Analyzing Strategies to Effectively Detect Changes in Content Delivery Networks

André Moreira, Moisés Rodrigues, Ernani Azevedo and Djamel Sadok
Federal University of Pernambuco
Recife, Brazil
{andre, moises, ernani, jamel}@gprt.ufpe.br

Arthur Callado
Quixadá Campus - UFC
Quixadá, Brazil
arthur@ufc.br

Victor Souza
Ericsson Research
Stockholm, Sweden
victor.souza@ericsson.com

***Abstract*— Content Delivery Networks have gained a popular role among application service providers (ASPs) and infrastructural companies. A CDN is an overlay network that gives more control of asset delivery by strategically placing servers closer to the end-user, reducing response time and network congestion. Many strategies have been proposed to deal with aspects inherent to the CDN distribution model. Though mostly very effective, a traditional CDN approach of statically positioned elements often fails to meet quality of experience (QoE) requirements when network conditions suddenly change. In this paper, we introduce a technique to detect significant changes in a monitored metric of a CDN in order to allow provisioning adaptation of resources, using easy access information with great practical viability. Results show decrease on network resources usage without considerable changes on quality of service (QoS). Furthermore, our technique has similar performance in comparison to an omniscient strategy.**

Keywords-component; Content Distribution Networks; Changes Detection; Strategies.

I. INTRODUCTION

Content Delivery Network (CDN) is a distribution system that allows content providers to offer a faster way to deliver their services to a large number of users, overcoming the inherent limitations of the Internet in terms of user perceived Quality of Service (QoS) [1]. CDNs are very successful in distributing content in the Internet, large enterprises are usually customers of traditional and high-performance CDNs who operate extensive networks across the globe, such as Akamai [2], LimeLight Networks and Mirror Image. According to a CISCO forecast for Internet traffic [3], due to the popularity of video streaming services, CDN will prevail as the dominant technology for content delivery.

Many strategies have been proposed to deal with aspects inherent to the CDN distribution model, however, they often fail to meet the high throughput demands of unexpected and unplanned events, like flash crowds [8]. A flash crowd is a sudden increase in the request rate that can deplete a content distribution system due to congestion or denial of service. It is becoming more frequent since the rapid spread of Internet capable devices, which makes information access more ubiquitous and demanding. The number of Internet-connected devices reached 8.7 billion in 2012 [7]. Because of a usually highly volatile content and short lifespan, an over-provisioning of the CDN is inefficient and uneconomical. The traditional CDN performs well with a relative constant workload, but the unanticipated surge in traffic provoked by a flash crowd renders traditional techniques ineffective at best. Furthermore, due to the distributed nature of CDNs and their varying demands, it remains difficult to predict an optimal configuration. The ability to handle flash crowds or significant changes in traffic pattern is an important aspect of CDN design.

Some solutions have already been proposed to address these problems. Promising solutions should incorporate a certain principle: changing the static CDN into an adaptive dynamic CDN [8]. The idea is not new, several network services and management systems allow the optimization of network and/or service operating by rearranging network configuration or resource allocation. Typically, such adaptations are performed on a periodical basis, and the success of adaptation algorithms often depends on the period between re-adaptations. Adaptation performed on long periods may operate on sub-optimal conditions most of time. On other occasions, the adaptation is performed very often and the costs are comparable or could sometimes even outweigh to the gains derived from the optimization itself.

In summary, the main challenges of a dynamic CDN include how to organize or reorganize a temporary overlay of surrogate servers quickly and without involving much overhead, but more important is how to detect the beginning and the end of flash crowds or other relevant changes in traffic behavior. Flash crowds are different from the normal workloads, whose magnitude and duration depend on the people's interest toward some triggering events, and it is difficult to make long-term predictions in advance.

In this work, we focus on efficiently detecting the need of adaptation in a CDN environment considering the inherent incompleteness of monitoring information. We present two approaches to detect changes in a network, using a content provision environment as scenario. The prime goal in detecting such changes accurately is to trigger the replacement algorithm (a CDN adaptation) only when truly necessary, ignoring insignificant changes ("false positives"). To validate the proposed enhancements, the strategies were implemented as plugins to the simulator tool described in [4], whose solutions were used as comparison baselines for the methods presented in this paper. Since the simulator was developed upon a content provision scenario as well, it is the fittest tool to evaluate the proposals.

The remainder of this paper is organized as follows. A research on dynamic CDNs is presented in Section II. Section III describes the case study used in the evaluation of the proposals. Sections IV and V present an overview, implementation and first results of the two strategies, respectively, Weighted Moving Average Vector and the simulator with the restriction of using only real Internet data.

