# Brezžični senzorji III

Sistemi Daljinskega Vodenja – Laboratorijske Vaje

1. december 2011

**Povzetek**

V tej nalogi se bomo spoznali z metodo za aktivno odkrivanje vstopne točke (angl. gateway) z brezžičnim vozliščem. Napisali bomo program, ki bo vozlišču omogočil preklapljanje med komunikacijskimi kanali, na vsakem kanalu bo čakal eno sekundo, in po njem poizvedoval po vstopni točki. V primeru odgovora vstopne točke si bo zapomnil njene podatke (port, itd.) in prenehal z iskanjem. Če od vstopne točke ne bo dobil odgovora v eni sekundi, bo preklopil na nov kanal in nadaljeval s poizvedovanjem. Vozlišče bo hkrati preverjalo ali se mu javlja prava vstopna točka. Vse dogajanje na vozlišču bomo spremljali preko serijskega terminala s programom cuteCom, izhajali pa bomo iz programa, ki ste ga napisali za prejšnjo nalogo. Naloga je razdeljena v več delov, ki so zaključene celote. Z vsakim delom se boste spoznali z enim od pomembnih elementov tipične implementacije odkrivanja vstopnih točk na brezžičnih vozliščih. Pomagajte si z vso literaturo, ki je povzeta na koncu teh navodil, in z rezultati vaših predhodnih nalog. Napišite poročilo, v katerem opišete kaj ste počeli in probleme s katerimi ste se srečevali. Poročilo morate oddate natisnjeno na naslednjih vajah. V Kolikor ne utegnete oddati v roku, se bo izhodiščna ocena tega poročila znižala na 8.

## 1 Zagon preko vmesnika VMWare

Najprej zaženite Kubuntu preko virtualnega stroja VMWare in v njem odprite Terminal:

1. Zaženite VMPlayer s klikom na ikono na vašem namizju. Prikaže se vmesnik VMWare kot na Sliki 1 za izbiro datoteke, ki vsebuje vaš operacijski sistem. Izberite ikono "Open"in odprite datoteko z operacijskim sistemom Kubuntu, oziroma, kliknite kar na napis
   `freshveem Kubuntu 9.04` pod "Recent Virtual Machines".
2. Ko se Kubuntu zažene, vas vpraša za username/password. Vsi imate v izhodišču enako uporabniško ime "SDVx" in geslo "SDVx" (brez navednic). !Pazite, da vpišete številko *vašega cikla* namesto "x"!
3. Ko se naloži namizje Kubuntu, lahko nastavite resolucijo (velikost namizja) s klikom na gumb za orodno vrstico v spodnjem levem kotu (ikona kjer je črka "K"). Nato izberite "System Settings", pod "Computer Administration" izberite "Display" in nastavite resolucijo v okencu "Size".
4. Sedaj najprej lahko zaženemo terminalski vmesnik "Terminal". Ponovno zaženite orodno vrstico preko ikone s črko "K" v spodnjem levem kotu. Odpre se okno z večimi zavihki (kot pri Windows "start").
5. Terminal lahko najdete preko grafičnih zavihkov kot je to običajno v operacijskem sistemu Windows, zaradi hitrejšega iskanja pa uporabite okence "Search" v prvem zavihku. V polje pod "Search" vpišite "Terminal".


Slika 1: Primer vmesnika VMWare .

# 2 Dogodki in odzivanje na vstopno točko (30%)

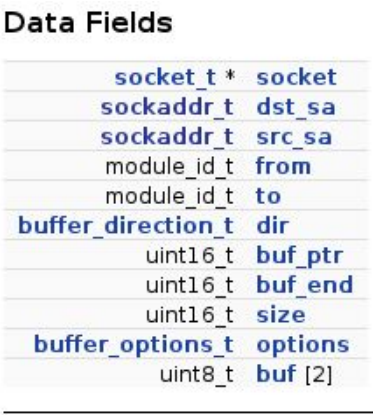
Motivacija: Če želimo pošiljati in prejemati pakete z vozlišča preko vstopne točke (angl. gateway), moramo najprej vzpostaviti komunikacijo med našim vozliščem in vstopno točko. Vstopna točka se fizično nahaja na nano usmerjevalniku – nanoRouterju. Sklad NanoStack nam omogoča vzpostavljanje komunikacije preko vmesnikov, ki implementirajo ICMP (Internet Cotrol Message Protocol). ICMP je nabor sporočil, katera lahko na vozlišču prejemamo preko tako imenovanih dogodkov (angl. event). Kot prvi korak k iskanju in sledenju vstopne točke bomo zato implementirali okvir kode, ki bo omogočala delo z dogodki.

