NDN over Wi-Fi Direct for Linux

Amar Chandole
Department of Computer Science, University of California, Los Angeles
amar.chandole@cs.ucla.edu

ABSTRACT

This project report outlines the work done in the development of a protocol that lets a Linux machine with Wi-Fi Direct capabilities talk to other Linux devices using Wi-Fi Direct standard [1] using Named Data Networking (NDN) [2]. The project has two main parts - first is to achieve a stable connection between the two devices at a link layer level, second being developing the protocol that works with the NFDs [3] on both the sides to help the connected nodes share the prefixes they serve, can connect to or wish to have. The project report currently focuses on single-hop communication but has a design that can be easily extended to multi-hop communication later.

The protocol uses naming rules for the prefixes that are consistent with those used in NDN Over Wi-Fi Direct for Android protocol [4] to facilitate interoperability between Linux and Android in the very near future. While now there is an immediate use-case of users who wish to use NDN applications and communicate to other local Linux devices, the long-term goal is to support seamless communication between any two devices, irrespective of the operating system they use. Implementation of the protocol resulted in a stable communication. The details of the protocol and the future challenges will be explored in this report.

Keywords

NDN, NFD, Wi-Fi Direct, Ad-hoc networking, wpa_supplicant

1. INTRODUCTION

1.1 Motivation

The following protocol design is intended for use in ad-hoc environments wherein there are no available Wi-Fi or cellular networks. This protocol provides a means to which two Wi-Fi Direct compatible Linux devices can connect to one another to exchange and maintain information about available NDN Data prefixes. The following specification does not touch upon multi-hop communication, but its design is such that it should easily accommodate it if implemented.

1.2 Scope

Even though the connection part of this project is limited to handling Linux to Linux connectivity right now, the NDN over Wi-Fi Direct protocol part works for all the devices that support NDN and NDN-CXX [5]. These devices need to be connected at link layer before they can simply run this protocol daemon and start communicating the prefix information to their peers using the same protocol.

Also, because there is already an implementation of a similar protocol in the NDN project for Android to Android communication, a few pending changes to this project will ensure connection and communication between Android and Linux machines as well. The prerequisite for all these scenarios is obviously that these devices are Wi-Fi Direct compatible and have suitable APIs that allow us to manage the connections. We shall talk in detail about all these topics in sections to follow.

1.3 Interfaces

The protocol assumes the existence of an underlying NDN forwarding layer (NFD), by which Interest and corresponding Data packets can be sent and received, among other common use cases. The following design also utilizes WPA Supplicant for Linux device connectivity

and NDN-CXX library for implementation and execution of the protocol.

2. RELATED WORK

2.1 "NDN Over Wi-Fi Direct for Android"

This project defines its own protocol design for running ad-hoc communication over Wi-Fi interface using Wi-Fi Direct standard for Android devices. The devices can exchange and maintain information about available NDN Data prefixes once the peer-to-peer communication is set up.

The protocol defined in this project is very similar to the one defined in our project. A few basic changes have been carefully made in our project to reduce the complexity of initial prefix exchange and registration, which will be explained in detail in the paper later. The design of *NDN Over Wi-Fi Direct for Android* utilizes the Android Wi-Fi Direct API as implemented by December 2016 and this project has been embedded inside the NFD application for Android smartphones available on Google Play Store [6].

2.2 wpa_supplicant

wpa_supplicant [7] is a WPA Supplicant for Linux with support for WPA and WPA2 (IEEE 802.11i / RSN). It is suitable for both desktop/laptop computers and embedded systems. Supplicant is the IEEE 802.1X/WPA component that is used in the client stations. It implements key negotiation with a WPA Authenticator and it controls the roaming and IEEE 802.11 authentication/association of the wlan driver.

wpa_supplicant is designed to be a "daemon" program that runs in the background and acts as the backend component controlling the wireless connection. wpa_supplicant supports a text-based frontend (wpa_cli) and a GUI (wpa_gui) and these are included with the wpa_supplicant software. It is freely available. The use of wpa_supplicant for this project has been discussed in the working of the protocol in the upcoming sections.

