day. This model assumes that during the first hours of the day, the cloud, and as a consequence the data center, experiences a minimum number of resource allocation requests, thus the data center resource utilization has a minimum value. In the early morning hours as people start to work, the cloud experiences a higher rate of resource allocation requests which leads to a higher resource utilization. The utilization reaches its highest level around midday. In the evening, users send increasing numbers of resource de-allocation requests which decrease the resource utilization. Finally, late at night, the utilization returns to the lowest level. This model is applicable to a data center where most of the users are located in a single geographical area and assumes that the demands are largely related to human users making requests, as opposed to batch processing.

The embedded data center simulation module behaves as a data center with some pre-defined capacity. We assume that the data center's capacity can be described by specifying the CPU's clock rate in GHz, the amount of RAM in GB, and the amount of storage in GB. The data center simulator serves user requests by allocating or releasing data center resources. The module simulates user requests based on a bounded request arrival rate. The mean of usable CPU capacity for this simulated data center over 24 hours is shown in Fig. 4.

D. Update policy

Within a cloud the demand for resources is dynamic in nature. To enable the CMS to make good resource allocation decisions, it is essential to select an appropriate policy for when to send Cloud LSAs. Sending very frequent Cloud LSAs updates results in more accurate information for the CMS at the cost of a higher traffic load, which may negatively impact the performance of the network embedded cloud. Conversely, sending infrequent Cloud LSA updates may result in losing synchronization between the extended TEDs of the OSPF routers in the network, and as a consequence the CMS may make a "bad" decision leading to over or under reservation.

In order to find an appropriate update policy, different update policies (with different key factor(s) values) were applied to the proposed data center resource utilization model of CPU capacity to see how each policy effects the number of updates. The results are shown in Fig. 5, Fig. 6, and Fig. 7.

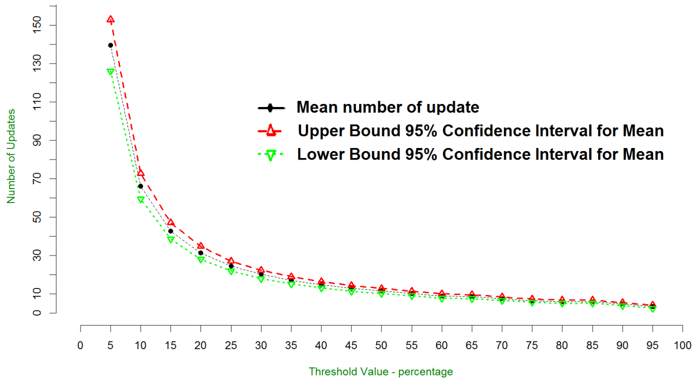

Fig. 6. Relative threshold-based update policy. Number of updates per different threshold values based on changes of a data center's CPU capacity.

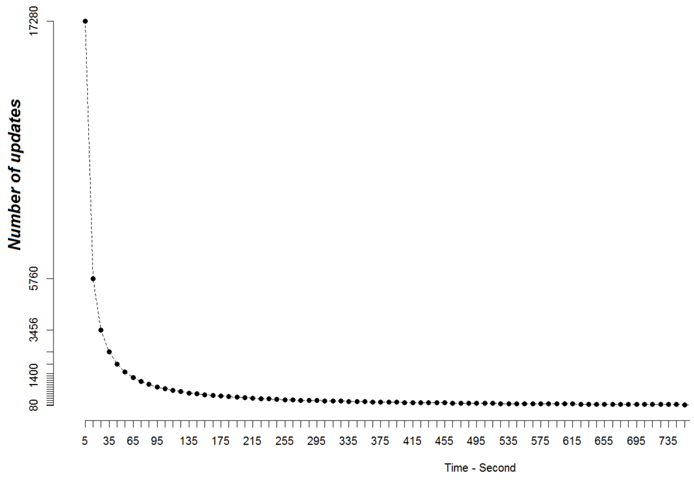

Fig. 7. Number of updates per different hold-down timer values.