Section VI descants the experience obtained from the experiments, the lessons learnt, discusses the possible evolution for each approach and draws some conclusions.

II. Related work

To detect the beginning of flash crowds, most of solutions rely on monitoring the load of surrogate servers and comparing with their overall load [10][11][14][19]. The work in [8] classified the solutions for dynamic adaptation CDN into three categories: server layer, intermediate layer and client layer. Intermediate-layer solutions utilize network resources to perform offload. An example is CoralCDN [15] that exploits overlay routing techniques using a P2P distributed sloppy hash table. Another example is multi-level caching [16]. Client-side solutions make clients help each other in sharing objects to distribute the load burden from a centralized server. An origin Web server can mediate client cooperation by redirecting a client to another client that has recently downloaded the objects [18]. Peer-Assisted CDNs fall into this category [10].

Most solutions are server layer. J. Jung, et al. [11], for instance, proposed an adaptive CDN using dynamic delegation to improve the protection of origin servers under heavy load. Another example is DotSlash [14]. It uses a mutual-aid rescue model. A web server joins a mutual-aid community by registering itself with a DotSlash service registry, and contributing its spare capacity to the community. A self-configuring service discovery is used to allow servers of different web sites to learn about each other dynamically, rescue actions are triggered automatically based on load conditions.

Other examples of server layer adaptation are the cloud-based CDNs [5][11][14][15]. A cloud-based CDN potentially reduces the effort and cost of producing a solution for content delivery. It merely uses the cloud to map the CDN infrastructure elements into virtual components across the cloud. For instance, a surrogate server can be mapped to an IaaS [6] storage and delivery service. An example is MetaCDN [9]. A similar approach is followed by [13] that allows content providers to dynamically deploy and instantiate CDN modules on core and edge routers with programming functionality over content distribution paths from the origin server to the users. The particular cloud-based CDN solution, called ActiveCDN, allows users to be redirected to the most appropriate node based on geolocation.

Finally, Moreira et al. [12] proposed the virtualization of a VoD CDN, allowing programmatic modifications in a CDN infrastructure designed for video distribution, in order to adapt it to new operating conditions. They also proposed a different technique to detect flash crowds by using information from CDN Monitor, instead of relying on the load of surrogates.

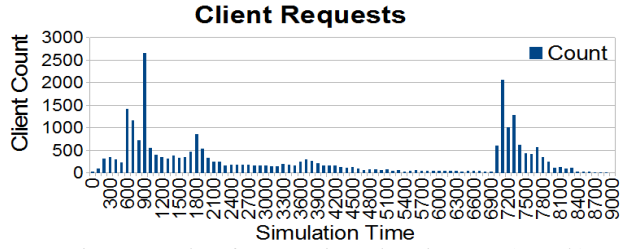

Figure 1: Number of requests triggered per timestamp (seconds)

III. Case Study

We extend the work in [12], which presented an algorithm to detect flash crowds or any other relevant changes in traffic of a CDN. It also, proposed the reorganization of the overlay by replacing the virtual replica servers using the algorithm in [17]. A scenario was designed to observe the behavior in situations that contemplate the necessity of the new developments described below. In the scenario, two flash crowds take place. In case the traffic boom is detected earlier, resource placement can be improved sooner and the CDN traffic, directly associated with maintenance cost, is diminished. Our works differs from the previous because we propose the use of real CDN data to detect new network operating conditions. The previous work uses omniscient network information, most likely not available in an actual CDN environment. What we call omniscient information was data collected through all nodes in the topology, representing network traffic perfectly, which is only possible in a simulated environment. In this paper we test the effectiveness of the algorithm in scenarios with restricted information. The validation of the simulator used in the experiments can be found in [4].

In a global topology, 13089 content consumers are spread among geographically remote areas (Autonomous Systems - AS), what causes the use of many hops and, consequently, yields a big traffic over the CDN network. In such a scenario, massive requests are more representative to the network state, and contribute to decaying the users' quality of experience. To prevent this decay, the surrogate replacement occurs, and as important as the replacement is its detection. This is the reason that leads us to use the flash crowd scenario to evaluate a sliding window proposal.

The 13089 requests that rule the simulation are split as follows: 8077 randomly divided requests spread among areas AS1 and AS2; a 100 second flash crowd of 1007 requests from AS1 users (from time 900s to time 1000s) and another flash crowd from time 7000s to time 9000s, holding 4005 requests from AS2 users. Figure 1 illustrates this distribution.

