Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Mark Hočevar

Usmerjevalni algoritmi v brezžičnih senzorskih omrežjih

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana, 2018

Zahvala

Zahvaljujem se mentorju izr. prof. dr. Patriciu Buliću za strokovno pomoč in vodenje pri izdelavi magistrskega dela.
Zahvaljujem se tudi družini za podporo na moji akademski poti.

Mark Hočevar, 2018

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
EAR	Energy Aware Routing	Energetsko učinkovito usmerjanje
FEAR	Fair Energy Aware Routing	Pravično energetsko učinkovito usmerjanje
LEACH	Low-energy adaptive clustering hierarchy	Nizkoenergetske adaptivne hierarhije gruče
MTE	Minimal transmition energy	Minimalna energija za pošiljanje
NED	Network Descripton	Opis omrežja
RSSI	Received signal strength indicator	Indikator moči sprejemnega signala

Povzetek

Naslov: Usmerjevalni algoritmi v brezžičnih senzorskih omrežjih

Brezžična senzorska omrežja so omrežja, sestavljena iz več naprav za zajem podatkov iz okolice, na primer temperature. Običajno so senzorske naprave baterijsko napajane, zato morajo algoritmi za usmerjanje paketov enakomerno obremenjevati vse naprave, da ne pride do predhodnega ugašanja le-teh.

V delu smo naredili pregled obstoječih algoritmov za usmerjanje v brezžičnih senzorskih omrežjih in simulacije časa odpovedi ter primerjavo treh algoritmov: algoritma energetsko učinkovitega usmerjanja (EAR), algoritma gradientnega usmerjanja in algoritma usmerjanja na principu nizkoenergetske adaptivne hierarhije gruče (LEACH). Simulacije smo izvedli na omrežjih, ki so se razlikovala po topologiji in velikosti.

Po naših primerjalnih kriterijih so algoritmi medsebojno konkurenčni. Osredotočili smo se na primerjavo časa odpovedi posameznih vozlišč in sicer prvo odpoved, odpoved 10% omrežja in odpoved polovice omrežja. Najbolje se je odrezal algoritem LEACH, njegova slabost pa je potreba po neposredni komunikaciji vseh vozlišč s prehodom. Pri algoritmu EAR in algoritmu gradientnega usmerjanja nismo omejeni z velikostjo nadzorovanega območja, vendar moramo zagotoviti dovolj vmesnih vozlišč za pot do prehoda.

Ključne besede

internet stvari, usmerjanje v omrežju, brezžični senzorji, topologija omrežja

Abstract

Title: Routing algorithms for wireless sensor networks

Wireless sensor networks consist of multiple smart devices which are capable of sensing various information from the environment, for example temperature. Nodes used in wireless sensor networks are powered by batteries which can lead to autonomy issues. The main reason for these issues is inefficient packet routing. Some routing algorithms do not load all nodes equally. This means that some of the nodes turn off sooner than others.

In this thesis we made a survey of commonly used routing algorithms for wireless sensor networks. We used different simulations to compare algorithms. In our work we focused on three algorithms Energy Aware Routing (EAR), gradient routing and Low-energy adaptive clustering (LEACH). We checked those algorithms on different topologies and network size to check how it affects them.

Based on our criteria algorithms are comparable in our simulations. We focused on comparison between time to first death of node, time to death of 10% of nodes and time to death of 50% of nodes. We found out that the LEACH algorithm is best, but it has a disadvantage that all nodes must be able to directly communicate with the gateway. With the EAR algorithm, and gradient routing we are not limited in network size.

Keywords

internet of things, network routing, wireless sensors, network topology

Poglavje 1

Uvod

Hiter razvoj elektronike nas je pripeljal v dobo, kjer ima skoraj vsaka naprava več procesorske moči, kot jo je imel glavni računalnik pri prvem poletu na luno petdeset let nazaj. Tak napredek v elektroniki je prinesel veliko novih načinov uporabe. V zadnjih letih vedno bolj napredujejo brezžične tehnologije. Prav napredek teh nam omogoča izdelavo brezžičnih senzorskih omrežij. Brezžično senzorsko omrežje je omrežje sestavljeno iz skupine naprav, ki med seboj komunicirajo prek radijskih signalov. Njihova glavna naloga je zaznavanje različnih parametrov, kot so temperatura, vlaga, pritisk, kemične koncentracije snovi in podobno, ki jih s pomočjo brezžične komunikacije posredujejo posebni napravi imenovani prehod (ang. *gateway*). Prehod te podatke nato obdela in predstavi uporabniku. Pri izdelavi teh senzorjev oziroma vozlišč moramo upoštevati nekaj omejitev. Ponavadi je želja, da so vozlišča čim manjša in čim cenejša. Posamezno vozlišče je napajano iz majhne baterije, kar pomeni, da ima omejeno količino električne energije. Zato je izbira brezžične tehnologije zelo pomembna, saj je le-ta ponavadi glavni porabnik električne energije.

