Can we improve Dropbox's LAN Sync feature with NAT Traversing?

A report by Kaustubh Welankar, Basu Dubey, Naren Surampudi, Chandrahas Aroori

Introduction

Dropbox is a file hosting service launched in 2007. It is used in many organisations as well as by individuals for their data storage needs. Dropbox is available as a web based service, where users can upload their files and view them, or as a native client, where files are synced automatically. Native clients are available for Windows, MacOS, Linux based PCs, and apps are available for Android and iOS.

The LAN sync feature has been available to users since 2015. The main goal of the feature is to increase syncing speeds significantly when the file you are trying to download exists on the network. Another benefit for Dropbox is the reduced load on their servers. This is done by searching if the file is available on a PC in the same subnet and downloading from the server only if the search returns negatively.

LAN Sync Functioning

IANA has reserved TCP & UDP ports 17500 for LAN Sync. LAN sync does not work if the clients lack an internet connection.

The LAN sync system has 3 components to it, that run on each client. We follow the notation that Dropbox mentions in the blogpost [1]. The components are Discovery Engine, LAN Sync Client, LAN Sync server. Refer to Figure 1.

Discovery Engine

The discovery engine does UDP broadcasts on the subnet it is on. It also listens on UDP port 17500 for incoming broadcasts.

LAN Sync Client

The LAN Sync client is invoked when the Dropbox client (do not confuse between the two) determines a new file can be downloaded. The LAN Sync client searches for the file with the clients discovered by the Discovery Engine.

LAN Sync server

The LAN Sync server is responsible for serving requests made to it by other LAN Sync clients. It verifies that the requesting client has authorisation to download the file. This authorisation is done with Dropbox servers. The LAN sync server binds to TCP port 17500. If the port is unavailable, the server binds to a random port. The randomly binded port number is added to the UDP packets broadcasted by the Discovery Engine.

Experiments

We set up our own test bed on the campus network, that emulates possible use cases. We have 3 Raspberry Pis and a laptop placed behind different routers to emulate use cases.

The Test Setup

Raspberry Pi 1 (aliased kpi) was behind router 172.16.8.251. Raspberry Pi2 (aliased gpi) was behind a router at 172.16.8.81. Raspberry Pi3 (aliased arbpi) was directly connected to the network, and had the 172.16.8.33. View figure 2.

Experiment 1

Experiment 1 involved verifying that the UDP broadcasts were being sent out by the client. We installed the dropbox client on a laptop (192.168.100.9). UDP broadcasts were observed. Broadcasts were made to 255.255.255.255 and 192.168.100.255 . Our hypothesis is that this was done to