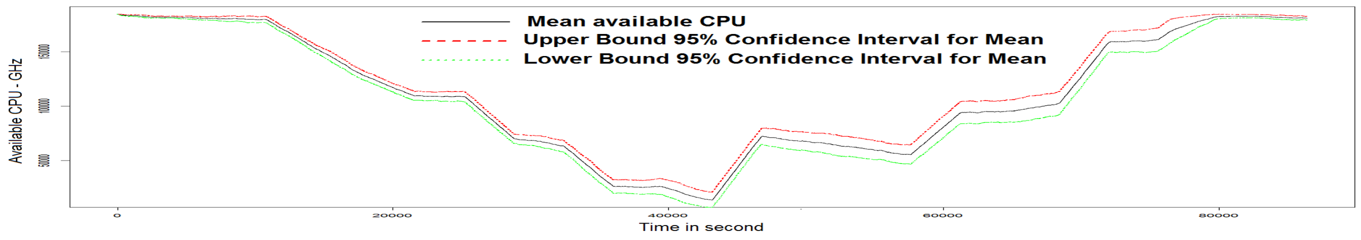

Fig. 4. Simulated data center's *CPU* capacity utilized per second.

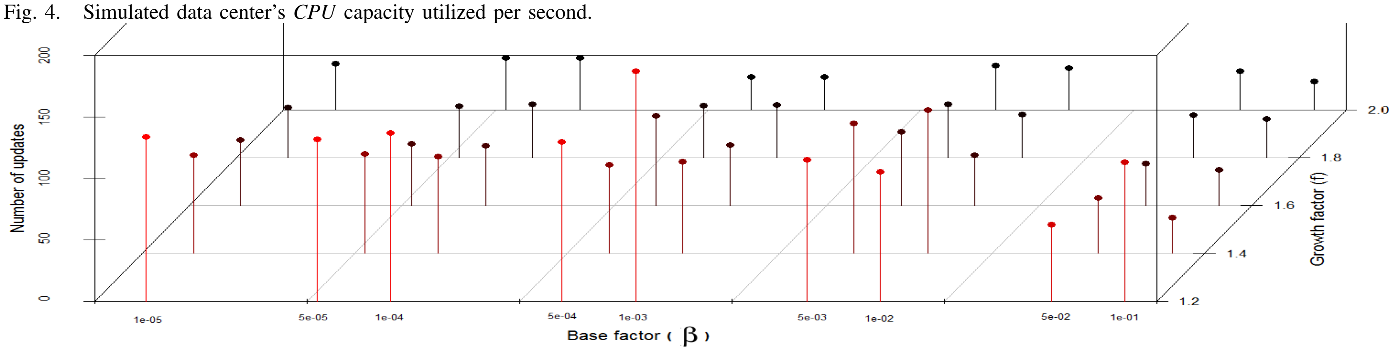

Fig. 5. Exponential-sized class-based update policy. Number of updates per different values for growth factor "*f*" and base factor "*β*". The base factor axis uses a logarithmic scale.

The number of updates is not a sufficient factor to make a decision upon an update policy and its key factor(s) value(s) as we must also consider how the CMS will view the resources with different update policies. We applied the four different update policies to the data center resource utilization model to see how the update policy reacts to the changes in the resource status and how well updates describe the data center's resources. Figures 8 to 11 enable us to assess the inaccuracy of the CMS's view of the state of the cloud's resources over time.

The complete analysis and study (see [17]) shows that when using a periodic update policy a significant change can be ignored (for a period of time), and unnecessary LSA updates can occur often (especially if the period is short). The immediate update policy maximizes the accuracy of resource management, but dramatically increases the network load. As the type and rate of resource requests change during the day, the available cloud resources fluctuate. In conclusion, periodic update and immediate update policies do not allow the update policy to adapt to the availability of resources.

A threshold-based policy based on a relevant change and a class-based policy with exponential-sized classes both try to avoid unnecessary updates by sending an update only when a *significant* change in resources occurs. Also, for both these update policies, updates are sent more frequently when there are few available resources - hence the CMS must be more careful in managing the available resources. Such characteristics make either of them a good candidate for a suitable update policy.

V. ANALYSIS

The Cloud LSA presented in this paper carries 44 bytes of data. In order to send a new LSA into the OSPF network as a link-state update (LSU), the LSU header (28 bytes), an IP header (20 bytes), and [if applicable] Ethernet header and trailer (18 bytes) should be added to Cloud LSA. Additionally, each LSU has an acknowledgment response. The link-state acknowledgment packet consists of the OSPF header (20 bytes) plus the link-state advertisement header (24 byte). In order to send an acknowledgment packet into the OSPF network, an IP header (20 bytes) and [if applicable] Ethernet header and trailer (18 bytes) should be added to this packet. When an OSPF router originates one Cloud LSA the network must transfer an additional 192 bytes.