The metrics explored in the experiments are network related and service related. Inner traffic and Cross Traffic represent the amount of bytes transferred, respectively, between routers inside the same domain, and between different domain routers. Lowering the Cross Traffic is a good measure to reduce traffic cost, once it indicates the network is able to handle its contents more effectively inside its own nodes, and reducing Inner Traffic is a pointer that the content is more efficiently distributed among the domain's nodes. For both metrics, the lower, the better, but one might rather prioritize increasing, if needed, the inner traffic than the cross traffic. Startup time (or startup delay) is the latency or the delay a user experiences when he requests for content and it effectively starts to play and is displayed. It is different from the "time to first byte" TTFB metric [20] because the startup delay must consider the buffering time.

IV. Weighted Moving Average Vector

A possible improvement to avoid sensitiveness to small duration pattern changes in replacement algorithm is a weighted average vector utility function. The goal is to smooth down the detection in pattern changes by combining multiple probes to represent a baseline scenario or current vectors. The

proposed utility function U_w is then defined as a combination of various vectors. The number of samples S is a parameter for the system.. The resulting vector of utility function is their weighted average. The weight is defined by a weight function $W(V, s)$ as in the following: the closer the probe vector is to the last replacement, the greater is the weight. The weight function should also comply with the following definitions:

$$\sum_{s=1}^{S} W(V,s) = 1 \qquad \sum_{s=1}^{S} dW(V,s) = 0 \rightarrow \text{Pareto optimal}$$

Figure 2 illustrates this condition, comparing the proposed improvement (a) to the approach described in [12] (b).

A. Implementation

The weight for the probes was defined according to an exponential function. The algorithm considers any number of windows and, after applying the formula, it normalizes the values to define the weights (see Algorithm 1). For the first proof of concept implementation, the formula was obtained empirically, and satisfied the case study.

The approach only begins to assess the traffic when the predefined amount of time windows is reached. Figure 3 compares the functioning of the sliding window sized 3, for a window length defined as 100 seconds, and its equivalent to a 300s sized window on the previous approach. While the single window approach produces the first usable collection in time 300s, and the next collections every 300s forth, the weighted strategy has a usable collection in the third 100s window as well, but is able to evaluate a new collection every 100s.

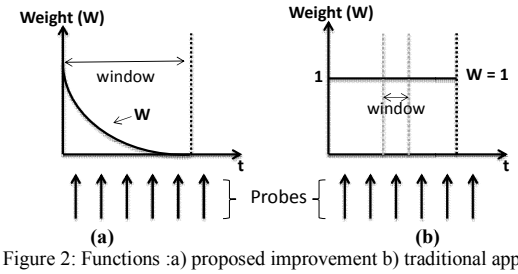

Figure 2: Functions :a) proposed improvement b) traditional approach

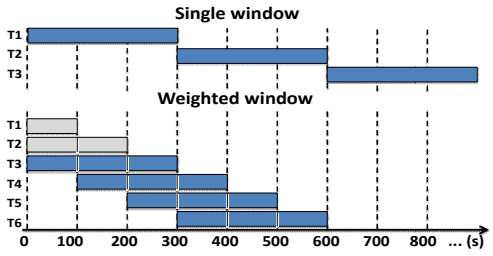

Figure 3: collection begins when a complete set of time slices has elapsed

```
INTEGER N: number of probes
INTEGER I: i^th Vector
baselineNetworkState <- INTEGER [0,0,...,0]
FOR EACH ProbeVector I < N
  baselineNetworkState <- baselineNetworkState + W(I) * ProbeVector
  I <- I + 1
ENDFOR;
baselineNetworkState <- Normalize(baselineNetworkState)
```

Algorithm 1: weighted average vector for baseline

B. Evaluation

The presented use case was executed for the traditional single window and for the weighted windows proposals, and the same time interval for traffic assessment was selected for both. Two variations were executed, but due to space constraints, only the most effective results are presented. The comparison with the less effective experiment is further shown in Discussion (Section VI).

The experiment with the following results was ran using a window width of 500s – 1x500s window for the single approach and 5x100s windows for the weighted – defined empirically from an observation of Figure 1. With this configuration, the weighted windows approach showed an improvement in placement. Figure 4 illustrates that, for the weighted windows, replacement took place more often and earlier if compared with the single window behavior.

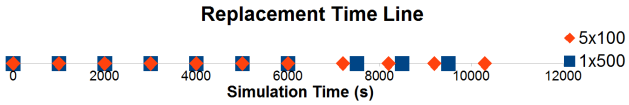

Figure 4: Content replacement occurrence in the two approaches

The new pattern of placements was noticed when one analyzes the performance of the network by the established metrics. Figure 5 and Figure 6 show that the single window approach has a worse performance (more bandwidth usage) when compared to the weighted windows approach.

