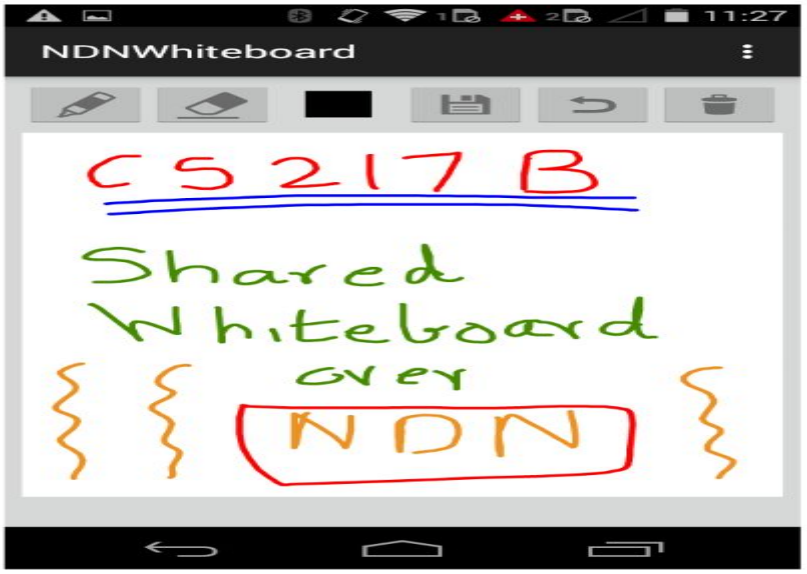
Robert Bielas, Michał Ćwiertnia, Krzysiek Burkat

Część pierwsza - NDN, architektura łatki whiteboardu

NDNWhiteboard to przykładowa aplikacja kliencka wykorzystująca koncepcję nazwanych sieci danych, którą można pobrać z GoogleStore'a lub zbudować ze źródeł projektu z repozytorium: https://github.com/named-data-mobile/apps-NDN-Whiteboard. W zamyśle, korzystając z NDNWhiteboard, użytkownicy mogą przyłączyć się do jednego z istniejących pokoi (reprezentowanych przez ndn-owe prefiksy) lub utworzyć nowy pokój i korzystać ze współdzielonego zasobu tablicy.



Jedyną wadą tej aplikacji jest fakt, że użytkownicy nie widzą zmian na tablicy poczynionych przez innych użytkowników w pokoju. Poniżej przedstawiamy łatkę która naprawia ten problem.

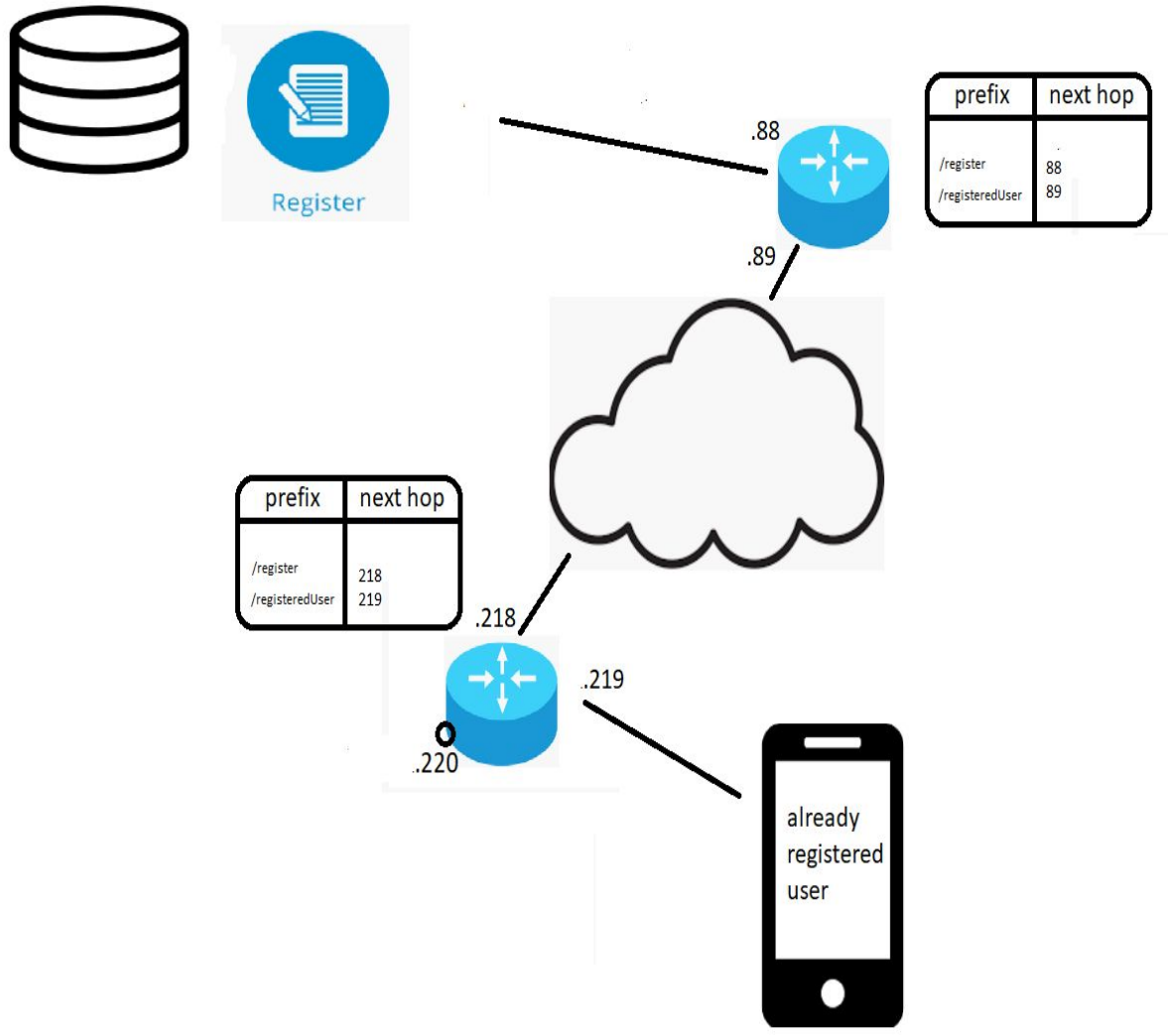
Na architekturę rozwiązania składa się kilka elementów:

1. sieć routerów ndn z demonami forwardującymi na każdym routerze; router składa się z tzw. face'ów (ndn-owych uogólnień interfejsów), do których są podpięte albo inne routery, albo końcowe urządzenia z odpowiednimi aplikacjami, w szczególności te wymienione dalej w punktach 2 i 3.
2. prosta usługa rejestracji użytkowników (register) - nie mylić z rejestracją prefiksów!
3. telefony z Androidem, z zainstalowaną kliencką aplikacją i możliwością komunikacji z (dopuszczalnie zdalnym) face'em demona forwardującego dowolnego routera będącego częścią sieci.

Zanim przejdziemy do szczegółów rozwiązania, ważne jest zarysowanie idei komunikacji w ndn-ach: w tym modelu istnieją dwie role - producenci i konsumenci danych. Konsumenci wyrażają **zainteresowanie** (Interest) danymi produkowanymi przez producentów identyfikowanych przez określone prefiksy. W kierunku konsument->producent domyślnie nie można przesłać żadnych danych, co najwyżej konsument jest w stanie "zaszyć" parametry zainteresowania w samym prefiksie. Dopiero producent, gdy dociera do niego zainteresowanie, tworzy i wysyła odpowiedni pakiet danych w kierunku, z którego przyszło

zainteresowanie. Żeby do producenta mogły docierać zainteresowania, musi zarejestrować swój prefiks w sieci, co czyni poprzez zalanie jej pakietem rejestracji swojego prefiksu.
API NDN-owe opiera się na systemie callbacków. System rejestruje kilka ich rodzajów w zależności od operacji, którą wywołujemy na obiekcie typu Face. Gdy urządzenie przyjmuje rolę producenta i rejestruje swój prefiks w sieci, musi równocześnie zarejestrować fragment kodu produkującego dane w momencie, kiedy nadchodzi zainteresowanie nimi (**onInterest**) - ten callback jest jedynym punktem kontaktu producenta z konsumentami; po zarejestrowaniu tego callbacku, API automatycznie inicjuje operację floodingu sieci pakietem rejestracji prefiksu nowego producenta.
Z kolei gdy urządzenie przyjmuje rolę konsumenta danych i wyraża swoje zainteresowanie nimi (**expressInterest**), musi ono zarejestrować callback **onData**, który jest wywoływany w momencie nadejścia pakietu z danymi wysłanego przez odpowiedniego producenta.