3. LINK-LAYER CONNECTIVITY

3.1 Wi-Fi Direct

Wi-Fi Direct is a certification mark certified by Wi-Fi Alliance for devices supporting a technology that enables Wi-Fi devices to connect Directly, making it simple and convenient to do things like print, share, sync and display. Products bearing the Wi-Fi Direct certification mark can connect to one another without joining a traditional home, office or hotspot network.

Although not all the Wi-Fi devices necessarily support transmission and reception of Wi-Fi Direct frames, most of the recent devices do. Wi-Fi Direct standard came into the market around the year 2008 for the first time. Devices supporting Wi-Fi Direct were rolled out by the leading manufacturers by 2010. Android devices supported Wi-Fi Direct starting from the year 2012 and Android also provided an API for developers in the same period. This led to a wide use of Wi-Fi Direct for fast and reliable P2P (point to point) communication, becoming an increasingly preferred choice over Bluetooth [8] that served a similar purpose.

Linux devices are understandably the primary interest for this project. These devices are capable of supporting Wi-Fi Direct connectivity but do not have an in-built manager software to establish and manage P2P connections. We searched for manager tools and found that wpa_supplicant is a software that can be used to suffice the primary needs of the project.

3.2 Using wpa_supplicant

wpa_supplicant is an open-source project that implements key negotiation and other authentication/association related steps for WPA and WPA2 protocols. Additionally, this supplicant can be configured to support P2P connections if a flag CONFIG_P2P is set in the defconfig file before building the code. Once it is configured to support P2P i.e. Peer-to-Peer connections, there are simple commands that can be used to turn P2P on, search nearby peers and connect to them at link layer.

Commands used in this project are:

A. p2p_find: scan for nearby peers who are finding other peers too.

B. p2p_peers: show discovered peers in vicinity.

C. p2p_connect <device addr> PBC go_intent = <0-15> auth: attempt to connect to the device whose device address (MAC) is provided as an argument. PBC is the Push Back Control mode, go_intent is the intent of the device to become a group owner. 1 is the lowest and 15 is the highest intent to become group owner.

A group owner works as a softAP or like a router who helps other client nodes in the group to communicate with each other by forwarding the messages. In this project, it is not important to have a specific node to be a group owner. So, all the nodes using this protocol can be configured to fire the p2p_connect command with a go_intent equal to 7, which means that the group owner will be decided based on the random group owner negotiation. If required, the choice to be a group owner or client can be easily given as an option to the user, which is done in the current implementation for ease of testing and focusing on the protocol.

Another thing that is important here is that the p2p_peers step fetches the p2p_name configured by the other device and its MAC address as well. Therefore, each device already has the knowledge of the MAC addresses of its peers. This MAC address knowledge is important for the protocol in a way that will be explained in the upcoming section. Connectivity is thus established at link layer after all these steps are executed.

4. PROTOCOL TO SYNC PREFIXES

4.1 Purpose of the protocol

It is important to understand the need of this protocol in our case of ad-hoc communication. In normal scenarios where devices are connected to the internet and are using NDN, they need to know where they can find the data their applications are looking for. Also, these nodes need to inform all the other nodes on the internet that they are serving some particular prefixes that their native NDN applications have announced. NDN Forwarding Daemon (NFD) has the task to do all this and it exchanges and updates all the prefix information by multicasting interest and data packets to all the neighboring nodes on the internet.

In case of an ad-hoc connection where all the clients are Directly connected only to the group owner, it is necessary to have a set of rules that makes the NFDs of these nodes aware of these connections and makes them exchange prefix information in a specific way. This is exactly why we need this protocol which is the prime part of this project.

4.2 Purpose of using IPv6 link-local addresses

After the link layer connection stage, the only addressing information known about peers is the Wi-Fi Direct interface MAC address of both the devices to each other. This information is sufficient to communicate using NDN for Linux devices, as NFD can simply use the ethernet face created towards the local Wi-Fi Direct network interface being used for this connection. However, we want to use the same protocol to make the communication of Linux devices with even Android devices possible, and Android devices cannot handle ethernet frames Directly without using the IP protocol (IPv4 or IPv6). Only if the Android smartphone is a rooted device can it be configured to handle ethernet frames Directly. Rooting is not a viable option to the common user, thus leaving this option infeasible. Thus, if we limit to using Direct ethernet faces for NDN communication, we lose out on extending the same protocol for Android communication.

