Enabling multi-source coded downloads

Patrik J. Braun*†, Derya Malak*, Muriel Médard*
*Research Laboratory of Electronics (RLE)
Massachusetts Institute of Technology
Cambridge, MA 02139 USA
{pbraun, deryam, medard}@mit.edu

Péter Ekler†
†Department of Automation and Applied Informatics
Budapest University of Technology and Economics
Budapest, 1111 Hungary
{patrik.braun, peter.ekler}@aut.bme.hu

***Abstract*—In this paper, we introduce two multi-source download protocols for loosely orchestrated edge cloud scenarios involving mobile devices. We focus on services with high bandwidth and low delay requirements, like video streaming. We propose MUlti-source Transmission Protocol (MUTP) for uncoded multi-source data delivery and extend it with network coding capabilities to create coded MUTP. We investigate their throughput using a custom designed testbed. We present measurement results collected over eight months that include more than 630 GBs of video download. We show that even when downloading from only two sources, our protocols can match the heavily optimized HTTP protocol. Furthermore, by increasing the number of sources to four or higher, MUTP protocols can outperform the HTTP approach, reaching an up to three-fold goodput increase.**

***Keywords*-Network coding, multi-source download, WebRTC, JavaScript, video download, edge cloud**

I. Introduction

Mobile data traffic increases continuously. According to Ericcson [1], in the third quarter of 2018, the grow-rate of mobile-based Internet traffic reached 79% on a year-on-year basis. In 2018, 60% of this data was video streaming, a figure that is expected to reach 75% by 2024.

The majority of services that are used over the mobile network are mostly based on a traditional client-server setup, through protocols like HyperText Transfer Protocol (HTTP) [2]. Furthermore, the servers are usually placed in the core of the network, far away from the client. When a user travels with high speed, its mobile network can have bandwidth fluctuations, because of losses or handovers [3]. When the user watches an online video, these fluctuations can lead to a reduction in video quality or even to stream interruptions [4].

A possible solution to this issue is to apply an edge cloud system [5]. An edge cloud system brings the content closer to the clients by caching it at the base stations. Furthermore, if clients can connect to multiple cell towers, they can download from all neighboring edge clouds simultaneously. To achieve multi-source download, conventional protocols like Transmission Control Protocol (TCP) are not sufficient. Furthermore, since in a mobile scenario the network configuration is continuously changing, it is challenging or not feasible to synchronize all edge servers so that they do not send the same packets to the client.

Bruneau et al. have proposed MS-streaming, a multiple-source streaming solution that splits video into multiple independent sub-streams and offers methods for bit-rate adaptation and server-switching [6]. Compared to optimal Dynamic Adaptive Streaming over HTTP (DASH) systems, MS-streaming can achieve up to a 74% mean bit-rate gain.

We have shown that Random Linear Network Coding (RLNC) [7] may also be used to improve the throughput of a multi-source network [8]. RLNC creates linear combinations of the original packets using random coefficients. The main advantage of RLNC is that it is a rateless code with recoding ability. Sørensen et al. have presented Network Coded Filesystem Shim (NCFSS), a filesystem-level solution for multipath, and multi-source download with RLNC [9]. They have provided a proof-of-concept implementation of their proposed solution and have shown that their solution improves access and download time by two to five-fold compared to downloading from a single source. RLNC has also been applied in transportation protocols by Kim at al [10]. They have presented a network coded implementation of TCP (CTCP) that was be able to improve throughput on a single lossy link.

In this paper, we propose two protocols for multi-source download. The first protocol is the MUlti-source Transmission Protocol (MUTP) that transfers uncoded packets from several servers to one client. The second protocol is the Coded MUTP that is based on MUTP and creates RLNC-encoded packets. Our solution differs from previous works in three main aspects. 1) We focus on unreliable scenarios where the servers cannot cooperate. 2) We propose protocols that can be applied in the transportation layer or over User Datagram Protocol (UDP) in the application layer. 3) We use RLNC as part of the protocol. To examine the performance of our protocols, we have designed a testbed that intercepts YouTube video downloads. We download the intercepted video through several servers, using one of our protocols or a naive parallel HTTP-based approach that starts multiple HTTP downloads for the same data and chooses the fastest among them.

II. Problem definition

In this paper we focus on a scenario that has M servers and one client. All M servers contain the same L original data packets that the client would like to download. The client connects to $N < M$ nodes and starts to download the original data. Connections between the client and the servers are unreliable in both directions.

We measure the client progress with Degrees of Freedom (DoF). DoF increases by one if the client receives a new, useful packet. The client sends cumulative feedback that contains information about all of its received packets.

The server nodes do not have information about each other, i.e. they do not know how many nodes the client is connected to, and the bandwidth of the nodes is also not available for packet scheduling. For packet scheduling, a server must rely on two information: 1) the previously sent packets, 2) the information from the feedback. Note that the feedback from the client to the server delayed. Thus, the servers never have full information about the system.