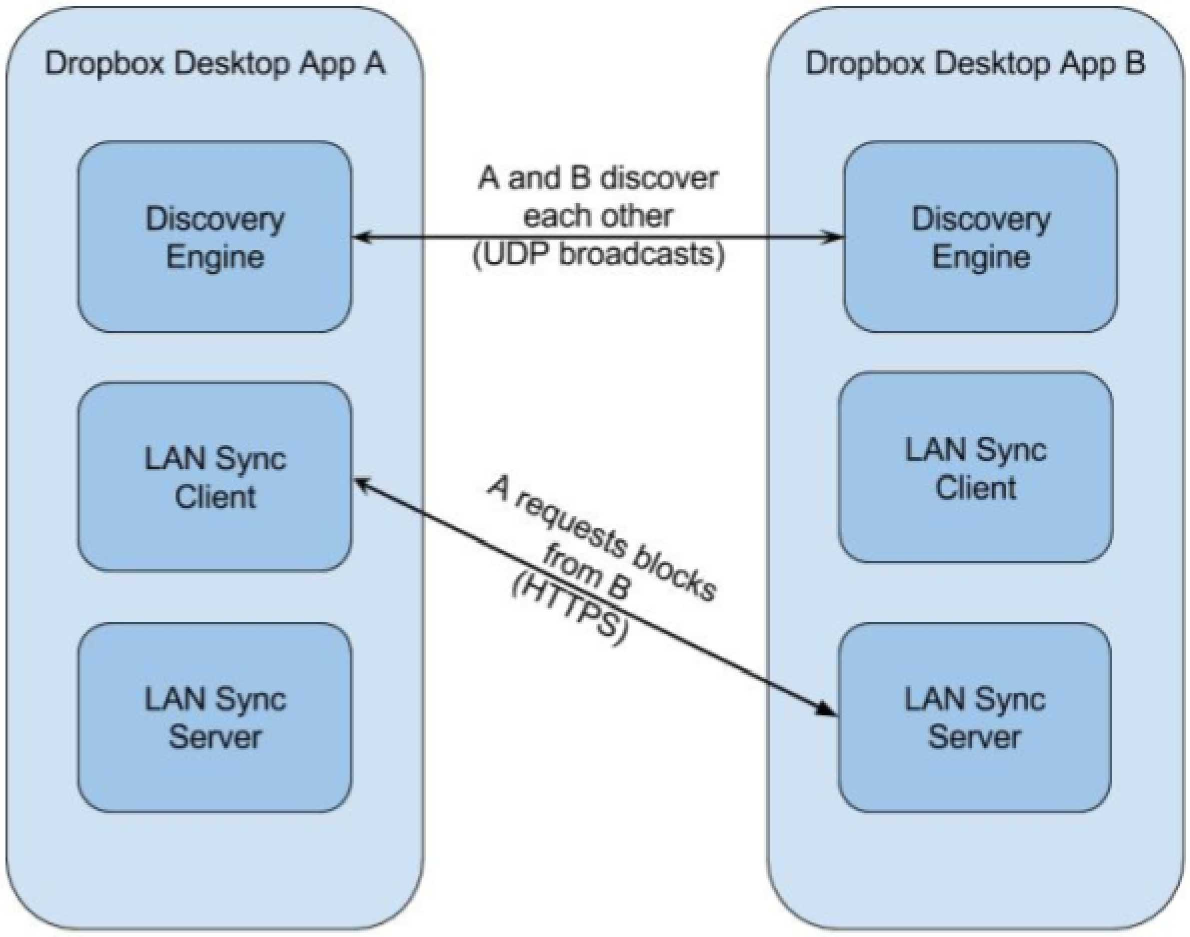


Figure 1: LAN Sync client architecture (from ref [1])

