Joint Multi-CDN and LT-Coding for Video Transport over HTTP*

Kai Tang, Chao Zhou, Xinggong Zhang, Zongming Guo†
Institution of Computer Science and Technology
Peking University, Beijing, 100080, China
Email: {tangkai, zhouchaoyf, zhangxg, guozongming}@pku.edu.cn

***Abstract*—Video transport over HTTP is becoming more and more popular. Many video service providers construct huge content distribution networks(CDN) to support HTTP streaming service, however, they seldom exploit the benefits of multiple servers to achieve higher bandwidth and reliability by parallel streaming. In this paper, we study the problem of jointing multi-CDN and LT-coding for video transport over HTTP. Using LT coding, a client could download the same video segment from multiple servers without considering data segmentation and server scheduling issue. Thus, we are able to treat all CDN servers as a virtual server with higher bandwidth and reliability. To reduce the ACK overhead, a stochastic model is designed to predict the amount of data to be sent from each server while guaranteeing the decoding probability. Compared with the existing schemes, the experimental results show that our proposed scheme obtains less overhead and fewer number of HTTP requests. Besides, we also achieve better video quality and better robustness to fluctuant bandwidth.**

I. Introduction

Recently, video transport over HTTP has drawn a lot of attention. Video transport upon HTTP/TCP makes it easy to traverse through firewalls and NAT. Moreover, the use of HTTP/TCP makes it possible to reuse existing network infrastructures such as geographically distributed CDN and widely deployed proxies. It has been widely accepted that CDN is a necessary part in building a modern media transport service. There are many CDN companies in the world such as Akamai, Level3, Limelight, and CloudFront etc. The multi-CDN strategy could provide service with higher reliability, lower cost and better user experience. In [1] V. Adhikari *et al.* have shown that Netflix has employed multi-CDN. They also show that, for a certain user, only one CDN server is used at a time in netflix. It further suggested that users could get better quality of video if concurrent transmission from multi-CDN is allowed. To the best of our knowledge, there has been no HTTP video transport system that implements concurrent downloading in multi-CDN.

There are two typical classes of concurrent downloading: the **Block-based** and **Fountain-based**. Rodriguez *et al.* proposed a Block-based parallel transport scheme in [2]. The authors proposed to fragment files into blocks of fixed size. Client requests one block at a time. However, there had been two issues. *I*) Each server will be idle between two consecutive block transmission. This is because each block transmission is initiated by a client request. It takes time for request to be received by the server. The idle time could be referred to as *interblock idle* time. If the segmentation size is too small, *interblock idle* would be too high to be ignored compared with transfer time. *II*) Servers may not finish transmission at the same time. We referred to the period of time, which starts from the moment when any of the servers stop transmission and ends at the moment when the whole file is received, as *termination idle* time. The bigger the block size is, the greater *termination idle* would be. During *termination idle*, bandwidth parallelism could not be fully utilized. These two issues contradict to each other, and the trade-off is hard to achieve. The work in [3] [4] follow the work in [2] and implement concurrent downloading in HTTP video transport scenario. G. Tian *et al.* [4] changes the granularity in [2] from block to the whole video segments. The block size, however, is too big and it increases the *termination idle* time. W. Pu *et al.* [3] splits the video file into blocks of various sizes based on approximate bandwidth of each server. Video segments may be fragmented into too many pieces, which makes the *interblock idle* issue even more serious.

Fountain-based concurrent downloading avoids the *interblock idle* and *termination idle* issues. Fountain codes are a class of erasure codes with the property that a potentially limitless sequence of encoding symbols can be generated from a given source file. The original file can be recovered from any subset of the encoding symbols with larger size than the original file. Thus, there is no need to segment the video file. The servers just keep sending the encoded packets until the client recovers the original file. M. Luby *et al.* among the first to employ Tornado codes, which also belong to the fountain codes, to speed up downloading by concurrently downloading Tornado encoded chunks in multiple servers in [5]. The scheme, however, barely considered the transmitting overhead, which will be explained later and it is referred as *ACK overhead* in this paper, incurred by the use of fountain codes. The servers would keep sending data until they receive ACK from the clients. During the period since the client sends the ACK till the servers receive it, the amount of transmitted data is referred as $ACKoverhead$. The $ACKoverhead$ grows with RTT and bandwidth, and it could not be simply ignored.