Zahtevana funkcionalnost: Brezžično vozlišče bomo sprogramirali tako, da bo na *ročno izbranem* komunikacijskem kanalu oddajalo signal za iskanje vstopne točke (VT). VT se bo odzvala na ta signal s sporočilom ICMP, v katerem bo posredovala svoje podatke. Sporočilo ICMP bo vozlišče prejelo in sprožilo dogodek. Naš program bo dogodek prestregel, ga prebral in izpisal naslov MAC VT na CuteComov terminal. Naslov MAC si napišite na list (tokrat je v pravilnem vrstnem redu in vsi znaki veljajo), ker ga boste rabili pri naslednji nalogi.

Naloga: Za izhodišče bomo vzeli program, ki smo ga napisali v prejšnji nalogi (ex0). Preden začnete s programiranjem, si preberite Poglavje 3.1, 3.2 v [2]. Poglejte si tudi primere za inicializacijo dogodkov v [2] na strani 10 in 15, vendar primera uporabljajte samo kot vodilo pri nalogi[1]. Z naslednjimi koraki bomo implementirali funkcionalnost, ki jo zahteva naloga: (i) najprej bomo definirali spremenljivko za dogodke, (ii) jo inicializirali, (iii) implementirali poslušanje dogodkov in (iv) analizo vsebine prestreženega dogodka:

1. Definicija: Definirajte **globalno spremenljivko** `stack_event_t stack_dogodki`, ki bo hranila vse dogodke.
2. Inicializacija: Pred vstopom v neskončno zanko v funkciji `mojaGlavnaFunkcija()` pokličite funkcijo `stack_dogodki = stack_service_init(NULL)` za inicializacijo dogodkov na skladu. Natančnejši opis funkcije najdete v [1] (`Files->Globals->Functions`).
3. Prestrezanje: Vstavite kodo za poslušanje in analizo dogodkov[2] tako, da sledite spodnjim točkam:
   - Definirajte globalno spremenljivko (kazalec na polje), v katero bomo shranjevali prihajajoče dogodke – definirajte globalni kazalec `moj_buffer` tipa `buffer_t` [3]. Tip `buffer_t` je struktura in vsebuje polja, kot prikazuje Slika 2.

[1] Ne dobesedno prepisovati, ker primeri vsebujejo napake in odvečno kodo.
[2] Primer lahko najdete v [2], stran 10.
[3] Torej, buffer_t * moj_buffer.




Slika 2: Polja, ki jih vsebuje struktura tipa buffer_t. Povzeto po [1].

- Koda za analizo dogodkov se začne s klicem funkcije, ki pogleda, če je med delovanjem prišlo do kakšnega dogodka. V neskončni zanki ste na prejšnji vaji napisali del za komunikacijo z UART. Takoj za to kodo (in pred pavziranjem za 500ms) dodajte klic `moj_buffer = waiting_stack_event(100)`, ki 100 milisekund posluša, če se je pojavil kak dogodek. Če se na skladu ni pojavil dogodek, bo funkcija vrnila vrednost `0`. V primeru, da se je na skladu pojavil dogodek, pa bo funkcija *rezervirala prostor v pomnilniku* za polje tipa `buffer_t` in vam vrnila kazalec nanj. Ta kazalec morate analizirati, po analizi pa *moramo sami sprostiti spomin*. To v našem primeru storimo s klicem `stack_buffer_free(moj_buffer)`. V [1] `Data Structures->Data Structures` (oziroma v Sliki 2) si poglejte vsebino strukture `buffer_t`. Vidimo, da struktura vsebuje mnoga polja, vključno s podatki, ki so vsebovani v dogodku. Pomembno si je zapomniti, da so v polju `src_sa` vpisani podatki o naslovu MAC vstopne točke.
- V primeru, da nam funkcija `waiting_stack_event(100)` vrne kazalec na polje z dogodkom, moramo ugotoviti kaj ta dogodek predstavlja. V naši aplikaciji nas bo zanimalo, ali smo odkrili VT, oziroma, ali smo jo izgubili. NanoStack nam omogoča preprosto analizo z uporabo funkcije `parse_event_message(moj_buffer)`, ki nam vrne značko dogodka `moj_buffer`. V primeru dogodka `našel sem VT` funkcija vrne `ROUTER_DISCOVER_RESPONSE`, v primeru dogodka `izgubil sem VT` pa vrne `BROKEN_LINK`. [4]; za več značk lahko pogledate [1] ali [2], Tabeli 11 in 12.
- Vaša koda naj v primeru, da se na dogodku nahaja značka `ROUTER_DISCOVER_RESPONSE` izpiše preko UART `Nasel sem nov gateway!`[5], in izpiše naslov MAC VT. Naslov MAC se nahaja v polju `src_sa` vaše strukture `moj_buffer`, torej `moj_buffer->src_sa`. Za izpis naslova MAC na UART uporabite funkcijo `void debug_address(sockaddr_t *)`. Pazite na dejstvo, da funkcija prejme *kazalec* na polje tipa `sockaddr_t`![6] V primeru, da se na dogodku pojavi značka `BROKEN_LINK`, naj izpiše na UART `Izgubil sem gateway!`. Uporabite `switch/case` stavek za realizacijo zgornje funkcionalnosti; izhajajte iz primera v [2], poglavje 3.1, na strani 8, točka 3.