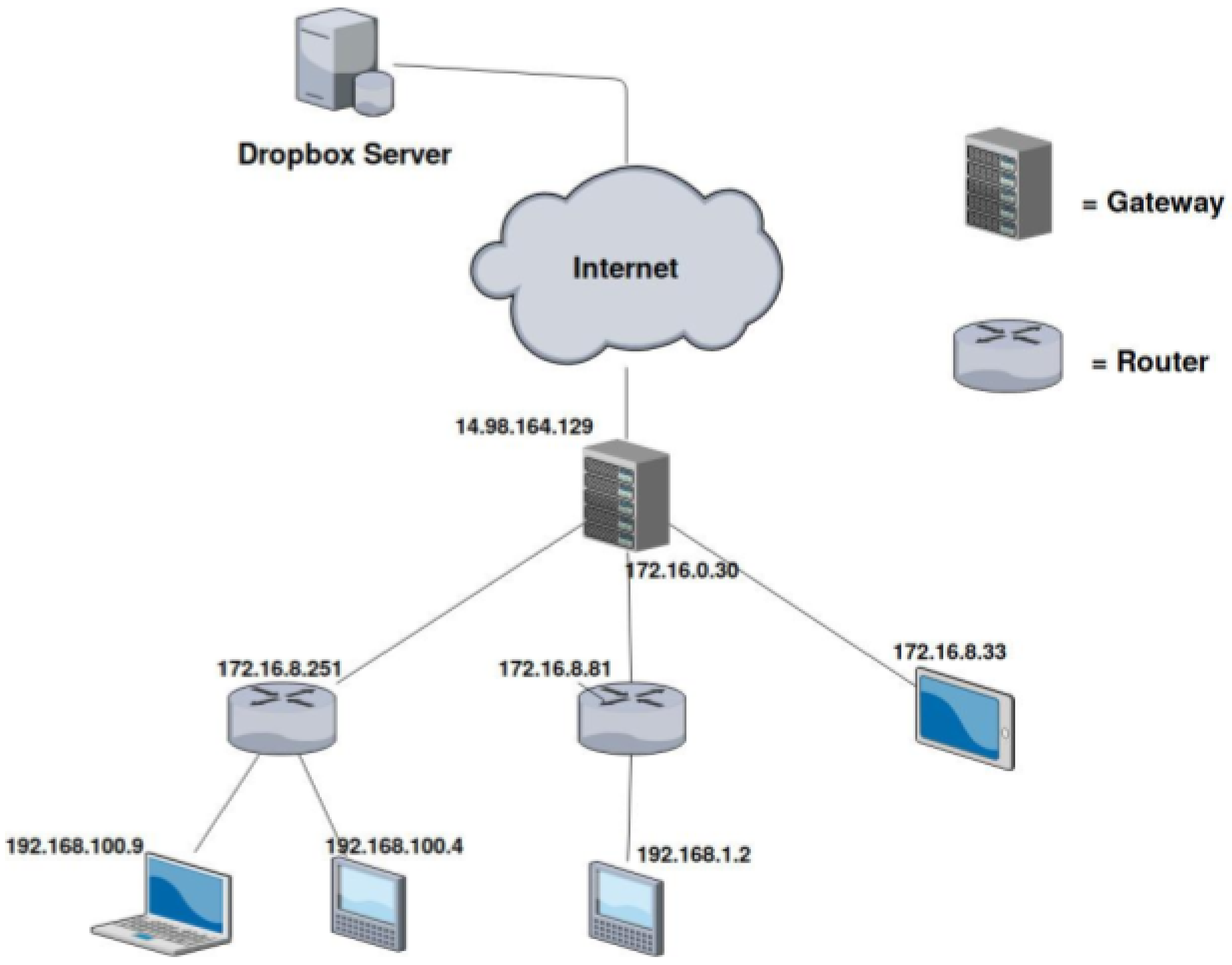


Figure 2: Testbed architecture

ensure higher chances of delivery. 6 packets were sent every 30 seconds. However, we do not investigate this any further. As mentioned in the Dropbox blog post [1], the UDP packets contain the Sync server port number, along with the namespace and other dropbox specific metadata.

We verified that the UDP discovery packets were received by another device on the same subnet. (See kw_pc.pcapng. Use display filter udp.port==17500. You will see discovery probes from 192.168.100.2, on which wireshark was run. The other was 192.168.100.5. You can see incoming packets from 192.168.100.5) This verifies that broadcasts work on the same subnet.

Experiment 2

The Dropbox article mentioned that LAN sync worked when the clients were on the same subnet. We hypothesized that this was due to the unreachability of the UDP packets. This was a correct guess. Clients at 192.168.100.9, 192.168.1.2 and 172.16.8.33 did not receive any UDP broadcasts from each other.

Current Limitations of the feature

Experiment 2 confirmed our suspicions. The subnetting in this case was limiting local syncing. Although all the endpoints here are on the same network (ie. all have the same gateway to the internet), local sync is not feasible. This is a significant issue in networks, which have many subnets behind the same gateway, like the BITS Hyderabad campus.

Imagine someone wanting to share large files, where each person is behind their own router. With the current solution, a lot of campus bandwidth would be used since one would have to download the file from the Dropbox server, instead of peering it locally.

Is a duct-tape solution possible ?

Port Forwarding
We could enable port forwarding at the router(172.16.8.251). We could ask it to forward UDP packets with port 17500 to some device.

That means that only one of the device behind 172.16.8.251 will know of other peers in the network. The problem with this is the packets can be forwarded to only one device. This will only allow devices like 192.168.100.9 to discover that 172.16.8.33 is available. However, the reverse situation, where 172.16.8.33 discovers 192.168.1.2 is not possible. Also, only one of 192.168.100.9 and 192.168.100.4 will be able to make use of this feature, at any given time. This makes this solution messy, while we are discarding the assumption that the end user knows how to set up port forwarding, and has the will to do it. Also, there is no way client behind 2 different routers(like 192.168.100.4 and 192.168.1.2) can detect each other.

Port Triggering
After an outbound connection is made on a particular port, the router will wait for a particular time, and allow an incoming connection on a prespecified port. However, this will require the endpoints to know the IPs of the routers they are behind. However, this is a very complicated setting that users will have to configure. Not all routers have this settings available for users to work.

To summarize, duct tape solutions do not work robusty, and require the user to have significant technical expertise.

NAT Traversals

What is NAT?
A Network Address Translation is a method of remapping one IP address space into another IP address space by modifying the IP headers. NAT-ing is done by routers, as they are connected to two different networks, with different IP address ranges.

NAT-ing allows one public IP address to be used for multiple devices on a private network. NAT works for both TCP and UDP.

When a client initiates and outbound request, the TCP/UDP packet sent has a the device’s local address in the header. The NAT, make an entry for that IP and port number and maps it to a public IP and port on the other interface of the router. That way, the server sees the request originating from the router, and can reply to it. The reply is NATted, and packets are sent to correct internal host.

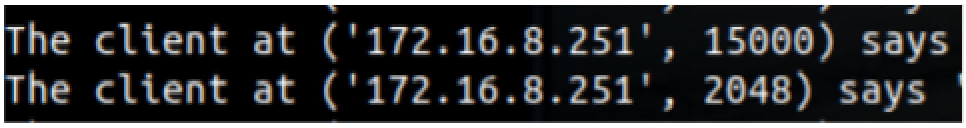
Eg: We hosted a normal UDP server on the device at 172.16.8.33 and, sent it packets from 192.168.100.4 and 192.168.100.11. Refer to figure 3 for the demo that shows NAT in action. Not how the server does not get the private IPs (192…..) but the IPs of the router (172….). This is how NATting happens.