In this paper, we propose a joint multi-CDN and LT-coding scheme to resolve the above challenges. With LT codes [6], the *interblock idle* and *termination idle* issues are disappeared as file fragmentation is unnecessary. To reduce the ACK overhead, we periodically predict the mean and variance of bandwidth using sophisticated ARMA/GARCH. With the mean and variance, a stochastic model is designed to minimize the ACK overhead while controlling the decoding probability. We formulate it into an optimization problem. By some approximation, an effective near-optimal solution is

†Correspondent Author.

*This work was supported by National High-tech Technology R&D Program (863 Program) of China under Grant 2013AA013504, National Natural Science Foundation of China under contract No. 61201442 and National Key Technology R&D Program of China under Grant 2014BAK10B02.

978-1-4799-3432-4/14/$31.00 ©2014 IEEE

presented. Our contribution is threefold.

1) We are among the first to study the problem of video transport over HTTP by jointly considering multi-CDN and LT-coding. The proposed video transport scheme greatly exploits the benefits of parallel transmission.
2) By building the bandwidth prediction model and a stochastic model, we are able to minimize the ACK overhead while guaranteeing the decoding probability.
3) We further formulate the problem into an optimization problem. Under proper approximation, we solve the problem effectively with low complexity.

The rest of the paper is organized as follows. We will present the multi-CDN architecture and identify the problem in Section II. In Section III, we give the formulation of the problem. In Section IV, we give a solution to the problem with reasonable approximation. Experiments are presented to validate our analysis in Section V.

II. System Architecture

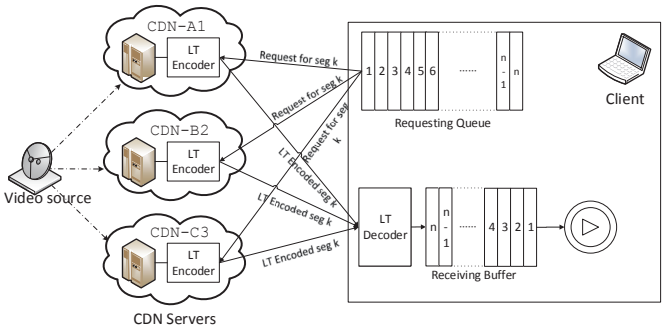

Fig. 1: Architecture of multi-CDN video transport system

We describe the multi-CDN DASH(aka Dynamic Adaptive Streaming over HTTP) architecture in Fig 1. Suppose there are three CDN servers. Video is transcoded into video files of various qualities/bitrates. These files are fragmented into video segments which then are replicated in all CDN servers. When a client sends HTTP request `GET` for a certain quality of video, it sends requests to all servers for the same segment simultaneously. Different from traditional DASH, servers then send the LT-encoded packets to clients. All received data is collected in the buffer at the client side waiting to be decoded.

To encode the video segments using LT codes, video segments have to be refragmented into subsegments called *Input Symbols*. To generate the encoded *Output Symbols*, the encoder has to randomly choose a degree d from Robust Soliton distribution. The encoder then chooses uniformly at random d distinct Input Symbols as neighbors and conducts XOR operation to compute the current Output Symbols. The neighbors ids are included in the header of the encoded packets. To avoid the duplicate encoded packets, different random seeds are assigned to different servers. We also implement an interface in the client to prevent the transmission of existing Output Symbols by checking the header of incoming packets. At the client side, the decoder progressively decodes Output symbols. As soon as original files are recovered, the client stops transmission, puts the decoded segment in the receiving buffer and moves into the next video segment. The client could safely download the video segment concurrently from all servers without worrying issues in block-based transmission. Although the TCP provides HTTP transmission with reliability, it could do nothing to cope with bandwidth deterioration or even link disconnection. These issues will cause assigned blocks to be downloaded overtime in block-based approaches. As a result, buffered media data would drain up quickly and video playout interruption would happen. The use of LT codes would increase robustness in case of any one of the link becomes congested. The reason is that the client retrieves data as if it retrieves from a single *virtual* server with higher bandwidth and deterioration in any of links would be compensated by other links timely.

