České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky
Študijný program: Otevřená informatika (magisterský)
Obor: Počítačové inženýrství

Martin Tuka

Podpora Wi-Fi sítí ve frameworku pro rezervace zdrojů

Diplomový projekt

Vedúci diplomového projektu: Ing. Michal Sojka, Ph.D.
Január 2012

Zadanie

Czech Technical University in Prague

FACULTY OF ELECTRICAL ENGINEERING

Support of Wi-Fi networks in resource reservation framework

Degree Course: Open informatics

Author: Martin Tuka

Supervisor: Ing. Michal Sojka, Ph.D.

1/2012

This thesis deals with resource reservation framework FRSH/FORB and its support for wifi networks. It extends it to possibility of adaptation for changes of transmission bit-rate during communication between applications using framework. Theoretical part of the thesis describes the way framework is working with resources, and current functionality of wifi networks. Also describes another technologies used in practical part. Practical part describes possibilities of framework extension, so it will be able to work with more different bit-rates of wifi network. Here are shown individual steps how to achieve this task – gathering wifi conditions for each station in network, evaluation of conditions and adaptation of application for this functionality.

České vysoké učení technické v Praze

FAKULTA ELEKTROTECHNICKÁ

Podpora Wi-Fi sítí ve frameworku pro rezervace zdrojů

Študijný program: Otevřená informatika

Autor: Martin Tuka

Vedúci bakalárskeho projektu: Ing. Michal Sojka, Ph.D.

1/2012

Táto práca pojednáva o frameworku FRSH/FORB na rezervovanie zdrojov a jeho podpore pre bezdrôtové siete. Rozširuje ho o možnosť adaptácie na zmeny prenosovej rýchlosti medzi aplikáciami využívajúcimi framework. Teoretická časť opisuje spôsob fungovania frameworku, postup vyjednávania kontraktov a súčasnú prácu s wifi sieťami. Taktiež obsahuje popis ďalších technológií použitých v praktickej časti. Praktická časť sa zaoberá možnosťami rozšírenia frameworku tak, aby bol schopný pracovať s viacerými rôznymi rýchlosťami bezdrôtových sietí. Opisuje jednotlivé kroky nevyhnutné na splnenie tejto úlohy – zisťovanie aktuálnej rýchlosti v sieti, zhromažďovanie a vyhodnocovanie dát a adaptácia aplikácií na túto funkcionalitu.

Čestné prehlásenie

Prehlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedených zdrojov.

V Prahe dňa 3. 1. 2012

.....

podpis

Poďakovanie

Na tomto mieste sa chcem poďakovať vedúcemu diplomovej práce Ing. Michalovi Sojkovi, Ph.D., za jeho cenné rady, ochotu a trpezlivosť.

Obsah

Úvod 8
1 Analýza problematiky 9
1.1 FRSH-FORB framework 9
1.1.1 Rezervovanie zdrojov založené na kontraktoch 9
1.1.2 Podpora Wi-Fi sietí vo framework-u pre rezerváciu zdrojov 17
1.2 Ďalšie použité technológie 17
1.2.1 Subsystémy Linuxu mac80211 a nl80211 17
1.2.2 Netlink komunikácia a knižnica libnl 18
1.2.3 mac80211 19
1.2.4 nl80211 20
1.2.5 WvTest 20
1.2.6 ULUT 21
1.2.7 COBRA IDL 22
2 Návrh riešenia a implementácia 24
2.1 Návrh riešenia 24
2.1.1 Zisťovanie stavu aktuálnej prenosovej rýchlosti 24
2.1.2 Zhromažďovanie informácií o stave siete 25
2.1.3 Rozšírenie vyhodnocovacieho testu 25
2.1.4 Prispôsobovanie aplikácií 25
2.2 Implementácia 26
2.2.1 Zisťovanie stavu aktuálnej prenosovej rýchlosti 26
2.2.2 Zhromažďovanie informácií o stave siete 28
2.2.3 Rozšírenie vyhodnocovacieho testu 29
2.2.4 Prispôsobovanie aplikácií na zmeny prenosovej rýchlosti 30
2.3 Testovanie 30
2.3.1 Test kontroly zoznamu staníc v manažérovi zdrojov 31
2.3.2 Test správnosti funkcie vyjednávajúcej zdroje 32
2.4 Demonštračná aplikácia 33
3 Zhodnotenie 38
4 Technická dokumentácia 39
4.1 Funkcia vyhodnocovacieho testu 39
4.2 Zhromažďovanie informácií o stave siete 42
4.3 Zisťovanie stavu aktuálnej prenosovej rýchlosti 43
Použité zdroje 48

Úvod

Dnes nás už všade obklopujú informačné technológie, bez ktorých si život vieme sotva ťažko predstaviť. Ich použitie je čoraz rozmanitejšie a nachádzame ich často aj na miestach, kde by sme ich sotva čakali. Veľká časť sa používa vo vstavaných systémoch, z ktorých mnohé sú operačnými systémami reálneho času, čo v praxi znamená, že musia spĺňať podmienky zamerané na splnenie časových podmienok vykonania úlohy.

Tieto systémy sa vyvíjajú v čoraz väčšom množstve a prostredia, do ktorých sa nasadzujú, sú čoraz rozmanitejšie. Navyše, s rastom trhu rastú aj požiadavky na vývojárov. Hotový produkt je preto často zhotovený z viacerých nezávisle vyvinutých komponentov, ktoré sú znova použiteľné vo viacerých projektoch. Tento prístup značne urýchľuje vývoj produktov.

Spĺňanie časových podmienok pri vývoji vôbec nie je jednoduché. Pokiaľ je systém jednoduchý, väčšinou nie je problém nájsť nejaký známy algoritmus a na problém ho priamo aplikovať. Množstvo systémov ale nie je jednoduchých, sú komplikované a distribuované, bežia v rozličných prostrediach. Taktiež podmienky, ktoré musia aplikačne spĺňať, sa môžu v čase meniť, čo prináša veľa komplikácií pri zaručení správnosti fungovania systému.

Jednou z možností, ako zabezpečiť splnenie podmienok v reálnych aplikáciách, je použitie frameworku FRSH/FORB. Tento je možné použiť na rezervovanie zdrojov, a o ktorého rozšírení v oblasti Wi-Fi sietí pojednáva táto diplomová práca.

Štruktúra práce je nasledujúca: V kapitole 1 sa zoznámime s už spomenutým frameworkom, so subsystémami Linuxu mac80211 a nl80211, a ďalšími použitými technológiami. Vo zvyšných kapitolách opíšeme navrhnuté riešenie na problematiku, ako je opísaná v zadaní.

1 Analýza problematiky

1.1 FRSH-FORB framework

FRSH/FORB je framework na rezerváciu zdrojov založený na kontraktoch pre distribuované aplikácie pracujúce v reálnom čase. Jeho cieľom je uľahčenie vývoja aplikácií tým, že splní ich požiadavky (ako sú napríklad deadline) a zabezpečí koordináciu zdrojov ako CPU, siete a podobne.[1]

Základný postup fungovania frameworku je nasledovný: Aplikácia špecifikuje svoje požiadavky na zdroje (CPU, pamäť, sieť atď.), a pošle ich frameworku. Framework sa postará o celý manažment prerozdelenia. V prípade dostatku zdrojov garantuje ich pridelenie aplikácii počas celého behu programu a izoluje ich od zvyšku systému. V prípade, že o zdroje požiada ďalšia aplikácia, môže si "vybrať" už len zo zostávajúcich zdrojov. Pokiaľ aplikácia žiada nad rámec možností, ktoré je hardvér schopný poskytnúť, framework nepovolí aplikácii beh. Službe, ktorá je poskytovaná aplikácii frameworkom, hovoríme vyjedávanie kontraktu. (angl. contract negotiation) Aplikácia požiada o zdroje, a framework odpovie na základe informácií, ktoré o zdrojoch zhromažďuje. Po úspešnom vyjednaní kontraktu sa rezervuje priestor pre aplikáciu. Vo frameworku sa vytvorí virtuálny zdroj (angl. virtual resource), ktorý reprezentuje zlomok zo zdrojov, ktoré aplikácia bude po dobu behu používať.

V prípade, že viacero aplikácií naraz bude žiadať o absolútne využitie rovnakého zdroja, môže dôjsť k prerezervovaniu zdrojov, čo znamená, že v čase vyjednávania aplikácií bude zdrojov dostatok, ale počas behu programov už nie. Aby sa tomuto zabránilo, framework používa vyhodnocovacie testy (angl. admission test), ktoré sú založené na analýze plánovateľnosti "state-of-art". (angl. state-of-the-art schedulability analysis)

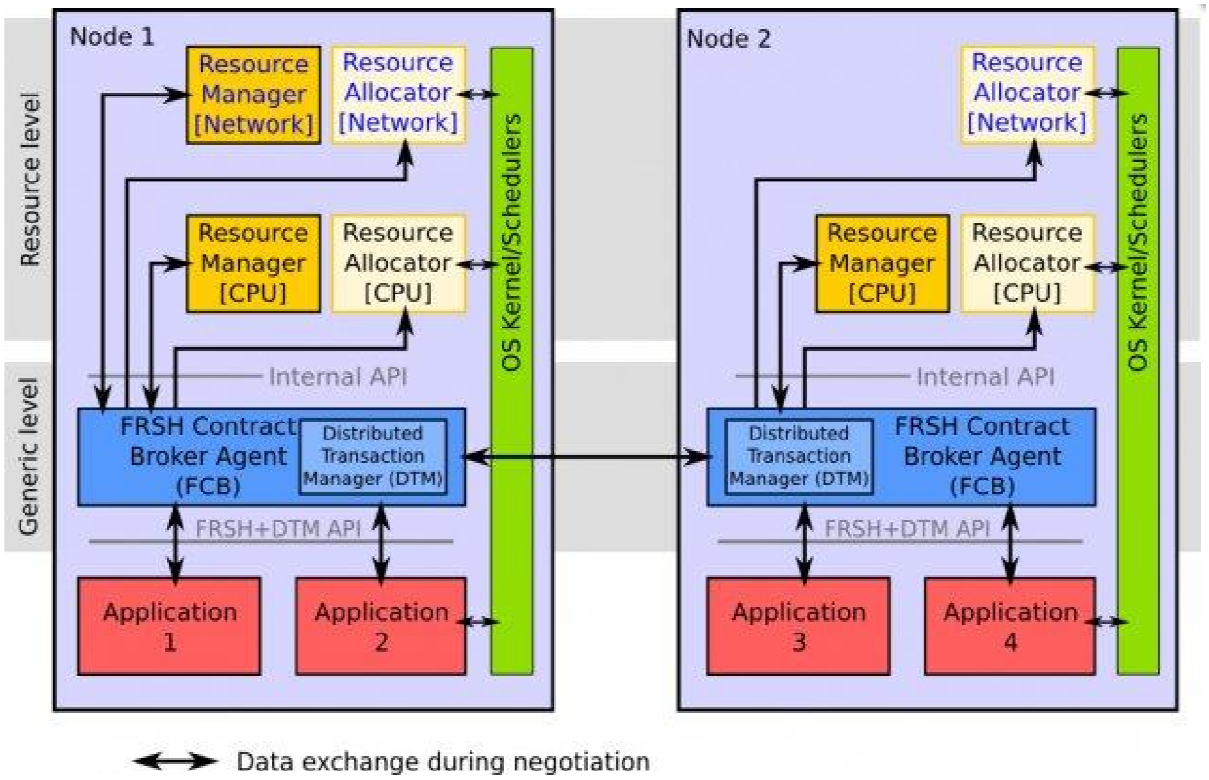
1.1.1 Rezervovanie zdrojov založené na kontraktoch

V tejto časti stručne opíšeme architektúru frameworku, základný algoritmus rezervovania zdrojov a vyjednávania aplikácie s frameworkom.

<u>Architektúra</u>

Vnútorná architektúra môže byť rozdelená do troch úrovní: [2]

- vonkajšie aplikačné rozhranie (API, angl. Application Programming Interface): Rozhranie medzi aplikáciou a frameworkom. Hlavná služba poskytovaná touto úrovňou je vyjednávanie kontraktov. Aplikácia svoje požiadavky vloží vo forme kontraktu (dátová štruktúra), a vloží ho cez API do frameworku. Pokiaľ je vyjednávanie úspešné, framework vytvorí virtuálny zdroj a priradí ho aplikácii. Aplikácia potom naviaže svoje požiadavky (reprezentované entitami) na tento virtuálny zdroj pomocou funkcií API.
- úroveň nezávislá na zdrojoch (angl. resource-independent level): Táto úroveň je reprezentovaná sprostredkovateľom kontraktov (angl. contract broker) a slúži na rozdistribuovanie voľných kapacít jednotlivých zdrojov, globálnu optimalizáciu, a spracovanie viac-zdrojových požiadaviek.
- úroveň špecifická pre jednotlivé zdroje (angl. resource-specific level): Táto úroveň je špecifikovaná manažérom a alokátorom zdrojov (angl. resource manager, resource allocator), ktorí už pracujú priamo s konkrétnymi zdrojmi. Spravujú pravidlá na rezervovanie zdrojov, zabezpečujú vykonanie úloh v reálnom čase a tiež zabezpečujú, aby beh úlohy na konkrétnom zdroji nebol prerušený.



