IJDCST @Oct-Nov-2015, Issue- V-3, I-7, SW-06
ISSN-2320-7884 (Online)
ISSN-2321-0257 (Print)

Dipping Cost of Cloud Bandwidth using Prediction-Based System

Shaik Sharmila[1], V Padmaja[2], Sayeed Yasin[3]
[1]M.Tech (IT), Nimra College of Engineering and Technology, A.P., India.
[2]Asst.professor, Dept. of Computer Science & Engineering, Nimra College of Engineering and Technology, A.P., India.
[3]Head of the Department, Dept. of Computer Science & Engineering, Nimra College of Engineering & Technology, A.P., India.

Abstract - In Cloud Computing environment we designed and present PACK (Predictive ACKs), end to end traffic redundancy elimination (TRE) system called for cloud computing customers.. PACK's main advantage is its capability of offloading the cloud-server TRE effort to end clients, thus minimizing the processing costs induced by the TRE algorithm. Unlike previous solutions, PACK does not require the server to continuously maintain clients' status. Cloud-based TRE needs to apply a judicious use of cloud resources so that the bandwidth cost reduction combined with the additional cost of TRE computation and storage would be optimized. This makes PACK very suitable for pervasive computation environments that combine client mobility and server migration to maintain cloud elasticity. We present a fully functional PACK implementation, transparent to all TCP-based applications and network devices. Finally, we analyze PACK benefits for cloud users, using traffic traces from various sources. PACK is based on a novel TRE technique, which allows the client to use newly received chunks to identify previously received chunk chains, which in turn can be used as reliable predictors to future transmitted chunks.

***Keywords* — Caching, cloud computing, network optimization, traffic redundancy elimination.**

I. INTRODUCTION

Cloud customers are increasing day by day in the world where Cloud customers has to pay only for the actual use of computing resources, storage, and bandwidth, according to their changing needs, utilizing the cloud's scalable and elastic computational capabilities. In particular, data transfer costs (i.e., bandwidth) is an important issue when trying to minimize costs [1], [2]. Consequently, cloud customers, applying a judicious use of the cloud's resources, are motivated to use various traffic reduction techniques, in particular traffic redundancy elimination (TRE), for reducing bandwidth costs.

Traffic redundancy stems from common end-users' activities,
such as repeatedly accessing, downloading, uploading (i.e.,
backup), distributing, and modifying the same or similar information items (documents, data, Web, and video). TRE is used to eliminate the transmission of redundant content and, therefore, to significantly reduce the network cost. In most common TRE solutions, both the sender and the receiver examine and compare signatures of data chunks, parsed according to the data content, prior to their transmission. When redundant chunks are detected, the sender replaces the transmission of each redundant chunk with its strong signature [3]–[5]. Commercial TRE solutions are popular at enterprise networks, and

involve the deployment of two or more proprietary-protocol, state synchronized middle-boxes at both the intranet entry points of data centers and branch offices, eliminating repetitive traffic between them While proprietary middle-boxes are popular point solutions Within enterprises, they are not as attractive in a cloud environment. Cloud providers cannot benefit from a technology whose goal is to reduce customer bandwidth bills, and thus are not likely to invest in one. The rise of “on-demand” work spaces, meeting rooms, and work-from-home solutions [3] detaches the workers from their offices. In such a dynamic work environment, fixed-point solutions that require a client-side and a server-side middle-box pair become ineffective. On the other hand, cloud-side elasticity motivates work distribution among servers and migration among data centers. Therefore, it is commonly agreed that a universal, software-based, end-to-end TRE is crucial in today’s pervasive environment [4], [5]. This enables the use of a standard protocol stack and makes a TRE within end-to-end secured traffic (e.g., SSL) possible.

Current end-to-end TRE solutions are sender-based. In the case where the cloud server is the sender, these solutions require that the server continuously maintain clients’ status. We show here that *cloud elasticity* calls for a new TRE solution. First, cloud load balancing and power optimizations may lead to a server-side process and data migration environment, in which TRE solutions that require full synchronization between the server and the client are hard to accomplish or may lose efficiency due to lost synchronization. Second, the popularity of rich media that consume high bandwidth motivates content distribution network (CDN) solutions, in which the service point for fixed and mobile users may change dynamically according to the relative service point locations and loads. Moreover, if an end-to-end solution is employed, its additional computational and storage costs at the cloud side should be weighed against its bandwidth saving gains.