However, there is a major drawback in the procedure above. The servers would only stop sending data when they received the ACK indicating that data has been successfully decoded. It takes time for ACK to be delivered from servers to client. During that time, the transmitted data is regarded as the *ACK overhead*. Fig 2 is a diagram to exhibit the $ACKoverhead$ in fluctuating network situation. Shaded part is the overhead. The main purpose of this paper is to calculate the transmission time of each server to minimize ACK overhead while controlling the decoding probability to be high enough.

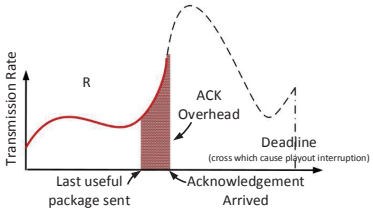

Fig. 2: Diagram of *ACK Overhead* using scheme in [5]

III. Problem Formulation

We denote BW_i as the bandwidth between the client and the i-th server. The transmission time of all servers is denoted as $\mathbf{T} = [t_1, t_2, \ldots, t_N]$, where t_i is the transmission time of server i, and N is the number of servers. We further denote R as overall data amount to be transmitted, thus:

$$R(\mathbf{T}) = \sum_{i=1}^{N} \int_0^{t_i} BW_i\,(t)\,dt \tag{1}$$

We define C as the size of a video segment, then the *ACK overhead* can be defined as $(E[R(\mathbf{T})]-C)/C$, where $E[R(\mathbf{T})]$ is the mean value of $R(\mathbf{T})$. Given the transmission time vector $\mathbf{T}$ and data amount r, the successfully decoding probability is denoted as $P(\mathbf{T}; r)$. $P(\mathbf{T}; r)$ is the joint distribution of $\mathbf{T}$ and r. We define $P(\mathbf{T})$ as the marginal probability density functions of $P(\mathbf{T}; r)$ on $\mathbf{T}$ and ε as excepted successfully decoding probability, then the problem can be formulated as follows:

$$\begin{aligned} \mathbf{T}^* &= \arg\min_{\{\mathbf{T}\}} (E[R(\mathbf{T})] - C)/C \\ \text{s.t. } & P(\mathbf{T}) \geq \varepsilon \end{aligned} \tag{2}$$

The problem aims to find the optimal $\mathbf{T}^*$ to minimize the ACK overhead while guaranteeing the decoding probability to be no less than ε.

To solve the constraint optimization problem, we need to calculate $E[R(\mathbf{T})]$ and $P(\mathbf{T})$. We begin with the analysis of $R(\mathbf{T})$. The work in [7] has pointed out that $BW_i(t)$ can be approximated as a strict-sense stationary gaussian process. As

a result, for any given $ts_i \in [0, t_i]$, $BW_i(ts_i)$ is a Gaussian random variable. i.e., $BW_i(ts_i) \sim \mathcal{N}(\mu_i, \sigma_i)$. According to math theorem, the integral of Gaussian variable is a Gaussian variable; the sum of several Gaussian variables is a Gaussian variable as well. Considering that the data sent by i-th server is independent from all the other servers, R satisfies $R(\mathbf{T}) \sim \mathcal{N}(\mu(\mathbf{T}), \sigma(\mathbf{T}))$, with mean $\mu(\mathbf{T}) = E[R(\mathbf{T})] = \sum_{i=1}^{N} \mu_i t_i$ and variance $\sigma^2(\mathbf{T}) = \sum_{i=1}^{N} \sigma_i^2 t_i^2$.