OBRÁZOK 1: FRSH/FORB - BLOKOVÝ DIAGRAM [3]

Celý framework beží na knižniciach operačných systémov, čo umožňuje nezávislosť behu na viacerých platformách. Na komunikáciu medzi jednotlivými objektmi a úrovňami sa používa FORB, čo je middleware typu "Object Request Broker", ktorý slúži na volanie funkcií iných programov alebo ich častí po sieti. (Detailnejšie je popísaný ďalej.) Tento zjednocuje komunikáciu medzi procesmi a komponentmi frameworku.

API

Ako už bolo spomenuté vyššie, aby bol framework použiteľný pre viaceré aplikácie, je potrebné definovať funkcie užívateľského rozhrania umožňujúce komunikáciu medzi aplikáciou a frameworkom. Keďže API poskytuje túto funkcionalitu nezávisle na tom, o rezerváciu akých zdrojov sa jedná (nezávisle na zvyšných úrovniach), bolo implementované rozdelenie do viacerých modulov na základe žiadanej funkčnosti. V každom module sú definované atribúty, ktoré je možné nastaviť kontraktu. Tieto atribúty sú podľa možností nezávislé na zdrojoch.

Príkladom atribútu môže byť budget, perióda, deadline a podobne.

Poskytované funkcie API môžu byť rozdelené nasledovne (za každým popisom sú príklady použitých funkcií):

- Služby obsluhovania kontraktov - Pracujú s dátovými štruktúrami jednotlivých kontraktov, nastavujú časové požiadavky kontraktov a podobne.

  `frsh_contract_set_resource_and_label` - nastaví zdroj a popis kontraktu

  `frsh_contract_set_basic_params` - nastaví základné parametre

  `frsh_contract_set_timing` - nastaví časové parametre
- Služby vyjednávania - Po pripravení kontraktu aj jeho atribútov sa volá funkcia vyjednávania. Prebehne vyjednávanie kontraktu, ktorého výsledkom je v prípade úspechu virtuálny zdroj, ktorý reprezentuje zdroj vo frameworku. Taktiež je možné existujúci kontrakt zrušiť alebo "prevyjednať" ten istý kontrakt znova.

  `frsh_contract_negotiate` - funkcia vyjednávania kontraktu

  `frsh_contract_renegotiate_sync` - funkcia "prevyjednávania" kontraktu

- Služby párovania virtuálnych zdrojov - Po vytvorení virtuálneho zdroja je potrebné vytvoriť pár kontrakt - virtuálny zdroj. Toto sa dosiahne volaním niektorých z nasledujúcich funkcií.

  `frsh_thread_bind` – vytvorí vlákno manažujúce virtuálny zdroj
- Služby na prácu s virtuálnymi zdrojmi - Funkcie na prácu s virtuálnymi zdrojmi počas behu programu

  `frsh_vres_get_remaining_budget` - funkcia vráti zostávajúci budget

  `frsh_vres_get_budget_and_period` - funkcia vráti zostávajúci budget a periódu

Virtuálny zdroj je pre aplikáciu kľúčový. Každý virtuálny zdroj, nezávisle od toho skutočného, poskytuje nasledovné dve služby:

- Garancia služby - Aplikácia, ktorá využíva skutočný zdroj spárovaný s príslušným virtuálnym zdrojom, má garantované splnenie požiadaviek ako boli definované v predtým úspešne vyjednanom kontrakte. Medzi takéto požiadavky patrí napríklad splnenie deadlineov, garantuje sa veľkosť pásma Wi-Fi siete a podobne.
- Detekcia a ochrana pred prekročením rezervovaného limitu - V zmysle dodržania garancie ako je spomenutá vyššie, framework detekuje pokusy o používanie väčšieho množstva zdrojov ako si pôvodne zarezervoval. Takéto pokusy sú bránené a využívanie väčšieho množstva nie je možné.

<u>Úroveň nezávislá na zdrojoch</u>

Sprostredkovateľ kontraktov (angl. contract broker) frameworku FRSH-FORB slúži ako sprostredkovateľ medzi aplikáciami a jednotlivými zdrojmi. Ide o distribuovanú aplikáciu s agentom bežiacim na každej stanici. Agenti spolu pracujú na vymieňaní si informácií o všetkých zdrojoch a kontraktoch v celom distribuovanom systéme. V najjednoduchšom prípade sprostredkovateľ len prepošle údaje do príslušného manažéra zdrojov a v prípade úspešného vyjednania kontraktov cez alokátor zdrojov späť aplikácii.

<u>Úroveň špecifická pre jednotlivé zdroje</u>

Táto časť frameworku obsahuje dva moduly, ako je aj možné vidieť na obrázku 1.

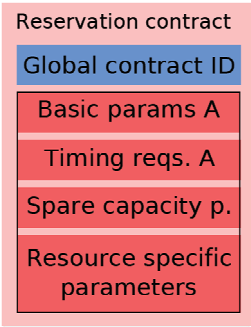
Prvou časťou je manažér zdrojov, ktorého hlavná úloha je poskytovať funkciu vyjednávania kontraktov pre jednotlivé konkrétne zdroje. Toto vyjednávanie je zvyčajne založené na nejakej analýze plánovania, a účel tejto analýzy je nasledujúci:

- Skontrolovať, či nový kontrakt reprezentujúci požiadavku na alokovanie zdrojov nejakého typu môže byť akceptovaný bez toho, aby boli nejakým spôsobom obmedzené virtuálne zdroje predstavujúce už vyjednané kontrakty.
- V prípade, že sa dopredu počíta s možnosťou, že kontrakt bude "prevyjednávaný" (napríklad v prípade, keď aplikácia používajúca zdroje beží vo viacerých módoch a predpokladá sa ich zmena), manažér zdrojov tiež zisťuje, či je možné takéto "prevyjednávanie" uskutočniť.
- V závislosti od analýzy môže manažér zdrojov pridávať informácie, potrebné neskôr pre alokátor alebo plánovač

Druhou časťou tohto modulu je alokátor zdrojov. (angl. resource allocator) Každý manažér zdrojov je sprevádzaný alokátorom zdrojov, ktorých môže byť pre jeden zdroj viacero. Alokátor má dve úlohy, spolupracovať s manažérom zdrojov na manažmente jednotlivých zdrojov podľa ním zadaných inštrukcií, a tiež sa podieľa na párovaní virtuálny zdroj - kontrakt.

Reprezentácia kontraktov

Aby bol framework dostatočne prispôsobivý na rôzne typy zdrojov, definícia ich rezervovania vo forme kontraktov je dostatočne prispôsobivá. V praxi to znamená, že pre každý zdroj je možná potreba rôznych informácií na rezervovanie. Príklady a rozdelenie jednotlivých atribútov kontraktu je na obrázku nižšie.



OBRÁZOK 2: KONTRAKT A JEHO BLOKY

Každý kontrakt je definovaný globálnym identifikátorom, označeným ako ID kontraktu, ktoré je jedinečné pre celý systém. Ostatné atribúty sú zoskupené do množín nazvaných bloky. Každý kontrakt obsahuje viacero týchto blokov a každý z nich reprezentuje skupinu informácií, ktoré sa použijú spolu. Navyše každý kontrakt typicky obsahuje blok, ktorý definuje informácie typické pre neho.

Najpoužívanejšími atribútmi v kontraktoch sú

- budget - Predstavuje rozpočet, množstvo ponúkanej služby. (veľkosť dát, potrebný čas vykonania a podobne)
- perióda - Doba v akej aplikácia žiada o pravidelné prideľovanie budgetu.
- deadline - Doba, v akej má byť úloha dokončená.
- typ pracovného zaťaženia (angl. workload type) - Popisuje spôsob pracovného zaťaženia, ktorý môže byť dvoch typov:
    - ohraničený (angl. bounded) - Aplikácia má vymedzené konkrétne množstvo práce, ktoré musí spraviť v každej perióde. (angl. job)
    - nedefinovaný (stredný, angl. intermediate) - Koncept vymedzeného množstva práce nie je definovaný.

Atribúty jednotlivých kontraktov môžu byť nastavované a upravované aj manažérom zdrojov a sprostredkovateľom kontraktov. Sprostredkovateľ kontraktov aj manažér zdrojov môžu na základe typu kontraktu odhadnúť, ktoré ďalšie informácie skrývajúce sa v atribútoch sú pre daný zdroj ešte potrebné.

Počas procesu vyjednávania kontraktov môžeme predpokladať 3 typy kontraktov (na základe ich funkcionality):

- Užívateľský kontrakt (angl. User contract) - Kontrakt, ako bol vytvorený aplikáciou. V tejto forme môže kontrakt obsahovať viacero možností, ako bude rezervovaný, pomocou bloku "náhradná kapacita" (angl. spare capacity)
- Kontrakt pripravený na rezerváciu (angl. Reservation contract) - Vznikne v sprostredkovateľovi kontraktov z užívateľského kontraktu. Sprostredkovateľ kontraktov vyberie jednu konkrétnu možnosť z bloku "náhradnej kapacity" pokiaľ je viac možností k dispozícii.

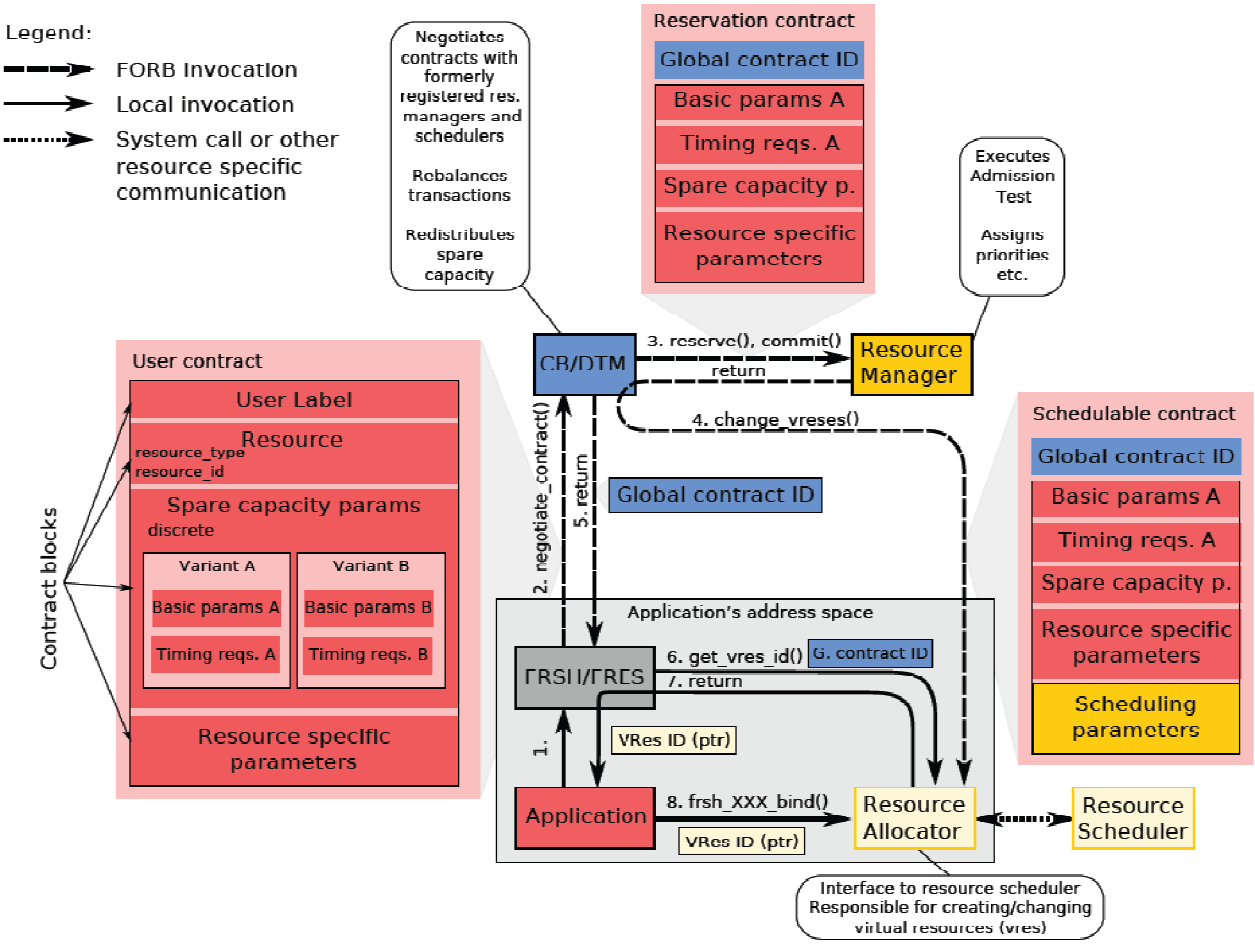
- Plánovateľný kontrakt (angl. Schedulable contract) - Je generovaný manažérom zdrojov z kontraktu pripraveného na rezerváciu, a vznikne po operácii vyjednávania kontraktu. Tento kontrakt je rozšírený o informácie pre alokátor zdrojov a slúži na vytvorenie virtuálneho zdroja.