Nadzorovano področje brezžičnega senzorskega omrežja je lahko večje kot doseg brezžične komunikacije posameznega vozlišča, zato vozlišča omogočajo tudi posredovanje sporočil drugih vozlišč do prehoda. Pri tem odpremo nov problem, kako usmerjati posamezna sporočila do prehoda. Prvo vodilo usmerjanja je, da vedno uporabimo najkrajšo pot do prehoda. Tak način izbire poti prekomerno obremenjuje vedno ista vozlišča, kar pripelje do večje porabe energije na teh vozliščih in posledično hitrejše odpovedi le-teh. S pomočjo pametnih usmerjevalnih algoritmov poizkušamo čimbolj enakomerno obremeniti vsa vozlišča in s tem zagotoviti čim daljše delovanje omrežja. Brezžična senzorska omrežja so uporabna za širok nabor situacij. Kje in kako se senzorji uporabljajo, je odvisno predvsem od tega, kakšno zaznavanje omogočajo. S pomočjo zaznavanja temperature in vlažnosti zemlje so raziskovalci postavili sistem avtomatičnega namakanja, ki optimizira porabo vode v kmetijstvu [1]. Podobno so s pomočjo zaznavanja temperature in vlažnosti

ozračja, raziskovalci zgradili brezžično senzorsko omrežje, ki zazna požare v gozdovih [2]. Tako brezžično senzorsko omrežje, za zaznavanje požara v gozdovih, uporabljajo tudi v Južni Koreji [3]. Na otoku Mavricij brezžično senzorsko omrežje uporabljajo za nadziranje onesnaženosti zraka [4].

Glavna tema naše naloge je pregled različnih usmerjevalnih algoritmov. V prvem delu naloge smo analizirali različne algoritme, ki so trenutno najbolj popularni. V drugem delu se osredotočimo na primerjavo algoritmov s pomočjo simulacije. Pri tem smo poizkušali ustvariti čimbolj realne pogoje. Zanimalo nas je tudi, kako se določeni algoritmi obnašajo pri različnem številu vozlišč. Zaključimo z analizo in našim komentarjem rezultatov simulacij.

Poglavje 2

Pregled usmerjevalnih algoritmov

Usmerjevalni algoritmi so algoritmi, ki skrbijo za usmerjanje sporočil v omrežjih. Vsakodnevno uporabljamo usmerjevalna algoritma EGP (Exterior Gateway protocol) [5] in RIP (Routing Information Protocol) [6], ki se uporabljata za usmerjanje paketov v internetnem omrežju. Algoritma pot določita glede na usmerjevalne tabele, ki jih hranijo usmerjevalniki. Cilj pa je čim hitrejša dostava paketov, torej so preferirane čim krajše povezave. Usmerjevalniki v internetnem omrežju se napajajo preko električnega omrežja, kar pomeni da poraba energije ni pomembna. Prav to je glavni razlog, da omenjena algoritma nista primerna za brezžična senzorska omrežja, saj je obremenitev usmerjevalnih vozlišč neenakomerna. Pri baterijsko napajanih napravah bi torej bolj obremenjena vozlišča ostala brez energije veliko prej kot neobremenjena vozlišča, česar si pa ne želimo. Cilj usmerjevalnih algoritmov, vključenih v pregled, je torej čim manjša in čimbolj enakomerna poraba energije na nivoju celotnega omrežja. S tem zagotovimo, da so v senzorskem omrežju čim dalj časa povezana vsa vozlišča.