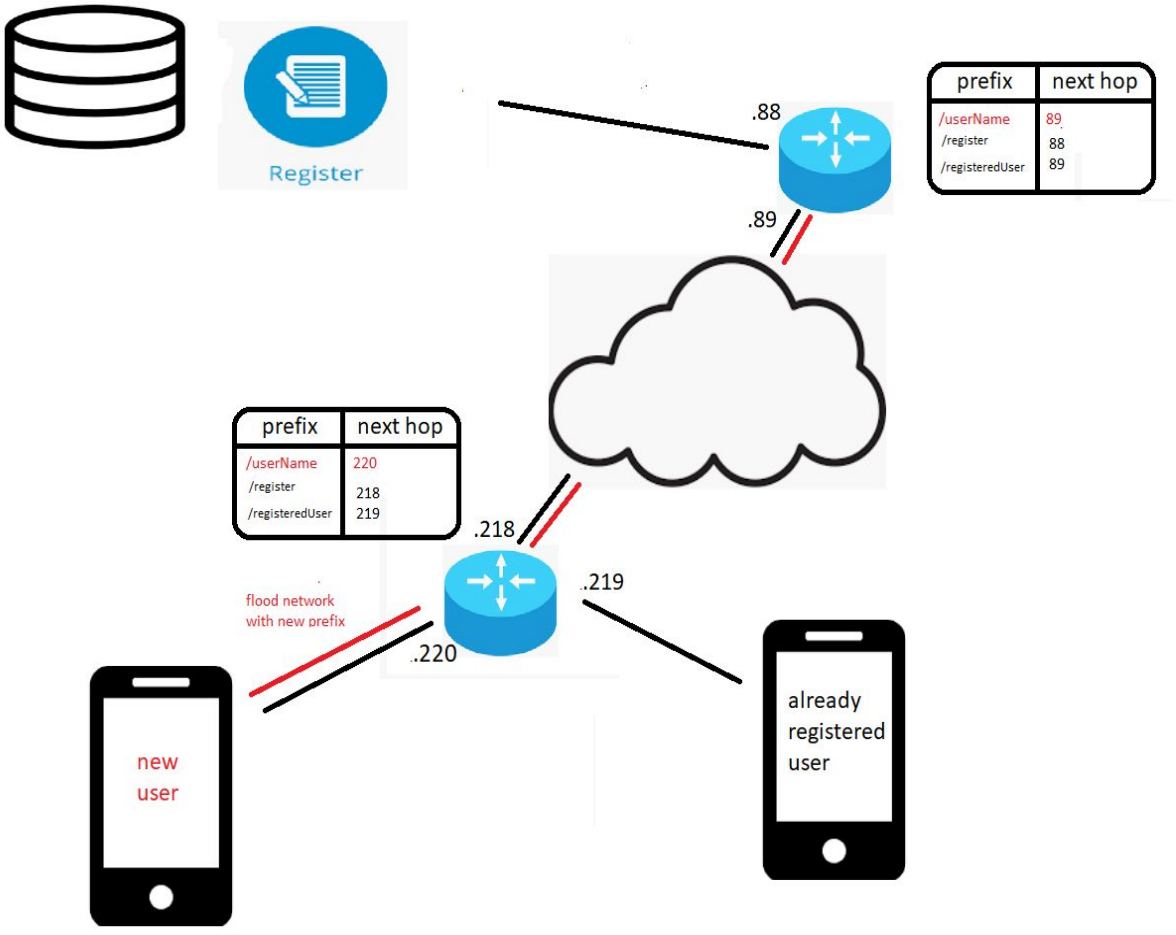
Przykładowy schemat architektury po niezbędnej konfiguracji może wyglądać np. tak jak poniżej:



Czyli istnieją:

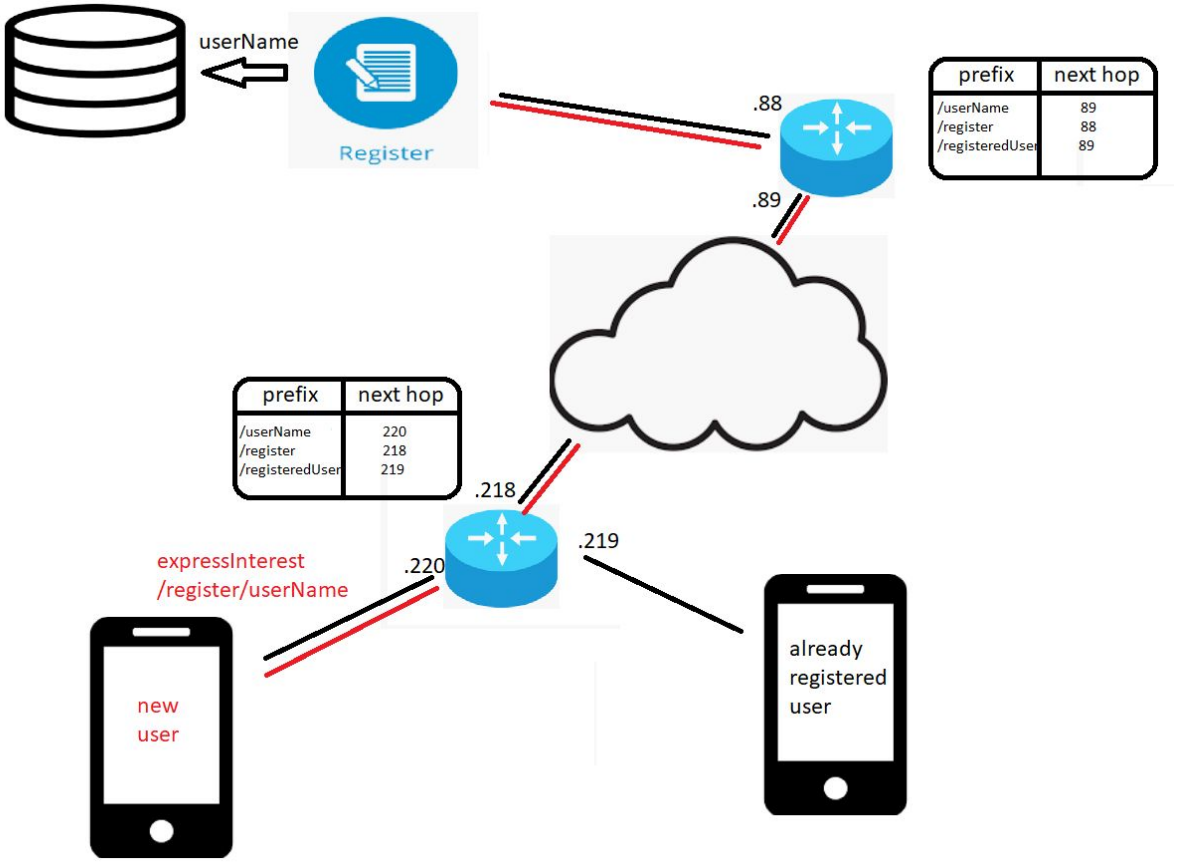
1. uruchomiony proces usługi rejestracji, zaś każdy router posiada trasę do niego zapisaną w postaci pary (/register, nextHop), gdzie /register to prefiks usługi, a nextHop to odpowiedni numer face'a, który przybliża wędrówkę do usługi o jeden skok.
2. jeden (lub więcej) zarejestrowanych klientów, posiadający swój własny unikalny prefiks zarejestrowany u wszystkich routerów w sieci