Servers maintain a window of size $w \leq L$ to limit the memory needed for transmission. In this multi-source scenario, we cannot use the conventional sliding window, since in that case, all servers would send the same packet. Therefore, in this paper we consider a *strict moving* window setup. A server may schedule any packets from its window. A packet can be removed from the window if the client successfully received and acknowledged its reception. To have a constraint on the packet delay, we define $\mathcal{W}(t)$, the set of packets in the window at time t the following way:

$$\begin{aligned} \mathcal{W}(t) &= \{i \in \mathcal{L} \,|\, w_{\text{min}} \leq i < w_{\text{min}} + w\} \\ \mathcal{L} &= \{0, \ldots, L\} \\ \mathcal{L}_{\text{acked}}(t) &= \{i \in \mathcal{L} \,|\, \text{ pkt. } i \text{ was acknowledged by time } t\} \\ w_{\text{min}} &= \min(\mathcal{L} - \mathcal{L}_{\text{acked}}(t)), \end{aligned} \tag{1}$$

where $\mathcal{L}$ is the set of original packets, $\mathcal{L}_{\text{acked}}(t)$ is the set of all received and acknowledged packets by time t and w_{min} is the not-yet-acknowledged packet with the lowest index.

In this paper, we focus on finding the achievable maximum goodput (useful throughput) of a system that fulfills the model that is presented in this section.

III. System description

We propose MUlti-source Transmission Protocol (MUTP) for uncoded data transfer from multiple sources. Based on MUTP, we propose Coded MUTP with network coding for encoded multi-source data transfer.

Figure 1 shows our system setup, consisting of an *Origo* server and several proxy servers and clients. The responsibility of the *Origo* server is to manage the proxy servers and serve as the entry point to the system.

We have designed a browser extension that intercepts YouTube video requests and downloads them over N proxies. It can download the content in three different ways:

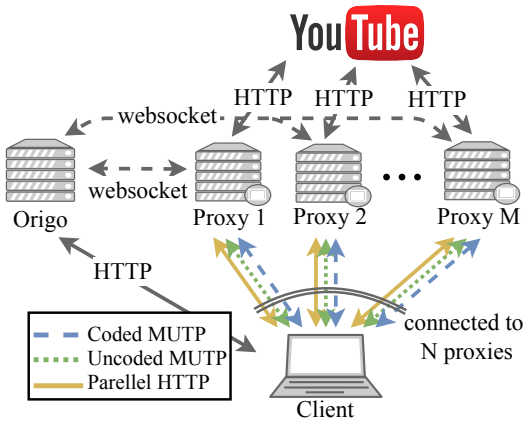

Figure 1. System overview

over a naive *Parallel HTTP*, or using Uncoded MUTP or Coded MUTP protocols. *Parallel HTTP* sends the same HTTP request over TCP to all connected proxies and uses the fastest response as the result of the download. Uncoded MUTP and Coded MUTP create an unreliable data-channel over Web Real-Time Communication (WebRTC) to connect to the proxies.

A. MUlti-source Transmission Protocol (MUTP)

Uncoded MUTP proxies maintains two lists: *in-window* packets, and *in-transit* packets. In-transit packets are those that have been sent, but no feedback has yet been received.

The client maintains a list of *received* packets that increased its DoF (DoF increases at the client if it receives a packet that was not present in its *received* list). The client sends cumulative feedback based on its *received* list.

Based on the obtained *received* list from the client and the *in-transit* list, each server creates a *sendable* list of packets. The servers use this to choose a packet for transmission. Since the proxies cannot communicate with each other, optimal scheduling is not possible. Therefore we implement a random scheduler, that chooses a packet uniformly at random from the *sendable* list without replacement.

B. Coded MUTP

Coded MUTP uses a similar approach as Uncoded MUTP to transmit packets. Instead of working on a packet level, Coded MUTP first organizes the original L packets into g sized groups, called *generations*. With the Coded MUTP protocol the *in-window*, *in-transit*, *received* and *sendable* lists contain generations instead of packets.

Coded MUTP uses a *rarest generation first* approach for packet scheduling. It sends a packet from a *generation* that has the least received DoF and in transiting packets. We chose this method of scheduling a *generation* for sending because it has already shown potential to improve throughput in RLNC enhanced distributed systems [8].