Due to OSPF's reliable flooding and the LSA N-squared problem, in a fully meshed connected network with N OSPF routers, flooding each LSU produces a maximum of $O(N^2)$ copies of the LSA. As a result, the total traffic load due to Cloud LSAs in the OSPF network (from the receivers' point of view) is:

$$Extra\ load = D \times R_{Netwok} \times N_{CloudLSA} \times Size_{CloudLSA}$$

Where D denotes the number of embedded data centers, R_{Netwok} denotes number of copies of the LSAs that OSPF generates to flood a single LSA, $N_{CloudLSA}$ denotes number of Cloud LSAs that Cloud-OSPF injects into the OSPF network, and $Size_{CloudLSA}$ denotes the size of a Cloud LSA.

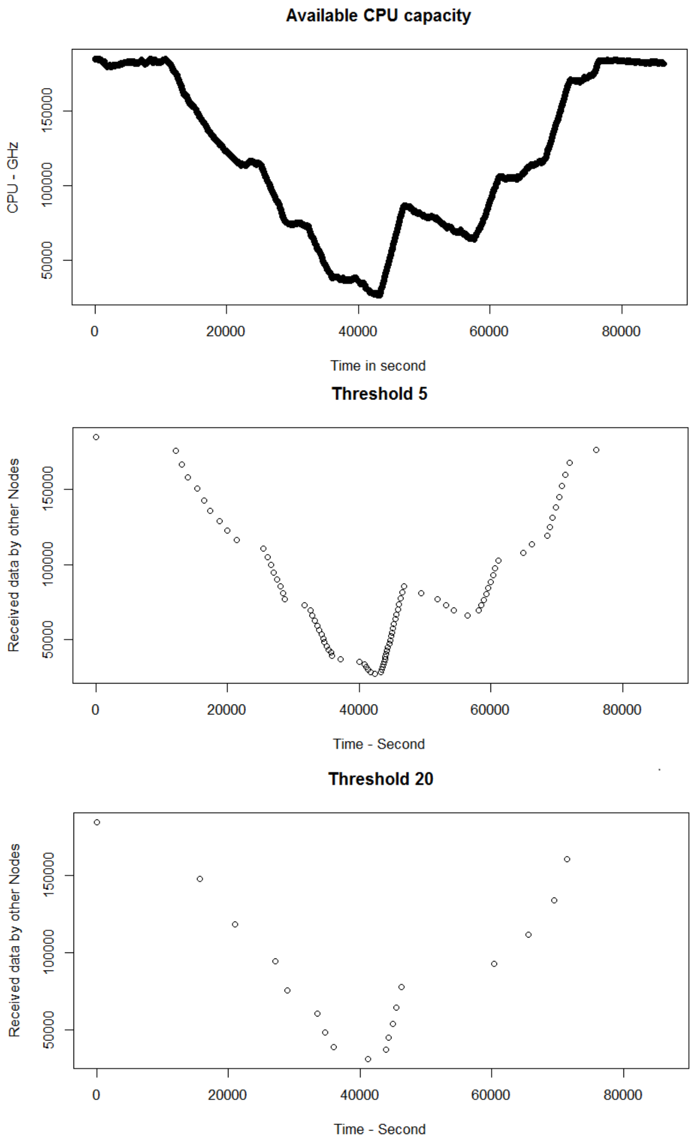

Fig. 8. Relative threshold-based update policy. The top curve shows the available CPU capacity and this curve should be compared to how a CMS views cloud resources with policies. In this case it should be compared with the relative threshold-based update policy different threshold values 5% and 20%, the updates points are plotted as circles in each graph.

The update policy module of the Cloud-OSPF-Sender is responsible for making decisions about when the CMS needs to be updated about the data center's resource information. Based on the OSPF protocol's MinLSInterval timer the OSPF router can produce a maximum of one LSA every 5 seconds. As a result, using any type of update policy, the maximum rate for sending Cloud LSAs by an extended OSPF router, is limited to one LSA every 5 seconds. Thus, the solution presented in this paper can handle cases where a data center's resource changes in such a way that a CMS need to be updated at most every 5 seconds. Hence, each Cloud-OSPF-Sender sends at most 192 bytes every 5 seconds corresponding to an average of 38.4 bytes per second (Bps).