Proces vyjednávania kontraktov

Proces ako je opísaný v [2]

1. Vyjednávanie začína v aplikácii, ktorá pripraví kontrakt a vyplní jeho atribúty (ID, typ zdroja, základné parametre (budget, perióda), časové požiadavky (deadline), a prípadné ďalšie požiadavky typické pre daný zdroj) Následne aplikácia zavolá službu užívateľského rozhrania.
2. Užívateľské rozhranie použije FORB na zavolanie funkcie vyjednávania v najbližšom, miestnom sprostredkovateľovi kontraktov.
3. Sprostredkovateľ urobí nasledovné:
    a. Priradí kontraktu globálny identifikátor
    b. Pokiaľ existuje blok "náhradná kapacita" (spare capacity), distribučným algoritmom sa vyberie najvhodnejší variant a pripraví sa kontrakt pripravený na rezerváciu (ako je popísaný vyššie)
    c. Následne sa podľa bloku definujúceho zdroj vyberie príslušný manažér zdrojov. (Okrem toho, že manažér zdrojov musí pracovať s rovnakými zdrojmi, nájde sa tiež najvhodnejší z relatívnych manažérov v distribuovanej sieti.)
    d. Potom manažér zdrojov vykoná vyhodnocovací test (angl. admission test). Pokiaľ je kontrakt odmietnutý, táto informácia je oznámená aplikácii. Pokiaľ je kontrakt rozvrhnuteľný, pridá sa blok definujúci rozvrhnuteľnosť kontraktu. (`FRES_BLOCK_FWP_SCHED`) Tým sa zmení na plánovateľný a posunie sa sprostredkovateľovi zdrojov. (Použije sa funkcia `fres_contract_add_block(contract,FRES_BLOCK_FWP_SCHED,sched);`)
4. Sprostredkovateľ zdrojov posunie plánovateľný kontrakt do alokátora zdrojov. Alokátor prakticky aplikuje zmeny, vytvorí virtuálne zdroje, nastaví ich parametre. Tieto operácie prebiehajú s dohľadom plánovača zdrojov.

5. Pokiaľ vytvorenie a naplánovanie prebehlo úspešne, aplikácia dostane ako návratovú hodnotu ID vyjednaného kontraktu.
6. Cez vrátené ID aplikácia pristupuje ku zdrojom. Aby bol prístup rýchly, funkčne sa všetky zdroje raz prehľadajú a informácia o virtuálnom zdroji sa udržuje dlhodobo pomocou ukazovateľa.



Obrázok 3: Diagram interakcie modulov FRSH-FORB počas vyjednávania kontraktu

1.1.2 Podpora Wi-Fi sietí vo framework-u pre rezerváciu zdrojov

Framework podporuje aj komunikáciu cez Wi-Fi sieť. Časť frameworku, ktorá sa týmto zaoberá a je za manažment zodpovedná, sa nazýva FRSH WLAN Protokol. (skrátene FWP) Tento protokol je založený na štandarde IEEE 802.11 a používa prístup Vylepšenej DFC prístupovej metódy ku kanálom. (angl. EDCA - Enhanced DCF Channel Access, DCF - Funkcia rozloženej koordinácie, angl. Distributed Coordination Function) Táto metóda definuje rozličný prístup k médiu na základe 4 rôznych kategórií (zvuk, video, najlepšia odozva – angl. best effort, pozadie – angl. background) Aby sa predišlo prípadným kolíziám, použitý je pravdepodobnostný algoritmus.

FWP poskytuje vlastné užívateľské rozhranie, cez ktoré sa dá jednoducho pristupovať k jednotlivým funkciám vyjednávania kontraktov, ako je opísané vyššie. Používa protokol UDP.

V súčasnosti pracuje FWP len pokiaľ je prenosová rýchlosť nemenná. Adaptácia na zmenu rýchlosti je úlohou tejto práce.

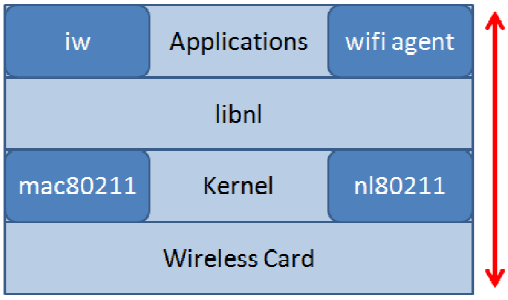
1.2 Ďalšie použité technológie

1.2.1 Subsystémy Linuxu mac80211 a nl80211

Subsystémy sú súčasťou jadra Linuxu. Ich zaradenie v štruktúre jednotlivých komponentov je popísané nižšie:

- Wi-Fi sieťová karta - prijíma a odosiela pakety
- Jadro - obsahuje ovládače na Wi-Fi kartu a podsystémy mac80211 a nl80211. Ich úloha je nasledovná:
    - mac80211 - je zodpovedný za plánovanie a tvorbu paketov
    - nl80211 - je zodpovedný za konfiguráciu rozhrania bezdrôtových sietí pre užívateľské rozhranie
- Knižnica libnl - transportná vrstva používaná na komunikáciu s jadrom pomocou netlink rozhrania

- Užívateľské programy - na prácu so sieťou volajú funkcie knižnice libnl, patrí sem napríklad nástroj iw, ktorý slúži na nastavanie sieťových rozhraní, alebo wi-fi agent, čo je časť frameworku FRSH-FORB a bude popísaná nižšie.



OBRÁZOK 4: ZARADENIE SUBSYSTÉMOV VZHĽADOM NA SOFTVÉROVÉ VRSTVY V STANICI

1.2.2 Netlink komunikácia a knižnica libnl

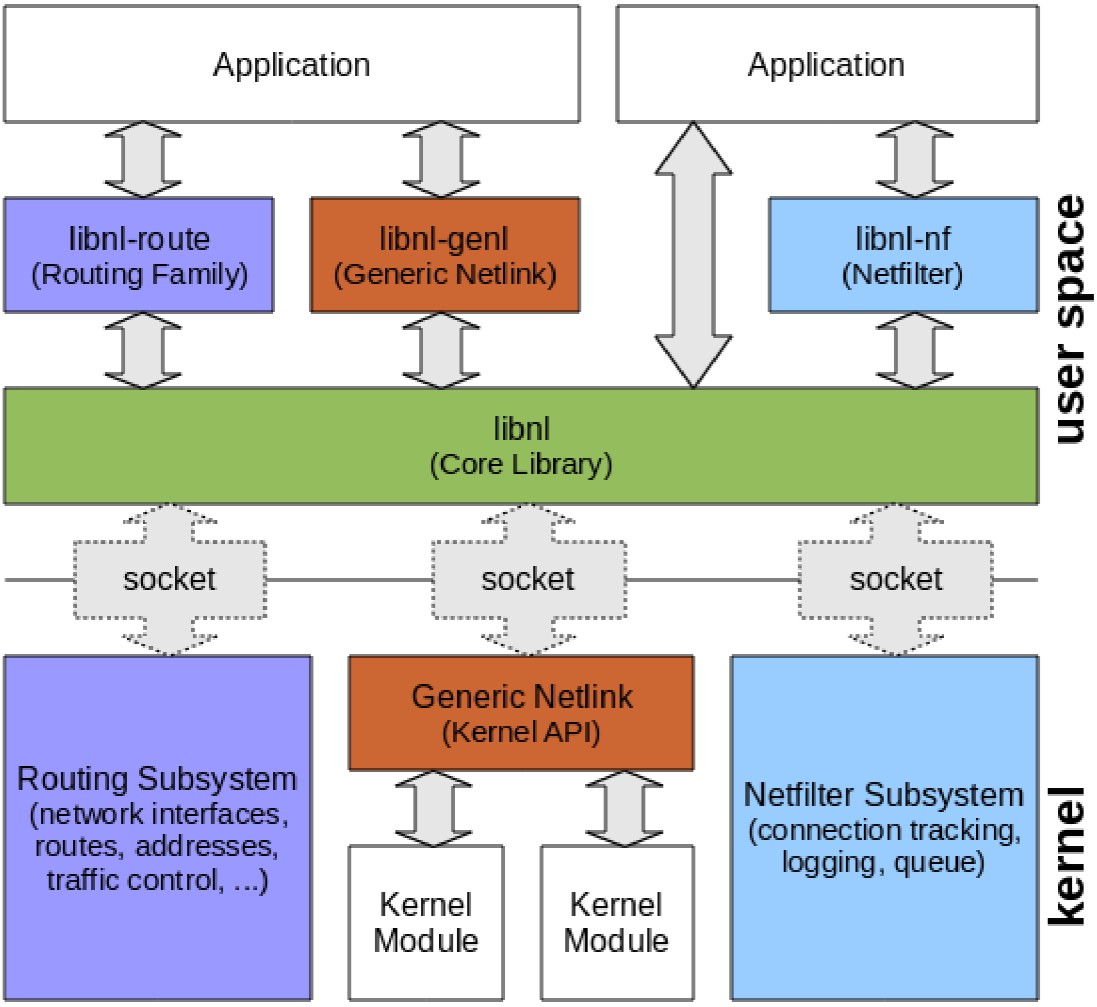
Jadro Linuxu používa monolitický dizajn, kde základné časti sú implementované v jadre operačného systému (angl. kernel space), zatiaľ čo subsystémy ako sieť a konfigurácia ovládačov sú v časti označenej ako užívateľský priestor. (angl. user space) Všeobecne platí, že subsystémy jadra poskytujú rozhrania procesom užívateľského rozhrania a tieto procesy tak dostávajú informácie o zdrojoch a službách jadra, prípadne jadro konfigurujú. Je prirodzené, že komunikácia medzi týmito časťami operačného systému je dôležitá a musí byť čo najefektívnejšia. [5]

Netlink je spôsob takejto komunikácie, ktorá prebieha medzi jadrom operačného systému a užívateľským rozhraním a naopak. Taktiež môže byť použitý ako spôsob komunikácie medzi procesmi. Komunikácia je obojstranná, a každý typ môže používať vlastný protokol definovaný v

`include/linux/netlink.h`

Netlink je implementovaný na infraštruktúre BSD socketov a preto podporuje aj základné primitívne funkcie ako `socket()`, `bind()`, `sendmsg()`, `recvmsg()` a podobne.

Knižnica libnl poskytuje API pre netlink komunikáciu. Jej štruktúra je zobrazená na obrázku nižšie. [7]



OBRÁZOK 5: SCHÉMA KNIŽNICE LIBNL

1.2.3 mac80211

mac80211 je framework, ktorý vývojári používajú na tvorbu ovládačov pre bezdrôtové zariadenia typu "SoftMAC". SoftMAC je taký typ bezdrôtovej sieťovej karty, kde sa očakáva softvérové ovládanie MLME. (Riadiaca entita podvstvy MAC, angl. Media Access Control (MAC) Sublayer Management Entity) [8] Umožňuje pohodlnejšie ovládanie softvéru pomocou manažmentu rámcov 802.11 priamo v softvéri. Väčšina prístrojov podporuje túto možnosť.

Pre registráciu a nastavenie sieťového podsystému používa mac80211 konfiguračné API cfg80211 v kombinácii s nl80211. Pre úplnosť, opačným prípadom ako "SoftMAC" je "FullMAC", kde je MLME ovládané hardvérovo. Pre bezdrôtové zariadenia tohto typu je framework mac80211 nepoužiteľný.

1.2.4 nl80211

nl80211 je protokol užívateľského priestoru založený na Netlinku. Používa sa na nastavenie bezdrôtových zariadení z pohľadu užívateľa. Konkrétna použitá funkcionalita je opísaná v časti implementácia.

1.2.5 WvTest

WvTest je testovací framework, ktorý volí najjednoduchší možný prístup pri testovaní. Je to protokol, nie užívateľské rozhranie. (Toto môže byť ľubovoľne prispôsobené podľa požiadaviek – rozhranie použité v tejto práci je popísané nižšie) Framework je použiteľný na viacerých operačných systémoch a podporuje takmer všetky programovacie jazyky. [6]

Základná myšlienka je v tom, aby bolo používanie čo najjednoduchšie a čo najviac užívateľsky príťažlivé. Preto sa nepoužíva objektová orientácia, ale funkcie sa testujú volaním testovanej funkcie ako parametra niektorej z testovacích funkcií frameworku. Framework funkciu spustí, skontroluje jej výstup a vráti výsledok testu.

Pokiaľ chceme testovať funkciu things.works(), otestujeme ju zavolaním WVPASS(thing.works()); a presvedčíme sa, že funkcia v poriadku prebehla. Existujú ďalšie makrá typu WV*, napríklad WVPASSEQ(a,b), ktoré vráti pozitívny výsledok testu, pokiaľ je a rovné b.

Tento prístup tiež podporuje testovanie viacero funkcií naraz, keď v zdrojovom súbore každú funkciu uzavrieme do vhodného makra. Wvtest presne ukáže, čo prebehlo správne a čo nie vo výstupe, ako je popísaný v ďalšom odstavci.