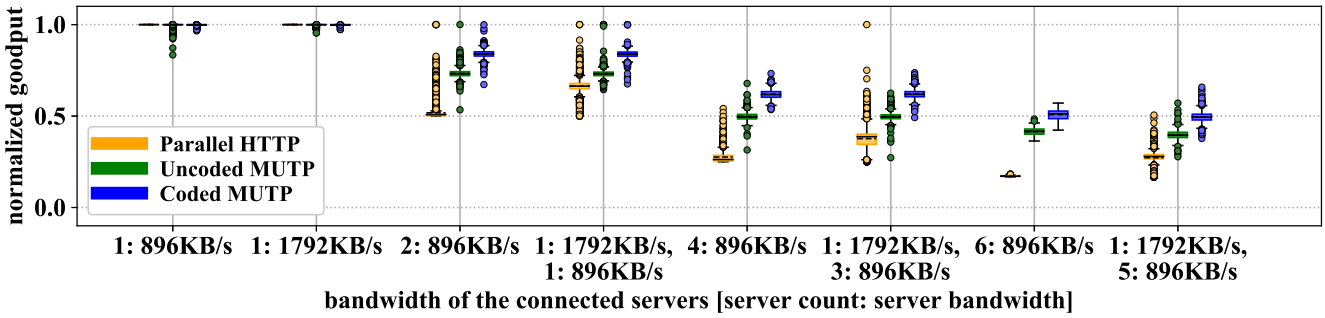

Figure 2. Grouped boxplot representation of downloading 1-2MB data from $N \in \{1, 2, 4, 6\}$ servers with 896 KB/s and 1,792 KB/s upload bandwidth with window size $w = 240$ and generation size $g = 24$.

IV. Results and Discussion

For our measurements, we used Amazon Web Services (AWS)[1] to host one *Origo* server and 18 proxy servers in the USA and Europe. We run our measurements for 8 months and collected 820,000 log records from more than 630 GBs of YouTube video watch. To better represent a multi-source scenario, where the bottleneck is the server, we limited 14 of our proxies to 896 KB/s and four proxies to 1,792 KB/s.

In this section we compare the *normalized goodput* of different setups. We define *normalized goodput* as the quotient of goodput (useful throughput) and throughout. We use this to compare amount of received packets that increase the DoF at the client to all received packets. We also compared our measurement results to our previous work [8], where we construct a model to analyze multi-source networks. Our analysis and our measurements show similar trends.

Figure 2 shows combined results of downloading 1-2MB sized chunks with different upload bandwidth. Results show that in case of a single connection, all three approaches perform the same way, while increasing the number of sources, our MUTP protocols have better performance compared to the HTTP approach. Uncoded MUTP has an up to two-fold performance increase compared to Parallel HTTP. Furthermore, Coded MUTP has an up to three-fold performance increase compared to Parallel HTTP and a 25% performance increase compared to Uncoded MUTP.

V. Conclusion

In this paper, we have proposed two multi-source download protocols for loosely orchestrated edge cloud scenarios: Uncoded MUTP and Coded MUTP. To test their performance, we created a testbed and carried out an extensive measurement campaign. Throughout our measurements, we achieved two- and three-fold normalized goodput increase with Uncoded MUTP and Coded MUTP, respectively, compared to the naive Parallel HTTP approach.

As future work, we plan to investigate different packet scheduling methods for both Uncoded and Coded MUTP protocols and further optimize our system.

[1]Amazon Web Services: https://aws.amazon.com/

References

[1] (2018, Nov.) Ericsson mobility report. Ericsson. [Online]. Available: https://www.ericsson.com/assets/local/mobility-report/documents/2018/ericsson-mobility-report-november-2018.pdf

[2] (2018) Global Internet Phenomena. Sandive. [Online]. Available: https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf

[3] R. Ahmad, E. A. Sundararajan, N. E. Othman, and M. Ismail, "Handover in LTE-advanced wireless networks: state of art and survey of decision algorithm," *Telecommunication Systems*, vol. 66, no. 3, pp. 533–558, Nov 2017.

[4] N. Wehner, S. Wassermann, P. Casas, M. Seufert, and F. Wamser, "Beauty is in the eye of the smartphone holder a data driven analysis of youtube mobile qoe," in *2018 14th International Conference on Network and Service Management (CNSM)*, Nov 2018, pp. 343–347.

[5] H. Chang, A. Hari, S. Mukherjee, and T. V. Lakshman, "Bringing the cloud to the edge," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2014, pp. 346–351.

[6] J. Bruneau-Queyreix, M. Lacaud, D. Négru, J. M. Batalla, and E. Borcoci, "Adding a new dimension to http adaptive streaming through multiple-source capabilities," *IEEE MultiMedia*, vol. 25, no. 3, pp. 65–78, July 2018.

[7] R. Ahlswede, N. Cai, S. Y. Li, and R. W. Yeung, "Network Information Flow," *IEEE Trans. Inf. Theor.*, vol. 46, no. 4, pp. 1204–1216, Sep. 2006.

[8] P. J. Braun, D. Malak, M. Médard, and M. Ekler, "Multi-Source Coded Downloads," in *2019 IEEE International Conference on Communications (ICC)*, May 2019, pp. 1–7.

[9] C. W. Sørensen, D. E. Lucani, and M. Médard, "On network coded filesystem shim: Over-the-top multipath multi-source made easy," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–7.

[10] M. Kim, J. Cloud, A. ParandehGheibi, L. Urbina, K. Fouli, D. J. Leith, and M. Médard, "Network coded TCP (CTCP)," *CoRR*, vol. abs/1212.2291, 2012.