NDNDrop - Device Agnostic File Sharing Using NDN

Extended Abstract

Nishant Sabharwal
UCLA
nsabharwal@ucla.edu

Carlos Santillana
UCLA
csant019@cs.ucla.edu

ABSTRACT

We explore the Named Data Networking architecture by designing a file sharing app which can share files utilizing the network layer. This app demonstrates the usefulness of the NDN architecture in a small, standalone application which is an important step before widespread adoption. The app also highlights the difficulties newcomers to NDN have in picking up the design principles and tools.

The application was written in C++ using the ndn-cxx library. A current copy of the code can be found at https://github.com/carlossantillana/ndn-drop.

KEYWORDS

Information Centric Networks, Named Data Networking, Decentralized

ACM Reference Format:
Nishant Sabharwal and Carlos Santillana. 2020. NDNDrop - Device Agnostic File Sharing Using NDN: Extended Abstract. In *ACM Conference on Information-Centric Networking (ICN '20), September 29–October 1, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 3 pages.

1 INTRODUCTION

Apple's AirDrop offers a useful way to share a file between two Apple devices without connecting to WiFi. [3] We use NDN to build an application which can implement this feature in a device agnostic manner. The application is able to share files across any medium. It was designed with a wirless medium such as WiFi or Bluetooth in mind but it can make use of any channel that an IP packet can use.

Throughout the paper, we use an example where Alice is a user with two devices: `alice/laptop` and `alice/phone` which compose her network.

1.1 Goals

The primary use case of the application is a user with many devices such as a laptop, a phone, and a desktop. These devices form a network which can exchange files. The design is for a simple application that can allow files to move between the devices. A user can place files in a designated folder that then become public to this network of devices. For example, a user can place files in a folder on their laptop. If they want to access them from their phone, they can list the files in that directory and then move them directly from the laptop to the phone across a broadcast medium.

Devices need to be able to join in an ad hoc manner. Devices are portable and some may be new. Alice may take her phone out with her or replace her laptop. IP makes this hard because the IP address of these devices must be known. When devices are exchanging packets, the packets should not go over the internet. Many TCP/IP applications use an always-on server at a known IP address for the two devices to connect to. This is wasteful because the packets travel longer than they need to and face an unnecessary risk of being sniffed. Network outages should not stop two nearby devices from communicating. The application has the two devices communicate directly. Rather than knowing a phone's IP address, forming a connection with it, and then exchanging data, a device simply makes a request to the network for that file. [5] The user should have full control over their security and decide which devices they want to be allowed into their file sharing network. Files are encrypted and only trusted devices have the keys to decrypt them. If Alice's phone and laptop are exchanging files in the library, a nearby unauthorized phone should not be able to eavesdrop and read those files. Worse yet, an unauthorized device should not be able to successfully request files.

1.2 Features

We identified key features that are necessary for this application and designed the backend infrastructure that can be used as an API. The application is fully distributed and directly peer to peer. Each node participates with the following actions.

- `discover`: A node needs to be able to find out which devices are currently nearby.
- `list`: Once a node knows which neighbors are available, it should be able to list which files are available. This would result in a list of filenames that can be moved accross the network.
- `receive`: Once a file is selected, it should be able to request it and receive it.
- `decrypt`: It should be able to decrypt files. It should also be able to get the necessary keys.

2 DESIGN

2.1 Discover

The name of the device is also important because it identifies the originator of the files. A user may want to list all the files from their phone. Rather than the user typing in "Phone," it would be useful for the device to show a list of available devices. This is accomplished using a soft state protocol. Each device maintains a Neighbor List consisting of a list of neighbors and a counter for each. Each node periodically sends out interest packets. Upon receiving

ICN '20, September 29–October 1, 2020, Virtual Event, Canada
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8040-9/20/09.
DOI: 10.1145/3405656.3420232

a discover interest, a node waits a random time and responds with it's Neighbor List. Each node maintains the Neighbor List by periodically decrementing the counter for each neighbor. When the counter reaches 0, the neighbor is assumed no longer reachable and removed. The counter is reset if it hears a discover packet from that neighbor or sees the neighbor in an incoming Neighbor List. Though not yet implemented, eventually neighbors should listen to the broadcast medium to see if other neighbors respond before responding with their list. This allows a dynamically list of all currently nearby neighbors to be maintained.

2.2 List and Send

A node needs to be able to list the files available for transfer of a given neighbor. This is accomplished by maintaining a file, `files.txt` which is simply a list of file names. A node can send an interest for that file. A node can send an interest for `/ndn/drop/alice/phone/files.txt` and get a list of available files. Each node periodically publishes this file to a local repo. Similarly, the files in the sharing directory are published to the repo and neighbors can request them.