To provide the latest information to the CMS of a network embedded cloud an extended OSPF router at each data center

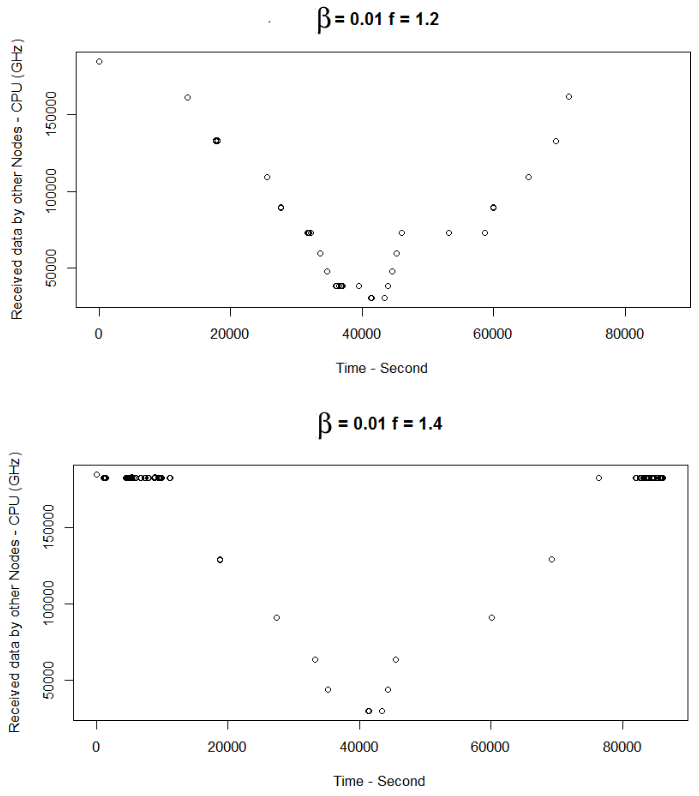

Fig. 9. Exponential-sized class based update policy. How a CMS views data center's CPU resources with fix base factor ($\beta = 0.01$) with different growth factor "f" values, the updates are plotted as circles in each graph. The available CPU resources were shown in the top curve of Figures 8.

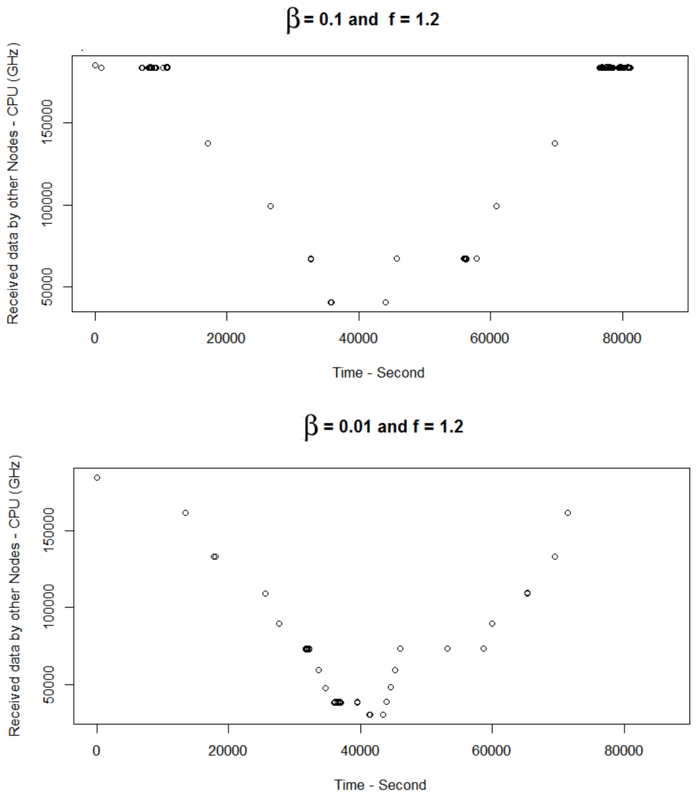

Fig. 10. Exponential-sized class based update policy. How a CMS views data center's CPU resources with a fix growth factor (*f* = *1.2*) with different base factor "β" values, the updates are plotted as circles in each graph. The available CPU resources were shown in the top curve of Figures 8.

needs to send Cloud LSAs. Given the arguments above, in a fully meshed connected network embedded cloud with D embedded data centers and N OSPF routers the worst case average traffic due to Cloud LSAs from a receiver's point of view will be $(D \times O(N^2) \times 38.4)$ Bps. If each of the data centers sends one LSU at the same time, then the peak overhead traffic in traffic caused by this protocol will be $(D \times O(N^2) \times 192)$ bytes every 5 seconds.