4. Aktivno iskanje: Za namene razhroščevanja dodajte klic `gw_discover()` tik pred klicem `waiting_stack_event(100)`. Ta funkcija vsakič generira in odda sporočilo, na katerega se VT, ki posluša na istem kanalu, odzove z ICMP RSM sporočilom

[4] Značke `ROUTER_DISCOVER_RESPONSE`, `BROKEN_LINK` so samo globalno predefinirane celoštevilske spremenljivke za lažjo interpretacijo vsebine sporočila.

[5] Uporabite funkcijo `debug();` kot ste storili v nalogi Brezžični Senzorji II.

[6] Torej morate funkcijo podati *referenco* na spremenljivko: `&(moj_buffer->src_sa)`.

o svoji prisotnosti. To sporočilo bo na našem vozlišču sprožilo dogodek, ki ga prestrežemo z `waiting_stack_event(100)`.

5. Testiranje: Priklopite programator na USB port, na programator pa izbrano vozlišče. Prevedite program in sprogramirajte vozlišče. Pozor: Pred programiranjem morate odklopiti morebitno komunikacijo s CuteCom. Preverite tudi, če imate v `app.rules` izbrani pravi USB port (tisti na katerega je priklopljen programator).
6. Iz kovčka vzemite nano usmerjevalnik in ga priklopite v USB port. Skonfigurirajte in zaženite `nRouted` (kot ste storili v nalogi "Brezžični Senzorji I"). Zaženite CuteCom in se povežite z vozliščem. Ročno nastavite (s tipkami 'c' in 'C') komunikacijski kanal vozlišča na kanal usmerjevalnika. Če ste vse storili pravilno, vam mora vozlišče izpisovati detektirano VT (njen MAC) na CuteComov terminal. Ko končate, pokličite asistenta, da pregleda program preden nadaljujete z naslednjo točko. Naslov MAC si zapišite v svoje zapiske za poročilo.

# 3 Sledenje vstopni točki (50%)

Motivacija: Sedaj bomo implementirali celotno avtomatsko sledenje in odkrivanje lastne vstopne točke. V prejšnjem delu naloge smo napisali kodo, s katero prestrezamo dogodke o najdenih in izgubljenih VTjih, vendar pa ima naš program naslednje pomanjkljivosti:

1. Komunikacijski kanal na vozlišču smo vedno nastavljali ročno.
2. Program je neprestano iskal VT, četudi jo je že našel in je še ni izgubil.
3. Nismo preverjali ali je vstopna točka, ki smo jo detektirali res tista, ki jo iščemo, ali je morda to vstopna točka vaših sosedov.

Nova funkcionalnost: Sedaj bomo spremenili obstoječo programsko kodo in dodali vozlišču naslednjo funkcionalnost:

1. Vozlišče bo periodično preklapljalo med komunikacijskimi kanali, dokler ne zazna prisotnost vstopne točke. Med kanali bo vozlišče preklapljalo s periodo ene sekunde, dokler ne odkrije vstopne točke. Če je zaznana vstopna točka res naša (ima pravilen MAC naslov), potem vozlišče ostane na izbranem komunikacijskem kanalu in preneha z iskanjem. Sicer nadaljuje s preklaplanjem.
2. V primeru, da vstopne točke še nismo odkrili in prestrežemo dogodek `ROUTER_DISCOVER_RESPONSE`, si bomo zapomnili njene podatke (port, itd.). Če pa smo že odkrili vstopno točko, pa bomo ignorirali to sporočilo. V primeru, da prestrežemo sporočilo `BROKEN_LINK`, bomo zopet aktivirali iskanje vstopne točke. To se nanaša na dodajanje nove kode v dele stavkov `switch/case` pod "`case ROUTER_DISCOVER_RESPONSE:`" in "`case BROKEN_LINK:`".