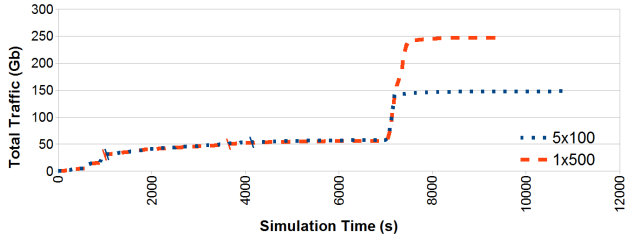

Figure 5: Compared cross network traffic

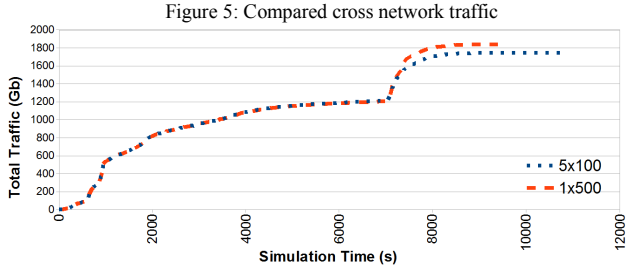

Figure 6: Compared inner network traffic

Likewise, because of the improved replacements, the weighted window approach also had better results concerning user experience. Figure 7 shows a slightly lower start up delay of weighted windows approach, compared with single window – 0.296 for the former and 0.311 for the latter.

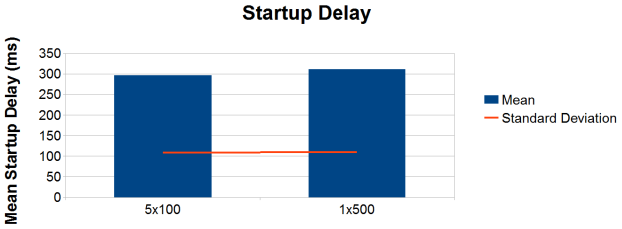

Figure 7: Comparison of startup delay of both approaches

V. REAL METRICS

The previous attempt of network change detection, weighted moving average vector, presented rather inconclusive results. As it is further discussed in Section VI, the experiments on this approach didn't lead to any confidence, since the strategy is highly dependent on the window length.

Once a set with all N relevant routers is defined, any interest metric of all involved routers can be collected and have its values organized in a vector. Ideally, many metric vectors can be gathered with each other. Take, for instance, vectors R and T, with the respective baseline representations B_R and B_T. The baselines have the same structure of their representing vectors, but hold the values of the last representative state. The baseline vectors are used to obtain Δ_R and Δ_T, the dot product between the baselines and the current state vectors, respectively from the round-trip delays and the total router traffic. Both vectors are then normalized to simplify the comparison of metrics, since the vectors may hold heterogeneous values.

Hereinafter, a function to join the Δ values for both metrics is used to weigh the most relevant attributes and come out with a single decimal value, ranging from 0 to 1:

$$f(\Delta_R, \Delta_T) = \Delta_R . w_R + \Delta_T . w_T$$

Where $w_R + w_T = 1$ and values of w_R and w_T are preferably obtained empirically. The resulting value of the function f is compared to a pre-defined threshold which will state whether an update of the baseline vectors will occur. In the positive case, the current vectors are now the most recent representative state and will replace their respective baselines.

The first solution concerning real metrics, considered effective but not practical, considers two metrics, one-way delay and aggregated network traffic. The new metrics, based on real internet data, should reflect the same meaning as the former ones and in addition, must be easily accessible by any CDN. To approximate network latency, client startup delay (D) was assigned. For network traffic, since it is feasible for a CDN operator to measure, the total traffic of the surrogates (T) was used. To shorten the distance between the old simulator-tied data and proposed real internet metrics, the proposal gathers network information and traffic information. The traffic is calculated by summing all sent data or storing bitrates information by time slice. This way, every T seconds will produce P values, able to compose the vector representing the status of the network regarding network traffic. Thus, network traffic can be represented as:

$$T_p = \sum_{i=1}^{M_p} Tc_i^p \, H_i^p$$

Where H_i^p is the number of hops between the client, or client's AS, and the surrogate server. The latency for each surrogate will be:

$$L_p = \sum_{i=1}^{M_p} S_i^p$$

Where S_i^p is the start up delay experienced by the client i when getting content from surrogate p. Again, calculating this metric should be very simple for the CDN.