Clearly, a TRE solution that puts most of its computational effort on the cloud side2may turn to be less cost-effective than the one that leverages the combined client-side capabilities. Given an end-to-end solution, we have found through our experiments that sender-based end-to-end TRE solutions [4], [3] add a considerable load to the servers, which may eradicate the cloud cost saving addressed by the TRE in the first place. Our experiments further show that current end-to-end solutions also suffer from the requirement to maintain end-to-end synchronization that may result in degraded TRE efficiency. In this paper, we present a novel receiver-based end-to-end TRE solution that relies on the power of predictions to eliminate redundant traffic between the cloud and its end-users. In this solution, each receiver observes the incoming stream and tries to match its chunks with a previously received chunk chain or a chunk chain of a local file. Using the long-term chunks’ metadata information kept locally, the receiver sends to the server predictions that include chunks’ signatures and easy-to-verify hints of the sender’s future data. The sender first examines the hint and performs the TRE operation only on a hint-match. The purpose of this procedure is to avoid the expensive TRE computation at the sender side in the absence of traffic redundancy. When redundancy is detected, the sender then sends to the receiver only the ACKs to the predictions, instead of sending the data.

On the receiver side, we propose a new computationally lightweight chunking (fingerprinting) scheme termed *PACK chunking*. PACK chunking is a new alternative for Rabin fingerprinting traditionally used by RE applications. Experiments show that our approach can reach data processing speeds over 3 Gb/s, at least 20% faster than Rabin fingerprinting. Offloading the computational effort from the cloud to a large group of clients forms a load distribution action, as each client processes only its TRE part. The receiver-based TRE solution addresses mobility problems common to quasi-mobile desktop/ laptops computational environments. One of them is cloud elasticity due to which the servers are dynamically relocated around the federated cloud, thus causing clients to interact with multiple changing servers. Another property is IP dynamics, which compel roaming users to frequently change IP addresses. In addition to the receiver-based operation, we also suggest a hybrid approach, which allows a battery-powered mobile device to shift the TRE computation overhead back to the cloud by triggering a sender-based end-to-end TRE similar to [5].To validate the receiver-based TRE concept, we implemented, tested, and performed realistic experiments with PACK within a cloud environment. Our experiments demonstrate a cloud cost reduction achieved at a reasonable client effort while gaining additional bandwidth savings at the client side. The implementation code, over 25 000 lines of C and Java, can be obtained from [6]. Our implementation utilizes the TCP Options field, supporting all TCP-based applications such as Web, video streaming, P2P, e-mail, etc. In addition, we evaluate our solution and compare it to previous end-to-end solutions using terabytes of real video traffic consumed by 40 000 distinct clients, captured within an ISP, and traffic brained in a social network service for over a month. We demonstrate hat our solution achieves 30% redundancy elimination without significantly affecting the computational effort of the sender, resulting in a 20% reduction of the overall cost to the cloud customer

II. RELATED WORK

Several commercial TRE solutions described in [6] and [7] have combined the sender-based TRE ideas of [4] with the algorithmic and implementation approach of [5] along with protocol specific optimizations for middle-boxes solutions. In particular,[6] describes how to get away with three-way handshake between the sender and the receiver if a full state synchronization is maintained. Several TRE techniques have been explored in recent years. A protocol-independent TRE was proposed in [4]. A large-scale study of real-life traffic redundancy is presented in [8], and [4]. In the latter, packet-level TRE techniques are compared [11].Our paper builds on their finding that “an end to end redundancy elimination solution, could obtain most of the middle-box’s bandwidth savings,” motivating the benefit of low cost software end-to-end solutions.Wanax is a TRE system for the developing world where storage and WAN bandwidth are scarce.. In this scheme, the sender middle-box holds back the TCP stream and sends data signatures to the receiver whether the data is found in its local cache. Data chunks that are not found in the cache are fetched from the by receiver middle. Naturally, such a scheme incurs three-way-and shake latency for no cached datacenter [5] is a sender-based end-to-end TRE for enterprise networks. It uses a new chunking scheme that is faster than the commonly used Rabin fingerprint, but is restricted to chunks as small as 32–64 B. Unlike PACK, Ender requires the server to maintain a fully and reliably synchronized cache for each client. To adhere with the server’s memory requirements, these caches are kept small, making the system inadequate for medium-to-large content or long-term redundancy. End RE is server-specific, hence not suitable for a CDN or cloud environment.