<u>Formát výstupu Wvtestu</u>

Použité užívateľské rozhranie, ako je použité v tejto práci, vyzerá nasledovne:

```
1 Testing "my test function" in mytest.t.cc:
2 ! mytest.t.cc:432     thing.works()         ok
3 This is just some output that was printed while counting.
4 ! mytest.t.cc.433     3 < 4                 FAILED
```

Existujú 4 typy výstupu:

- Hlavička testovacej funkcie (riadok 1) - Určuje začiatok testovanej funkcie. Riadok výstupu začína so slovom "Testing" (Testovaná) bez akýchkoľvek bielych znakov pred slovom, potom nasleduje názov funkcie, potom "in" a názov súboru (v akom súbore je funkcia definovaná)
- Žiadané (správne) vyhodnotenie testovaného výrazu (riadok 2) - Pokiaľ začiatok riadku výstupu začína "!" a končí "ok", vyhodnocovací výraz bol vyhodnotený testom rovnako ako bolo požadované. V riadku je taktiež definovaná poloha vyhodnoteného výrazu a prípadný komentár.
- Akýkoľvek výpis (riadok 3) - Pokiaľ riadok výstupu nezačína ! a ani nevyzerá ako hlavička testu (nezačína kľúčovým slovom "testing"), ide len o ladiaci výstup, ktorý bol vložený užívateľom pre prípad, ak by test neprešiel.
- Vyhodnotenie testovaného výrazu ako nesprávne (riadok 4) - Formát výstupu je rovnaký ako pri žiadanom vyhodnotení testovaných výrazov s tým rozdielom, že posledné slovo nie je "ok", ale "FAILED".

1.2.6 ULUT

Knižnica ULUT, uLan Utilities Library obsahuje predpripravené konštrukcie a funkcie na vytváranie a správu dátových štruktúr (spájaný zoznam, prioritné rady, AVL stromy atď.) [10] Keďže potrebuje len málo pamäte, môže byť použitá aj v malých vstavaných aplikáciách. V tejto práci je použitá na uchovávanie údajov o všetkých staniciach v sieti.

Niektoré z využitých možností tejto knižnice:

Knižnica obsahuje možnosť definovania vlastných funkcií tým, že zadefinujeme jednotlivým funkciám manažmentu dátových štruktúr vlastný prefix.

```
#define UL_LIST_CUST_DEC(cust_prefix, cust_head_t, cust_item_t,\
            cust_head_field, cust_node_field) \
\
static inline cust_item_t * \
cust_prefix##_node2item(const cust_head_t *head, const ul_list_node_t
*node) \
  {return UL_CONTAINEROF(node, cust_item_t, cust_node_field);}\
\
static inline void \
cust_prefix##_init_head(cust_head_t *head)\
{\
```

```
  INIT_LIST_HEAD(&head->cust_head_field);\
}\
```

Tieto vytvorené funkcie odkazujú na štandardné funkcie práce s dátovými štruktúrami, ako sú napríklad:

```
static inline void list_add(struct list_head *newe, struct list_head
*head)
static inline void list_add_tail(struct list_head *newe, struct
list_head *head)
static inline void list_del_init(struct list_head *entry)
```

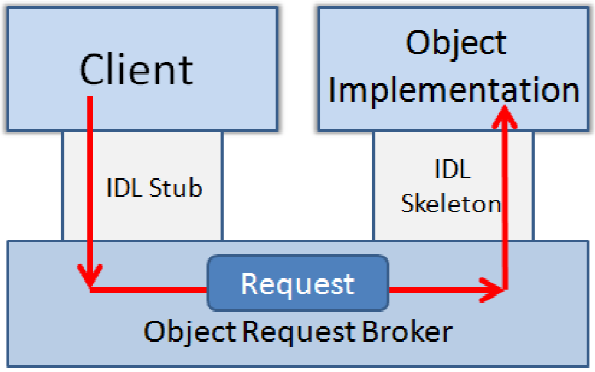
1.2.7 COBRA IDL

COBRA (angl. Common Object Request Broker Architecture), je platformovo a softvérovo nezávislá architektúra a infraštruktúra používaná na komunikáciu medzi aplikáciami alebo ich časťami po sieti. Používa protokolový štandard IIOP, a preto je jej použiteľnosť neobmedzená. [12]

IDL (angl. Interface Definition Language [11]) je jazyk popisujúci rozhranie, ktorý sa používa na definovanie spôsobu ako sa jednotlivé komponenty spájajú. Je možné spájať komponenty napísané v rôznych jazykoch, alebo komunikujúce po sieti.

Aplikácie fungujúce na princípe tejto technológie sú poskladané z objektov, ktorý kombinuje funkcionalitu a dáta. Pre každý objekt je vytvorené rozhranie (opísané v jazyku IDL), ktoré server ponúka klientom, aby mali možnosť ho zavolať. Toto rozhranie definuje požiadavky a odpovede na ne.

Implementácia je od rozhrania oddelená a je schovaná pred stranou klienta. Klient preto môže volať objekty len tak, ako ich vidí v rozhraní. Obrázok nižšie ukazuje, ako je implementácia spojená s rozhraním. IDL kód sa skompiluje do častí nazvaných "IDL stubs" a "IDL Skeletons", kde "IDL stubs" predstavuje kostru objektu na strane klienta a "IDL Skeletons" predstavuje kostru objektu na strane serveru. Implementácia objektu volaného klientom je na strane serveru. Pretože IDL definuje rozhrania striktne, nemá definícia na strane klienta problém "sedieť" so stranou servera, aj v prípade ak implementácia prebehla v rôznych programovacích jazykoch.



OBRÁZOK 6: BLOKOVÁ SCHÉMA DEFINÍCIE A VOLANIA IDL OBJEKTOV

Každá inštancia objektu má jedinečný odkaz na objekt, ktorý zabráni zámene inštancií. Klient sa správa akoby chcel vyvolať operáciu na objekte, ale v skutočnosti vyvoláva len IDL definíciu na svojej strane. Toto vyvolanie pokračuje v objekte ORB, na ktorý sa odkazuje, kde sa zavolá operácia na objekte ako je definovaná na strane servera.

2 Návrh riešenia a implementácia

2.1 Návrh riešenia

Ako bolo spomenuté vyššie, v súčasnom stave pracuje framework so statickou hodnotou prenosovej rýchlosti v bezdrôtovej sieti. Zo zadania vyplýva priamy postup návrhu a implementácie adaptácie na zmenu rýchlosti, čo opíšeme v tejto kapitole.

Na splnenie adaptácie sú potrebné nasledujúce kroky:

- Zisťovanie stavu aktuálnej prenosovej rýchlosti na všetkých staniciach v sieti
- Zhromažďovanie informácií o stave siete v manažérovi zdrojov
- Rozšírenie vyhodnocovacieho testu tak, aby počítal s rozličnými rýchlosťami
- Nájdenie spôsobu, akým sa budú aplikácie prispôsobovať aktuálnej prenosovej rýchlosti

2.1.1 Zisťovanie stavu aktuálnej prenosovej rýchlosti

Zisťovanie aktuálnej prenosovej rýchlosti vykonáva samostatný program, ktorý beží na každom počítači v sieti. Tento program, nazvaný wifi agent, zbiera informácie zo sieťovej karty počítača, na ktorom beží. Následne ich posiela do manažéra zdrojov. Dvojica údajov, ktorá sa pre každú stanicu pošle, je teda MAC adresa, ktorá definuje počítač v sieti, a aktuálna rýchlosť, akou je táto stanica schopná vysielať.

Aby sme zabezpečili zisťovanie zmeny prenosovej rýchlosti, je potrebné zisťovanie zmeny rýchlosti pravidelne aktualizovať. V pravidelných intervaloch preto agent zisťuje, či sa prenosová rýchlosť nezmenila a v prípade zmeny posiela údaje o novej hodnote do manažéra zdrojov. Dôležitým parametrom je perióda zisťovania, aby v prípade zmeny rýchlosti nedošlo k zastaveniu komunikácie na sieti.

2.1.2 Zhromažďovanie informácií o stave siete

Manažér zdrojov prijíma informácie zo všetkých počítačov v sieti v samostatnom vlákne a ukladá informácie do vhodnej dátovej štruktúry. V tomto prípade je to spájaný zoznam staníc, kde každý prvok spájaného zoznamu uchováva kombináciu MAC adresa a aktuálna prenosová rýchlosť.

Tento spájaný zoznam je prístupný aj pre vyhodnocovací test, kde sa s ním pracuje, ako je spomenuté nižšie.

2.1.3 Rozšírenie vyhodnocovacieho testu

Vyhodnocovací test pôvodne pracoval tak, že predpokladal statickú rýchlosť, ktorá bola nastavená na 1 Mbps. Rozšírenie očakáva vyjednávanie kontraktu na základe aktuálnej informácie o rýchlosti uloženej v manažérovi zdrojov. Preto sa pred vyjednávaním spájaný zoznam prehľadá a výpočet samotného využívania konkrétneho zdroja (v tomto prípade je to rýchlosť bezdrôtovej siete) prebehne s aktuálnou hodnotu.

2.1.4 Prispôsobovanie aplikácií

Prispôsobovanie aplikácií rôznym zmenám prenosovej rýchlosti je možné viacerými spôsobmi:

Prvým a asi aj najelegantnejším spôsobom je ten, keď sa aplikácia o zmenu rýchlosti vôbec starať nemusí a o prerozdeľovanie zdrojov sa automaticky postará manažér zdrojov. V tomto prípade pri zmene rýchlostí manažér zdrojov zmení veľkosť pridelených zdrojov a upovedomí o tom aplikáciu. (prípadne aplikácia sa spýta, či nebolo niečo zmenené)

V praxi to znamená, že pokiaľ sa zmení hodnota rýchlosti vysielania, po zápise zmeny do spájaného zoznamu sa skúsi vyjednať lepší kontrakt. Pokiaľ to nie je možné, alebo naopak dôjde k spomaleniu prenosovej rýchlosti, nové vyjednanie prebehne a o výsledku bude aplikácia informovaná.

Druhým možným prístupom je možnosť, keď sa aplikácia o manažment zmeny rýchlosti postará sama, a to vyjednaním nového kontraktu po tom, ako zistí zmenu rýchlosti. V

tomto prípade by musela mať prístup k zoznamu staníc, alebo aspoň k aktuálnej prenosovej rýchlosti, akou sama disponuje. V prípade zmeny by potom požiadala o zrušenie pôvodného kontraktu a vyjednala by kontrakt nový na základe novej prenosovej rýchlosti.

2.2 Implementácia

V tejto kapitole opíšeme, ako bol logický návrh zapracovaný do frameworku. Pri opise algoritmov sú použité útržky kódu, ktoré sa nemusia úplne zhodovať s vytvoreným kódom, a majú len informatívny charakter.

2.2.1 Zisťovanie stavu aktuálnej prenosovej rýchlosti

Zisťovanie stavu aktuálnej prenosovej rýchlosti je implementované podľa návrhu. Na zistenie aktuálnej rýchlosti sa používa knižnica nl80211.

Wifi agent zisťuje potrebné údaje podobne ako nástroj iw [9]. Najskôr prebehne inicializácia potrebných zdrojov na používanie netlinku. Konkrétne sa alokuje miesto pre netlink sockety, prebehne pripojenie k netlinku a alokuje sa pamäť potrebná pre operácie. Taktiež sa vyhľadá podsystém nl80211.

```
      state->nl_sock = nl_socket_alloc(); //alokovanie
      genl_connect(state->nl_sock);       //pripojenie
      genl_ctrl_alloc_cache(state->nl_sock, &state->nl_cache); //cache
      state->nl80211 = genl_ctrl_search_by_name(state-
>nl_cache,"nl80211");
```

Nasleduje výmena informácií pomocou preddefinovaných funkcií knižnice nl80211. Použité funkcie sú `genlmsg_put()`, `NLA_PUT_U32()` a `nl_recvmsgs()`.

Prijatá správa sa rozdelí na menšie časti podľa požadovanej informácie. Pomocou `nla_parse_nested` získame požadované informácie definované na základe atribútov popisujúcich jednotlivé polia prijatej správy. Pre nás podstatné a použité sú

- `NL80211_STA_INFO_TX_BITRATE` - súčasná rýchlosť vysielania stanice
- `NL80211_RATE_INFO_BITRATE` - maximálna rýchlosť (u16, 100kbit/s)
- `NL80211_ATTR_IFINDEX` - číselný index sieťového rozhrania zariadenia

- `NL80211_ATTR_MAC` - MAC adresa
- `NL80211_CMD_GET_STATION` - vráti atribúty pre stanicu definovanú na základe MAC adresy

Pri posielaní dát bol použitý FORB, ktorého funkcie sú zadefinované v .idl súbore. Odkaz na bežiacu inštanciu FORB na strane manažéra zdrojov je textový a má tvar `"net.sourceforge.frsh-forb.wai"`

Celkový algoritmus:

```
1 wai=forb_resolve_reference(fwp_orb,"net.sourceforge.frsh-forb.wai");
2  while(1)
3  {  /* Get tx info */
4     err = get_tx_bitrate(&nlstate);
5     if (err < 0)
6     fprintf(stderr,"command failed:%s (%d)\n", strerror(-err), err);
7
8     /* Call server functions and send wifi info */
9     if((wai != NULL) && (rate_glob_new != rate_glob_old))
10    {     printf("%d \n", rate_glob);
11          wifi_agent_idl_add(wai, rate_glob_new / 10,
12                mac_addr_string_to_number(mac_addr_glob), &env);
13
14          wifi_agent_idl_print_state(wai, &env);
15    }
16    //TODO: Optimalize sending interval
17    sleep(1);
18 }
```

Ako je z popisu zrejmé, v neustále bežiacom cykle (riadok 2) program zisťuje aktuálny stav vysielanej rýchlosti (riadok 4), ktorý posiela do manažéra zdrojov (riadok 11). Aby sme sa presvedčili o správnosti prístupu, po každom pridaní sa v manažérovi zdrojov vypíše aktuálny zoznam staníc. (riadok 14) Táto informácia je čisto informatívna a nepotrebná, a slúži len na testovacie účely.