Five experiments were proposed to detect the changes comparing the previously presented method with variations of the proposed improvement. The experiment named NONE is the baseline for the further experiments, and it is defined to run without resource reallocation during the simulation. Surrogates are placed, based on prediction of all future requests, in the beginning of the simulation and remain there. The second experiment, labeled BITSENT, also establishes an observation baseline, since it was built upon an optimal omniscient environment – the previous work [12]. The three remaining experiments are variations of the real network data approach. It combines the use of real traffic and start up delay metrics. Configurations considering only traffic (REALDATA_T100_SD0) and only start up delay (REALDATA_T0_SD100) were evaluated, so the contribution of each metric can be individually compared with the established baselines. Additionally, an empiric configuration of mixing up both metrics to obtain a weighted network behavior point of view was set up, using 80% relevance for traffic and 20% relevance for startup delay (REALDATA_T80_SD20). With this configuration we hope to superficially observe the impact in using more than one metric to evaluate the network.

A. Evaluation

The scenario described in the case study of Section III was used to validate the use of real data to assess and manage CDNs. In order to obtain the best configuration parameters for the simulation, a number of experiments were run, so it is possible to extract how the use case will behave, to set the parameters properly. Using these results, the setup of another needed configuration was assigned – the threshold for network status changes was set to 0.75 in the real data configuration, the best value for the configuration, as observed empirically.

Excited about the promising results (further detailed above), two variations of the regular scenario were designed specifically for this strategy, in order to validate the changes detection along with differently dimensioned scenarios. In the STRETCHED SCENARIO, the goal was to observe the behavior of the usage of real metrics to notify network changes for management purposes in a more lasting simulation and more user requests. The SHRUNK SCENARIO, on the other hand, is an attempt to intensify the amount of requests per time – a bigger amount of requests in smaller simulation time.

Both new scenarios follow the guideline of having flash crowd situations. Due to clarity issues, if in any of the figures shown in this section there is more than one strategy labeled with the same color that means that both strategies had statistically equivalent results. The test used was the Kolmogorov–Smirnov [21] and strategies considered equivalent had a p-value < 1. The following sections present the results of the three scenario variations.

1) Regular scenario

The timeline below (Figure 8) permits an overview of which configurations lead to the expected replacements, which in this scenario, should happen about time 900s and time 7000s (the flash crowds). The results show that NONE configuration doesn't end up with replacements, as expected, since it has no data collection to do so. Also as expected, the first flash crowd (time 900s) isn't noticed because of the single 500s data collection window, which ignores the 100s duration traffic boost. BITSENT configuration, our baseline, performs the replacements after detecting enough amount of traffic after the second flash crowd, while REALDATA_T100_SD0 and

REALDATA_T80_SD20, even using limited data, are able to follow the replacements of the baseline by the same timestamp. REALDATA_T0_SD100, due to assigning too much weight to a not so relevant metric, only performs the replacement two collection frames (two 500s windows) later – which is an indicative that weighting adequately many metrics is probably better than using a chosen one individually.

The traffic metric charts show the impact of each collection strategy in network's overall performance. As expected, the behavior of Cross Traffic (Figure 9) and Inner Traffic (Figure 10) follow equal until the first relevant resource reallocation action is taken (from Figure 8 one can see it is in time 7500s).

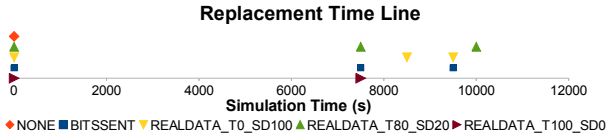

Figure 8: Surrogate placement for all the configurations along time

In Cross traffic chart, the difference becomes noticeable from the point that the first replacement occurs, by that time, all the traffic was inside the requester AS, the same behavior reflected in Inner traffic chart. In time 7500s, the occurrence of the first surrogate replacement, sets a milestone in simulation. From this point forward the approaches diverge: NONE and REALDATA_T0_SD100 show a traffic increase whereas BITSENT, REALDATA_T100_SD0 and REALDATA_T80_SD20 configurations keep similar, using less bandwidth. Figure 8 also shows that when the demand is critical (flash crowd at time 7000s), the configuration that considers only start up delay fails to trigger a replacement – issue possibly corrected fine tuning the startup delay threshold. Therefore, REALDATA_T0_SD100 behaves like NONE (no surrogate replacement), whereas the other three configurations successfully detect the need for replacing, improving the afterwards bandwidth usage observed in the chart below.