Usmerjevalne algoritme lahko razdelimo na podskupine glede na način iskanja poti in glede na vlogo posameznih naprav. Iskanje je lahko proaktivno, reaktivno ali pa mešanica obojega [7]. Proaktivni algoritmi pot izračunajo, preden je ta res potrebna, ponavadi v začetni fazi vzpostavitve omrežja. Reaktivni algoritmi pa računajo pot, ko je ta potrebna, torej ko neko vozlišče pošlje sporočilo. Hibridni algoritmi uporabljajo različne kombinacije obeh pristopov. Pri statični postavitvi vozlišč so proaktivni algoritmi bolj primerni, saj se poti ne spreminjajo. Če pa so naša vozlišča premikajoča, je poti potrebno izračunat ob vsakem prenosu, ker ne vemo, ali prejšnja pot še obstaja. V našem pregledu predpostavljamo statično postavitev vozlišč, zato smo se usmerili na proaktivne algoritme.

Drugi način je delitev glede na vloge posameznih naprav [8]. O ravninskem (ang. *flat-based*) usmerjanju govorimo, če imajo vse naprave enako vlogo pri usmerjanju in ne potrebujemo posebnih vmesnih naprav, kot so usmerjevalniki in podobno. Primer ravninskega usmerjanja je gradientno usmerjanje [9]. Pri hierarhičnem usmerjanju pa razlikujemo med napravami, ki zajemajo podatke (senzorji), in napravami, ki usmerjajo pakete (usmerjevalniki). Odličen primer hierarhičnega usmerjanja je popularno omrežje Zigbee. V omrežju Zigbee imamo koordinatorja, usmerjevalnik in končno napravo [10]. Sledijo algoritmi, ki za usmerjanje uporabljajo podatke o fizičnih lokacijah vozlišč (ang. *location-based*). Ta vozlišča potrebujejo način za zaznavanje lokacije, kot je na primer GPS. To pa pomeni dodatno strojno opremo, kar pa naprave, katerih cilj je čim nižja cena, zelo podraži.

Pregled področja bomo začeli z najbolj osnovnim usmerjevalnim algoritmom poplavljanja. Poplavljanje je enostaven algoritem, pri katerem sporočilo posredujejo kar vsa vozlišča. Težko torej govorimo o izbiranju poti. Nadaljujemo z algoritmom energetskega učinkovitega usmerjanja. Ta algoritem ob vzpostavitvi omrežja potrebuje nekaj poslanih sporočil, da zgradi usmerjevalne tabele za vsako posamezno vozlišče. Na podlagi teh usmerjevalnih tabel se potem v delovanju omrežja pošiljajo sporočila. Sledi gradientno usmerjanje, ki posameznim vozliščem med usmerjanjem spreminja ceno vozlišča. Poslan paket torej sledi največjemu gradientu do prehoda, ki ima ceno nič. Cena vozlišč se med delovanjem spreminja, ponavadi glede na preostalo energijo, lahko pa tudi glede na drug parameter. Oba omenjena algoritma spadata v skupino ravninskega usmerjanja. Kot zadnji algoritem za primerjavo smo si ogledali algoritem iz skupine hierarhičnega usmerjanja. Pri njem se med delovanjem tvorijo različne skupine, v katerih eno izmed vozlišč prevzame vlogo vodje. Vozlišča svoje podatke najprej pošljejo vodji svoje skupine, ta pa jih posreduje prehodu. Vloga vodje je energetsko bolj potratna, zato se vodje menjajo.

2.1 Algoritem poplavljanja in opravljanja

Najosnovnejši algoritem za usmerjanje je poplavljanje (ang. *Flooding*) [11]. Poplavljanje spada med ravninske algoritme usmerjanja, saj imajo vsa vozlišča enako nalogo. Algoritem je zelo enostaven. Vsako vozlišče posreduje sporočilo vsem svojim sosedom. To prejeto sporočilo vsa vozlišča ponovno posredujejo vsem svojim sosedom. S tem se sporočilo širi čez celotno omrežje in doseže vsa vozlišča, in s tem tudi naslovnika. S tem načinom usmerjanja z vsakim sporočilom obremenimo celotno omrežje. Še posebej je to problematično pri baterijsko napajanih napravah, saj vsako sporočilo troši energijo vseh vozlišč. S povečevanjem števila vozlišč v omrežju se ta problem še bolj izpostavi.

Pri algoritmu poplavljanja moramo zagotoviti, da vsako vozlišče določeno sporočilo posreduje samo enkrat. To zagotovimo z enoličnim identifikacijskim številom, ki ga ima vsako sporočilo. V primeru, da tega mehanizma ni, pride do efekta implozije, kjer se eno sporočilo pošilja po omrežju v nedogled.