Je na uvážení, ako často je potrebné zisťovať a aktualizovať informácie o zmene rýchlosti. Predpokladáme ale, že náklady na zistenie sú na lokálnom počítači minimálne a pokiaľ zmena nenastala, údaje v manažérovi zdrojov sa neaktualizujú. Preto nedochádza k zahlteniu siete údajmi o nej samotnej.

2.2.2 Zhromažďovanie informácií o stave siete

Na strane manažéra zdrojov beží serverová časť aplikácie, ktorá zbiera údaje o všetkých staniciach v sieti. Obsahuje definície funkcií vo formáte COBRA IDL, ktoré potom wifi agent volá, ako bolo spomenuté vyššie.

Aby bolo možné prijímať dáta, do pôvodného manažéra zdrojov bolo potrebné pridať ďalšie vlákno, ktoré sa o toto stará (riadok 13), spolu s registrovaním odkazu FORB. Názov je prirodzene rovnaký ako vo wifi agentovi `"net.sourceforge.frsh-forb.wai"` (riadok 5)

Rozšírená inicializácia manažéra:

```
1  /* Cration of an object implementing myinterface */
2  wai = forb_wifi_agent_idl_new(orb, &wifi_agent_impl, &intern_data);
3 /*We can register our object under a name, so that other processes
4     use it. */
5  forb_register_reference(wai, "net.sourceforge.frsh-forb.wai");
6  /* Initialize head of linked list */
7  sta_list_init_head(&priv.sta_list);
8  /* Create executors and register 2 interfaces */
9  ret = forb_executor_init(&executor_wai);
10    if (ret) goto error;
11 ret = forb_executor_init(&executor_frm);
12    if (ret) goto error;
13 /* 1. interface - forb_execute_object(wai); execute */
14 ret = forb_executor_register_object(&executor_wai, wai);
15    if (ret) goto destroy_and_error;
16 /* 2. interface - frm_register_and_run(orb, &frm); execute */
17 fres_block_register_fwp();
18 frm_ret = frm_register(orb, &frm_data, &executor_frm, &frm);
19    if (!frm_ret) {
20          ret = -1;
21          goto destroy_and_error;
22    }
```

Na záver ešte v krátkosti opíšeme samotné pridávanie údajov do spájaného zoznamu. Postup je jednoznačný. Po spracovaní dát zadaných ako parameter funkcie na strane klienta wifi agenta (riadok 5) prehľadáme spájaný zoznam pre prípad, že ide o aktualizáciu dát a stanica už predtým vysielala. Potom nastane aktualizácia údajov. (riadok 15) V opačnom prípade sa vytvorí a vloží nový prvok spájaného zoznamu. (riadok 24)

```
1  /* Add information about AP (mac_address & bit rate) */
2  CORBA_long wifi_agent_idl_add(wifi_agent_idl _obj, const CORBA_long
3  rate, const CORBA_long_long client_mac_addr, CORBA_Environment *ev)
4  {
5     struct fwp_sta *ed = forb_instance_data(_obj);
6     fwp_sta_t *sta;
```

```
7     fwp_sta_t *new_to_add;
8     new_to_add = (fwp_sta_t*)malloc(sizeof(fwp_sta_t));
9     /*client transmission bitrate (wifi_agent -> fwp_manager)*/
10    ed->rate = rate;
11    ed->client_mac_addr = client_mac_addr;
12    /*fill the linked list here*/
13    ul_list_for_each(sta_list, &priv.sta_list, sta){
14          /*client was sending info before*/
15          if(sta->client_mac_addr == ed->client_mac_addr) {
16                /*mac address is the same change only rate*/
17                sta->rate = ed->rate;
18                return 0;
19          }
20    }
21    /*client was not sending info yet*/
22    new_to_add->rate = ed->rate;
23    new_to_add->client_mac_addr = ed->client_mac_addr;
24    sta_list_ins_tail(&priv.sta_list, new_to_add);
25    return 0;
```

2.2.3 Rozšírenie vyhodnocovacieho testu

Rozšírenie vyhodnocovacieho testu počíta s rozličnými prenosovými rýchlosťami spôsobom, že na začiatku výpočtov vyjednávania kontraktu prehľadá spájaný zoznam a vyberie rýchlosť relatívnej stanice. Pre prehľadávanie je opäť použitá knižnica ULUT. (riadok 10)

Pokiaľ by došlo k nejakej chybe a v kontraktoch by sa nenachádzal blok FWP s MAC adresou a nebolo by možné určiť, aká je rýchlosť, použije sa prednastavená hodnota 1 MBps (riadok 8).

```
1  int fwp_admctrl_utilization(struct fres_sa_scenario *scenario, void
2      *priv, bool *schedulable)
3  {
4     int rate = 1*1000*1000; //default initialization is 1 Mbps
5     fres_sa_scenario_for_each_no_cancel_contract(scenario, c) {
6           fwp_block = fres_contract_get_block(c->contract,
7                 FRES_BLOCK_FWP);
8     if(fwp_block != NULL){
9           /* from list of all stations choose the relative one */
10          ul_list_for_each(sta_list, &pr->sta_list, sta){
11                if(sta->client_mac_addr == fwp_block->mac_address){
12                      rate = sta->rate*1000*1000;
13                      break;
14                }
15          }
16    }
17    /*... continue processing admission test */
```

2.2.4 Prispôsobovanie aplikácií na zmeny prenosovej rýchlosti

Ako bolo spomenuté v návrhu riešenia, existuje viacero prístupov, ako môžu aplikácie reagovať na zmenu rýchlosti.

Návrh, kde framework reaguje na zmeny rýchlostí automaticky a upovedomuje o tom aplikáciu, by vyzeral nasledovne: Po aktualizácii rýchlostí v manažérovi zdrojov sa zavolá funkcia, ktorá preozdelí reálne zdroje a zmení virtuálne zdroje pridelené aplikácii. Aplikácia pravidelne kontroluje, akú má rezervovanú periódu a budget.

Od tohto návrhu sme nakoniec upustili a zvolili sme jednoduchšie riešenie. V tomto aplikácia sama reaguje na zmenu rýchlosti a stará sa o manažment s tým spojený manažment. Kontrakt sa vyjednáva v každom kroku posielania (riadok 12) a po posielaní sa zruší. (riadok 20) Budget a perióda sa odhadujú automaticky, a pokiaľ sa ho nepodarí vyjednať, hodnoty sa upravia smerom nadol alebo nahor. (riadok 3)

```
1  for(int i=0; i<data; i++) {
2     /* guess budget value */
3     budget = guess_value(budget_old);
4     /* initialization of contracts */
5     frsh_contract_init(&contract1);
6     /* create virtual resource */
7     printf("Create vres1\n");
8     if (fwp_vres_create(&vparam1, &vres1) < 0) {
9           return -1;
10    }
11    /* contract negotioate function */
12    if ((frsh_contract_negotiate(&contract1, &vres1)) != 0) {
13          frsh_contract_cancel(vres1);
14          continue;
15    }
16    /* endpoint bind */
17    frsh_send_endpoint_bind(vres1, sepoint1);
18    /* send data */
19    frsh_send_async(sepoint1, msg1, sizeof(msg1));
20    frsh_contract_cancel(vres1);
21    frsh_contract_destroy(contract1);
22 }
```

2.3 Testovanie

Pre overenie funkčnosti a správnosti sme pripravili nasledujúce testy. Použitý je testovací framework wvtest.

2.3.1 Test kontroly zoznamu staníc v manažérovi zdrojov

Testovaná je správnosť zoznamu staníc v spájanom zozname. Pevné hodnoty zoznamu staníc pošleme do manažéra zdrojov pomocou funkcií COBRA IDL, a presvedčíme sa o ich správnosti vypísaním. Porovnáme vypísané dáta s očakávanými.

Test sa skladá z dvoch súborov, shellový skript vytvorí očakávaný výstup (riadok 2) a porovná ho s aktuálnym (riadok 10) Tiež zavolá samotný test "list_of_station_test". (riadok 8)

```
1  WVPASS fcb -dfcb.pid
2  cat > expected << EOF
3  Actual_list_of_stations
4  12:09:76:29:10:EF_-_10
5  ------------
6  #... some more data
7  EOF
8  WVPASS sh -c "frm_fwp -dfwp.pid 2> actual"
9  WVPASS list_of_stations_test
10 kill $(cat fwp.pid)
11 WVPASS diff -u expected actual
```

Samotný test v jazyku C naplní spájaný zoznam staníc vopred zadefinovanými stanicami (riadok 6) a vypíše ho. (riadok 8) Test kontroluje aj to, či sa správne prepíše hodnota v spájanom zozname pri aktualizácii hodnôt. (riadok 9)

```
1  /* Find our implementation */
2  wai = forb_resolve_reference(fwp_orb, "net.sourceforge.frsh-
forb.wai");
3  if(wai == NULL)
4     fprintf(stderr, "failed to create connection\n");
5  if(wai != NULL)
6  {  wifi_agent_idl_add(wai, 10, "12:09:76:29:10:EF", &env);
7                                            /* new station */
8     wifi_agent_idl_print_state(wai, &env);   /* print all nodes */
9     wifi_agent_idl_add(wai, 12, "12:09:76:29:10:EF", &env);
10                                           /* changed rate */
11    wifi_agent_idl_add(wai, 21, "00:02:76:A2:1B:67", &env);
12                                           /* new station */
13    wifi_agent_idl_print_state(wai, &env);   /* print all nodes */
14    /* ...some more changes */
15 }
```

2.3.2 Test správnosti funkcie vyjednávajúcej zdroje

Test správnosti funkcie vyjednávajúcej zdroje (angl. admission control) prebehne tak, že vytvoríme niekoľko kontraktov (riadok 5), nastavíme staticky ich parametre a ako parameter ich vložíme do testovanej funkcie. (riadok 16) Podmienky siete taktiež vytvoríme staticky (riadok 10), a výsledky porovnáme s očakávanými výsledkami. (riadok 25)

```
1  /* define scenario and its contracts */
2  WVPASS((scenario = fres_sa_scenario_new()) != NULL);
3  WVPASS((c1 = fres_sa_contract_new()) != NULL);
4
5  /* initialization of contracts */
6  WVPASS(frsh_contract_init(&contract1) == 0);
7
8  /* ... set resource, label and other details of contracts */
9
10 /* create conditions of network (mac address and rate) */
11 new_to_add1.rate = 1;
12 new_to_add1.client_mac_addr = p->mac_address1;
13 sta_list_ins_tail(&priv.sta_list, &new_to_add1);
14
15 /* tested function */
16 WVPASS(fwp_admctrl_utilization(scenario,&priv,&schedulable) == 0);
17
18 /* check results */
19 WVPASS((c1 = fres_sa_scenario_find_contract(scenario, id1))!=NULL);
20 WVPASS((fwp_sched = fres_contract_get_block(c1->contract,
21                              FRES_BLOCK_FWP_SCHED)) != NULL);
22 WVPASSEQ(fwp_sched->ac_id, FWP_AC_VO);
23
24 /* compare actual and expected results */
25 WVPASSEQ(schedulable, schedulable_expected);
```

Ilustratívny výstup testu vyzerá nasledovne (nie všetky riadky výstupu sú zahrnuté)

```
! wvtestmain.c:67    true                                        ok
Testing "fwp_admctrl_utilization_test" in
/home/martin/skola/iw/cast26/frsh-
forb/build/aquosa/fwp/mngr/test/fwp_adm_test.c:235:
! fwp_adm_test.c:260   frsh_init() == 0                          ok
! fwp_adm_test.c:265   (p[num] = malloc(sizeof(*p[0]))) != NULL ok
! fwp_adm_test.c:90    (scenario = fres_sa_scenario_new()) != NULL ok
! fwp_adm_test.c:116   frsh_contract_set_resource_and_label(
&contract2, FRSH_RT_NETWORK, FRSH_NETPF_FWP, NULL) ==0 ok
! fwp_adm_test.c:154   frsh_contract_set_timing_reqs(&contract1, fal-
se, &deadline1) == 0 ok
! fwp_adm_test.c:229   schedulable == 1 == p->schedulable_expected ==
1 ok
! fwp_adm_test.c:269   fwp_admctrl_utilization_test(p[num]) == 0 ok
! wvtest.c:214   new_valgrind_errs == old_valgrind_errs   ok
memleaks: sure:0 dubious:0 reachable:0 suppress:0
```

2.4 Demonštračná aplikácia

Jednotlivé súčasti pridané do frameworku sme otestovali spolu vytvorením jednoduchej aplikácie, ktorá využíva všetky skôr popísané súčasti.

Cieľom aplikácie je posielanie dát medzi dvoma stanicami na základe aktuálnej prenosovej rýchlosti. Ide o náhodne vytvorené dáta, ktoré nemajú žiadny konkrétny obsah.

Na začiatku aplikácia nastaví parametre a vyjedná kontrakt. Pokiaľ je vyjednávanie úspešné, nastane posielanie dát druhej strane. Po čiastkovom posielaní sa kontrakt zruší a vyjednaním sa stanica presvedčí o tom, či je stále možné posielať dáta druhej stanici.