Hole Punching

Hole punching is a method of P2P communication used. We use a rendezvous server that exchanges the public ips of the devices. The clients now have the public endpoints of the other clients. They can try to initiate an connection, abusing the fact that the NAT entries in each router remain. We use the method as specified in [2]. However, we do not exchange the private IPs of the devices. The reason for this is that private IPs are being discovered in each subnet through the already existing mechanism. We expect, as the authors of [2] call it, a hairpin translation at the campus NAT.

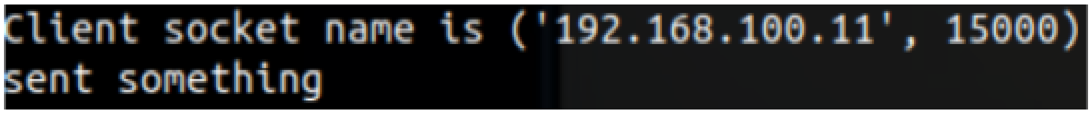
UDP Hole Punching

Since UDP packets are blocked at the gateway on the campus network, we use the Raspberry pi @ 172.16.8.33 as the rendezvous server. < Demo to be shown >



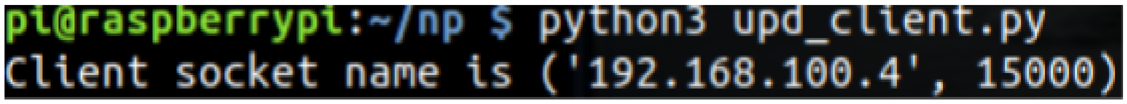
```
The client at ('172.16.8.251', 15000) says
The client at ('172.16.8.251', 2048) says
```

Server sees 2 clients, from the same IP



```
Client socket name is ('192.168.100.11', 15000)
sent something
```

Client 2 uses port 15000, but gets NAT-ed to port 2048
Any incoming packets to port 2048 will be forwarded to client 2



```
pi@raspberrypi:~/np $ python3 upd_client.py
Client socket name is ('192.168.100.4', 15000)
```

Client 1 uses port 15000 and gets NAT-ed to port 15000
Any incoming packets to port 15000 will be forwarded to client 1

Figure 3

TCP Hole Punching
TCP hole punching uses the same mechanism. As the paper specifies, we use the TCP client socket with the SO_REUSEADDR option set. This allows us to reuse a client socket immediately after closing a connection. TCP allows for simultaneous open, which helps with peer connection establishment. **Our final demo involves showing that data can be transferred between 2 clients using TCP hole punching. We propose that this is a feasible method.** < Demo to be shown > We also believe that our demo showing TCP hole punching working is a PoC of the system.

The rendezvous server, as the authors of [2] call it, are now called as STUN servers [5].

The Solution

We propose a solution that can address this shortcoming of Dropbox's LAN sync solution.

1. In each subnet behind a router, we have a device that maintains a TCP connection with the rendezvous server.
    a. This server can be chosen by consensus methods, that are common.
    b. For eg, of the 2 devices behind 172.16.8.251, one of them can be elected as the 'leader' [6]
2. If a client realizes that it needs a file, it first searches in its own subnet, using the UDP discovery mechanism.
3. If the required file is not found, the client contacts the rendezvous server, which returns the address of a client that has the file and is behind the same gateway.
    a. If such a client does not exist, the client that needs the file downloads the file from Dropbox.
    b. The Dropbox rendezvous server can take ask the elected leader to search in its subnet, to ensure a comprehensive search.

Peering methods like these have been used by many torrent like clients. This is why we claim that Dropbox should implement such a solution

We have implemented the NAT hole punching in Python.

Possible failures of this method

As the results from [2] show, NAT Hole punching is possible, but success depends on the type of NAT. NAT traversals, despite many RFCs, haven't been incorporated in all, if not most, routers.
NAT implementation is sometimes vendor specific, and may lead to issue with certain implementations.

References

[1] Inside LAN Sync (https://blogs.dropbox.com/tech/2015/10/inside-lan-sync/)

[2] Peer to Peer Communication Across Network Address Translators (http://bford.info/pub/net/p2pnat/)

[3] Port Triggering (https://en.wikipedia.org/wiki/Port_triggering)

[4] Network Address Translation (https://en.wikipedia.org/wiki/Network_address_translation)

[5] STUN Protocol (https://en.wikipedia.org/wiki/STUN)

[6] Leader election (https://en.wikipedia.org/wiki/Leader_election)