Then, we come to derive $P(\mathbf{T})$. As $R(\mathbf{T})$ is a Gaussian variable, the probability of transmitting r (bits) of data is:

$$P_{Gaussian}(R(\mathbf{T}) = r) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(r - \mu(\mathbf{T}))^2}{2\sigma^2(\mathbf{T})}\right) \quad (3)$$

According to [6], given r (bits) of LT encoded data, the decoding probability $P_{LT}(r)$ is:

$$P_{LT}(r) = \max\left(1 - C\exp\left(-\sqrt{\frac{r - C}{\alpha\sqrt{C}}}\right), 0\right) \quad (4)$$

where α is a constant. The decoding probability $P(\mathbf{T}; r)$ is affected by two factors: the first is probability of transmitting r (bits), the second is the LT decoding probability for the given data size r. Thus, we have $P(\mathbf{T}; r) = P_{Gaussian}(R(\mathbf{T}) = r)P_{LT}(r)$. At last, since $P(\mathbf{T})$ is the marginal distribution of $P(\mathbf{T}; r)$ on $\mathbf{T}$, we have:

$$P(\mathbf{T}) = \int_{-\infty}^{+\infty} P_{Gaussian}(R(\mathbf{T}) = r) P_{LT}(r)\, dr \quad (5)$$

IV. Solution

LT codes have good performance. In practice, if the number of *Input Symbols* K satisfies $1000 \le K \le 8192$, only $K + 2$ *Output Symbols* are needed to recover the original file [8]. So it is quite reasonable to use a step-function as approximation of $P_{LT}(r)$. If we denote the amount of data that is needed to successfully decode as M, we will have:

$$P_{LT}(r) \approx \begin{cases} 0 \text{ if } r < M \\ 1 \text{ if } r \ge M \end{cases} \quad (6)$$

Together with (3) (5) (6), (2) could be rewritten as:

$$\begin{aligned} \mathbf{T}^* &= \arg\min_{\{\mathbf{T}\}} \mu(\mathbf{T}) \\ \text{s.t.} &\int_{M}^{+\infty} P_{Gaussian}(R(\mathbf{T}) = r)\, dr \ge \varepsilon \end{aligned} \quad (7)$$

We rewrite the constraints formula in (7) as follows:

$$\beta\sigma(\mathbf{T}) + \mu(\mathbf{T}) \ge M \quad (8)$$

where $\beta = \Phi^{-1}(1 - \varepsilon)$ and Φ is the CDF of standard normal distribution. Using method of Lagrange multiplier, the problem can be rewritten as an unconstraint optimization problem:

$$\mathbf{T}^* = \arg\min_{\{\mathbf{T}\}} \mu(\mathbf{T}) + \lambda(\beta\sigma(\mathbf{T}) + \mu(\mathbf{T}) - M) \quad (9)$$

The solution is as follows:

$$t_i = \frac{M\beta\sqrt{\sum_{i=1}^{N} \frac{\mu_i}{\sigma_i^2}} + M\sum_{i=1}^{N} \frac{\mu_i^{3/2}}{\sigma_i^2}}{-\beta^2 \sum_{i=1}^{N} \frac{\mu_i}{\sigma_i^2} + \left(\sum_{i=1}^{N} \frac{\mu_i^{3/2}}{\sigma_i^2}\right)^2} \frac{\sqrt{\mu_i}}{\sigma_i^2} \quad (10)$$

V. Experiment

In this section, we designed a group of experiments to prove the validity and performance of the Proposed scheme. We compare with the Proposed scheme with *ACK with LT* [5] scheme, *Parallel* [4] scheme and *CMSS* [3] scheme.

A. Interblock Idle and Termination Idle

The motivation of using fountain coding based concurrent downloading scheme is the problem of *interblock idle* and *termination idle* caused by the block based concurrent downloading scheme. To study the trade-off of these two idles, we have done some experiments. We have three servers. Their bandwidth are 800, 400, 280 kbps and RTT are 10, 70, 120 ms respectively. The result is shown in Fig. 3. From the figure, we find that the smallest total idle (i.e. the sum of interblock idle and termination idle) is higher than 500ms. The idle time is fluctuated heavily under various block numbers. Generally, the optimal block number (corresponding to the minimal total idle) is also changed with many factors, such as the server number and time-varying bandwidth. For space limit, we omit the experiments under different server number and bandwidth conditions. Thus, how to set the block number is no easy job. Instead of setting the best block number which is necessary for the block based concurrent downloading scheme, we adopt LT-coding to prevent such problems. In the following experiments, we demonstrate the effectiveness of our proposed video transport scheme and algorithm.