To avoid this Android-Linux communication incompatibility problem, we make use of the IPv6 link-local address [9][10]. In IPv6 protocol, link-local addresses are a special scope of address which can be used only within the context of a single layer-two domain, that is for a one hop local communication. These addresses are useful for establishing communication across a link in the absence of a globally routable prefix or for

intentionally limiting the scope of traffic which should not be routed (for example, routing protocol advertisements). These IPv6 link-local addresses are auto-configured as a part of IPv6 protocol for each of the network interfaces. What is most important is the fact that these addresses can be calculated by transforming MAC address in a rule-based method. Thus, it obviates the need of something like DHCP server because both client and group owner have their own addresses already. Also, as MAC address is already known to the peers, nothing related to the IP address needs to be sent over to start the conversation and the link-local IPv6 address is calculated at both ends as soon as the connection is established.

4.3 Model of operation

The protocol needs to take care of two major things - to notify all connected peers about the prefixes being served locally, and to request other connected nodes to send the prefix information they have. In Wi-Fi Direct groups, the group owner is connected to all other nodes, whereas the client nodes are only connected to the group owner. These clients can talk to each other if the group owner relays their prefix requests and replies to the rest of the network by acting as a router.

However, there is still no need to make any distinction in the protocol for group owner and clients, because irrespective of the role, each node should share only the information it has learnt from all other nodes except the one it is talking to. This is identical to a well-known method called Split Horizon Route Advertisement [11]. Thus, in the case of non-group owner (client) nodes, they will only send the prefixes announced at their NFD by their local NDN applications. Group owners, similarly, will only share the prefix information received from either other connected nodes or announced by local NDN applications. Sharing information back onto the same interface from which it was learnt will lead to well-known issues like routing-loops [12] or poison reverse [13].

Assuming that the reader has basic knowledge like faces, routes, prefixes and their role in NDN communication [3], we move ahead to look at an example that explains the model of operation in a way that is easy to understand. In this example, a probe is an interest packet sent after a constant interval to ask for prefix information present at the receiver node. More about probing is explained later.

EXAMPLE SCENARIO: Let's say that there are two appropriate Linux devices, A and B, that wish to connect and communicate using this protocol. They will first have to connect to each other at link layer as discussed in Section 3.2. Following are the steps that are the part of the protocol to sync prefix information or in NDN terms, to sync the Forwarding Information Base (FIB):

1) After successful group formation, both devices will register their local-hop probe routes to their own NFDs, by which they will satisfy incoming probe interests meant for them. As an example, let us consider that node A has an IPv6 address <GO-IP>, then A registers /localhop/wifidirect/ <GO-IP> expecting other nodes to use this prefix to probe it.
2) Now add a udp6 route towards each other to let applications send out information to the correct peer using the correct face in the NFD. Route will be added using a command like
nfdc route add /localhop/wifidirect/other-IP udp6://[other-IP]:6363 by both nodes.
3) Both devices then begin the Probe Procedure, where an interest packet is sent out after a fixed interval of time. Details of probe procedure are mentioned in section 4.4.
4) On receiving probe interest, prepare a reply data packet containing all prefix entries from the FIB entry except the ones that have next hop face equal to the face-ID from where the probe was received (to avoid poison reverse as discussed before). Send this data packet back through the same face as probe was received.
5) Repeat all the steps for each of the newly added peers.

4.4 Probe Procedure

4.4.1 Sending probe interests:

a. On connecting with a peer, start the probe procedure and repeat it after every 15 seconds.
b. Construct a probe interest with prefix /localhop/wifidirect/other-IP/probeSeqNo?MustBeFresh=True
c. Send this probe as an interest packet to the peer and wait for the corresponding data packet.
d. Reset the timer for next probe immediately.

The probe basically says to the peer:
"Hey, please tell me all the prefixes that you can serve, so that NDN applications running at my end can look for the data with those prefixes from you if needed."