OSPF and its flooding protocol are applicable only for small and medium sized networks because of the LSA N-squared problem. Studies show the maximum number of OSPF routers per AS is limited to around 50 [18]–[20]. As a consequence, if the network is fully-meshed and the number of OSPF routers is limited to 50, and if there is an embedded data center co-located with each OSPF routers, then the maximum average additional traffic due to Cloud LSAs can be calculated as 4.610 MBps with a peak of 23.050 MB every 5 seconds. To ignore unnecessary Cloud LSA messages forwarding in an OSPF network we suggested that each router do a simple analysis of each Cloud LSA (e.g., calculate a hash value of the Cloud LSA's information or look at the Cloud LSA information). This analysis enables the OSPF routers in a network to recognize whether they have recently see this LSA or not. As a result, the OSPF router can discard redundant Cloud LSAs by avoiding forwarding them to other interfaces when this is not required, thus reducing the unnecessary Cloud LSA traffic in OSPF network but at the cost of additional processing at each router. This should be studied in future work.

As network embedded clouds are not yet a common practice, there is no clear answer for the number of embedded data centers that may need to be supported. To enable cloud functionality telecommunication providers would add computing resources to each of their network nodes (or co-located with these nodes). As a result, the number of embedded data centers in a network may be more than 50. Whether the optimization proposed in the previous paragraph can be applied to ameliorate the effects of these much larger numbers of nodes remains to be seen.

VI. EVALUATION

To evaluate the prototype, a simulated network embedded cloud, consisting of six linux routers, six simulated embedded data centers, and nine links was deployed, as shown in Fig. 12. This enabled us to analyze the Cloud-OSPF traffic overhead. Oracle's VM VirtualBox program [21] was used to create a network topology, where each linux router runs an OSPF implementation of the Quagga suite (extended with Cloud-OSPF functionality) and the OSPF API option was enabled. All the OSPF routers are configured in a backbone area (area zero). For the purposes of testing the Cloud-OSPF module was configured to use a threshold-based update policy

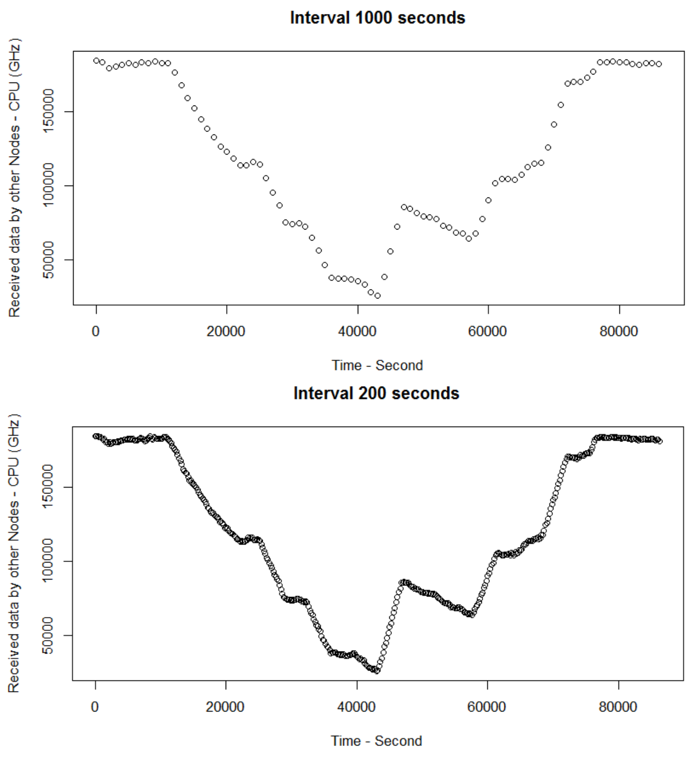

Fig. 11. Periodic update policy. How a CMS views data center's CPU resources with different hold-down timer values 1000 and 200 seconds, the updates points are plotted a circles in each graph.