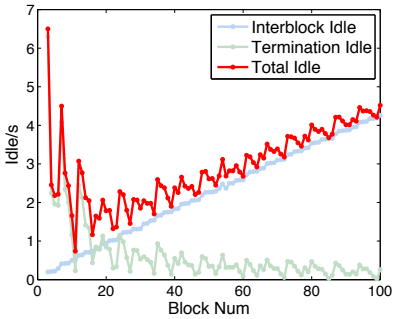

Fig. 3: Total Idle time with given block number

B. ACK Overhead

We aim to minimize the ACK overhead under a given decoding probability. The definition of ACK overhead indicates that it only exists in Fountain based approaches: *ACK with LT* and the Proposed.

The parameters of Proposed approach are as follows. According to the three sigma principle, we expect the minimum decoding probability ε = 99.7%. With (8), β = 2.0985. Considering the LT codes performance mentioned in Section IV, α could reasonably be set as 10^{-4}. The experiment is set up in LAN with one client and three servers. $netem$ is deployed on each server to manipulate the network condition. In Fig. 2, it is obvious that the area of shaded part(ACK overhead) is affected by the width(RTT) and the height(bandwidth). By adjusting the RTT and the bandwidth, we get the amount of ACK overhead in Fig. 4. Overhead in $Propose$ algorithm is merely affected by RTT and bandwidth while Overhead in *ACK with LT* grows with RTT and bandwidth. However, it has a certain upper bound as $netem$ maintains a drop tail queue. When RTT or bandwidth becomes too large, the queue

tends to overflow which leads to packet loss and HTTP/TCP throughput declines. In most cases, Our $Proposed$ algorithm has a better performance. It indicates that $Proposed$ algorithm is better in higher RTT and bandwidth situation.

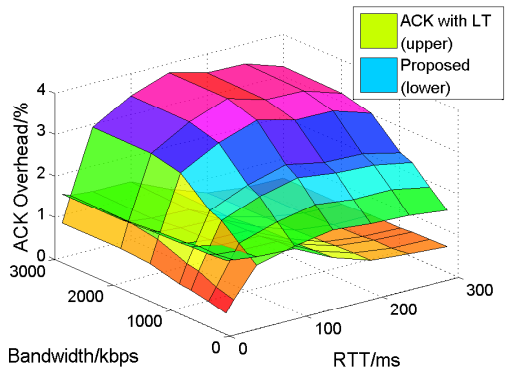

Fig. 4: ACK Overhead v.s. different network parameter

C. Real world experiment

We set up the real world experiment using $Planetlab$. The client is deployed at PKU with LT decoder, while three servers are deployed at HUST, SYSU and BUPT with LT encoder. Video files whose bitrate range from 300kbps to 2000kbps are fragmented into $4s$ length of video segments. We set $L = 128bytes$, $M = 1.002C$. Therefore the number of $InputSymbol$ and $OutputSymbol$ in a video segment meets the requirement we discussed in section IV. We conducted a 6-hour-long experiment by video loop playback.

1) ACK Overhead: We show the average ACK overhead within every 15 minutes in Fig. 5. Although RTT and bandwidth varied with time among three different sites, the Proposed approach always achieves at least 1/3 less ACK overhead than the *ACK with LT*.