Note that here we have set the time interval between 2 probes to be 15 seconds, which is same as in the Android protocol as well. This interval can be further reduced to something between 5 to 10 seconds as Linux laptops do not have as much of a power consumption issue as Android smartphone devices do. However, it would be worthy to understand if there is any problem caused by two different time intervals in Android and Linux when they interoperate. Time interval can be easily reconfigured to choose by simply changing a variable in the code.

4.4.2 Receiving probe interests and replying with data packet:

a. On receiving an interest, the peer finds out all the prefixes that it has learnt (own or other) and adds them to a data packet and sends this data packet back through the same face. These prefixes exclude the ones starting with /localhop/wifidirect and /localhost as these prefixes are not useful and not meant for the peer.
b. As discussed earlier, all the prefixes that have been learnt from the peer who probes, are to be excluded from the data packet. This is to avoid Poison Reverse problem.

So, the data packet replies saying:
"These are all the prefixes that I can help you with. Send me interest packets if they begin with one of these prefixes."

4.4.3 Receiving data packet as a reply to probe:

a. Parse all the prefixes sent by the peer in the data packet.
b. Update all the records in the local RIB based on the data prefixes parsed - add a new entry if there is no existing entry for a prefix or update the entry if it already exists.
c. The next hop i.e. the face ID of these prefixes is the face from which this packet was received (face towards the peer itself), as interests for these prefixes need to be sent to this very peer.
d. All the added or updated entries have an expiration period of 100 seconds. This ensures that if this NFD does not hear back from the peer for more than 100 seconds, it should mark all the entries towards it as unreachable.
e. Expect a fresh view of the prefix availability in next 15 seconds.

4.5 Packet formats

4.5.1 Probe Interest:

Probe Interests should always have the form:
/localhop/wifidirect/<OtherIP>/probe?MustBeFresh=True

Recall all IP addresses here are IPv6 link-local addresses (format is fe80::/10). The MustBeFresh flag is set to True so that nodes will always return their most up to date Data prefixes.

4.5.2 Probe Data:

Probe Data packets should be of the form:

Prefix1\n
Prefix2\n
.
.
PrefixN\n

If we wanted to add new sections to the response, we would need to start the packet with the number of prefix entries N, so that we immediately know where the beginning and ending of the Data prefixes were. There is no N in the current format because adding a new section does not seem necessary.

5. CHALLENGES/FUTURE WORK

There are two major challenges that we faced during this project. First is the smooth merging of connectivity and protocol parts of the code such that they run as an easy to use utility for the end user. Second is facilitating Linux to Android intercommunication. The latter challenge falls out of the original scope of the project, which is to make Linux to Linux communication possible, but is still an important issue as resolving it will make Wi-Fi Direct communication more flexible and useful for common users and will open doors for a variety of NDN apps. Both the challenges are a part of the future work of this project.

5.1 Merging connectivity with protocol for ease of access

The final and ideal aim of the project is to make it as simple as possible for the user to use Wi-Fi Direct and communicate with peers using NDN. However, no Linux distribution has a facility to connect to Wi-Fi Direct peers using the NetworkManager GUI like it is possible to connect to Wi-Fi networks or even pair Bluetooth devices. Also, adding the Wi-Fi Direct connection facility to the NetworkManager GUI is a non-trivial task to achieve on Linux distros.

Another way to achieve the goal of simplistic connection is to add this protocol bundled with the connectivity part to NDN Control Center. Currently, the project has as a command line interface that lets the user select the network interface to use for the Wi-Fi Direct connection, then choose the peer to connect to and finally choose between being a group owner or a client. We look forward to tackling this challenge after solving the next challenge as the latter one would result in more overall impact.

5.2 Linux to Android interoperability

As discussed in Section 4.2, the purpose of using IPv6 link local addresses in this protocol was that Linux to Android communication will be possible using this protocol. However, each node needs to create a udp6 face (as seen in Section 4.3) towards the other inside NFD, which is not possible right now due to an NFD bug.