with the relevant change threshold set to 20%. Additionally, the Cloud-OSPF-Sender module in this test scenario sends a new Cloud LSA when the relative changes in available CPU capacity or RAM capacity or storage capacity exceeds the threshold value as described in the algorithm below.

```
**if** (CPU relative change $\geq$ Threshold Value, OR
RAM relative change $\geq$ Threshold Value, OR
CPU relative change $\geq$ Threshold Value) **then**
    Send a new Cloud LSA
**end if**
```

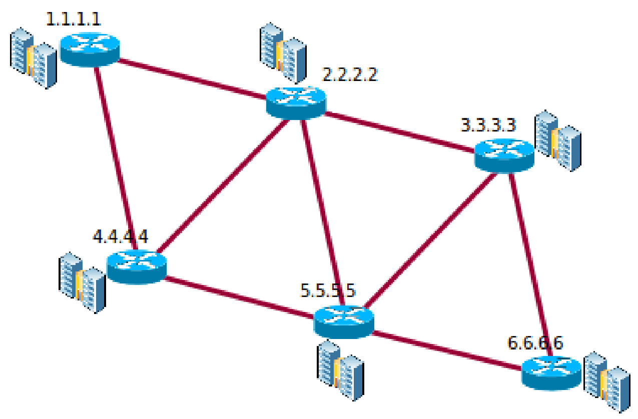

Fig. 12. Network topology used to evaluate the prototype.

Fig. 13 shows CMS's views of the simulated CPU resources for this particular case. Network traffic was captured using a network analyzer application for 24 hours. Fig. 14 shows the OSPF protocol terrific pattern in bytes per 10 minute interval over 24 hours in this simulated network embedded cloud.

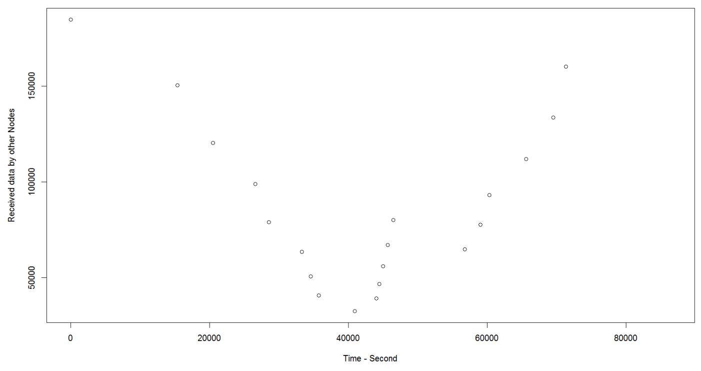

Fig. 13. CMS's views of the CPU resources with a relative threshold value 20% when the algorithm is used, the updates are plotted as circles.

The traffic patterns show one peak around midday. As discussed in the data center utilization model at midday the simulated data center's available resources are very limited. In this condition, based on the relative threshold update policy (i.e. as implemented by the Cloud-OSPF-Sender module policy in test scenario), the extended OSPF router sends more frequent Cloud LSUs when the resources are low. The measurements show that the proposed solution produces 7 Bps of additional OSPF traffic. A more general comparison is difficult due to the dependency on the topology, number of embedded data centers, number of nodes, update policy, and data center utilization model.

It is worth remembering that the OSPF protocol produces less additional traffic than OSPF-TE, when the network is stable. Although our simple test scenario shows that the solution proposed in this paper produces less additional traffic than pure OSPF, we expect the proposed solution to produce less additional traffic than OSPF-TE when the size of the network scales up. Further study and experiments are required to understand the implication of the proposed Cloud-OSPF extension when deployed in a network with more nodes, different topology, different update policy, and/or a more realistic data center utilization model.