Załóżmy, że na nowym telefonie ("new user") zainstalowaliśmy aplikację kliencką i chcemy się przyłączyć do pokoju. Po pierwsze, taki klient musi powiadomić sieć, że jest nowym producentem danych (bo będziemy nim produkować kreski na tablicy). Jak wspomniano powyżej, producent ndn-owy robi to poprzez zalewanie sieci pakietem rejestracji swojego prefiksu.



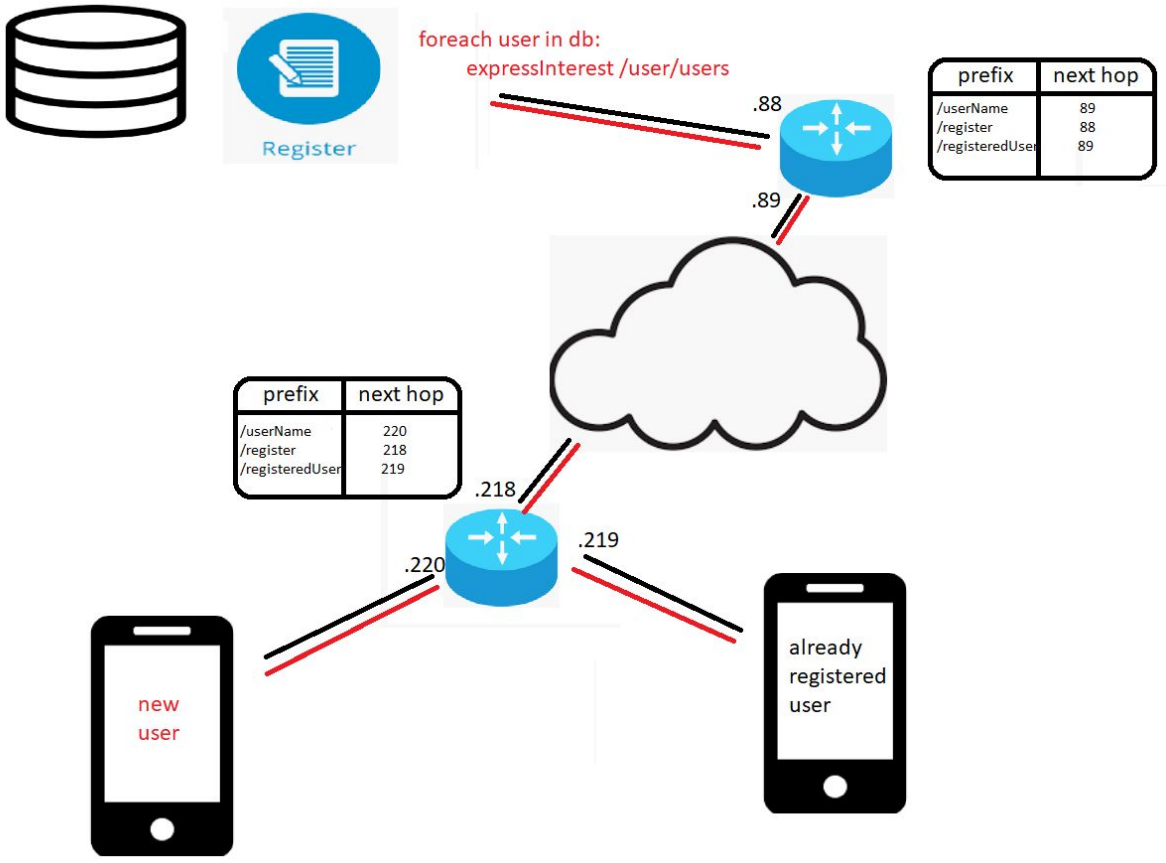
W efekcie każdy router sieci persystuje informację w tablicy forwardującej, mówiącą na jaki face należy wysyłać kierowane do tego klienta Interesty.