Posielanie dát

Na začiatku v rámci úvodných inicializácií aplikácia vytvorí vysielajúci endpoint, kde dostane pôvodnú hodnotu budgetu a periódy. Perióda sa počas behu programu nemení. Potom aplikácia vytvorí nový kontrakt (riadok 15), ktorý sa pokúsi vyjednať. (riadok 27)

V prípade kladného výsledku aplikácia pošle dáta (riadok 39) a po poslaní požiada o zrušenie kontraktu. (riadok 45) V ďalšom kroku posielania opäť vytvorí pred posielaním kontrakt s tými určenými hodnotami budgetu a periódy. Vyjednanie kontraktu prebehne nanovo, čím sa zaistí, že podmienky v bezdrôtovej sieti sú aj po prípadnej zmene postačujúce.

Pokiaľ podmienky postačujúce nie sú (nastala zmena v rýchlosti posielania), kontrakt je odmietnutý a zrušený. (riadok 29) Dochádza ku zmenšovaniu budgetu o 100 B. (riadok 30) Následne sa aplikácia pokúsi vyjednať kontrakt nanovo, až kým nie je úspešná. Hneď ako je kontrakt vyjednaný, dochádza k posielaniu dát.

```
1 int send_data(int period_ms,int budget,char *dest_station, int port)
2  {
3     /* ... variables definitions */
4
5     /* FWP initialization */
6     fwp_init();
7     frsh_send_endpoint_protocol_info_t send_pinfo;
8 if(frsh_send_endpoint_create(FRSH_NETPF_FWP,inet_addr(dest_station),
```

```
9     port,send_pinfo, &sepoint)< 0) {
10          return -1;
11    }
12    printf("Send endpoint created\n");
13    while(1) {
14          /* initialization of contracts */
15          frsh_contract_init(&contract);
16                /* set resource and label of contracts */
17                /* set basic parameters of contracts */
18                /* set fwp parameters of contract – mac address */
19          /* set virtual resource parameters and create it */
20          printf("Create VRES\n");
21          if (fwp_vres_create(&vparam, &vres) < 0) {
22                printf("Unable to create VRES\n");
23                return -1;
24          }
25          printf("VRES created \n");
26          / * contract negotioate function */
27          if ((frsh_contract_negotiate(&contract, &vres)) != 0) {
28                /* contract not negotiated */
29                frsh_contract_cancel(vres);
30                size_of_budget_actual = size_of_budget_actual - 100;
31                frsh_contract_cancel(vres);
32                frsh_contract_destroy(contract);
33                continue;
34          }
35          /* contract negotiated – continue sending */
36          frsh_send_endpoint_bind(vres, sepoint);
37          /* Prepare message and send data */
38          msg = prepare_msg(size_of_budget_actual);
39          frsh_send_async(sepoint, msg
40                      size_of_budget_actual);
41          printf("Sent %d. msg of size %d \n", num,
42                            size_of_budget_actual);
43          num++;
44          free(msg);
45          frsh_contract_cancel(vres);
46          frsh_contract_destroy(contract);
47    }
48    return 0;
49 }
```

Prijímanie dát

Druhá aplikácia slúži na prijímanie dát. Táto časť demonštrácie je o niečo jednouchšia. Po inicializácii FWP vytvoríme endpoint na prijímanie dát a priradíme mu port, na ktorom počúva. Prijaté dáta sú uložené do vyrovnávacej pamäte, ktorá je dostatočne veľká, aby obsiahla všetky dáta. Následne dochádza k vyhodnocovaniu dát.

```
1  int main()
2  {  ssize_t len;
3     int i;
4     char buffer[1000000];
5     struct fwp_endpoint *sepoint1, *sepoint2, *repoint1, *repoint2;
6     fwp_endpoint_attr_t attr;
```

```
7     fwp_addr_t from;
8
9     fwp_endpoint_attr_init(&attr);
10    /* FWP initialization */
11    printf("Start\n");
12    if (fwp_init() != 0) {
13          printf("FWP initialization failed!\n");
14          return -1;
15    }
16    /* Creation of receiving endpoint. Listening on port 7777. */
17    /* local_addr should be handled when creating socket */
18    if (fwp_receive_endpoint_create(7777, &attr,&repoint1) < 0){
19          return -1;
20    }
21    printf("Receive endpoint created\n");
22    /* receiving data */
23    while (1) {
24    if ((len = fwp_recv(repoint1, buffer, sizeof(buffer), &from,
25                0)) < 0) {
26                perror("Error while receiving data");
27                return -1;
28          }
29                else printf("Received - %d\n", len);
30    }
31    return 0;
32 }
```

Toto vyhodnocovanie prebehne porovnávaním prijatých a poslaných dát:

```
WVPASS diff -u send recieved
```

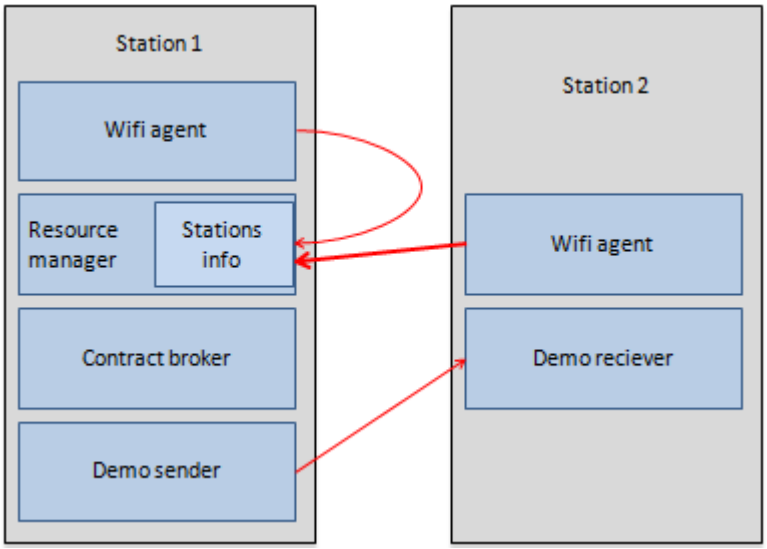
V konzole pozorujeme veľkosť prijatých dát, tak ako komunikácia prebieha po sieti. Túto hodnotu obdobným spôsobom porovnáme.

Experimenty

Po vytvorení demonštračnej aplikácie sme vykonali niekoľko experimentov, ktorými sme sa presvedčili o funkčnosti nami vytvorených zmien vo frameworku. Vytvorené testovacie prostredie bolo jednoduché, a pozostávalo z dvoch staníc, ktoré si posielali dáta medzi sebou.

Na prvej stanici bežal vyjednávateľ kontraktov, manažér zdrojov, wifi agent zaisťujúci informácie o stanici a časť demonštračnej aplikácie posielajúcej dáta.

Na druhej stanici bežal wifi agent zaisťujúci informácie o druhej stanici a časť demonštračnej aplikácie prijímajúca dáta.



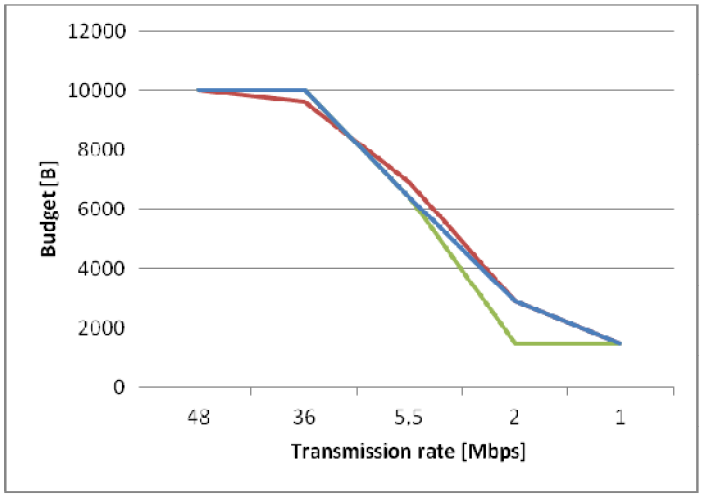
OBRÁZOK 7: ROZMIESTENIE BEŽIACICH APLIKÁCIÍ NA STANICIACH POČAS TESTOVANIA

Oba wifi agenti posielajú informácie o sieti do manažéra zdrojov, kde sú uložené ako dvojice. Pokiaľ sa zmení vysielajúca rýchlosť niektorej zo staníc, manažér zdrojov o zmene informuje konzolovým výpisom:

```
Actual list of stations:
00-16-EA-77-56-E4_-_54_Mbps
8C-A9-82-8E-D9-CE_-_54_Mbps
```

Nastavíme počiatočný budget a periódu tej časti demonštračnej aplikácie, ktorá posiela dáta. V tomto prípade sme nastavili budget rovný 10000 B a periódu 30 ms. Prirodzene, pomocou IP adresy a portu tiež definujeme, kam sa budú dáta posielať.

Postupom, ako je opísaný vyššie posielame dáta a zväčšovaním vzdialenosti medzi stanicami meníme kvalitu bezdrôtovej siete a tým aj možnú rýchlosť prenosu. Spravili sme viaceré merania, ktoré sme zaznamenali do grafu. (obrázok 8)



OBRÁZOK 8: EXPERIMENTÁLNA ZÁVISLOSŤ VEĽKOSTI POSIELANÝCH DÁT OD RÝCHLOSTI PRENOSU

Výsledky sú očakávané, so znižujúcou sa rýchlosťou prenosu sa znižuje veľkosť posielaných dát. Testovanie prebehlo vo viacerých rôznych prostrediach, a najčastejšia zmena je vyznačená modrou farbou. Keďže testovanie neprebehlo v uzavretom prostredí bez akejkoľvek inej komunikácie na sieti, výsledky môžeme pokladať len za orientačné.

Skresľovanie mohlo spôsobiť aj to, že samotná zmena rýchlosti neprebiehala úplne riadene, to znamená rušenie záviselo len od prostredia. Občas sa preto rýchlosť a s tým aj celá práca frameworku menila dosť neočakávane.

3 Zhodnotenie

Po naštudovaní frameworku frsh-forb sme rozšírili funkcionalitu frameworku tak, aby bol schopný pracovať s rôznymi zmenami rýchlostí. Počas návrhu a implementácie sme sa stretli s viacerými možnosťami riešenia aj problémami, ktoré bolo potrebné riešiť. Napriek tomu rôzne rýchlosti a ich zmena v bezdrôtovej sieti boli zapracované tak, aby framework o nich vedel a počítal s nimi.

Zbieranie údajov s aktuálnymi rýchlosťami malo pomerne jednoznačné riešenie, a to čo sa zisťovania rýchlostí týka aj jeho prenosu do manažéra zdrojov. Možno sa vynára otázka, čo všetko by bolo nevyhnutné o sieti uchovávať, ako napríklad okrem MAC adresy definujúcej vysielajúcu stanicu aj prijímajúcu MAC adresu, a prípadne rýchlosť opačným smerom. Pre jednoduchosť však predpokladáme, že rýchlosť prijímania je rovná rýchlosti vysielania.

Zhromaždenie potrebných dát v manažérovi zdrojov má tiež jednoznačné riešenie. Dátová štruktúra na ukladanie dát, ktorou je spájaný zoznam, je dynamická a použitím už vopred aplikovaného a otestovaného modulu sme zabezpečili jej maximálne využitie. Jej použitie v procese vyjednávania kontraktov je preto veľmi jednoduché.

Najväčším problémom s pomerne najrozsiahlejšími možnosťami riešenia tak ostala adaptácia zmien priamo pri použití v aplikáciách. Komplikovanosť prístupu nás nakoniec prinútila použiť nie práve najefektívnejšie riešenie tak, že aplikácia využívajúca framework sa sama stará o to, aby správne využívala možnosť rôznych prenosových rýchlostí a jej zmeny. Preto tu stále ostáva možnosť vylepšenia a zdokonalenia frameworku a nadviazanie na už hotovú prácu.

Počas vývoja sme sa stretli s viacerými problémami, a presvedčili sme sa, že vývoj aplikácií tohto typu nie je jednoduchý. Už len samotné správne nastavenie testovacieho prostredia nám zabralo veľa času, a keďže niektoré použité súčasti (ako nl80211) sú ešte stále vo vývoji, ich nasadenie tiež trvalo dlhšie, ako sme očakávali. Napriek tomu, že vidíme možné vylepšenia, si ale myslíme, že stanovený cieľ sme splnili.

4 Technická dokumentácia

Všetky implementované súbory sú k dispozícii vo verejnom gitovom repozitári `git://rtime.felk.cvut.cz/frescor/frsh-forb.git`, vo vetve `wifi-adaptive`

Tiež je možné zdrojové súbory prezerať online, a to na pod odkazom `tree` poslednej aktualizácie na
`http://rtime.felk.cvut.cz/gitweb/frescor/frsh-forb.git/shortlog/refs/heads/wifi-adaptive`

Zisťovanie stavu aktuálnej prenosovej rýchlosti (2.2.1)
`\frsh-forb\src\fwp\fwp\wifi_agent\*.*`
Zhromažďovanie informácií o stave siete (2.2.2) -
`\frsh-forb\src\fwp\fwp\mngr\wifi_agent*.*`
Rozšírenie vyhodnocovacieho testu (2.2.3) -
`\frsh-forb\src\fwp\fwp\mngr\fwp_admctrl.c`
Prispôsobovanie aplikácií na zmeny prenosovej rýchlosti (2.2.4) -
`\frsh-forb\src\fwp\fwp\demo\*.*`
Test kontroly zoznamu staníc v manažérovi zdrojov (2.3.1) -
`\frsh-forb\src\fwp\fwp\mngr\test\list_of_stations_test.*`
Test správnosti funkcie vyjednávajúcej zdroje (2.3.2) -
`\frsh-forb\src\fwp\fwp\mngr\test\fwp_adm_test.c`
Demonštračná aplikácia (2.4) -
`\frsh-forb\src\fwp\fwp\demo\*.*`

Prikladáme výpis celých zdrojových súborov tých implementačných častí, ktorých fragmenty kódu boli v kapitole 2.2 najviac skrátené. Všetky súbory sú k dispozícii na webovej adrese ako je to uvedené vyššie.