VII. CONCLUSION

The architecture of the network embedded cloud distributes computational and storage resources geographically and integrates these resources operationally throughout the carrier's network and both network and computational resources become cloud resources. To optimize resource management in this new cloud model a cloud management system requires information not only about the resources available at each distributed data centers, but also needs details of the network conditions. To provide this information we experimented with a new type of TE LSA, named Cloud LSA. This Cloud LSA allows more efficient management of

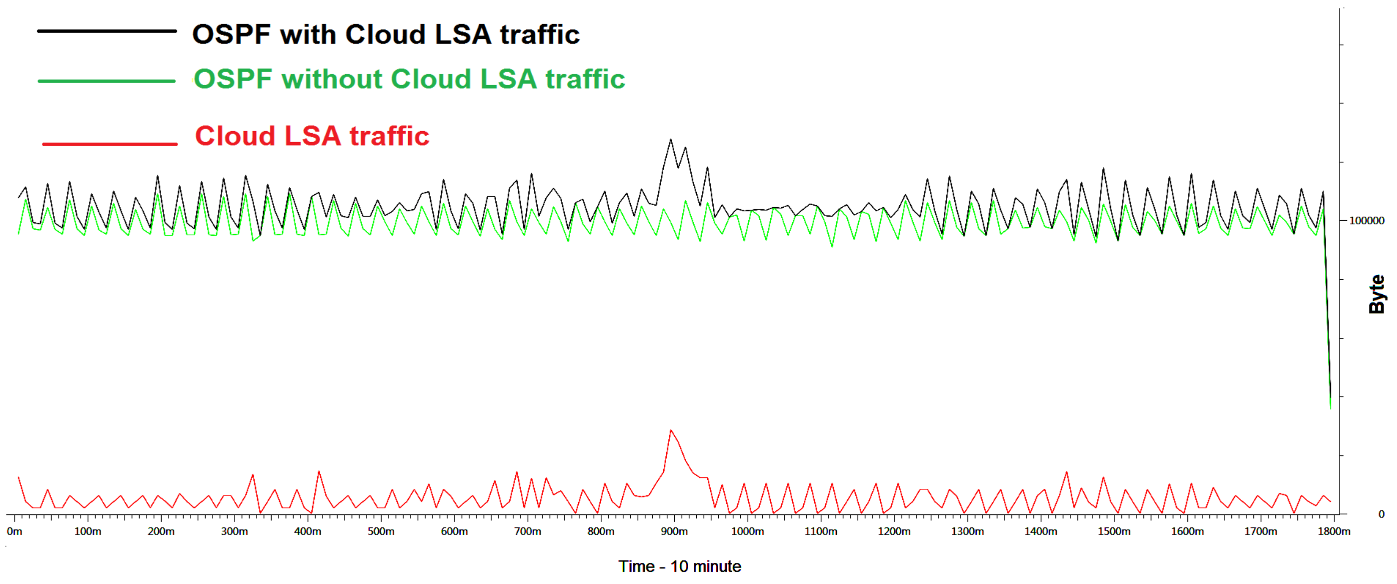

Fig. 14. Measured OSPF protocol traffic in a simulated network embedded cloud - bytes per 10 minute interval. Protocol traffic for OSPF without Cloud LSA (middle curve), OSPF with Cloud LSA (top curve), and the Cloud LSA traffic (bottom curve).

a network embedded cloud by providing available resource information (CPU, RAM, storage, and data center's location) and network conditions. Analysis shows the proposed solution can handle cases where updates to a data center's resource changes occur at most every 5 seconds producing at most $D \times O(N^2) \times 192$ bytes every 5 seconds, where D denotes the number of embedded data centers and N denotes the number of OSPF nodes.

References

[1] K. Church, A. Greenberg, and J. Hamilton, "On Delivering Embarrassingly Distributed Cloud Services," in *HotNets*, Oct. 2008, pp. 55–60, Available: http://conferences.sigcomm.org/hotnets/2008/papers/10.pdf.

[2] Heavy Reading Service Provider IT Insider, "Big Vendors Race to Establish Carrier Cloud," *Heavy Reading Insiders*, Mar. 2012.

[3] P. Endo, A. de Almeida Palhares, N. Pereira, G. Goncalves, D. Sadok, J. Kelner, B. Melander, and J. Mangs, "Resource allocation for distributed cloud: concepts and research challenges," *Network, IEEE*, vol. 25, no. 4, pp. 42–46, July-August 2011, Available: http://dx.doi.org/10.1109/MNET.2011.5958007.

[4] J. Moy, "OSPF Version 2," *Internet Request for Comments, RFC 2328*, Apr. 1998, Available: http://www.rfc-editor.org/rfc/rfc2328.txt.