Następnym krokiem algorytmu klienta jest powiadomienie usługi rejestrującej (register), że jest nowym użytkownikiem w pokoju. Robi to poprzez wyrażenie zainteresowania prefiksem /register/userName, gdzie pod userName kryje się dowolna nazwa wprowadzona przez użytkownika. W momencie gdy zainteresowanie dojdzie do usługi, ta zapisuje nowego użytkownika w swojej lokalnej bazie danych:

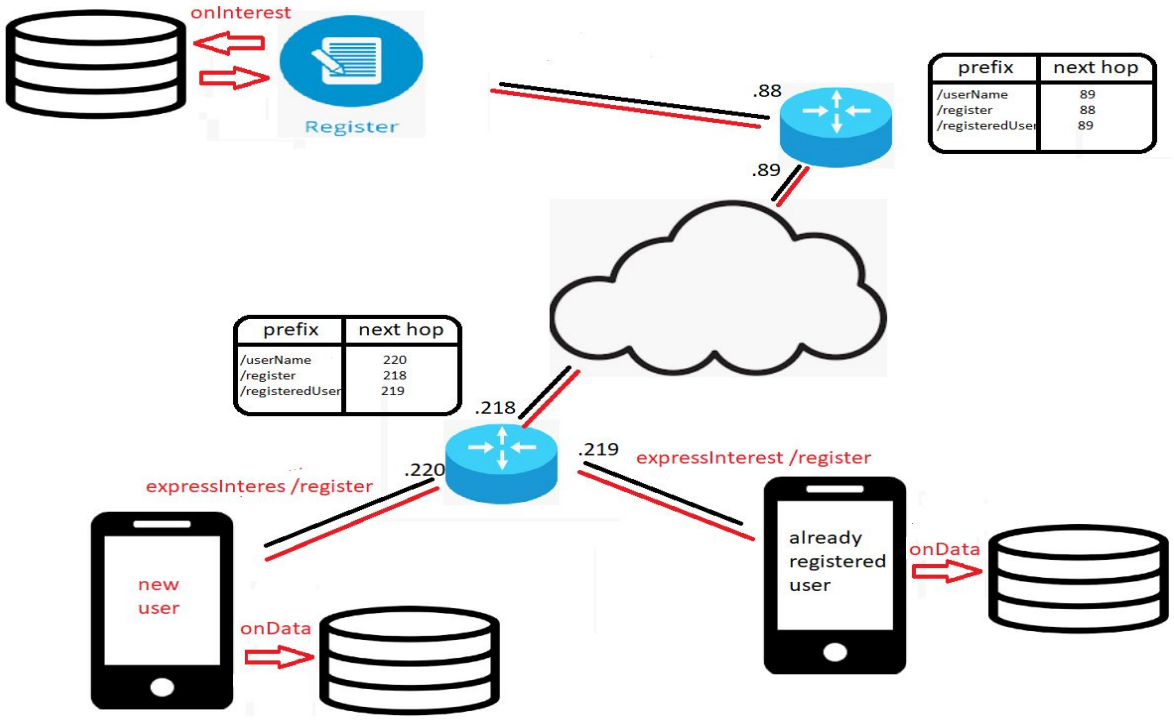


Tutaj następują dwie rzeczy:

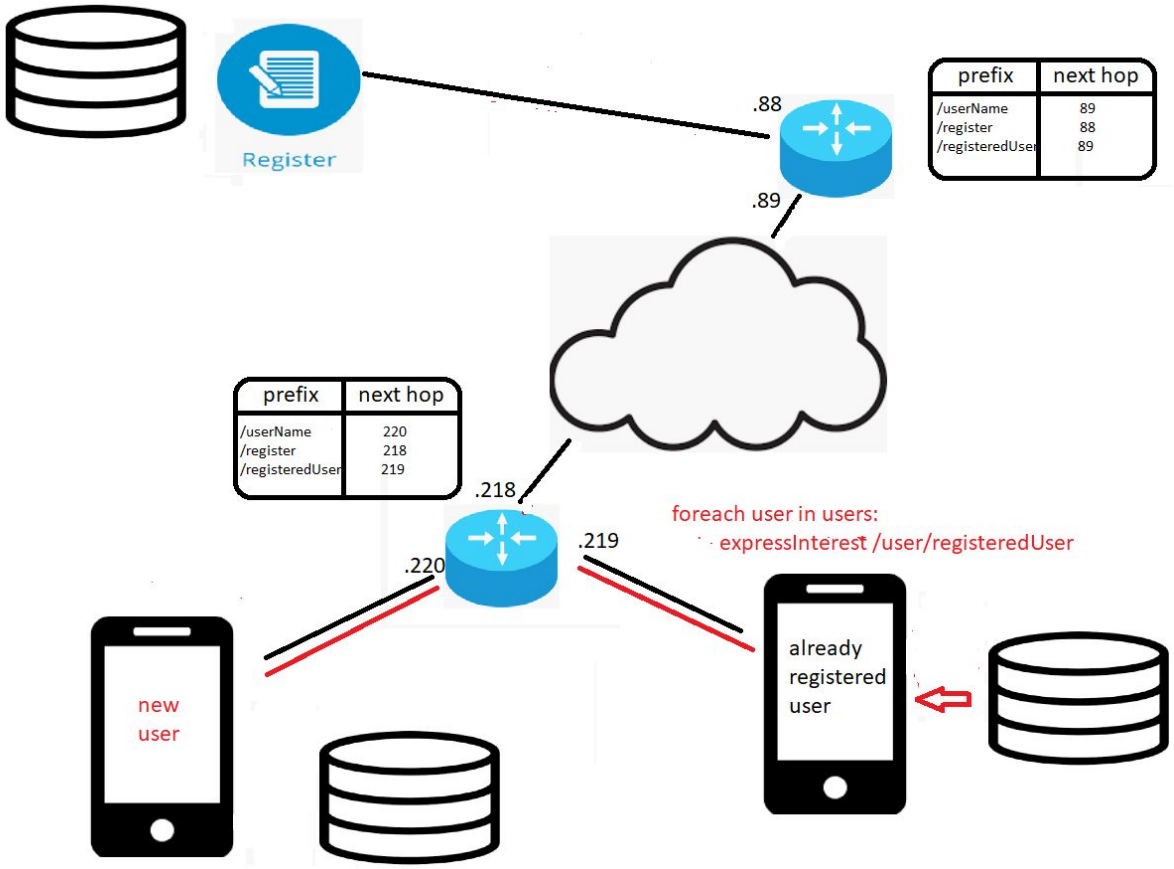
1. Ze względu na to, że użytkownik przesłał zainteresowanie, usługa przesyła mu pakiet z tekstem “ok”, co klient loguje w na standardowym wyjściu
2. dla każdego obecnego w lokalnej bazie danych użytkownika (w tym nowo zarejestrowanego), usługa “pinguje” go wyrażając zainteresowanie prefiksem /user/users, gdzie user to jedna z nazw użytkowników:



Każde z urządzeń orientuje się, że zainteresowanie prefiksem jest parametryzowane argumentem “users” (ostatni człon prefiksa). Jest to znak, że muszą one pobrać całą aktualną kolekcję użytkowników. Robią to poprzez wyrażenie zainteresowania prefiksem /register i obsługując dane zawarte w pakiecie w callbacku onData:

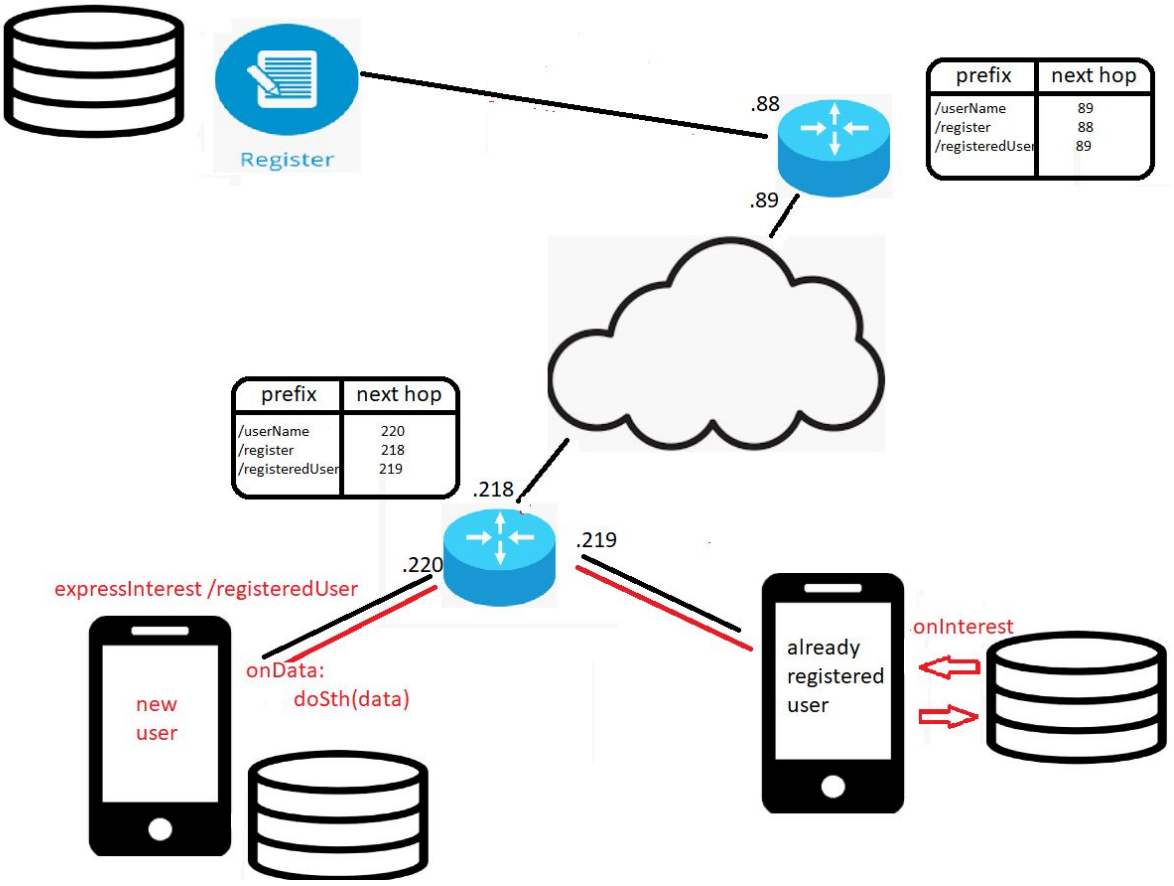


Po wyjściu z callbacku onData, urządzenia są gotowe do wymiany informacji pomiędzy sobą. Powiedzmy, że na urządzeniu registeredUser (tym po prawej) użytkownik narysował kreskę. Punkty stanowiące początek i koniec kreski są zapisywane w pamięci urządzenia. Następnie urządzenie wykorzystuje dopiero co pobraną kolekcję użytkowników do powiadomienia ich pingiem, że zaktualizowano historię kresek tego urządzenia; w tym celu, dla każdej nazwy użytkownika z pobranego zbioru, parametryzuje swoją nazwą prefiks nazwy tego użytkownika, by wiedział on, od kogo zaciągnąć historię. W efekcie, prefiks utworzony na tym etapie (dla urządzenia registeredUser) ma postać /user/registeredUser, gdzie user to nazwa jednego z pozostałych zarejestrowanych urządzeń (jeśli podczas iteracji user==registeredUser, oczywiście pomijamy operację pobrania danych):



Każde urządzenie które dostanie tego pinga, orientuje się o obecności ostatniego członu i zdaje sobie sprawę, że oznacza to konieczność pobrania historii kresek urządzenia identyfikowanego przez ten parametr. Następuje ostatnia faza komunikacji, gdzie pingowane urządzenie:

1. Wysyła potwierdzający tekst “ok”
2. Wyraża zainteresowania kierowane do producenta kreski (którym w tym konkretnym przypadku jest registeredUser) poprzez expressInterest
3. Każda z zapamiętanych w pamięci urządzenia registeredUser kresek jest serializowana jsonem w callbacku onInterest i wysyłana pakietem do zainteresowanego urządzenia i obsługiwana w nim w callbacku onData:



W ten sposób dokonuje się synchronizacji historii narysowanych kresek na urządzeniach zarejestrowanych w usłudze register.
Opisane mechanizmy pokrywają się z tym, co dzieje się w kodzie z dokładnością do:

1. nazw prefiksów
2. topologii sieci ndn

W tym ostatnim przypadku zamiast z całej sieci, korzystamy tylko z jednego routera (w postaci maszyny wirtualnej) z zainstalowanym demonem forwardującym, pod który są podpięte wszystkie urządzenia wraz z usługą rejestracji. Każde z urządzeń jest podpięte pod inny face tego routera, zaś to “podpięcie” jest realizowane poprzez zwykłe sockety UDP (tzn. oprócz mapowania prefix->face, w demonie są też obecne mapowania face->socket_deskryptor). Od momentu podpięcia reszta routingu i wymiana informacji odbywa się w całości poprzez wykorzystanie mechanizmów NDN.

Po zaimplementowaniu powyższej łatki, NDNWhiteboard jest w pełni funkcjonalny.