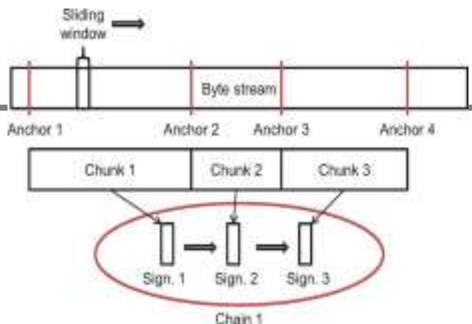

Fig. 1. From stream to chain

To the best of our knowledge, none of the previous works have addressed the requirements for a cloud-computing- friendly, end-to-end TRE, which forms PACK's focus.

III. PACK ALGORITHM

For the sake of clarity, we first describe the basic receiver driven operation of the PACK protocol. Several enhancements and optimizations are introduced. The stream of data received at the PACK receiver is parsed to a sequence of variable-size, content-based signed chunks similar to [3] , [9] and [5]. The chunks are then compared to the receiver local storage, termed *chunk store*. If a matching chunk is found in the local chunk store, the receiver retrieves the sequence of subsequent chunks, referred to as a *chain*, by traversing the sequence of LRU chunk pointers that are included in the chunks' metadata. Using the constructed chain, the receiver sends a prediction to the sender for the subsequent data. Part of each chunk's prediction, termed a *hint*, is an easy-to-compute function with a small-enough false-positive value, such as the value of the last byte in the predicted data or a byte-wide XOR checksum of all or selected bytes. The prediction sent by the receiver includes the range of the predicted data, the hint, and the signature of the chunk. The sender identifies the predicted range in its buffered data and verifies the hint for that range. If the result matches the received hint, it continues to perform the more computationally intensive SHA-1 signature operation. Upon a signature match, the sender sends a confirmation message to the receiver, enabling it to copy the matched data from its local storage.

A. Receiver Chunk Store

PACK uses a new *chains* scheme, described in Fig. 1, in which chunks are linked to other chunks according to their last
received order. The PACK receiver maintains a *chunk store*,
which is a large size cache of chunks and their associated metadata. Chunk's metadata includes the chunk's signature and a (single) pointer to the successive chunk in the last received stream containing this chunk. Caching and indexing techniques are employed to efficiently maintain and retrieve the stored chunks, their signatures, and the chains formed by traversing the chunk pointers. When the new data are received and parsed to chunks, the receiver computes each chunk's signature using SHA-1. At this point, the chunk and its signature are added to the chunk store. In addition, the metadata of the previously received chunk in the same stream is updated to point to the current chunk. The unsynchronized nature of PACK allows the receiver to map each existing file in the local file system to a chain of chunks, saving in the chunk store only the metadata associated with the chunks.3 Using the latter observation, the receiver can also share chunks with peer clients within the same local network utilizing a simple map of network drives. The utilization of a small chunk size presents better redundancy elimination when data modifications are fine-grained,

such as sporadic changes in an HTML page. On the other hand, the use of smaller chunks increases the storage index size, memory usage, and magnetic disk seeks. It also increases the transmission overhead of the virtual data exchanged between the client and the server. Unlike IP-level TRE solutions that are limited by the IP packet size (B) , PACK operates on TCP streams and can therefore handle large chunks and entire chains. Although our design permits each PACK client to use any chunk size, we recommend an average chunk size of 8 kB *B.*

Receiver Algorithm

Upon the arrival of new data, the receiver computes the respective signature for each chunk and looks for a match in its local chunk store. If the chunk's signature is found, the receiver determines whether it is a part of a formerly received chain, using the chunks' metadata. If affirmative, the receiver sends a prediction to the sender for several next expected chain chunks. The prediction carries a starting point in the byte stream (i.e., offset) and the identity of several subsequent chunks (PRED command). Upon a successful prediction, the sender responds with a PRED-ACK confirmation message. Once the PRED-ACK message is received and processed, the receiver copies the corresponding data from the chunk store to its TCP input buffers, placing it according to the corresponding sequence numbers. At this point, the receiver sends a normal TCP ACK with the next expected TCP sequence number. In case the prediction is false, or one or more predicted chunks are already sent, the sender continues with normal operation, e.g., sending the raw data, without sending a PRED-ACK message.

