# Effect of Chunk Size on Router Caches for Video Based Workloads in Information-Centric Networks

Harshad Shirwadkar, Da-Yoon Chung
harshad@cmu.edu, dayoonc@andrew.cmu.edu

Carnegie Mellon University
Pittsburgh, PA 15213.

## ABSTRACT

Caching in routers has been widely explored, with some models of networking such as Content Centric Networking already providing implementations of such functionality. While the effects of many different parameters of caching has been quantified and studied, one aspect has largely been ignored: the chunk size of the content being cached. Our work attempts to explore this single aspect of caching and its effect on previously established metrics on both user experience and server performance.

### Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Miscellaneous; C.4 [**Performance of Systems**]: Measurement Techniques

### General Terms

Measurement, Performance, Experimentation

### Keywords

ICN, CCN, NDN, Video

## 1. INTRODUCTION

The idea of introducing caching to network routers is not a novel one. As the type of network traffic shifted from mostly host-to-host connections to content based traffic, it became clear that caching could provide savings as popular content was requested frequently by many different clients. Some recent work by Sun et al focuses on streaming video as a major source of traffic in modern networks [2], and as the size of these videos and available user bandwidth both increase today, we will see more and more redundant data make itâĂŹs way across the internet. By introducing network caches, it has been shown that a significant reduction in network traffic is achievable. However, while these publications take into account many different aspects of router caches such as cache placement and size, they sidestep a potentially important parameter: the size of the individual chunks themselves.

In this project, we explore the effects of chunk size on a number of previously established metrics for both user quality of experience and server performance. More specifically, we measure DUBBLE CHECK join-time, buffering ratio, average latency of requests, and server load reduction. Using the ndnSIM module for the NS-3 network simulator, we designed a client and server application to quantify these metrics. We run our experiments on a small, straightforward topology which while not accurately representative of the real world, demonstrates the type of results we would expect in practice.

## 2. BACKGROUND

In order to better introduce the question we are trying to answer, we will talk about our work in relation to two previously published papers. The first is called "Developing a Predictive Model of Quality of Experience for Internet Video" [1]. This paper redefines the metrics used to predict the quality of experience that a user experiences depending on a number of measurements of video streaming. Although not directly related to caching, we used this work to determine which metrics we would try to measure to make sure our results translated to effects on the user.

The second paper is called "Trace-Driven Analysis of ICN Caching Algorithms on Video-on-Demand Workloads " [3]. This work is actually where we got the idea for our own project. We chose to explore one of the few aspects of caching that the paper did not take into account.

## 3. APPROACH

We discussed two different possibilities for running our experiments. The first option we considered was using real hardware to run the client, router, and server nodes for our experiments. The advantage of this approach is that it would be as close to a real, albeit very small, network in the real world. The obvious downside to this approach is the difficulty in setting up experiments, especially in implementing caching in commercial routers if we wanted to go all the way with this approach. There are test beds like emulab which we had previous experience using that allow experiments to easily define physical nodes and links in a network, but these do not provide any customization capabilities for the routers involved in the links. We quickly decided not to pursue this route.

The second approach, which was ultimately better suited to our purposes, is a simulation and trace-driven approach, similar to [2]. Using simulation software allowed us to run many different configurations of experiments on individual machines, and to predefine workloads that we could keep constant across runs. Because we did not have to worry about acquiring and transferring real files in our experiments, we could instead just simulate their effects on the network. This approach was easier than the former, while still being able to produce valid results.

### 3.1 ndnSIM

ndnSIM is a NS-3 based simulator for Named Data Networking (NDN), a networking model very similar to Content Centric Networking (CCN). It allows users to set up custom topologies, and define the behavior of individual nodes in the network using custom applications. In terms of our intended use of simulating the transfer of chunks between servers and clients in a network, ndnSIM is perfect because it already abstracts these content chunks into data and interests. Another big draw of ndnSIM is that it already has an implementation of a cache in the form of a Content Store structure. These Content Store objects are more general than the router caches we want to test, because they can be placed on any nodes in the network (not just routers). They are minimally configurable, allowing users to set the eviction policy and the max size in terms of packets.

## 4. EXPERIMENTATION

### 4.1 Topology

The most simple example of a topology involving a client, router, and server would be a three node setup with links between the client and router and router and server. However, we would only expect to router cache hits if the same video was requested multiple times, which while not unlikely due to our zipf-distribution of video popularity, is a poor representation of reality, and is also very boring.