4.1 Funkcia vyhodnocovacieho testu

Súbor `\frsh-forb\src\fwp\fwp\mngr\fwp_admctrl.c`

```
#include <fres_sa_scenario.h>
#include <fwp.h>
#include "fwp_idl.h"
#include "fwp_admctrl.h"
#include <ul_log.h>
#include <ul_logreg.h>
#include "fwp_utils.h"

UL_LOG_CUST(ulogd_fwp_admctrl);
ul_log_domain_t ulogd_fwp_admctrl = {UL_LOGL_MSG, "fwp_admctrl"};
UL_LOGREG_SINGLE_DOMAIN_INIT_FUNCTION(fwp_admctrl_logreg_domains, ulogd_fwp_admctrl);

/* TODO: Find out the real value and determine what influences it (MTU
 *      of the interface minus header sizes?).  */
#define MTU 1472
#define UDP_HEADER_SIZE 8
#define IP_HEADER_SIZE 20
#define LLC_HEADER_SIZE 2     /* ??? */
#define MAC_HEADER_SIZE 26    /* With QoS field added */
```

```
#define MAC_FCS_SIZE 4
#define RTS_FRAME_SIZE 20
#define CTS_FRAME_SIZE 14
#define ACK_FRAME_SIZE 14
#define ASLOTTIME_USEC 20     /* 802.11g-2003 p. 46 */
#define ASIFSTIME_USEC 10
#define ADIFSTIME_USEC (ASIFSTIME_USEC + 2*ASLOTTIME_USEC)
#define AAIFSTIME_USEC(n) (ASIFSTIME_USEC + (n)*ASLOTTIME_USEC)

/* Default values from 802.11e */
const int aifsn[FWP_AC_NUM] = { 2, 2, 3, 7 };
const int cwmin[FWP_AC_NUM] = { 3, 7, 15, 15 };
/* Experimental konstants - weight of backoff */
const int pisvejc[FWP_AC_NUM] = { 6, 5, 2, 2 };
int fwp_reserved_utilization;

/**
 * Calucaltes frame duration in microseconds. If the real duration is
 * represented by a fractional number, the value is rounded up.
 *
 * @param length Number of bytes in PSDU.
 * @param rate_bps Transmit rate of PSDU. The rate should correspond
 *            to other parameters accoring to 19.3.2 (802.11g)
 * @param erp_ofdm Whether Extended Rate PHY (part of 802.11g) and ERP-OFDM
 *            modulation is used.
 * @param short_preamble Whether short preamble (HR/DSSS/short) is
 *            used (802.11b)
 *
 * @return The number of microseconds or a negative number in case of error.
 */
static int frame_duration(uint16_t length, int rate_bps, bool erp_ofdm, bool
short_preamble)
{
        uint32_t duration_usec = 0;

        if (!erp_ofdm) {
                duration_usec = ((int64_t)(length) * 8 * SEC_TO_USEC / rate_bps);
                if (short_preamble) {
                        /* For HR/DSSS/short (2, 5.5 and 11 Mbit,
                         * 802.11b-1999), also for DSSS-OFDM rates and
                         * ERP-PBCC rates.
                         * Preamble sent at 1 MBit, header at 2 Mbit */
                        duration_usec += 72/*bits*/+48/*bits*//2;
                } else {
                        /* For DSSS PHY (1 and 2 Mbit rates,
                         * 802.11-1999) and for DSS-OFDM and ERP-PBCC
                         * rates. Always sent at 1 MBit */
                        duration_usec += 144/*bits*/+48/*bits*/;
                }
        } else {
#define MBIT_TO_INDEX(rate_Mbps)  ((rate_Mbps)/3 - 2)
                const int N_dbps[] = {
                        [MBIT_TO_INDEX(6)] = 24,
                        [MBIT_TO_INDEX(9)] = 36,
                        [MBIT_TO_INDEX(12)] = 48,
                        [MBIT_TO_INDEX(18)] = 72,
                        [MBIT_TO_INDEX(24)] = 96,
                        [MBIT_TO_INDEX(36)] = 144,
                        [MBIT_TO_INDEX(48)] = 192,
                        [MBIT_TO_INDEX(54)] = 216
                };
                int rate_idx = MBIT_TO_INDEX(rate_bps/1000/1000);
#undef MBIT_TO_INDEX
                if (rate_idx < 0 ||
                    rate_idx >= sizeof(N_dbps)/sizeof(*N_dbps) ||
                    N_dbps[rate_idx] == 0)
                        return -1;

                duration_usec += 16 + 4; /* Preamble, SIGNAL */
                int bits =
                        16 +       /* SERVICE */
                        length * 8 +
                        6;         /* tail bits */
                int Nsym = (bits + N_dbps[rate_idx] - 1)/N_dbps[rate_idx];

                duration_usec += Nsym * 4;
```

```
                duration_usec += 6; /* signal extension */
        }
        return duration_usec;
}

int fwp_admctrl_utilization(struct fres_sa_scenario *scenario, void *priv,
                            bool *schedulable)
{
        long long utilization = 0;
        struct frm_fwp_priv * pr = priv;
        int rate = 1*1000*1000; //default initialization is 1 Mbps //= pr-
>rate_mbps*1000*1000;
        struct fres_sa_contract *c;
        long int period_usec;
        fres_block_fwp_sched *fwp_sched;
        size_t bytes;
        long int duration_usec, tmp_usec;
        int fragments;
        int ac = FWP_AC_VO;
        fwp_sta_t *sta;

        fres_sa_scenario_for_each_no_cancel_contract(scenario, c) {
                fres_block_basic *basic;
                fres_block_fwp *fwp_block;
                char id[40];
                fres_contract_id_to_string(id, &c->contract->id, sizeof(id));
                basic = fres_contract_get_basic(c->contract);
                 fwp_block = fres_contract_get_block(c->contract, FRES_BLOCK_FWP);

                if(fwp_block != NULL){
                        /* from list of all stations choose the one which has requested
mac address */
                        ul_list_for_each(sta_list, &pr->sta_list, sta){
                                /* if there's a requested mac address in node */
                                if(sta->client_mac_addr == fwp_block->mac_address){
                                        rate = sta->rate*1000*1000;
                                        break;
                                }
                        }
                }
                frsh_network_budget_to_bytes(FRSH_NETPF_FWP,&basic->budget,&bytes);
                ul_logmsg("processing: id=%s, period=%ld ms, budget=%lu bytes\n",
                        id,
                        fosa_rel_time_to_msec(basic->period),
                        (long unsigned int)bytes);

                period_usec = basic ? fosa_rel_time_to_msec(basic->period)*1000 : 0;
                /* Calculate protocol overhead */
                fragments = (bytes + MTU - 1) / MTU;

                if (fragments == 0 && period_usec != 0)
                        continue;
                const int data_overhead = UDP_HEADER_SIZE + IP_HEADER_SIZE +
                        LLC_HEADER_SIZE + MAC_FCS_SIZE;
                duration_usec = frame_duration(data_overhead + bytes%MTU, rate,
                                               pr->erp_ofdm, pr->short_preamble);
                tmp_usec = frame_duration(data_overhead + MTU, rate,
                                          pr->erp_ofdm, pr->short_preamble);
                duration_usec += tmp_usec*(fragments-1);
                /* Add average backoff - assume there is no collision */
                tmp_usec = ASLOTTIME_USEC*fragments*pisvejc[ac];
                duration_usec += (aifsn[ac] + cwmin[ac]/2)*tmp_usec;
                /* We use ACK and ignore burst */
                tmp_usec = frame_duration(ACK_FRAME_SIZE, rate,
                                          pr->erp_ofdm, pr->short_preamble)
                        + ASIFSTIME_USEC;
                duration_usec += fragments * tmp_usec;
                //printf("duration: %ld ms\n", duration_usec/1000);
                /* TODO: If STA-to-STA, multiply it by two. Note that
                 * AP may use different values for backoff. */
                duration_usec *= 2; /* For demo, we have always STA-to-STA */
                if (c->contract == c->new) {
                        if (period_usec == 0) {
                                ul_logmsg("Period is zero!\n");
                                goto not_schedulable;
                        }
```

```
                        frsh_rel_time_t deadline;
                        long int d;
                        if (fres_contract_get_deadline(&c->contract, &deadline)) {
                                d = fosa_rel_time_to_msec(deadline)*1000;
                        } else {
                                d = 100/*sec*/*1000*1000;
                        }
                        if (d < 30/*msec*/*1000) {
                                ul_logmsg("Deadline shorter than 30 ms!\n");
                                goto not_schedulable;
                        };
                        fwp_sched = malloc(sizeof(*fwp_sched));
                        fwp_sched->ac_id =
                                d <  100*1000 ? FWP_AC_VO :
                                d <  500*1000 ? FWP_AC_VI :
                                d < 1000*1000 ? FWP_AC_BE :
                                FWP_AC_BK;
                        fres_contract_add_block(c->contract, FRES_BLOCK_FWP_SCHED,
fwp_sched);
                }
                utilization += (long long)duration_usec * 10000 / period_usec;
        }
        if (utilization >= 10000 * 96/100 && !pr->bypass) {
                goto not_schedulable;
        }
        scenario->utilization = utilization/100; /* For GUI */
        ul_logmsg("accepted (utilization=%lld%%)\n", utilization/100);
        *schedulable = true;
        return 0;
not_schedulable:
        ul_logmsg("rejected (utilization=%lld%%)\n", utilization/100);
        *schedulable = false;
        return 0;
}
```

4.2 Zhromažďovanie informácií o stave siete

Hlavičkový súbor `\frsh-forb\src\fwp\fwp\mngr\wifi_agent.h` s definovanými funkciami manažujúcimi spájaný zoznam s údajmi staníc.

```
#ifndef WIFI_AGENT_H
#define WIFI_AGENT_H

#include "wifi_agent_idl.h"
#include "fcb.h"

#define WAI_SERVER_ID (forb_server_id){ .uuid = { 0x22, 0x22, 0x22, 0x22, 0x22, 0x22,
0x22, 0x22 } }
#define WAI_TCP_PORT 24943

extern struct forb_wifi_agent_idl_impl wifi_agent_impl;
CORBA_Environment env;
fres_contract_broker fcb;

#endif
```

Samotné funkcie, ako sú definované v súbore `\frsh-forb\src\fwp\fwp\mngr\wifi_agent.c`

```
/*
 *      @brief server side (frm_fwp) functions for communication
 *      resource manager - station
 */

#include <frm_generic.h>
#include <forb.h>
#include <error.h>
#include <errno.h>
#include <getopt.h>
#include <fres_sa_scenario.h>
```

```
#include <stdbool.h>
#include <ul_list.h>
#include <ul_log.h>
#include <ul_logreg.h>
#include <fwp_res.h>
#include <stdio.h>
#include <string.h>
#include "wifi_agent_idl.h"
#include "fwp_admctrl.h"
#include "wifi_agent.h"

struct forb_wifi_agent_idl_impl wifi_agent_impl = {
        .add = wifi_agent_idl_add,
        .print_state = wifi_agent_idl_print_state,
};

/**
 *      Print list of stored stations
 */
void wifi_agent_idl_print_state(wifi_agent_idl _obj, CORBA_Environment *ev)
{
        fwp_sta_t *sta2;

        dprintf(2, "Actual_list_of_stations\n");
        ul_list_for_each(sta_list, &priv.sta_list, sta2){
                dprintf(2, "%llu_-_%d\n", sta2->client_mac_addr, sta2->rate);
        }
        dprintf(2, "------------\n\n");

}

/**
 *      Add information about AP (mac_address & bit rate)
 */
CORBA_long wifi_agent_idl_add(wifi_agent_idl _obj, const CORBA_long rate, const COR-
BA_long_long client_mac_addr, CORBA_Environment *ev)
{
        struct fwp_sta *ed = forb_instance_data(_obj);
        fwp_sta_t *sta;
        fwp_sta_t *new_to_add;

        new_to_add = (fwp_sta_t*)malloc(sizeof(fwp_sta_t));

        /*client transmission bitrate (wifi_agent -> fwp_manager)*/
        ed->rate = rate;
        ed->client_mac_addr = client_mac_addr;

        /*fill the linked list here*/
        ul_list_for_each(sta_list, &priv.sta_list, sta){

                /*client was sending info before*/
                if(sta->client_mac_addr == ed->client_mac_addr) {
                        /*mac address is the same change only rate*/
                        sta->rate = ed->rate;

                        return 0;
                }
        }
        /*client was not sending info yet*/
        new_to_add->rate = ed->rate;
        new_to_add->client_mac_addr = ed->client_mac_addr;

        sta_list_ins_tail(&priv.sta_list, new_to_add);

        return 0;
}
```