[5] K. Kompella and B. Fenner, "IANA Considerations for OSPF," *Internet Request for Comments, RFC 4940*, Jul. 2007, Available: http://www.rfc-editor.org/rfc/rfc4940.txt.

[6] L. Berger, I. Bryskin, A. Zinin, and R. Coltun, "The OSPF Opaque LSA Option," *Internet Request for Comments, RFC 5250*, Jul. 2008, Available: http://www.rfc-editor.org/rfc/rfc5250.txt.

[7] D. Katz, K. Kompella, and D. Yeung, "Traffic Engineering (TE) Extensions to OSPF Version 2," *Internet Request for Comments, RFC 3630*, Sep. 2003, Available: http://www.rfc-editor.org/rfc/rfc3630.txt.

[8] A. V. Aho and D. Lee, "Hierarchical networks and the LSA N-squared problem in OSPF routing," in *IEEE Global Telecommunications Conference, 2000. GLOBECOM '00.*, vol. 1, 2000, pp. 397–404, Available: http://dx.doi.org/10.1109/GLOCOM.2000.892036.

[9] A. Shaikh, J. Rexford, and K. G. Shin, "Evaluating the impact of stale link state on quality-of-service (QoS) routing," *IEEE/ACM Trans. Netw.*, vol. 9, no. 2, pp. 162–176, Apr. 2001, Available: http://dx.doi.org/10.1109/90.917073.

[10] G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, and S. Tripathi, "Intradomain QoS routing in IP networks: a feasibility and cost/benefit analysis," *Network, IEEE*, vol. 13, no. 5, pp. 42–54, Sept./Oct. 1999, Available: http://dx.doi.org/10.1109/65.793691.

[11] X. Yuan, W. Zheng, and S. Ding, "A comparative study of QoS routing schemes that tolerate imprecise state information," in *Eleventh International Conference on Computer Communications and Networks, 2002*, 2002, pp. 230–235, Available: http://dx.doi.org/10.1109/ICCCN.2002.1043071.

[12] Tae-il Kim, Jeong-Ju Yoo, Hae-Won Jung, Hyeong-Ho Le, Min Young Chung, and Seong-Il Jin, "Adaptive threshold-based link status update mechanism," in *The 8th International Conference Advanced Communication Technology, 2006. ICACT 2006*, vol. 1, Feb. 2006, pp. 888–891, Available: http://dx.doi.org/10.1109/ICACT.2006.206105.

[13] G. Apostolopoulos, R. Guérin, S. Kamat, and S. K. Tripathi, "Quality of service based routing: a performance perspective," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 17–28, Oct. 1998, Available:http://doi.acm.org/10.1145/285243.285251.

[14] "SQLite," Accessed June 2013. [Online]. Available: http://www.sqlite.org/

[15] "Quagga Software Routing Suite," Accessed June 2012. [Online]. Available: http://www.nongnu.org/quagga/

[16] R. Keller, "Dissemination of Application-Specific Information Using the OSPF Routing Protocol," TIK Report Number 181, TIK, ETH Zurich, Nov. 2003.

[17] A. Roozbeh, "Resource monitoring in a Network Embedded Cloud: An extension to OSPF-TE," Master's thesis, KTH Royal Institute of Technology, School of Information and Communication Technology, Trita-ICT-EX; 2013:85, 2013,Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-124367.

[18] K. Hutton, J. Tiso, M. Schofield, and D. Teare, *Designing Cisco Network Service Architectures (ARCH): (Ccdp Arch 642-874)*, ser. Foundation Learning Guides. Cisco Press, 2011, ch. 6: Designing Scalable OSPF Design.

[19] ImageStream Internet Solutions, Inc., "The OSPF Routing Protocol," Accessed June 2013. [Online]. Available: http://support.imagestream.com/OSPF.html

[20] Data Network Resource, "OSPF routing protocol, Dijkstra algorithm, OSPF stub area," Accessed Jun 2013. [Online]. Available: http://www.rhyshaden.com/ospf.htm

[21] J. Watson, "Virtualbox: bits and bytes masquerading as machines," *Linux J.*, vol. 2008, no. 166, Feb. 2008, Available: http://dl.acm.org/citation.cfm?id=1344209.1344210.