Soroden algoritem poplavljanju je tudi algoritem usmerjanja s pomočjo opravljanja (ang. *Gossiping*). V tem primeru vozlišče ne pošlje sporočila vsem svojim sosedom, ampak naključno izbere enega in mu pošlje sporočilo. Tak način usmerjanja manj obremeni celotno omrežje, vendar je obremenitev popolnoma naključna, in zato ne moremo reči, da je algoritem energetsko učinkovit.

2.2 Algoritem energetsko učinkovitega usmerjanja

V članku [12] avtorja Rahul C. Shah in Jan M. Rabaey predlagata usmerjevalni algoritem energetsko učinkovitega usmerjanja (ang. *Energy Aware Routing*, krajše EAR). Ideja algoritma je uporaba različnih, ne vedno najcenejših, poti za pošiljanje. S tem zagotovimo enakomerno porabo energije v vseh vozliščih in se izognemo konstantni obremenitvi vozlišč na najkrajši poti. Delovanje algoritma razdelimo na več faz delovanja. Prva faza je vzpostavitev usmerjevalnih tabel. V usmerjevalni tabeli posameznega vozlišča imamo več poti in verjetnosti za izbiro posamezne poti. V drugi fazi, fazi normalnega delovanja, vsako vozlišče za posamezno sporočilo, glede na verjetnost izbere eno izmed poti v svoji usmerjevalni tabeli. Ideja je, da v tabeli cenejše poti označimo kot bolj verjetne, dražje pa kot manj verjetne.

Izdelava usmerjevalne tabele se začne tako, da prehod pošlje sporočilo s ceno $Cost(N_{prehod}) = 0$. Vsako vozlišče k tej ceni doda svojo ceno povezave. Sporočilo potem posreduje naprej z novo ceno. Posamezno vozlišče N_j ceno povezave do prehoda preko vozlišča N_i izračuna po naslednji enačbi.

$$C_{N_j,N_i} = Cost(N_i) + Metric(N_j, N_i) \tag{2.1}$$

$Cost(N_i)$ nam predstavlja ceno povezave od prehoda do vozlišča N_i. $Metric(N_j, N_i)$ pa je izračun cene povezave med vozliščema N_i in N_j, ki jo izračunamo po naslednji enačbi,

$$Metric(N_j, N_i) = e^{\alpha}_{N_i,N_j} R^{\beta}_{N_i} \tag{2.2}$$

kjer je e_{N_i,N_j} energija, ki je potrebna za pošiljanje sporočila na tej povezavi (tako energija pošiljanja, kot energija sprejemanja). R_{N_i} pa je preostala energija naprave normalizirana na začetno energijo naprave. Imamo še dve utežni spremenljivki α in β, s katerima kombiniramo iskanje poti, ki za pošiljanje potrebuje najmanj energije, in iskanje poti, na kateri

imajo naprave največ preostale energije. Posamezno vozlišče vse te poti shranjuje v tabelo, v kateri ima zapisano ceno povezave in naslednje vozlišče na določeni poti. Vozlišče mora tem povezavam dodeliti verjetnost uporabe. Verjetnost uporabe izračunamo glede na ceno izbrane povezave in ceno vseh ostalih povezav po naslednji enačbi.

$$P_{N_j,N_i} = \frac{\frac{1}{C_{N_j,N_i}}}{\sum_{k \epsilon FT_j} \frac{1}{C_{N_j,N_k}}} \tag{2.3}$$

S to tabelo in verjetnostmi si vozlišče izračuna še povprečno ceno povezave (2.4). To ceno potem uporablja tudi pri vzpostavljanju poti drugih vozlišč, ki ga uporabljajo za posrednika.