We therefore ran our experiments on a more complex topology involving five clients linked to a single router, which in turn is linked to a server.

The bandwidth of the links between the nodes is 10 mbps with 10 ms of latency. With our workload, we do not see enough traffic for congestion to affect our results. (RIGHT?????????)

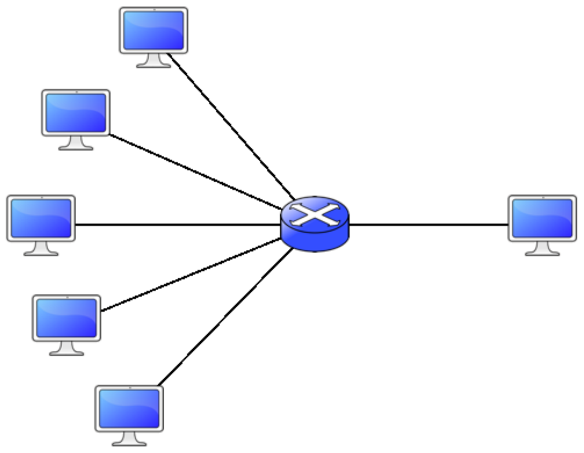
Figure 1: Topology for the experiments

### 4.2 Client Application

In order to run our experiments, we had to define a custom client application. Our client application is simple. First, we pre-generated trace files for each client, indicating which videos the each client would request, in what order, and to what ending duration. As we previously mentioned, the videos are chosen from a zipf distribution of popularity. Because we use these traces, each experiment we run for different chunk sizes requests the same videos. This ensures that our results will not have any discrepancies due to random differences in viewing, but on the downside, our results might not be representative of the average behavior. We made this compromise in the interest of time, but ideally, we could have run many randomized experiments multiple times and found an average. In order to better replicate real-world video streaming applications, we define a buffer which is 0.3 times the length of the video. Instead of beginning the simulating watching of the video once the first chunk is received, which we had done originally, we begin the watching of the video once the buffer is full for the first time. Once the video has begun to be watched, we track the current play time of the video and the amount of data fetched separately. The buffer slides along with the current play time such that the client tries to always have the next 30% of the chunks of the video pre loaded. The client would be considered to be in a buffering state if the current play time catches up to the last chunk fetched in the buffer.

Once a client reads in a video to watch from its corresponding trace file, it begins requesting chunks of the video to the server by sending interests. The clients do not specify the size of the chunk in the requests; this is defined in the server application. Instead, the client determines the size of the chunk by checking the size of the data received. Once data is received, the client updates the buffer, checks if its buffer is full, and requests another chunk if it is not. If it is full, the client waits until another chunkâĂŹs worth of play time has passed, and then requests another chunk.

### 4.3 Server Application

The server waits for an interest to be received, and when it does, it returns of chunk to the client, which is a data buffer of the chunk size being measured for that experiment. The content of the chunk is currently not checked by either the server or the client, so there is no content verification in place. This is really the extent of what the server does.

### 4.4 Router

The router is not something we programmed, but we did configure it. As was mentioned before, the size of the router is defined in terms of the number of packets, rather than an absolute size. Therefore, we set the size to be equal to 7 MB, a value small enough that on average, an entire video could not exist in the cache at one time.

We also used the router to observe the pending interests table, a data structure which tracks which interests were received but not yet served by the router. This is unique to these content centric network models.

### 4.5 MTU

ndnSIM has a default MTU value of 1500 bytes, which is in line with the maximum packet size allowed by ethernet. Unfortunately, this means that if we experiment with chunks larger than this value, ndnSIM automatically breaks them up into multiple packets, which are then individually cached by the content store. If we then want to evict a logical chunk from the cache, we have to modify ndnSIM to ensure that all of the chunks corresponding to the chunk is evicted.

In order to circumvent this complication, we configured ndnSIM to have an MTU much larger than any of the chunk sizes we experimented with. Although this does not model the real world perfectly, we felt this was an acceptable compromise because it would allow us to trust the cache to behave as we expected.

### 4.6 Batching

In some initial runs of our experiment, we did not implement any batching of requests by the client. Individual chunks were requested in order, one after the other. Unsurprisingly, certain metrics we chose like start-up time suffered miserably as RTT for the packets (40 ms, with our particular topology) consumed the vast majority of the time before the initial buffer filled and the video started being watched.