Proc. 1: Receiver Segment Processing

```
1. if segment carries payload data then
2. calculate chunk
3. if reached chunk boundary then
4. activate predAttempt()
5. end if
6. else if PRED-ACK segment then
7. processPredAck()
8. activate predAttempt()
9. end if
```

Proc. 2: predAttempt()

```
1. if received chunk matches one in chunk store then
2. if foundChain(chunk) then
3. prepare PREDs
4. send single TCP ACK with PREDs according to
Options free space
5. exit
6. end if
7. else
8. store chunk
9. link chunk to current chain
10. end if
11. send TCP ACK only
```

Proc. 3: processPredAck()

```
1. for all offset PRED-ACK do
2. read data from chunk store
3. put data in TCP input buffer
4. end for
```

IV. IMPLEMENTATION

In this section, we present PACK implementation, its performance analysis, and the projected server costs derived from the implementation experiments. Our implementation contains over 25 000 lines of C and Java code. It runs on Linux with Net filter Queue [11]. Fig. 2 shows the PACK implementation architecture.

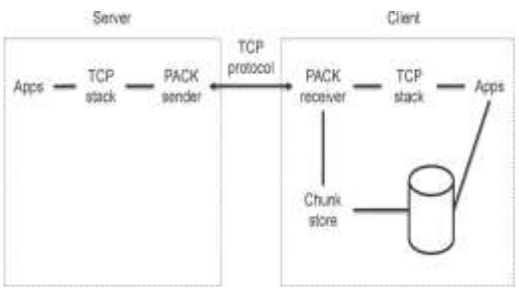

Fig. 2: Overview of the PACK implementation

At the server side, we use an Intel Core 2 Duo 3 GHz, 2 GB of RAM, and a WD1600AAJS SATA drive desktop. The clients laptop machines are based on an Intel Core 2 Duo 2.8 GHz, 3.5 GB of RAM , and a WD2500BJKT SATA drive. Our implementation enables the transparent use of the TRE at both the server and the client. PACK receiver–sender protocol is embedded in the TCP Options field for low overhead and compatibility with legacy systems along the path. We keep the genuine operating systems' TCP stacks intact, allowing a seamless integration with all applications and protocols above TCP. Chunking and indexing are performed only at the client's side, enabling the clients to decide independently on their preferred chunk size. In our implementation, the client uses an average chunk size of 8 kB. We found this size to achieve high TRE hit-ratio in the evaluated datasets, while adding only negligible overheads of 0.1% in metadata storage and 0.15% in predictions bandwidth. For the experiments held in this section, we generated a workload consisting of datasets: IMAP e-mails, HTTP videos, and files downloaded over FTP. The workload was then loaded to the server and consumed by the clients. We sampled the machines' status every second to measure real and virtual traffic volumes and CPU utilization. A. Server Operational Cost We measured the server performance and cost as a function of the data redundancy level in order to capture the effect of the TRE mechanisms in real environment. To isolate the TRE operational cost, we measured the server's traffic volume and CPU utilization at maximal throughput without operating a TRE. We then used these numbers as a reference cost, based on present Amazon EC2 [10] pricing. The server operational cost is composed of both the network traffic volume and the CPU utilization, as derived from the EC2 pricing. We constructed a system consisting of one server and seven clients over a 1-Gb/s network. The server was configured to provide a maximal throughput of 50 Mb/s per client. We then measured three different scenarios: a baseline no-TRE operation, PACK, and a sender-based TRE similar to End RE's Chunk-Match [12], referred to as End RE-like. For the End RE-like case, we accounted for the SHA-1 calculated over the entire outgoing traffic, but did not account for the chunking effort. In the case of End RE-like, we made the assumption of unlimited buffers at both the server and client sides to enable the same long-term redundancy level and TRE ratio of PACK.

Presents the overall processing and networking cost for traffic redundancy, relative to no-TRE operation. As the redundancy grows, the PACK server cost decreases due to the bandwidth saved by unsent data. However, the End RE-like server does not gain a significant cost reduction since the SHA-1 operations are performed over non redundant data as well. Note that at above 25% redundancy, which is common to all reviewed datasets, the PACK operational cost is at least 20% lower than that of End RE-like.