$$Cost(N_j) = \sum_{i \epsilon FT_j} P_{N_j,N_i} C_{N_j,N_i} \tag{2.4}$$

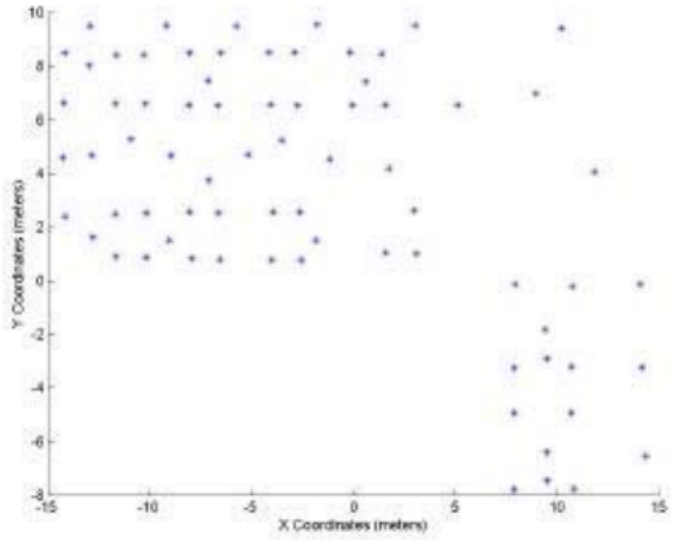
V fazi pošiljanja vozlišče pošlje sporočilo enemu izmed sosedov iz svoje tabele za posredovanje glede na verjetnost v tabeli. Ta sosed ponovi postopek izbire iz tabele za posredovanje. To se ponavlja dokler sporočilo ne pride do prehoda.

Avtorji so usmerjevalni algoritem EAR testirali s simulacijo. Za modeliranje okolja so predpostavili komunikacijo v prostoru s pregradami, ki slabijo signal. Za eno oddajanje so definirali, da potrebujejo $20nJ/bit + 1pJ/bit/m^3$ energije. $20nJ/bit$ je energija vezja med pošiljanjem, ki jo označimo z E_{elec}, $1pJ/bit/m^3$ pa potrebna energija ojačevalnika med pošiljanjem večsmernem prostoru, ki jo označimo z ϵ_{mp}. To pomeni, da je energija upadala s kubom razdalje. Naslednja enačba nam prikazuje potrebno energijo za oddajanje,

$$E_{tx} = l \cdot E_{elec} + l \cdot \epsilon_{mp} \cdot d^3 \tag{2.5}$$

kjer je l število bitov, ki jih pošiljamo, d predstavlja razdaljo. Za primerjavo v idealnih pogojih, torej pošiljanje v prostem prostoru (ang. *free space*), je upad energije s kvadratom razdalje.

Za simulacijo so uporabili 76 naprav, ki so razporejene v topologijo, ki jo vidimo na sliki 2.1. Topologija predstavlja pisarniško okolje, v katerem imamo senzorje temperature, ki oddajajo vsakih 30 sekund, in senzorje svetlobe, ki oddajajo vsakih 10 sekund.

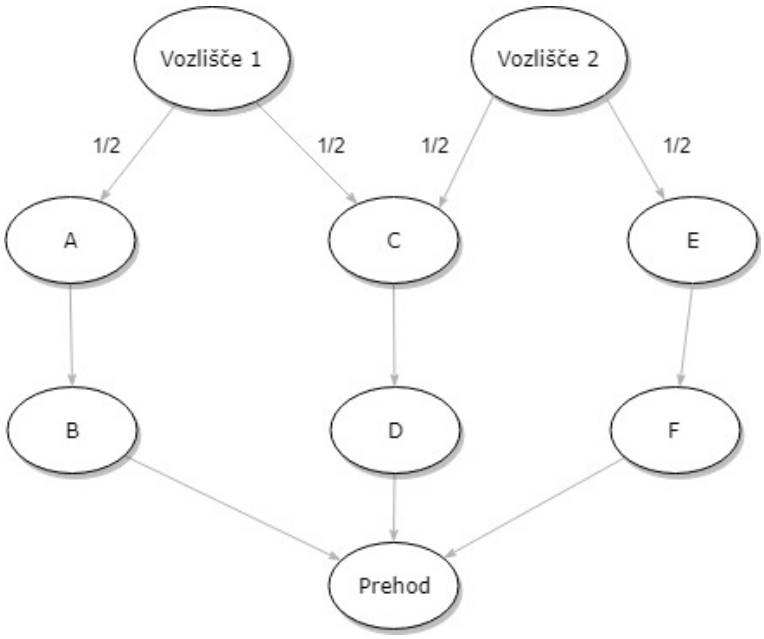


Slika 2.1: Postavitev vozlišč v simulaciji algoritma EAR [12]

