Switch Assisted Peer to Peer Transfer

1. Abstract

This document describes the need for a switch assisted peer to peer transfer and illustrates the pros and cons of this approach. Document also explains the high-level design of the switch assisted peer to peer transfer system that will be implemented in Cloudlab using Dell S4048-ON switch.

2. Introduction

Cloudlab is a cloud-based research platform that facilitates users from various universities to carry out their experiments. One of the main challenges faced by the Cloudlab infrastructure is to reduce the latency during allocation of computation resources to the researchers.

When a user requests a PC, Cloudlab checks for the availability of the type of PC and installs the operating system of user's choice. Cloudlab has an image server called as Frisbee server that contains all the supported operating systems. Based on the user's request the Frisbee client running on the target PC downloads the operating system image from the Frisbee server and installs the same.

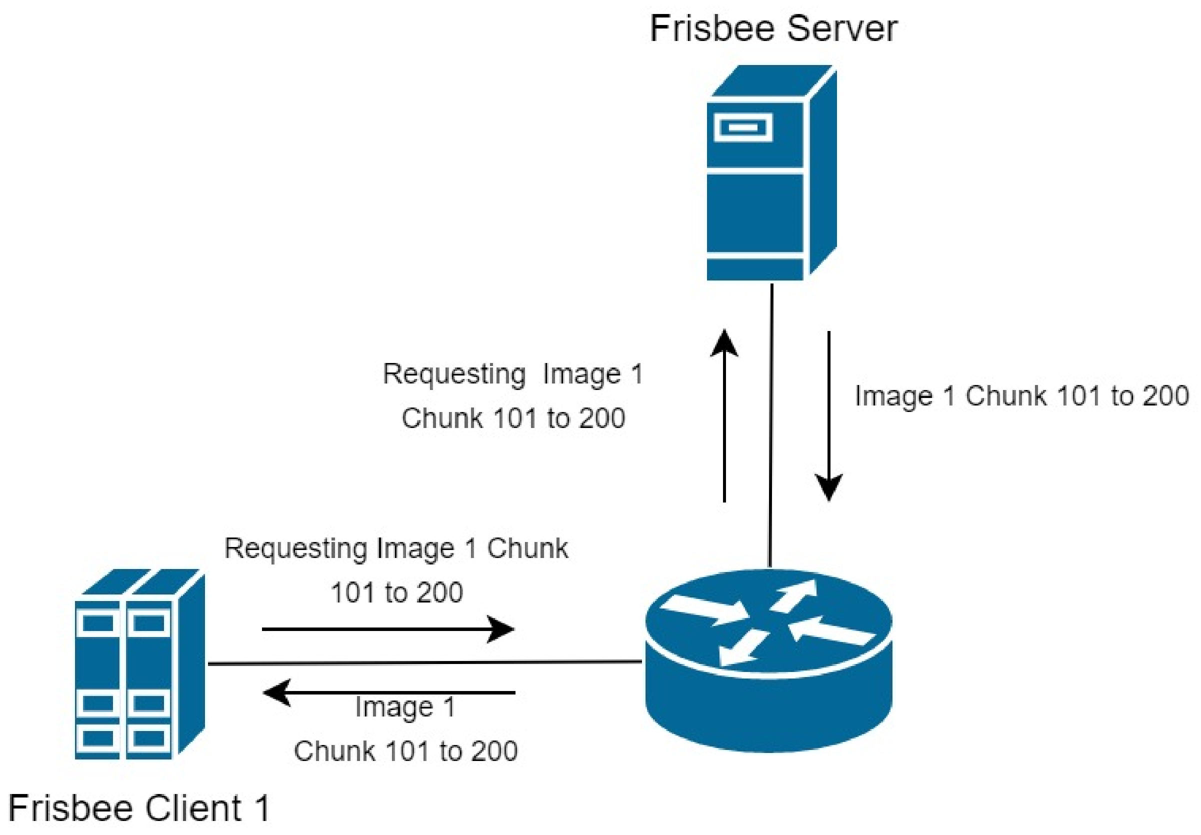
Frisbee server can serve multiple clients at the same time. If there are multiple clients requesting the same image, the traffic is duplicated in the network using multicast protocols such as IGMP. However, Frisbee server can become slow if there are simultaneous requests for different images.