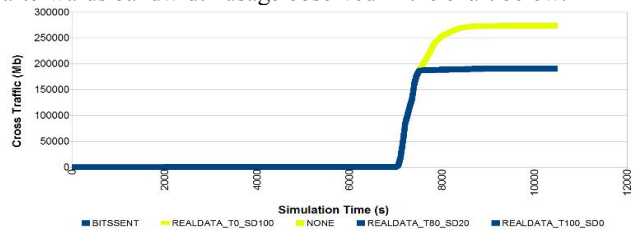

Figure 9: Cross traffic comparison for the real metrics experiments

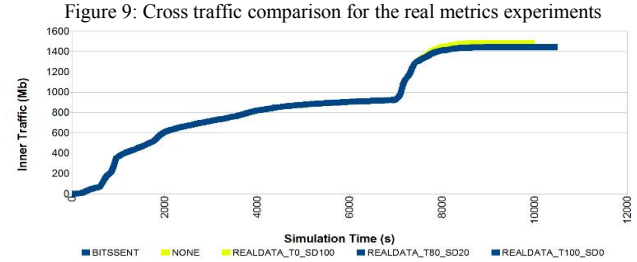

Figure 10: Inner traffic comparison for the real metrics experiments

2) Stretched scenario

The STRETCHED SCENARIO is composed of 30047 requests during 25000 seconds, split as follows: 20026 request as background traffic in AS1 and AS2, 1007 requests from AS1 representing the first flash crowd during the interval between times 900s to 1000s, 4005 requests from AS2 in the second flash crowd from times 7000s to 9000s and the last flash crowd comprising 5009 requests from AS2 from time 20000s to time 22000s. The first flash crowd is positioned in a high background traffic moment, the second one in an intermediate network usage state and the third, in a lull state.

The purpose of this experiment was to observe the power of the metrics to detect changes in a larger scenario. For the sake of space, and because the point of this experiment is to come up with a valid Internet-based metric, only the most relevant comparisons will be presented. Figure 11 and Figure 12 show respectively cross traffic and inner traffic for the omniscient (BITSSENT) and the total traffic real metric (REALDATA_T100_SD0). The results were similar to the regular scenario, but the BITSSTENT monitor performed slightly better than REALDATA_T100_SD0. On the other hand, the real metric monitor was better in keeping the traffic local, resulting in more inner traffic and less cross traffic.

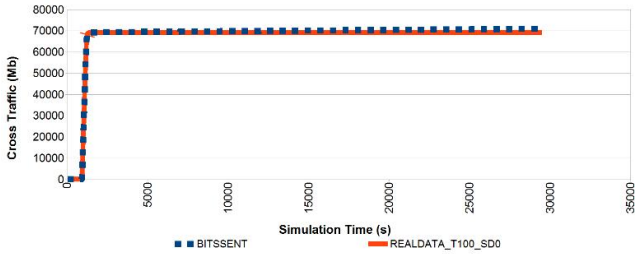

Figure 11: Cross traffic bandwidth usage along simulation time

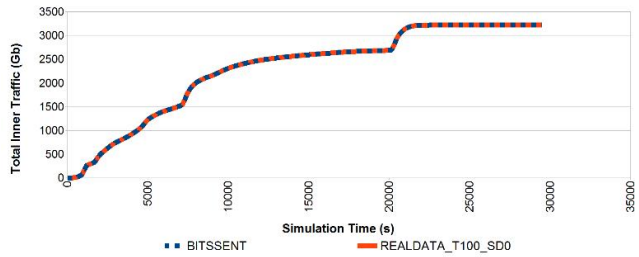

Figure 12: Inner traffic bandwidth usage along simulation time

3) Shrunk scenario

The SHRUNK SCENARIO has 30215 requests spread among 15000 seconds of simulation. As observed, this scenario is much denser than the previous ones, that is, it has a bigger number of requests per second. The requests are divided in 16199 background requests departing from AS1 and AS2, 7008 requests from AS1 between the times of 5000s and 7000s, during the rush time, and the second flash crowd, 7008 requests from AS2 between times 12000s and 14000s, during the lull time.

Charts in Figure 13 and Figure 14 compare the performances of all experiments altogether. As the lines show, the real metrics scenarios (REALDATA*) performed as good as the omniscient one (BITSSENT). In this scenario, REALDATA metrics were able to identify the changes in network conditions on the expected times (flash crowds), reducing considerably the cross traffic when comparing to the NONE scenario. However, likewise the STRETCHED scenario, the first flash crowd event was not detected. We assume this was consequence of the way the base traffic (normal operation) was distributed, since it concentrated on the same ASes that generated the flash crowd.