B. PACK Impact on the Client CPU

To evaluate the CPU effort imposed by PACK on a client, we
measured a random client under a scenario similar to the one used for measuring the server's cost, only this

time the cloud server streamed videos at a rate of 9 Mb/s to each client. Such
a speed throttling is very common in real-time video servers that aim to provide all clients with stable bandwidth for mooth view.
Table IV summarizes the results. The average PACK-related
CPU consumption of a client is less than 4% for 9-Mb/s video
with 36.4% redundancy. Fig. 12(a) presents the client CPU utilization as a function of the real incoming traffic bandwidth. Since the client chunks the arriving data, the CPU utilization grows as more real traffic enters the client's machine. Fig. 12(b) shows the client CPU utilization as a function of the virtual traffic bandwidth. Virtual traffic arrives in the form of prediction approvals from the sender and is limited to a rate of 9 Mb/s by the server's throttling. The approvals save the client the need to chunk data or sign the chunks and enable him to send more predictions based on the same chain that was just used successfully. Hence, the more redundancy is found, the less CPU utilization incurred by PACK.

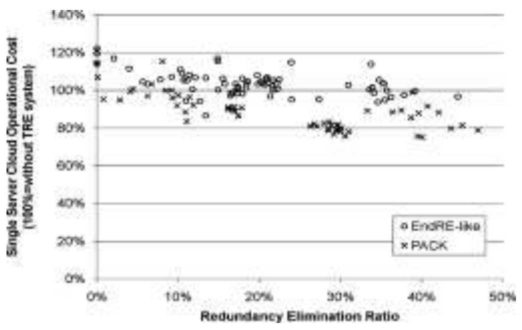

Fig 3: PACK versus End RE-like cloud server operational cost as a function of redundancy ratio.

V. CONCLUSION

In this paper, we have presented PACK , a receiver-based, Cloud-friendly, end-to-end TRE that is based on novel speculative principles that reduce latency and cloud operational cost. PACK does not require the server to continuously maintain clients' status, thus enabling cloud elasticity and user mobility while preserving long-term redundancy. Moreover, PACK is capable of eliminating redundancy based on content arriving to the client from multiple servers without applying a three-way handshake. Our evaluation using a wide collection of content types shows that PACK meets the expected design goals and has clear advantages over sender-based TRE, especially when the cloud computation cost and buffering requirements are important. Moreover, PACK imposes additional effort on the sender only when redundancy is exploited, thus reducing the cloud overall cost. Two interesting future extensions can provide additional benefits to the PACK concept. First, our implementation maintains chains by keeping for any chunk only the last observed subsequent chunk in an LRU fashion. An interesting extension to this work is the statistical study of chains of chunks that would enable multiple possibilities in both the chunk order and the corresponding predictions. The system may also allow making more than one prediction at a time, and it is enough that one of them will be correct for successful traffic elimination. A second promising direction is the mode of operation optimization of the hybrid sender–receiver approach based on shared decisions derived from receiver's power or server's cost changes.

References

[1] E. Zohar, I. Cidon, and O. Mokryn, "The power of prediction: Cloud bandwidth and cost reduction," in Proc. SIGCOMM, 2011, pp. 86–97.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph,R.Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," Commun. ACM, vol. 53, no. 4, pp. 50–58, 2010.

[3] U. Manber, "Finding similar files in a large file system," in Proc. USENIX Winter Tech. Conf., 1994, pp. 1–10.

[4] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in Proc. SIGCOMM, 2000, vol. 30, pp. 87–95.

[5] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in Proc. SOSP, 2001, pp. 174–187.

[6] E. Lev-Ran, I. Cidon, and I. Z. Ben-Shaul, "Method and apparatus for reducing network traffic over low bandwidth links," US Patent 7636767, Nov. 2009.

[7] S.Mccanne andM. Demmer, "Content-based segmentation scheme for data compression in storage and transmission including hierarchical segment representation," US Patent 6828925, Dec. 2004.

[8] R. Williams, "Method for partitioning a block of data into subblocks and for storing and communicating such subblocks," US Patent 5990810, Nov. 1999.

[9] A. Flint, "The next workplace revolution," Nov. 2012 [Online]. Available: http://m.theatlanticcities.com/jobs-and
economy/2012/11/
nextworkplace-revolution/3904/

[10] B. Aggarwal, A. Akella, A. Anand, A. Balachandran, P. Chitnis, C. Muthukrishnan, R. Ramjee, and G. Varghese, "EndRE: An end-system redundancy elimination service for enterprises," in Proc. NSDI, 2010, pp. 28–28.

[11] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet caches on routers: The implications of universal redundant traffic elimination," in Proc. SIGCOMM, 2008, pp. 219–230.

[12] A. Anand, V. Sekar, and A. Akella, "SmartRE: An architecture for coordinated network-wide redundancy elimination," in Proc. SIGCOMM, 2009, vol. 39, pp. 87–98.