Our goal is to implement a switch assisted P2P transfer mechanism to take care of the scenario mentioned above.

<u>Scope Limitation</u>
During the first phase of this project, only the Frisbee clients that are connected to the same switch can take advantage of this switch assisted P2P mechanism.

3. Basic switch assisted P2P transfer

For this project, we will use OPX running on Dell S4048-ON switch. Frisbee client is designed to request the data in chunks and we will take advantage of this implementation. To successfully conduct the switch assisted P2P transfer, we need to build a chunk availability database in the switch. Let us analyze the below example.

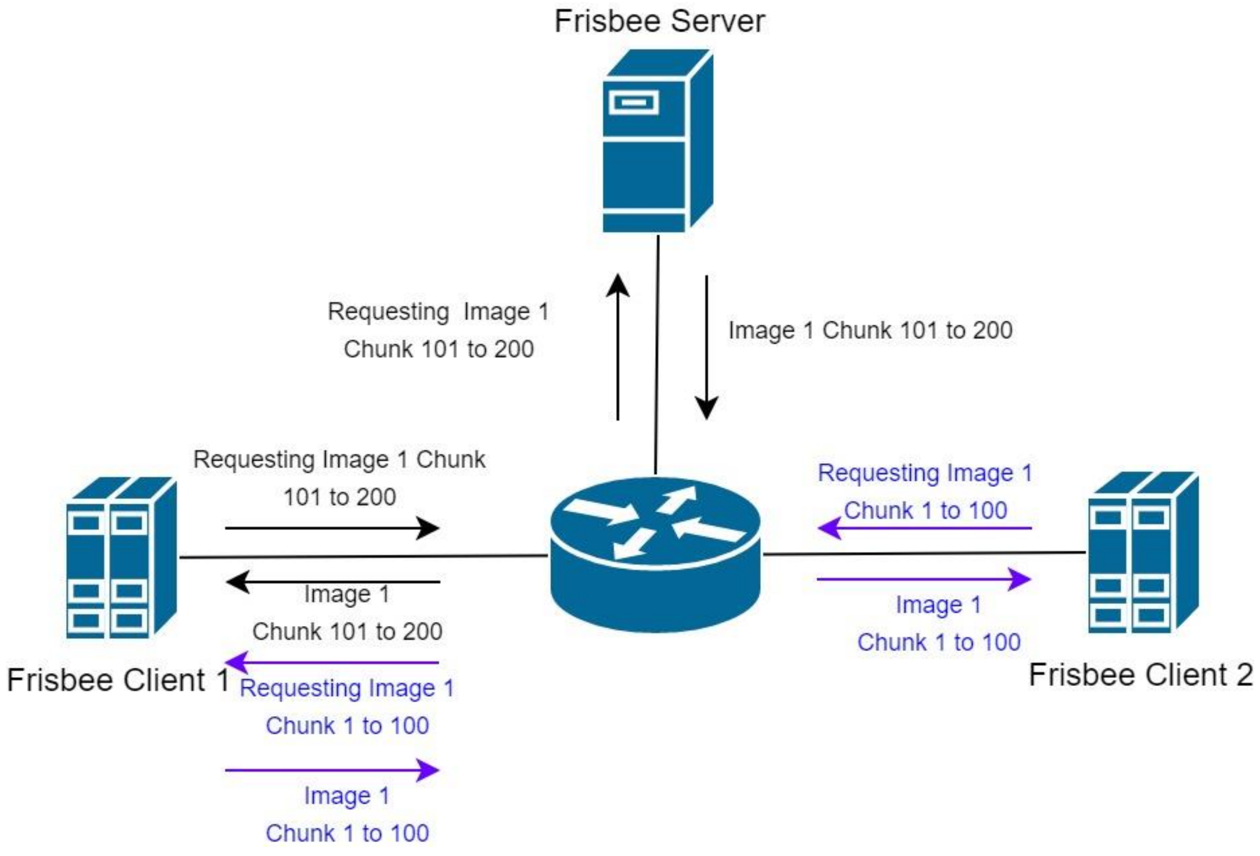


In the above diagram we can see that Frisbee Client 1 is requesting for Image 1 – Chunk 1-100. With the proposed switch assisted P2P implementation, switch should check if there is any entry in the chunk availability database for chunks 1-100. If there is none, then the request should be forwarded to the Frisbee Server.

Now after the Frisbee Client 1 has the chunk 1-100, it will request for the next set of chunks 101-200. During this time, if we have another Frisbee Client 2 that is requesting chunks 1-100 for Image 1, switch should forward Frisbee Client 2's request to Frisbee Client 1.

On receiving the request from Frisbee Client 2, Frisbee Client 1 should retrieve the chunks 1-100 and respond to the request. Currently the Frisbee clients don't remember the chunks that it has already received, because as soon as they receive the chunks they write to the disk and remove it from main memory. However, for our purposes Frisbee client should keep the received chunks in main memory.

Above example is illustrated as follows,



The primary set of tasks that we need to do are as follows,

- Redesign Frisbee protocol on client side to save the chunks.
- Processing Frisbee Packets at the switch CPU.
- Building Image-Chunk availability database at the switch.
- Design a load balancing mechanism at the switch.
- Redirecting the requests based on the forwarding database at the switch.

4. Frisbee Protocol changes

Current implementation of Frisbee protocol does not have support to save the received image chunks in main memory. Instead, the current implementation erases the chunks as soon as they are written on the hard disk drive.

For the switch assisted P2P mechanism, we need to make modifications at the Frisbee client side to save the chunks in the main memory. However, this introduces the following challenges.

- How much to store in the main memory?