Za primerjavo so uporabili algoritem difuzijskega usmerjanja (ang. *diffusion routing*). Algoritem difuzijskega usmerjanja ima drugačen način delovanja. Namesto periodičnega pošiljanja podatkov s strani vozlišč, pri difuzijskem usmerjanju prehod po celotnem omrežju pošlje željo po nekem podatku. Ciljno vozlišče na to sporočilo odgovori s podatkom. Za izbiro poti sporočila od vozlišča nazaj do prehoda pa uporabi najcenejšo pot, glede na uporabljeno cenovno funkcijo. Primerjava algoritma EAR z difuzijskim usmerjanjem je zanimiva izbira. Na prvi pogled sta algoritma namenjena drugačni uporabi v brezžičnih senzorskih omrežjih. Algoritem EAR je bolj primeren za uporabo v omrežjih pri katerih gre za periodično pošiljanje sporočil. Tak scenarij je tudi opisan v simulaciji. Difuzijsko usmerjanje pa je bolj primerno pri omrežjih, kjer podatke potrebujemo na zahtevo. Iz tega lahko sklepamo, da je samo del prihranka pri energiji posledica pametnejšega usmerjanja, del prihranka pa pride iz manjšega dodanega stroška komunikacije (ang. *overhead*).

Kot ocenjevalne parametre pri simulaciji, so uporabili povprečno porabo posameznega vozlišča v omrežju in pa čas do odpovedi omrežja. Odpoved omrežja definiramo kot čas do odpovedi prvega vozlišča. Rezultati so pokazali, da je povprečna poraba energije posameznega vozlišča za 21,5% manjša pri uporabi algoritma EAR v primerjavi z difuzijskim usmerjanjem. Prav tako je pri EAR daljši čas do odpovedi omrežja, in sicer za kar 40%.

V članku [13] avtorji izpostavijo potencialno pomanjkljivost algoritma EAR, in sicer pri računanju cene posamezno vozlišče ne upošteva, koliko vozlišč ga uporablja na svoji poti. Na sliki 2.2 vidimo topologijo enostavnega omrežja.



Slika 2.2: Topologija z različno obremenitvijo poti

Poti AB, CD in EF so enakovredne po porabi energije. Vozlišče ena dodeli poti AB in CD petdeset odstotno verjetnost za njuno uporabo. Prav tako naredi vozlišče dva za poti CD in EF. V tem primeru je torej pot CD bolj obremenjena kot AB oziroma EF, kar pripelje do hitrejše odpovedi omrežja. V idealnem primeru bi vozlišče ena pot AB uporabilo v $\frac{2}{3}$ primerov, pot CD pa v $\frac{1}{3}$ primerov. Prav tako bi vozlišče dva pot CD uporabilo v $\frac{1}{3}$ primerov, pot EF pa v $\frac{2}{3}$ primerov. Tako bi bile vse poti enakomerno obremenjene. Ta pomanjkljivost je zelo opazna pri opisani topologiji, v topologiji enakomerno razporejenih vozlišč pa je manj opazna. V članku [13] avtorji algoritem EAR spremenijo tako, da v fazi vzpostavljanja omrežja upoštevajo tudi to, koliko vozlišč uporablja določeno pot. Ta algoritem so poimenovali pravično energetsko učinkovito usmerjanje (ang. *Fair Energy Aware Routing* krajše FEAR).

2.3 Algoritem gradientnega usmerjanja

Gradientno usmerjanje (ang. *Gradient-Based routing*) spada med ravninske proaktivne algoritme usmerjanja. Vsakemu vozlišču dodelimo ceno [14]. Razliko med cenama dveh vozlišč imenujemo gradient. Paket po omrežju potuje preko vozlišč z največjimi gradienti do cilja, ki ima ceno nič. Osnovni algoritem ceno določa kot število skokov od vozlišča do ponora. S tem, ko sporočilo potuje po poti z največjim gradientom, izbere najkrajšo pot. Lahko pa na primer za izbiro cene vozlišča izberemo samo trenutno preostalo energijo vozlišča, in s tem zagotovimo, da sporočilo potuje po poti, ki ima največ energije. Pri brezžičnih senzorskih omrežjih je smiselno ta dva parametra združiti s ciljem, da sporočilo potuje po čim krajši poti, ki ima največ energije. S tem omogočimo enakomerno porabo energije med vozlišči in čim daljšo življenjsko dobo omrežja.