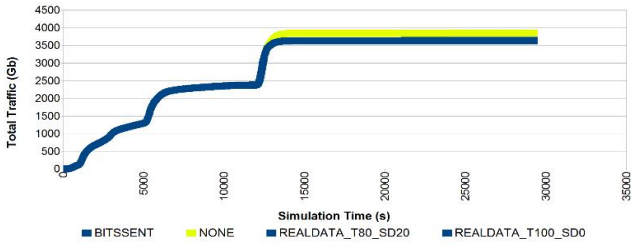

Figure 13: Total network traffic along simulation time

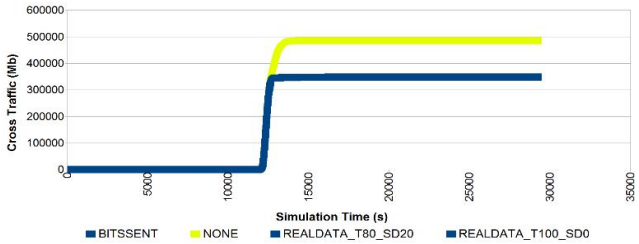

Figure 14: Total cross traffic along simulation time

As for the QoE point of view, in this scenario, as expected, the result was better than REGULAR scenario where QoE was considerably worst in the experiment with replica servers' replacement. Likely, this behavior occurs due to requests being concentrated on the same ASes that generated the flash crowd. Due to the considerable increase on localized requests during a flash crowd, the placement strategy replaces all the replica servers together in the flash crowd region. Considering the REGULAR scenario, requests aside from localized flash crowds were scattered on all ASes thus, reallocating all replica servers to the flash crowd´s region resulted in QoE problems to all other clients. Figure 15 shows mean startup delay for the SHRUNK scenario. There is no considerable difference between scenarios with and without cache replacement.

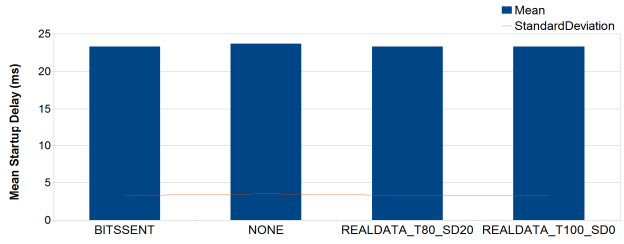

Figure 15: Mean startup delay for the SHRUNK SCENARIO

VI. Discussion, Conclusions and Future Work

The following subsections present our critical point of view concerning the presented strategies, possible solutions for the found issues and concluding remarks about this work.

A. Weighted moving average window

The experiments on the weighted moving average vector approach pointed out that window length clearly is an impacting factor and, so is the scenario, once a different request distribution could make even the best performing configuration to fail. A first row of experiments, using a window length of 300 seconds presented lower performance than the presented results of length 500s window. A possible justification for this is that the weighted windows can take advantage on more data to analyze and consider the recency of each time frame whereas the single window approach "dilutes" information among the whole interval.

Therefore, in order to fully validate the proposal as useful to the intended purposes, either an empirically found window configuration must be used, to comply with most of the scenarios, or a self-configuring approach has to be implemented, so it adapts to the dynamic of the events accordingly. It could use AI techniques to self-fixation of window size.

B. Real metrics

The main contribution of this work is a method to detect significant changes in an overlay network, similar to the previous mentioned work [12], but with limited monitoring information, as in real CDN environments. We showed that event with this limited information, we can achieve results similar to an optimal omniscient environment where the system has full information of all possible monitored data.

Our second and third attempts to validate the strategy already presented animating results. Increasing the number of requests and the observed experiment time (STRETCHED) resulted in detecting changes consistently with the regular scenario approach. Likewise, when the designed scenario was denser (SHRUNK) than the regular, we could observe a similar performance in change detection.

For future work, we plan to extend our experiments with bigger scenarios (more requests) and test different combination of weighted metrics used, in order to find out the best balance between the considered metrics as well as include other feasible CDN metrics. Different request distributions (more scattered or more concentrated in the ASes) are also matter for further studies, as well as experiment the detection strategy variations along with different replacement approaches.

C. Round-trip delay

Since the main idea was to detect network changes through Round-trip Delay (RTD) variation, the network topology was configured in such a way that collected RTD values could be used as an indication for network changes. Tuning this turned out to be a difficult task. Although it is considerably easy to configure a scenario where RTD values change according to the network changes and raise the flag of a threshold, to do that in a simulator [4] we had to change variables in way that the result would be a nonrealistic scenario. For instance, it was necessary to use lower link capacities and increase link latencies to values incoherent to the size of the simulated network.

A conclusion on the matter is that a suitable way to deal with this issue would be to use a network traffic generator along with the simulator to include background traffic in the experiments. The extra network usage in addition with CDN overlay traffic should result in a considerable RTD variation triggering network change detection. Considering the time that could be spent incorporating the network traffic generator to the simulator, and the importance of having results from other techniques, coupled the troubles regarding feasibility of this strategy for now, authors decided leaving such fixes as future work.