- How does the client lets the switch know about clearing the main memory?

Above questions can be addressed in two ways.

Approach 1

Each client can be configured with how much they can store on their main memory based on their capability

Client lets the switch know, when it is deleting the chunks from main memory.

Switch removes the client from the availability database on receipt of above message.

Approach 2

Select the global minimum, considering the capabilities of all the switches in the Cloudlab.

Consider the global minimum as the sliding window at the switch.

Example:

Global minimum is 500

When the client 1 is receiving chunks 1-500, it can be listed as available in the switch for 1-500 chunks.

Once the client 1 receives chunk 501-600, it will automatically delete chunks 1-100 from main memory.

Simultaneously the switch should remove the client 1's availability for chunks 1-100, without any notification from client 1.

Approach 1	Approach 2
Pros 1. No need for global minimum. High end systems will not be affected due to the low memory availability on inferior systems. 2. No need to process the data packets to understand the pattern, which in turn results in low load on CPU.	Pros 1. Global minimum eliminates the need to configure on per system basis. 2. Auto-sliding window mechanism will help reduce control traffic between switch and Frisbee Clients
Cons 1. Introduces more control traffic in the network. 2. Requires implementation of new messages to let the switch know of the update.	Cons 1. Requires snooping the Chunk messages from the server to client. 2. Makes it difficult to expand beyond single switch.

Selected approach: Yet to be decided

Note to self:
At this point I'm thinking that the only way to check if the client has got the chunk is by snooping the chunks sent by the server.

If that is the case, then the snooping pros and cons on the above table becomes invalid

5. Processing Frisbee Packet at the switch

To build an Image-Chunk availability database, the switch must understand the current state of the clients. State of the clients can be understood by snooping the Frisbee control packets that are passing through the switch. Following are the packets that needs to be snooped and the reason why.

Packet Type	Switch Action
Client_Req_Image_Chunk	Check if there is any other Frisbee client that already has the information. If true, then forward request to client. Else, forward request to server.
Server_Resp_Chunk	Register that the client has the chunks. If sliding window is active, then take care of updating the availability list.