Naloga: Začnimo z implementacijo. Najprej se bomo posvetili (i) kodi za periodično preklapljanje med komunikacijskimi kanali, nato pa (ii) kodi za analizo vsebine dogodkov.
Koda za periodično preklapljanje med kanali:

- Iskanje VT smo v prejšnjem delu naloge implementirali s preprosto vrstico `gw_discover()`. S tem je vozlišče neprestano pošiljalo sporočila ICMP RSM. Zgledujte se po psevdokodi v Algoritmu 1, in namesto vrstice `gw_discover()` napišite ustrezno kodo, ki preverja ali smo v fazi iskanja VT. Če smo res v fazi skeniranja kanalov, potem naj program preveri, če je od zadnjega preklopa kanala minila več kot *ena sekunda*. Če je, potem naj preklopi na nov kanal in resetira časovnik.
  Namig: Definirajte globalne spremenljivke `portTickType to, t1 ;` kamor boste shranjevali število procesorskih ciklov (čas) in globalno spremenljivko, `uint8_t skeniraj`, ki vam bo označevala ali ste v fazi iskanja vstopne točke ali ne.

- Ko napišete ta del programa, poskusite prevesti, (`make`), da vidite ali se vaša koda sploh prevede.

**Algorithm 1** Psevdokoda za periodično preklapljanje med komunikacijskimi kanali.

```
1: if skeniraj == 1 then
2:    t_1 ← xTaskGetTickCount();
3:    if t_1 − t_0 > 1sekunda then
4:       Preklopi na nov kanal
5:       Pošlji sporočilo ICMP RSM (gw_discover();)
6:       t_0 ← t_1;
7:    end if
8: end if
```

Koda za analizo vsebine dogodkov: Implementirali bomo psevdokodo, ki je prikazana v Algoritmu 2. Naša trenutna koda omogoča detekcijo dogodka z vsebino `ROUTER_DISCOVER_RESPONSE`. Vstopna točka nam sporoči tudi svoj naslov MAC, ki smo ga izpisali na ekran. Podatki o naslovu MAC so shranjeni v `moj_buffer->src_sa`.

Če si pogledamo specifikacije na [1] vidimo, da je spremenljivka `src_sa` tipa `sockaddr_t`. Na [1] poglejmo vsebino tipa `sockaddr_t` in vidimo, da gre za strukturo, ki vsebuje tri spremenljivke:

- `addr_type` ... v našem primeru tip naslova vstopne točke,
- `address` ... v našem primeru naslov vstopne točke,
- `port` ... v našem primeru port na katerem vstopna točka posluša.

V [2], Poglavje 4.4, najdemo specifikacijo strukture `sockaddr_t`, kjer lahko tudi najdete možne značke in tipe naslovov (`addr_type`). S spodnjimi koraki bomo dodali zahtevano funkcionalnost:

- Definicija in inicializacija: Najprej bomo spremenili program tako, da si ob izpisu naslova VT njene podatke tudi zapomnemo. Zato najprej definirajte globalno spremenljivko za hranjenje naslova `naslov_gw` tipa `sockaddr_t` – globalno spremenljivko bomo hkrati inicializirali takole:

```
sockaddr_t naslov_gw =
{
    ADDR_802_15_4_PAN_LONG,
    {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
    61630
};
```

  Zgornja koda postavi `addr_type` na 64-bitni naslov po protokolu 802.15.4, vpiše v `address` *splošni naslov* `{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}` in postavi `port` na 61630.

- Implementirati moramo tudi preverjanje identitete vstopne točke preko njenega naslova MAC. Definirajte gobalno spremenljivko, in vanjo vpišite vseh deset heksadecimalnih številk naslova MAC vašega usmerjevalnika:

```
address_t moj_router = {0x01, 0xAD, 0x05, itd. } ;
```

- Sedaj lahko v "`case ROUTER_DISCOVER_RESPONSE:`", po kodi ki izpiše na UART detektirane naslove, vstavite kodo, ki preveri, če je MAC naslov detektirane VT enak naslovu vašega usmerjevalnika. To lahko storite s funkcijo
  `int memcmp(void *s1, void s2, int num_bytes)`, ki preveri ali prvih `num_bytes` bajtov v polju `s1` ustreza prvim `num_bytes` v polju `s2`. Če sta polji enaki, vrne `0`. Primer uporabe:
  `memcmp(moj_router, &(moj_buffer->src_sa.address), sizeof(address_t))`