The NFD FaceUri parser should recognize link-local IPv6 addresses in the form of udp6://[fe80::d6ae:52ff:fecd:260c%25wlan0]:6363. Here, %wlan0 part is required to specify the NIC being used and it is written as %25 to escape the percentage sign, according to the rules in the RFC3986 [14]. TCP and UDP factories should allow creating faces using link-local IPv6 address with explicitly specified NIC name as user might have more than one NIC that he wishes to choose, and not specifying the NIC leaves NFD clueless as to which interface to use for face creation. Currently, the parser in NFD does not parse this format correctly and results in an error. A bug [15] has been reported for this issue to the NDN Redmine project issue tracking system. We plan to submit code for this issue very soon, to get this interoperability working. Currently, we are simply using Ethernet faces created by default in the NFD for each of the interfaces, and this works perfectly well for Linux to Linux scenario.

6. IMPLEMENTATION

The complete protocol is implemented as a C++ program that requires NFD and NDN-CXX libraries installed on the system to run. The connectivity part needs wpa_supplicant to be installed and running in the background and our code contains tweaked parts of the wpa_cli program that talks to the wpa_supplicant daemon and fires the necessary p2p commands. No other special installations is necessary. One might need to configure the built-in NetworkManager to stop managing the Wi-Fi Direct interface we plan to use as internally even NetworkManager software commonly used by many Linux distros uses wpa_supplicant and two instances cannot manage the interface together. This automation is a part

of the challenges we discussed in Section 5.1 but is not a crucial roadblock.

The current implementation is much simpler and efficient than the pre-existing Android protocol implementation, mainly because of the use of IPv6 addressing that obviates the need to have a DHCP server and waiting/exchanging initial packets to get knowledge of peer's IP address and introduces interoperability between Linux and Android.

7. CONCLUSION

The paper has presented the work done in this project that aims to make Linux laptops using NDN applications capable of communicating with each other over Wi-Fi Direct. This implementation is the first C++ implementation of the protocol and has a design that will make it easy to extend the same protocol over different platforms without the need of making any major changes. However, the protocol still has plenty of room to be improved and extended, especially on the Linux distribution integration side of it. We hope that this project will be greatly useful to promote the use of NDN over the edge and attract more developers to build apps that can take advantage of this infrastructure-less mode for flexible connectivity and peculiar use-cases.

8. ACKNOWLEDGEMENTS

Many thanks to Alex Afanasyev for discussing, guiding and providing valuable feedback at every step of this project. Thank you to Lixia Zhang for advising and motivating me whenever I needed help for this project. Special thanks to Wesley Minner, Rachel Chu and Da Teng for giving me their valuable time and suggestions during the project.

9. REFERENCES

[1] http://www.wi-fi.org/discover-wi-fi/wi-fi-direct

[2] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," ACM SIGCOMM Computer Communication Review, July 2014.

[3] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Huang, J. P. Abraham, S. DiBenedetto, C. Fan, C. Papadopoulos, D. Pesavento, G. Grassi, G. Pau, H. Zhang, T. Song, H. Yuan, H. B. Abraham, P. Crowley, S. O. Amin, V. Lehman, , and L. Wang, "NFD Developers Guide," NDN Project, Tech. Rep. NDN- 0021, Revision 5, oct 2015.

[4] https://redmine.named-data.net/projects/nfd-android/wiki/NDN_Over_WiFi_Direct_Protocol_Specification

[5] NDN Project Team, "NDN Client Library for C++ and C," Available at https://github.com/named-data/ndn-cpp.

[6] https://play.google.com/store/apps/details?id=net.named_data.nfd&hl=en

[7] https://w1.fi/wpa_supplicant/

[8] https://www.bluetooth.com/specifications/bluetooth-core-specification

[9] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <http://www.rfc-editor.org/info/rfc4291>

[10] Behringer, M. and E. Vyncke, "Using Only Link-Local Addressing inside an IPv6 Network", RFC 7404, DOI 10.17487/RFC7404, November 2014, <http://www.rfc-editor.org/info/rfc7404>

[11] https://technet.microsoft.com/library/Cc940478

[12] https://en.wikipedia.org/wiki/Routing_loop_problem

[13] Hedrick, C., "Routing Information Protocol", RFC 1058,DOI 10.17487/RFC1058, June 1988, <http://www.rfc-editor.org/info/rfc1058>

[14] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>

[15] https://www.google.com/url?q=https%3A%2F%2Fredmine.named-data.net%2Fissues%2F1428&sa=D&sntz=1&usg=AFQjCNF5Fus0YCee1e0Utm25GO_l1aTSiw