We need to install appropriate ACL rules to lift all the Frisbee protocol packets to the switch CPU. ACL rules can be installed depending on the TCP or UDP destination port information.

When lifting the request packets the ACL action should be TRAP_TO_CPU and while lifting the response packets the ACL action should be COPY_TO_CPU.

For implementing this module, we need to write a new module in OPX which will act as a CPS client. This module will listen to all the packets that are lifted by our ACL rules and forward packets accordingly.

Let us refer to this module as ***swp2pd*** from here on.

6. Image-Chunk availability database and load balancing

One of the main responsibilities of the proposed *swp2pd* module is to maintain the image-chunk availability and balancing the load.

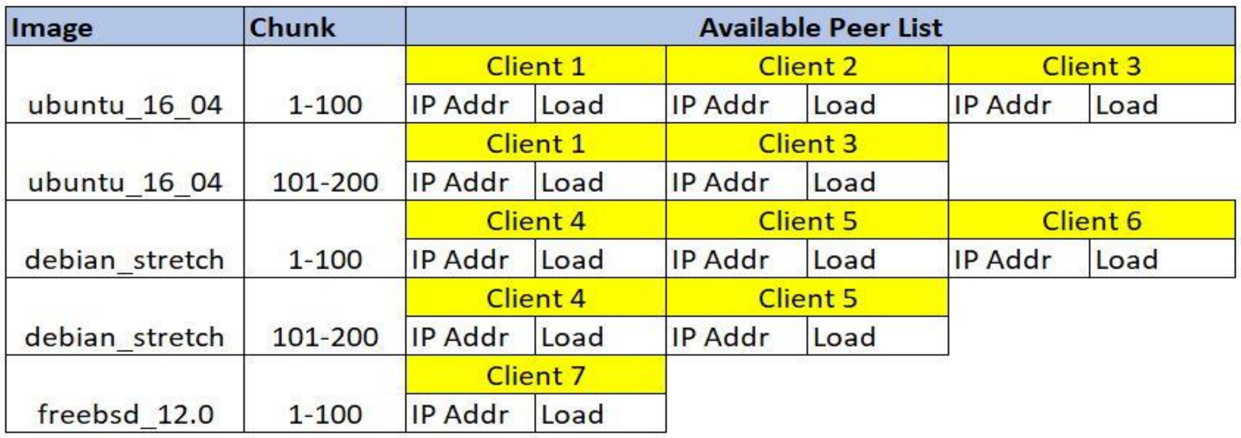


Image	Chunk	Available Peer List					
		Client 1		Client 2		Client 3	
ubuntu_16_04	1-100	IP Addr	Load	IP Addr	Load	IP Addr	Load
		Client 1		Client 3			
ubuntu_16_04	101-200	IP Addr	Load	IP Addr	Load		
		Client 4		Client 5		Client 6	
debian_stretch	1-100	IP Addr	Load	IP Addr	Load	IP Addr	Load
		Client 4		Client 5			
debian_stretch	101-200	IP Addr	Load	IP Addr	Load		
		Client 7					
freebsd_12.0	1-100	IP Addr	Load				

Above is an example of the proposed image-chunk availability database that will be present inside the *swp2pd.* Key can be a combination of image_name and chunk range. Availability peer list will be the data present inside image-chunk node.