Część druga - NDN, instrukcja instalacji, konfiguracji i obsługi

Archiwum z projektem zawiera dwa spakowane foldery: NDN-Whiteboard i jndn.
Pierwszy folder zawiera zmodyfikowane źródła aplikacji mobilnej, które można otworzyć w Android Studio i zbudować z nich podpisany certyfikatem autora plik z rozszerzeniem .apk.
Drugi folder zawiera repozytorium źródeł klientów NDN napisanych w javie wraz z przykładami aplikacji desktopowych korzystających z nazwanych sieci. Wśród autorskich przykładów znajduje się w nich także implementacja opisanej w części pierwszej usługi rejestrującej nowe urządzenia. Znajduje się ona na ścieżce (startując z katalogu zawierającego folder jndn): jndn\examples\src\net\named_data\jndn\tests\UsersListener.java

Żeby oba artefakty mogły zadziałać, niezbędne jest zainstalowanie demona forwardującego NDN. Do tego wymagany jest Linuxo-podobny system operacyjny. Testowanie projektu odbyło się na Fedorze z numerem wersji jądra 18, nic nie stoi jednak na przeszkodzie żeby zainstalować go także na innych systemach jak np. Ubuntu. Nie poleca się instalować demona na Windowsie, gdyż autorzy popełnili mnóstwo błędów w konfiguracji instalatora na tej platformie.
Do skompilowania demona forwardującego NDN (którego od tego momentu nazywamy NFD) potrzebne są źródła biblioteki ndn-cxx oraz źródła samego NFD. Najpierw trzeba zainstalować tą pierwszą, zgodnie z instrukcją znajdującą się pod adresem:
https://github.com/named-data/ndn-cxx/blob/master/docs/INSTALL.rst
Kiedy projekt zbuduje się poprawnie, należy zbudować samego NFD, którego instrukcja instalacji znajduje się pod adresem:
https://github.com/named-data/NFD/blob/master/docs/INSTALL.rst.

W momencie poprawnego zbudowania należy dokonać kilku rzeczy.
Po pierwsze, jeśli odpaliliśmy demona zgodnie z instrukcją instalacji NFD, ubijamy go komendą

nfd-stop

gdyż musimy zmodyfikować domyślną konfigurację NFD przed jego startem (jeśli nie wystartowaliślmy demona oczywiście nie musimy odpalać tej komendy).
Zakładając że skopiowaliśmy plik `nfd.conf.sample do pliku nfd.conf` w folderze `/usr/local/etc/ndn`
to otwieramy vima na tej ścieżce z uprawnieniami roota i modyfikujemy zawartość pliku nfd.conf tak, aby usunąć średniki komentarzy przed linijkami 409-415 (sekcja localhop_security umożliwiająca zdalnym użytkownikom rejestrować w tablicy tego demona swoje prefiksy, bez tego nic nie chce działać). Dopiero po tym jak zapiszemy tak zmodyfikowaną konfigurację, możemy ponownie odpalić NFD komendą:

nfd-start

Od teraz możemy komunikować się z demonem poprzez utilsowy program nfdc który został zainstalowany wraz z dystrybucją NFD. Na przykład możemy wylistować wszystkie face'y komendą:

nfdc face

a wszystkie trasy w tablicy routingu komendą:

nfdc route

Pełnię opcji możemy poznać konsultując się z pomocą narzędzia.

Mając zainstalowanego NFD, czasami ważne jest ubicie firewalla, żeby ten nie blokował nadchodzących połączeń. Na Fedorze robimy to komendą

sudo systemctl stop firewalld

Dopiero teraz możemy uruchamiać resztę programów w projekcie. Najpierw stawiamy usługę rejestracji (można na dowolnym komputerze, my postawiliśmy na Fedorze z odpalonym NFD). Modyfikujemy kod źródłowy pliku UsersListener.java w odpowiednim podfolderze folderu examples jndn, zastępując adres IP w linijce 99 adresem hosta, na którym działa NFD (na fedorze możemy go poznać komendą **ifconfig**). Potem przechodzimy do folderu jndn/examples i z tego poziomu odpalamy w konsoli komendę:

mvn -q test -DclassName=UsersListener

Oczywiście musimy mieć przedtem zainstalowanego mavena. Czekamy aż usługa rejestracji jest gotowa, co jest sygnalizowane pojawieniem się w konsoli tekstu: **"Working at /users".**

Na samym końcu możemy skompilować i uruchomić na naszych telefonach aplikację tablicy. Najważniejsza zmiana (plus parę mniejszych) w stosunku do źródeł na repozytorium to klasa:
edu.ucla.cs.ndnwhiteboard.tasks.WhiteboardListener, która musiała zostać w całości dopisana przez nas, tak aby urządzenia mogły się poprawnie komunikować ze sobą, i która zawiera zaimplementowany protokół z opisu w części pierwszej.

W Android Studio ładujemy rozpakowany folder NDN-Whiteboard i budujemy zawarty w nim projekt, ewentualnie instalując wszystko to co nam podpowiada środowisko. Po zainstalowaniu i uruchomieniu aplikacji jesteśmy witani ekranem logowania. Ważne jest, aby w tym ekranie podać przede wszystkim poprawny adres IP hosta z odpalonym NFD (wygodnie jeśli wszystkie urządzenia znajdują się w tej samej sieci prywatnej IP). Dobrze jest także, aby urządzenia miały **różne** nazwy użytkowników oraz **te same** identyfikatory współdzielonej tablicy.

Po zastosowaniu tych wszystkich kroków, aplikacja powinna działać jak należy.