Prva faza algoritma je vzpostavitvena faza. V tem delu se izmenjajo sporočila, s pomočjo katerih se ustvari začetno omrežje in poti za pošiljanje. Ob uspešni prvi fazi vozlišča začnejo pošiljati svoje senzorske podatke. Med običajnim delovanjem potrebujemo tudi način za posodabljanje omrežja, saj s tem zagotovimo, da se poti sporočil ustrezno spreminjajo glede na preostalo energijo v omrežju.

2.3.1 Predlagana cenovna funkcija

Naša želja je cenovna funkcija, ki bi upoštevala tako dolžino poti, kot preostalo energijo vozlišč. V literaturi nismo našli take cenovne funkcije. Za simulacijo gradientnega usmerjanja smo zato predlagali svojo cenovno funkcijo. Naša želja je, da vozlišče izbere najkrajšo pot, a le če ta pot ni veliko bolj energijsko izčrpana kot druge poti. Cenovna funkcija mora torej upoštevati preostalo energijo vozlišča in oddaljenost vozlišča do prehoda. Ker ne vemo, kateri od obeh parametrov je bolj pomemben, nam cenovna funkcija omogoča nastavljanje utežnih parametrov. Kako izbrati utežni parameter za najboljše rezultate bomo preverili s simulacijo v poglavju 3.1.3.

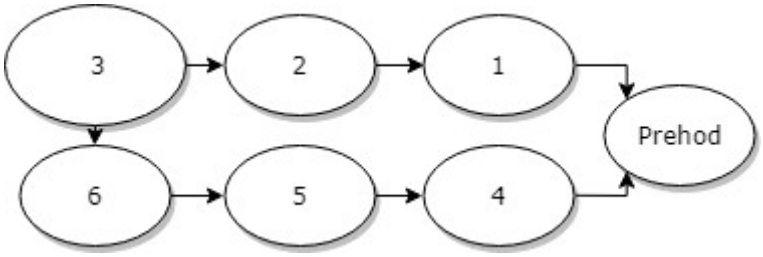
Enačba (2.6) predstavlja predlagano cenovno funkcijo, kjer je *bat* normalizirana vrednost preostale energije baterije, *hop* število skokov do ponora ter α in β utežni spremenljivki.

$$Metric(N_i, N_j) = \frac{1}{bat} \times \alpha + hop \times \beta \tag{2.6}$$

Celotna cena poti od vozlišča N_i do ponora preko vozlišča N_j je vsota cen poti do vozlišča N_j in rezultat enačbe (2.6). Celotna cena, na podlagi katere se vozlišče odloči za določeno pot je torej:

$$Cost(N_i) = Metric(N_i, N_j) + Cost(N_j) \tag{2.7}$$

Cilj spremenljivk α in β je možnost nastavljanja obnašanja naše cenovne funkcije. S spreminjanjem razmerja teh dveh spremenljivk lahko določimo, kako hitro bo vozlišče uporabilo daljšo pot, ki ima več energije. Za primer vzemimo dve različni možnosti. Pri prvi imamo vrednosti $\alpha = 1000$ in $\beta = 1$, pri drugi možnosti pa $\alpha = 1$ in $\beta = 1000$. Vzemimo vozlišče, ki ima dve možni poti, eno dolgo tri skoke in drugo dolgo štiri skoke, kot vidimo na sliki 2.3. Predpostavljamo, da je cena obeh poti do vozlišča enaka (vrednost $Cost(N_j)$), torej je izbira poti odvisna samo od stanja baterije vozlišča in števila skokov. Na tem mestu je vredno poudariti, da če imamo dve poti, eno s tremi in eno s štirimi skoki, ki imata enako ceno, to pomeni, da ima pot s tremi skoki manj energije kot tista s štirimi.



Slika 2.3: Topologija enostavnega omrežja z različno dolžino poti od vozlišča tri do prehoda

Oglejmo si torej nekaj vrednosti cen poti glede na stanje baterije. V naslednji tabeli 2.1 vidimo, kako se spreminjajo vrednosti pri 100%, 75%, 50%, 25% in 10% preostale baterije.

1. primer		
$\alpha = 1000; \beta = 1$		
% \št. skok	3	4
100	1003	1004
75	1336	1337
50	2003	2004
25	4003	4004
10	10003	10004

2. primer		
$\alpha = 1; \beta = 1000$		
% \št. skok	3	4
100	3001	4001
75	3001,3	4001,3
50	3002	4002
25	3004	4004
10	3010	4010

Tabela 2.1: Primer cen poti glede na utežni spremenljivki, dolžino poti in stanje baterije

Vidimo torej, da v prvem primeru hitro omogočimo menjavo na daljšo pot, saj je že v primeru, da ima vozlišče 75% energije in tri skoke, njegova cena višja kot pri polni bateriji in štirih skokih. V drugem primeru se to ne zgodi nikoli. To pomeni, da v drugem primeru ne bo prišlo do menjave na daljšo pot.