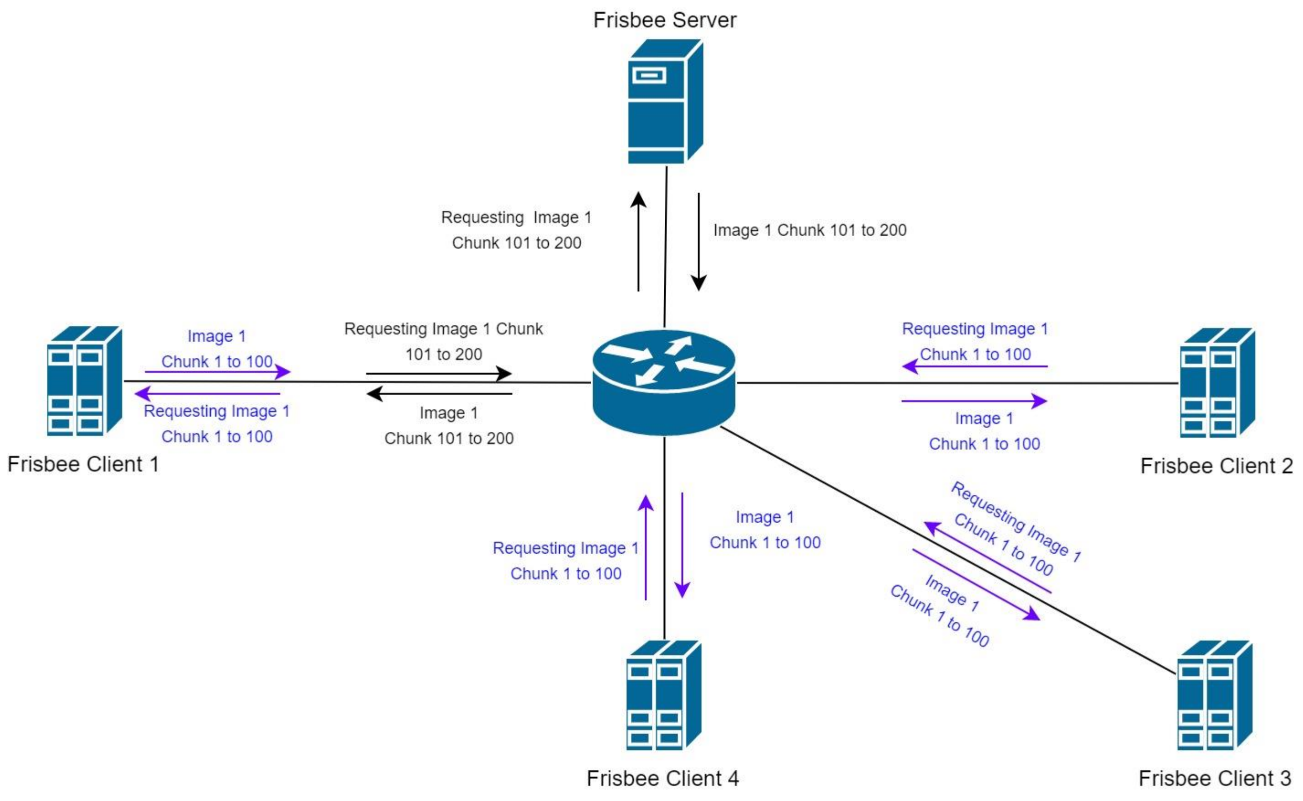
Availability peer list will have the following fields for each client that has the chunk.

Client IP Address	Client Current Load

To maintain the current load status of any client, the load should be incremented when a request is forwarded to the client. Load should be decremented when the request is served by the client. But this adds more snooping to the overall design causing more load on CPU.

Note to self:
Need to come up with a better way to maintain the current load status.

Following is an example of overloading, in this case Frisbee Client 1 is overloaded.


Frisbee Server
Requesting Image 1 Chunk 101 to 200
Image 1 Chunk 101 to 200
Image 1 Chunk 1 to 100
Requesting Image 1 Chunk 1 to 100
Requesting Image 1 Chunk 101 to 200
Image 1 Chunk 101 to 200
Requesting Image 1 Chunk 1 to 100
Image 1 Chunk 1 to 100
Frisbee Client 1
Frisbee Client 2
Requesting Image 1 Chunk 1 to 100
Image 1 Chunk 1 to 100
Requesting Image 1 Chunk 1 to 100
Image 1 Chunk 1 to 100
Frisbee Client 4
Frisbee Client 3