Acknowledgment

This work was supported by Ericsson Telecomunicações S.A., Brazil.

REFERENCES

[1] Pathan, A. K., and Buyya, R. 2007. "A taxonomy and survey of content delivery networks." Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report.

[2] DATANYZE. 2013. Available at http://blog.datanyze.com/cdn-market-share-update-december-2013/ . Last access in July, 2014.

[3] CISCO. 2014. "Cisco Visual Networking Index: Forecast and Methodology, 2012–2017". Last access in February, 2014.

[4] Rodrigues, M. B. E. ; Moreira, A. L. C. ; Azevedo, E. ; Neves, M.; Sadok, D.; Callado, A. ; Moreira, J. . On Learning How to Plan Content Delivery Networks. In: 46th ANSS, 2013, San Diego.

[5] Papagianni, C.; Leivadeas, A; Papavassiliou, S., "A Cloud-Oriented Content Delivery Network Paradigm: Modeling and Assessment," Dependable and Secure Computing, IEEE Transactions on , vol.10, no.5, pp.287,300, Sept.-Oct. 2013.

[6] Buyya, R., et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems. Volume 25, Number 6, June 2009.

[7] Dave Evans. The Internet of Things. How the next evolution of Inetnet is changing everything. Cisco Internet Business Solutions Group (IBSG). Available at. http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf April, 2011.

[8] Norihiko Yoshida. Dynamic CDN Against Flash Crowds. Book Chaper. Content Delivery Networks. Lecture Notes Electrical Engineering Volume 9, 2008, pp 275-296.

[9] James Broberg, Rajkumar Buyya, Zahir Tari. MetaCDN: Harnessing 'Storage Clouds' for high performance content delivery. Journal of Network and Computer Applications, Volume 32, Issue 5, September 2009, Pages 1012-1022.

[10] Dongyan Xu, Sunil Suresh Kulkarni, Catherine Rosenberg, Heung-Keung Chai. Analysis of a CDN-P2P Hybrid Architecture f or Cost-Effective Streaming Media Distribution. Multimedia Systems. April 2006, Volume 11, Issue 4, pp 383-399.

[11] Jung J, Krishnamurthy B, Rabinovich M (2002) Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In: Proc. 11th Int. World Wide Web Conf., 252-262.

[12] Moreira, A., Moreira, J., Sadok, D., Callado, A., Rodrigues., Neves, M., Souza, V., and Karlsson, P. 2011. "A Case for Virtualization of Content Delivery Networks", In: IEEE. Winter Simulation Conference.

[13] Srinivasan, J.W.J. Lee, D. Batni, and H. Schulzrinne, "Active-CDN: Cloud Computing Meets Content Delivery Networks," technical report, Columbia Univ., 2012.

[14] Zao W, Schulzrinne H (2004) DotSlash: A Self-Conguring and Scalable Rescue System for Handling Web Hotspots Effectively. In: Proc. Int. Workshop on Web Caching and Content Distribution, 1-18.

[15] Freedman MJ, Freudenthal E, Mazieres D (2004) Democratizing Content Publication with Coral. In: Proc. 1st USENIX/ACM Symp. on Networked Systems Design and Implementation.

[16] Arlitt M, Cherkasova L, Dilley J, Friedrich R, Jin T (1999) Evaluating Content Management Techniques for Web Proxy Caches. In: ACM SIGMETRICS Performance Evaluation Review, 27(4):3-11.

[17] P. Krishnan, D. Raz, and Y. Shavitt, "The cache location problem," IEEE/ACM Transactions on Networking (TON), vol. 8, no. 5, 2000.

[18] Padmanabhan VN, Sripanidkulchai K. (2002) The Case for Cooperative Networking. In: Proc. 1st Int. Workshop on Peer-to-Peer Systems, 178-190.

[19] Dán, G. & Carlsson, N. (2014). Dynamic Content Allocation for Cloud-assisted Service of Periodic Workloads. In: 33rd Annual IEEE (INFOCOM'14); Toronto, Canada, 27 April - May 2, 2014.

[20] Elkotob, M.; Andersson, K., "Challenges and opportunities in content distribution networks: A case study," Globecom Workshops (GC Wkshps), 2012 IEEE , vol., no., pp.1021,1026, 3-7 Dec. 2012.

[21] M. Hollander, D. A. Wolfe, and E. Chicken. "Nonparametric statistical methods". Vol. 751. John Wiley & Sons, 2013.