- Kopiranje naslova MAC: Če je vstopna točka res naša, potem si zapomnimo njene podatke tako, da kopiramo vsebino `moj_buffer->src_sa` v `naslov_gw`: `memcpy(&naslov_gw, &(moj_buffer->src_sa), sizeof(sockaddr_t))`
- Ustavimo skeniranje: Ko si zapomnete naslov vstopne točke, ustavite skeniranje. Na primer tako, da postavite vaš indikator za skeniranje na nič. Prav tako sporočite preko UART `Nasel sem nas usmerjevalnik!`. Pri implementaciji zgornjih točk si pomagajte s psevdokodo v Algoritmu 2.
- Ponovni zagon skeniranja: V primeru, da smo izgubili vstopno točko, označite, da nimamo naslova usmerjevalnika: `naslov_gw.addr_type = ADDR_NONE` (glej [2], stran 11), in ponovno zaženem skeniranje. Na primer, postavimo vrednost indikatorja za skeniranje na ena. Prav tako sporočite preko UART `Izgubil sem usmerjevalnik!`.
- Zaenkrat samo prevedite kodo, da vidite, če niste naredili kakšne sintaktične napake. *Ne programirajte vozlišča!* Če se koda prevede brez napak, nadaljujte z naslednjo točko.
- V nalogi s prejšnjih laboratorijskih vaj ste implementirali komunikacijo preko UART porta tako, da vozlišče prejema ukaze v obliki črk in se nanje odziva. Tako ste lahko, na primer, spreminjali vrednost komunikacijskega kanala vozlišča. Sedaj dodajte še kodo, ki se odzove na ukaz `'s'` tako, da ponovno zažene iskanje vstopne točke (npr., postavi indikator za skeniranje na ena).
- Testiranje: Prevedite kodo in sprogramirajte vozlišče. V enem od USB portov je še vedno priklopljen nano router. Sedaj ni več potrebno posebej preko CuteComa nastavljati vrednost komunikacijskega kanala. Vozlišče mora samo preskenirati kanale, odkriti pravi usmerjevalnik in ostati na istem kanalu. Zaženite cuteCom in poglejte, ali odkrivanje deluje pravilno. Če deluje pravilno, pokličite asistenta, da pregleda nalogo.

**Algorithm 2** Psevdokoda za analizo sporočil in zaznavanje usmerjevalnika.

```
1: moj_buffer = waiting_stack_event(100);
2: if moj_buffer! = 0 then
3:    Začetek switch stavka za parsanje vsebine moj_buffer ;
4:    case BROKEN_LINK:
5:    Sporoči, da si izgubil usmerjevalnik ;
6:        skeniraj = 1 ;
7:    naslov_gw.addr_type = ADDR_NONE ;
8:    case ROUTER_DISCOVER_RESPONSE:
9:    if skeniraj == 1 then
10:      Sporoči, da si zaznal eno od vstopnih točk ;
11:      Izpiši MAC_x vstopne točke ;
12:      if MAC_x je enak našemu usmerjevalniku then
13:        Sporoči, da si našel naš usmerjevalnik ;
14:          skeniraj = 0 ;
15:          Skopiraj podatke zaznane vstopne točke v naslov_gw ;
16:      end if
17:   end if
18:   Konec switch stavka za parsanje vsebine moj_buffer ;
19:   Sprosti polje moj_buffer ;
20: end if
```

# 4 Donatna funkcionalnost 20%

Dodajte še naslednje funkcionalnosti vašemu vozlišču:

- Ko vozlišče odkrije vašo (pravilno) vstopno točko, naj ostane ledica s številko 2 prižgana. Če izgubi vozlišče, pa naj ponovno prične utripati.
- Komunikaciji preko CuteCom dodajte možnost, da ob pritisku tipke "g" izpiše naslov MAC detektirane vstopne točke.
- Medtem, ko vozlišče išče vstopno točko in preklaplja med kanali naj izpisuje preko UART ob vsakem preklopu kanala tudi trenutno vrednost kanala.

# Literatura


[1] Sensinode, Work/Sensinode/NanoStack/NanoStack-v1.1.0/Docs/html/index.html, *The NanoStack documentation*, 0.6 ed., 2008.

[2] Sensinode Ltd., *NanoStack Reference*, 2008.