For later experiments, we implemented batching on the client. Instead of individual interests, the client computes which chunks in the buffer are not yet fetched, and sends individual interests for those chunks all at once. Once it receives all of the data for this batch of requests, it repeats the process. The effects of this are quantified in the results section.

## 5. EVALUATION

### 5.1 Metrics Evaluated

Join Time: This metric was used in the QoE paper as a measure of user experience. It refers to the time it takes for the video to start playing after it has been requested. Because we also used a video-based workload, we believed this was still a very relevant metric.

Buffering Ratio: This metric was also used in the QoE paper. This is the ratio between the time that the user spends waiting for the video to buffer and the total time to finish watching the video. We felt this would also be an interesting metric, because we expected it to vary inversely with chunk size, as it would take longer for increments of the video to be buffered. However, it is possible that our buffer size is large enough that we prefetch enough chunks that this is not the case.

Pending Interest Table Size: This structure is import to us because we expected the number of pending interests to vary inversely with the chunk size. With smaller chunks, chunk request are made more frequently because the buffer empties in smaller increments, so more requests should pile up on the router node.

Server Load Reduction: This is fraction of requests that was served by the cache rather than the server. This is a standard way to measure the benefits to the server due to caching.

### 5.2 Chunk Size

We ran experiments for the following chunk sizes (in bytes): 128, 256, 512, 1024, 1940, 4096, 6144, 8192. 1940 is an odd value because it is not 2048 as would be expected given the other values. We had to use value because for reason we could not identify, the data could not be received by the clients for any chunk sizes between roughly 2000 to 3000. The server would receive the interests, but the experiments would stop at that point. There were no other error messages or information we could use to debug, so in the interest of time, we used a value close to 2048 just to have a datapoint in the range of values.

### 5.3 Experiments without batching

Not batching has the most extreme effect on the join time metric. For 128 byte chunks, we see the join time is about 30 seconds, which is longer than the length of most videos themselves. To deduce why this is the case, we can do some simple calculations. Given we defined a buffering window of 10% of the total video size, for a video of size 10 MB, the client downloads 1 MB of the video before starting the video. This is equal to $1 * 1024 * 1024/128 = 8192$ individual requests. The $RTT$ of each request is $4 * 10 = 40$ ms because each link has 10 ms latency. Therefore, just the time the chunks spend in transit is equal to $8192 * 40 = 327680$ milliseconds, or 328 seconds.

With larger chunk sizes, we see the linear decrease in join time as we would expect.

## 6. RELATED WORK

## 7. FUTURE WORK

In order to obtain some early results and verify that our experimentation platform worked, we had to make some simplifications to our experiments. If we had more time, we would have liked to explore the following improvements to our model.

### 7.1 Topologies

We used a very rudimentary topology in the experiments evaluated in this project. We would have ideally liked to run our experiments on a topology more representative of a real network. Rocketfuel provides some PoP-level ISP maps which we could try to emulate in future experiments [2]. Of course, with a larger topology, is it more difficult to collect data and then draw conclusion based on factors like connectedness of the router and how far it is from the edge.

### 7.2 User behavior

We would also like to try make our model account for real-world user behavior better. For example, in our experiments, we had a very simple view of user viewing behavior. We assumed the user had a uniform probability of stopping the video at any point from the start to the end. While this accounts for the fact that not all videos are watched in entirely, in reality, people are more likely to watch the front part of the video than the end. Also, people do not necessarily start watching the video from the beginning and may skip around. Although these factors might not drastically affect our results, if we accounted for them, we could say our experiments more closely model the real world.

### 7.3 non-simulated Video Traces

For the sake of our experiments, we generated an artificial workload based on a zipf distribution of popularity and an arbitrary range of lengths and bitrate. In future experiments, we would like to parse and interpret some real traces of video watching in order to compute the effects of chunk size on them.

## 8. CONCLUSION

## 9. REFERENCES

[1] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. Developing a predictive model of quality of experience for internet video. *SIGCOMM Comput. Commun. Rev.*, 43(4):339–350, Aug. 2013.

[2] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring isp topologies with rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, Feb. 2004.

[3] Y. Sun, S. K. Fayaz, Y. Guo, V. Sekar, Y. Jin, M. A. Kaafar, and S. Uhlig. Trace-driven analysis of icn caching algorithms on video-on-demand workloads. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 363–376, New York, NY, USA, 2014. ACM.