4.3 Zisťovanie stavu aktuálnej prenosovej rýchlosti

Súbor \frsh-forb\src\fwp\fwp\wifi_agent\wifi_agent.c

```
/*
 *      @brief: Program finds out mac address and tx bitrate of station
```

```
 *      and sends it to frm_fwp manager
 *
 */

#include "wifi_agent.h"
#include <string.h>
char mac_addr_glob[20];
int rate_glob;

void mac_addr_n2a(char *mac_addr, unsigned char *arg)
{       int i, l;
        l = 0;
        for (i = 0; i < ETH_ALEN ; i++) {
                if (i == 0) {
                        sprintf(mac_addr+l, "%02x", arg[i]);
                        l += 2;
                } else {
                        sprintf(mac_addr+l, ":%02x", arg[i]);
                        l += 3;
                }
        }
}

#ifndef CONFIG_LIBNL20
/* libnl 2.0 compatibility code */
static inline struct nl_handle *nl_socket_alloc(void)
{
        return nl_handle_alloc();
}
static inline void nl_socket_free(struct nl_sock *h)
{
        nl_handle_destroy(h);
}
static inline int __genl_ctrl_alloc_cache(struct nl_sock *h, struct nl_cache **cache)
{
        struct nl_cache *tmp = genl_ctrl_alloc_cache(h);
        if (!tmp)
                return -ENOMEM;
        *cache = tmp;
        return 0;
}
#define genl_ctrl_alloc_cache __genl_ctrl_alloc_cache
#endif /* CONFIG_LIBNL20 */

/*
 *      nl80211 functions
 */
static int nl80211_init(struct nl80211_state *state)
{       int err;
        state->nl_sock = nl_socket_alloc();
        if (!state->nl_sock) {
                fprintf(stderr, "Failed to allocate netlink socket.\n");
                return -ENOMEM;
        }
        if (genl_connect(state->nl_sock)) {
                fprintf(stderr, "Failed to connect to generic netlink.\n");
                err = -ENOLINK;
                goto out_handle_destroy;
        }
        if (genl_ctrl_alloc_cache(state->nl_sock, &state->nl_cache)) {
                fprintf(stderr, "Failed to allocate generic netlink cache.\n");
                err = -ENOMEM;
                goto out_handle_destroy;
        }
        state->nl80211 = genl_ctrl_search_by_name(state->nl_cache, "nl80211");
        if (!state->nl80211) {
                fprintf(stderr, "nl80211 not found.\n");
                err = -ENOENT;
                goto out_cache_free;
        }
        return 0;
 out_cache_free:
        nl_cache_free(state->nl_cache);
 out_handle_destroy:
        nl_socket_free(state->nl_sock);
        return err;
}
```

```
static void nl80211_cleanup(struct nl80211_state *state)
{
        genl_family_put(state->nl80211);
        nl_cache_free(state->nl_cache);
        nl_socket_free(state->nl_sock);
}

/*
 *      handler functions
 */
static int error_handler(struct sockaddr_nl *nla, struct nlmsgerr *err,
                        void *arg)
{       int *ret = arg;
        *ret = err->error;
        return NL_STOP;
}
static int finish_handler(struct nl_msg *msg, void *arg)
{       int *ret = arg;
        *ret = 0;
        return NL_SKIP;
}
static int ack_handler(struct nl_msg *msg, void *arg)
{       int *ret = arg;
        *ret = 0;
        return NL_STOP;
}

/*
 *      Mac address conversion functions
 */
/**
 *      Char to number conversion
 */
int atoint(char a)
{       if( (a >= '0') && (a <= '9') )
                return a - '0';
        else if ( (a >= 'A') && (a <= 'F') )
                return a - 'A' + 10;
        else if ( (a >= 'a') && (a <= 'f') )
                return a - 'a' + 10;
        return 0;
}

/**
 *      Mac addres char to number conversion
 *      @param mac mac address in string format
 *
 *      @return Mac address in long long format
 */
long long mac_addr_string_to_number(char * mac)
{       long long number;
        long long help;
        int ix;
        int ix2;
        char mac_num[12];
        mac_num[0] = mac[0];  mac_num[1] = mac[1];
        mac_num[2] = mac[3];  mac_num[3] = mac[4];
        mac_num[4] = mac[6];  mac_num[5] = mac[7];
        mac_num[6] = mac[9];  mac_num[7] = mac[10];
        mac_num[8] = mac[12]; mac_num[9] = mac[13];
        mac_num[10] = mac[15];mac_num[11] = mac[16];

        number = 0;
        for(ix = 0; ix <= 12 ; ix++)
        {
                help = 1;
                for(ix2 = 1; ix2 < 12 - ix; ix2++)
                        help *= 16;
                number += atoint(mac_num[ix]) * help;

        }
        printf("%llu \n", number);
        return number;
}

/*
 *      program functions
```

```
 */
static int print_data(struct nl_msg *msg, void *arg)
{
        struct nlattr *tb[NL80211_ATTR_MAX + 1];
        struct genlmsghdr *gnlh = nlmsg_data(nlmsg_hdr(msg));
        struct nlattr *sinfo[NL80211_STA_INFO_MAX + 1];
        struct nlattr *rinfo[NL80211_RATE_INFO_MAX + 1];
        char mac_addr[20], dev[20];
        static struct nla_policy stats_policy[NL80211_STA_INFO_MAX + 1] = {
                [NL80211_STA_INFO_INACTIVE_TIME] = { .type = NLA_U32 },
                [NL80211_STA_INFO_RX_BYTES] = { .type = NLA_U32 },
                [NL80211_STA_INFO_TX_BYTES] = { .type = NLA_U32 },
                [NL80211_STA_INFO_RX_PACKETS] = { .type = NLA_U32 },
                [NL80211_STA_INFO_TX_PACKETS] = { .type = NLA_U32 },
                [NL80211_STA_INFO_SIGNAL] = { .type = NLA_U8 },
                [NL80211_STA_INFO_TX_BITRATE] = { .type = NLA_NESTED },
                [NL80211_STA_INFO_LLID] = { .type = NLA_U16 },
                [NL80211_STA_INFO_PLID] = { .type = NLA_U16 },
                [NL80211_STA_INFO_PLINK_STATE] = { .type = NLA_U8 },
                [NL80211_STA_INFO_TX_RETRIES] = { .type = NLA_U32 },
                [NL80211_STA_INFO_TX_FAILED] = { .type = NLA_U32 },
        };
        static struct nla_policy rate_policy[NL80211_RATE_INFO_MAX + 1] = {
                [NL80211_RATE_INFO_BITRATE] = { .type = NLA_U16 },
                [NL80211_RATE_INFO_MCS] = { .type = NLA_U8 },
                [NL80211_RATE_INFO_40_MHZ_WIDTH] = { .type = NLA_FLAG },
                [NL80211_RATE_INFO_SHORT_GI] = { .type = NLA_FLAG },
        };
        nla_parse(tb, NL80211_ATTR_MAX, genlmsg_attrdata(gnlh, 0),
                  genlmsg_attrlen(gnlh, 0), NULL);

        if (!tb[NL80211_ATTR_STA_INFO]) {
                fprintf(stderr, "sta stats missing!\n");
                return NL_SKIP;
        }
        if (nla_parse_nested(sinfo, NL80211_STA_INFO_MAX,
                             tb[NL80211_ATTR_STA_INFO],
                             stats_policy)) {
                fprintf(stderr, "failed to parse nested attributes!\n");
                return NL_SKIP;
        }
        mac_addr_n2a(mac_addr, nla_data(tb[NL80211_ATTR_MAC]));
        if_indextoname(nla_get_u32(tb[NL80211_ATTR_IFINDEX]), dev);
        printf("%s (%s)", mac_addr, dev);
        strcpy(mac_addr_glob, mac_addr);
        if (sinfo[NL80211_STA_INFO_TX_BITRATE]) {
                if (nla_parse_nested(rinfo, NL80211_RATE_INFO_MAX,
                                     sinfo[NL80211_STA_INFO_TX_BITRATE], rate_policy)) {
                        fprintf(stderr, "failed to parse nested rate attributes!\n");
                } else {
                        printf("\ttx bitrate:\t");
                        if (rinfo[NL80211_RATE_INFO_BITRATE]) {
                                int rate = nla_get_u16(rinfo[NL80211_RATE_INFO_BITRATE]);
                                printf("%d.%d MBit/s", rate / 10, rate % 10);
                                rate_glob = rate;
                        }
                }
        }
        printf("\n");
        return NL_SKIP;
}


static int get_tx_bitrate(struct nl80211_state *state)
{       struct nl_cb *cb;
        struct nl_msg *msg;
        int devidx = 0;
        int err;
        devidx = if_nametoindex("wlan0");
        if (devidx == 0)
                devidx = -1;
        if (devidx < 0)
                return -errno;
        msg = nlmsg_alloc();
        if (!msg) {
                fprintf(stderr, "failed to allocate netlink message\n");
                return 2;
```

```
        }
        cb = nl_cb_alloc(NL_CB_DEFAULT);
        if (!cb) {
                fprintf(stderr, "failed to allocate netlink callbacks\n");
                err = 2;
                goto out_free_msg;
        }
        genlmsg_put(msg, 0, 0, genl_family_get_id(state->nl80211), 0,/*cmd-
>nl_msg_flags*/700, NL80211_CMD_GET_STATION, 0);
        NLA_PUT_U32(msg, NL80211_ATTR_IFINDEX, devidx);
        nl_cb_set(cb, NL_CB_VALID, NL_CB_CUSTOM, print_data, NULL);
        err = 0;
        if (err)
                goto out;
        err = nl_send_auto_complete(state->nl_sock, msg);
        if (err < 0)
                goto out;
        err = 1;
        nl_cb_err(cb, NL_CB_CUSTOM, error_handler, &err);
        nl_cb_set(cb, NL_CB_FINISH, NL_CB_CUSTOM, finish_handler, &err);
        nl_cb_set(cb, NL_CB_ACK, NL_CB_CUSTOM, ack_handler, &err);
        while (err > 0)
                nl_recvmsgs(state->nl_sock, cb);
 out:
        nl_cb_put(cb);
 out_free_msg:
        nlmsg_free(msg);
        return err;
 nla_put_failure:
        fprintf(stderr, "building message failed\n");
        return 2;
}

void delay(int i)
{
        int x;
        i *= 1000;
        for(x = 0; x < i; x++)
                ;
}
int main(int argc, char **argv)
{
        struct nl80211_state nlstate;
        int err;
        forb_orb fwp_orb;
         wifi_agent_idl wai;
         CORBA_Environment env;
        fwp_orb = forb_init(&argc, &argv, NULL); /* FORB initialization */
        /* Find our implementation */
         wai = forb_resolve_reference(fwp_orb, "net.sourceforge.frsh-forb.wai");
        if(wai == NULL)
                fprintf(stderr, "failed to create connection\n");
        err = nl80211_init(&nlstate);
        if (err)
                return 1;
        while(1)
        {       err = get_tx_bitrate(&nlstate);
                if (err < 0)
                        fprintf(stderr, "command failed: %s (%d)\n", strerror(-err), err);
                /* Call server functions and send wifi info */
                if(wai != NULL)
                {       printf("%d \n", rate_glob);
                        wifi_agent_idl_add(wai, rate_glob / 10,
mac_addr_string_to_number(mac_addr_glob), &env);
                        //wifi_agent_idl_print_state(wai, &env);
                }
                //TODO: Optimalize sending interval
                delay(10000);
        }
        nl80211_cleanup(&nlstate);
        return err;
}
```

Použité zdroje

[1] FRSH/FORB, domovská stránka projektu
http://frsh-forb.sourceforge.net/

[2] SOJKA M.: Resource Reservation and Analysis in Heterogeneous and Distributed Real-Time Systems, České vysoké učení technické v Praze, 2010. 112 s.

[3] FRSH/FORB, domovská stránka projektu, vzorové schémy
http://sourceforge.net/projects/frsh-forb/

[4] SOJKA M. et al.: Frescor, Framework for Real-time Embedded Systems based on COntRacts, České vysoké učení technické v Praze, 2008. 55 s.

[5] AYUSO P. N. et al.: Communicating between the kernel and user-space in Linux using Netlink sockets, University of Seville, 23.9.2002, 17 s.
http://1984.lsi.us.es/~pablo/docs/spae.pdf

[6] Wvtest, domovská stránka projektu
http://www.ohloh.net/p/wvtest

[7] Netlink Protocol Library Suite (libnl), dokumentácia ku knižnici
http://www.infradead.org/~tgr/libnl/

[8] Linux Wireless wiki, dokumentácia
http://linuxwireless.org/

[9] Dokumentácia k iw, domovská stránka projektu
http://linuxwireless.org/en/users/Documentation/iw

[10] uLan Utilities Library (ULUT), Píša Pavel, 2003-2006, dokumentácia ku knižnici
http://cmp.felk.cvut.cz/~pisa/ulan/ulut.pdf

[11] Tutoriál k IDL, Interface Definition Language, kolektív autorov
http://jmvidal.cse.sc.edu/csce590/spring02/corba-idl-intro.pdf

[12] Tutoríal ku COBRA, kolektív autorov, posledná zmena 17.10.2011
http://www.omg.org/gettingstarted/corbafaq.htm