2.3 Access Control

Access control is implemented by encrypting files with symmetric keys. Users are authenticated using NDN's methods of establishing a trust anchor and authenticating signatures. We assume that Alice is the trust anchor and all her devices are preconfigured with her public key as part of the bootstrapping proccess when NDNDrop is installed. As long as nodes encrypt every file they publish, we can assume privacy is maintained. Access control becomes a matter of key distribution. Each node has it's own symmetric session key to encrypt data. If the phone requests a file from the laptop, it needs the session key. It then requests the session key with an interest `/ndn/drop/alice/laptop/session_key`. The phone must sign this interest with it's own private key. The phone's certificate, signed by Alice, is used to authenticate. The certificate is the phone's public key signed by Alice which can be used to verify the interest's signature. This proves the identity of the phone and it's trust in our simple trust model. [4] The session key can then be returned. Importantly, it is encrypted with the phone's public key so only the phone can decrypt it.

3 BENEFITS OF NDN

There are many existing decentralized, peer to peer applications which run on the TCP/IP stack. Complicated protocols have been designed to make these work. NDN's features make the design much simpler. When designing a peer to peer application, IP address management becomes difficult. In order to connect with a peer, it needs an IP address to send an IP packet. For example, Freenet is an adaptive peer to peer network application that serves as distributed data store that uses the IP stack. A new node needs the IP address of an existing node to join. This requires it to be configured out of band or the use of central seed servers. [1][2] This was accomplished in NDN by naming the network a node wants to join via discover interests. This allows the users to add new devices easily without needing to maintain a list of IP addresses.

NDN's security features were useful for the design. Since all data packets are signed, a user can verify the file is the file that was requested. Signing data packets is not unique to NDN but since it is built-in, it required little extra thought when designing the app. The built-in key infrastructure of NDN proved useful for protecting the privacy of the user. Through the use of certificates, each device's public key can be requested so data only they can decrypt can be shared across the medium. The app was able to easily pass around symmetric keys used to encrypt the files. This leaves room to change out keys to ensure privacy is protected through the lifetime of the app. Again, encryption and key infrastructures are not unique to NDN but NDN made it much easier to incorporate. [4]

By naming the data, the network matched the application layer which simplified design. For example, to share files, each node maintained a list of available files in a file, `files.txt`, and a node could simply request that file. This is much more in line with the high level design rather than thinking of it as sending a TCP message to a node requesting the available filenames and responding to such messages.

4 FUTURE WORK

There are remaining tasks to improve the app. The current method of detecting neighbors is wasteful. When a node sends a discover interest, all neighbors reply. In a broadcast medium, neighbors can listen for another's reply and suppress it's response if another neighbor responds. The application currently uses threads to listen for interests and responds with data packets. It needs to take advantage of the repo model and instead publish files to a local repo. The trust model needs to be more automated. This is more of a code improvement. Currently, it manually follows the chain of signatures but there is a useful feature in the library to take advantage of schematized trust.[4] Finally, designing a GUI will be the final step to make it a full application.

5 CONCLUSION

NDN is a proposed replacement for the internet architecture that must be gradually adopted. By designing NDNDrop, we show how NDN's features can facilitate the design of distributed systems and simplify the implementation of security. Most developers have experience with TCP/IP. NDNDrop's design highlights the learning curve of NDN for newcomers and highlights the parts that are simple to pick up as well as the parts that were harder to understand. For example, the app was originally designed like a web server listening for requests while the more NDN-centric approach is to publish all the data to a local repo. NDN lends itself nicely to distributed applications with ad hoc connections by eliminating the need for determining IP addresses. Security ended up being greatly simplified because NDN made confirming identity and exchanging keys easy.

REFERENCES

[1] Freenet project: 2020. Retrieved August 29, 2020 from https://freenetproject.org/pages/help.html.

[2] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. *Freenet: A Distributed Anonymous Information Storage and Retrieval System.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 46–66.

[3] Cohen, J. How to use airdrop, June 2019. Retrieved August 16, 2020 from https://www.pcmag.com/how-to/how-to-use-airdrop.

[4] Yu, Y., Afanasyev, A., Clark, D., claffy, k., Jacobson, V., and Zhang, L. Schematizing trust in named data networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking* (New York, NY, USA, 2015), ACM-ICN '15, Association for Computing Machinery, p. 177–186.

[5] Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., claffy, k., Crowley, P., Papadopoulos, C., Wang, L., and Zhang, B. Named data networking. *SIGCOMM Comput. Commun. Rev. 44*, 3 (July 2014), 66–73.