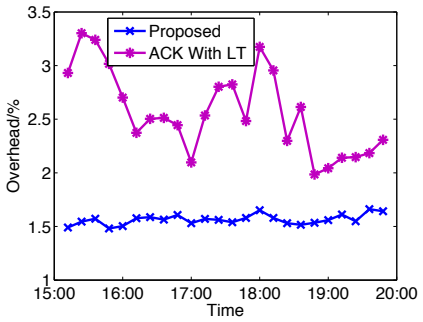

Fig. 5: Overhead in real world experiment

2) Overall quality: We set up the experiment by measuring the video quality within DASH client. Due to great fluctuation of network from the three sites, rate adaptation algorithm in OSMF is of bad performance. In this case, we employed the buffer-based algorithm proposed in our previous works [9] as a replacement. In this algorithm, q_{min} indicates the expected minimum buffer. When the buffer is less than q_{min}, playout interruption is more likely to happen. Results are shown in Table I. As video was encoded uniformly before transmission, the video bitrate reflects the video quality directly. DASH achieves different effective throughput in different parallel transmission framework within the same network condition. As a result, they have different bitrate choice and different video quality. Moreover, the buffer min value and time below q_{min} reflects the robustness to bandwidth fluctuation. From Table I, we could conclude that the Proposed scheme achieves higher bitrate than other schemes. Also, the Proposed has higher buffer to cope with any potential network deterioration. The minimal buffer using *Parallel*(proposed in [4]) is the smallest. That is because in *Parallel* a segment is downloaded by a single CDN server rather than by all simultaneously, and it took longer time to update buffer. During that time, buffer is more likely to be drained up. In *CMSS*(proposed in [3]), client sends more HTTP requests, which leads to significant interblock idle. That would be detrimental to both transmission throughput and video quality. In *ACK with LT*, the use of LT codes makes the transmission more robust to bandwidth fluctuation. However, ACK overhead in *ACK with LT* undermines the video quality.

TABLE I: Video Quality Overview

Algorithm Type	Average Bitrate	Buffer min	Times when buffer is less than q_{min}
Proposed	1579.2 kbps	4.5s	6
ACK with LT [5]	1509.0 kbps	3.6s	12
Parallel [4]	1416.4 kbps	0.9s	25
CMSS [3]	1483.2 kbps	1.1s	21

VI. Conclusion

In this paper, we studied multi-CDN video transport over HTTP. We identified the challenge faced in multi-CDN HTTP video transport. To resolve the issues above, we present a joint multi-CDN and LT-coding transport scheme. Experimental results show that the proposed method could reach less ACK overhead, better video quality and better robustness to network fluctuation.

References

[1] V. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z.-L. Zhang, Unreeling Netflix: Understanding and improving multi-CDN movie delivery, in INFOCOM, 2012 Proceedings IEEE, march 2012, pp. 1620 1628.

[2] P. Rodriguez and E. W. Biersack, Dynamic parallel access to replicated content in the internet, IEEE/ACM Trans. Netw., vol. 10, no. 4, pp. 455465, Aug. 2002.

[3] W. Pu, Z. Zou, and C. W. Chen, Dynamic adaptive streaming over HTTP from multiple content distribution servers, in Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, dec. 2011, pp. 1 5.

[4] G. Tian and Y. Liu, Towards agile and smooth video adaptation in dynamic HTTP streaming, in Proceedings of the 8th international conference on Emerging networking experiments and technologies, ser. CoNEXT 12. New York, NY, USA: ACM, 2012, pp. 109120.

[5] J. Byers, M. Luby, and M. Mitzenmacher, Accessing multiple mirror sites in parallel: using tornado codes to speed up downloads, in INFOCOM 99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, vol. 1, Mar, pp. 275 283 vol.1.

[6] M. Luby, LT codes, in Proceedings of the 43rd Symposium on Foundations of Computer Science, ser. FOCS 02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 271.

[7] M. Wang, X. Meng, and L. Zhang, Consolidating virtual machines with dynamic bandwidth demand in data centers, in INFOCOM, 2011 Proceedings IEEE, april 2011, pp. 71 75.

[8] J.-P. Wagner, J. Chakareski, and P. Frossard, Streaming of scalable video from multiple servers using rateless codes, in Multimedia and Expo, 2006 IEEE International Conference on, 2006, pp. 15011504.

[9] C. Zhou, X. Zhang, L. Huo, and Z. Guo, A control-theoretic approach to rate adaptation for dynamic http streaming, in Visual Communications and Image Processing (VCIP), 2012 IEEE, 2012, pp. 16.