Izračunamo lahko tudi, pri katerem odstotku baterije se bo pot zamenjala. Kot že omenjeno, imamo vozlišče, recimo s številko tri, ki ima do ponora dve poti. Prva pot je dolga tri skoke - preko vozlišča dva in vozlišča ena. Druga pot pa je dolga štiri skoke - preko vozlišč šest, pet in štiri. Ponor označimo s številko nič. Pri predpostavki, da so baterije na vozliščih štiri, pet in šest popolnoma polne, velja enačba (2.9), in pri predpostavki, da se baterije vozlišč ena, dva in tri praznijo z enako hitrostjo velja enačba (2.8). Zanima nas torej, pri katerem odstotku preostale energije baterij vozlišč ena, dva in tri bo druga, daljša pot, postala cenejša. S $Cost_3(N_3)$ označimo pot s tremi skoki, s $Cost_4(N_3)$ pa daljšo pot, s štirimi skoki.

$$bat_{N_1} = bat_{N_2} = bat_{N_3} \tag{2.8}$$

$$bat_{N_4} = bat_{N_5} = bat_{N_6} \tag{2.9}$$

$$\begin{aligned}
Cost_3(N_3) &= Metric(N_3, N_2) + Metric(N_2, N_1) + Metric(N_1, N_0) \\
&= (\frac{1}{bat_{N_3}} \times \alpha + hop_{N_3} \times \beta) + (\frac{1}{bat_{N_2}} \times \alpha + hop_{N_2} \times \beta) \\
&+ (\frac{1}{bat_{N_1}} \times \alpha + hop_{N_1} \times \beta) \\
&= \frac{3}{bat_{N_3}} \times \alpha + \beta \times (3 + 2 + 1) \\
&= \frac{3}{bat_{N_3}} \times \alpha + 6\beta \\
Cost_4(N_3) &= Metric(3, 6) + Metric(6, 5) + Metric(5, 4) + Metric(4, 0) \\
&= (\frac{1}{bat_{N_3}} \times \alpha + hop_{N_3} \times \beta) + (\frac{1}{bat_{N_6}} \times \alpha + hop_{N_6} \times \beta) \\
&+ (\frac{1}{bat_{N_5}} \times \alpha + hop_{N_5} \times \beta) + (\frac{1}{bat_{N_4}} \times \alpha + hop_{N_4} \times \beta) \\
&= (\frac{3}{bat_{N_6}} + \frac{1}{bat_{N_3}}) \times \alpha + (4 + 3 + 2 + 1) \times \beta \\
&= (\frac{3}{bat_{N_6}} + \frac{1}{bat_{N_3}}) \times \alpha + 10\beta
\end{aligned} \tag{2.10}$$

Veljati mora torej naslednje:

$$\begin{aligned} Cost_3(N_3) &> Cost_4(N_3) \\ \frac{3}{bat_{N_3}} \times \alpha + 6\beta &> (\frac{3}{bat_{N_6}} + \frac{1}{bat_{N_3}}) \times \alpha + 10\beta \\ \frac{3\alpha}{bat_{N_3}} - \frac{\alpha}{bat_{N_3}} &> \frac{3\alpha}{bat_{N_6}} + 10\beta - 6\beta \\ \frac{2\alpha}{bat_{N_3}} &> \frac{3\alpha}{bat_{N_6}} + 4\beta \\ bat_{N_3} &< \frac{2\alpha}{\frac{3\alpha}{bat_{N_6}} + 4\beta} \end{aligned} \tag{2.11}$$

Rekli smo, da so baterije na daljši poti polne, torej lahko vstavimo $bat_{N_6} = 1$ in dobimo enačbo

$$bat_{N_3} < \frac{2\alpha}{3\alpha + 4\beta} \tag{2.12}$$

Sedaj vstavimo spremenljivki $\alpha = 1000$ in $\beta = 1$ iz našega prvega primera. Vidimo, da se bo pot spremenila, ko imajo vozlišča na krajši poti manj kot 67% baterije.

$$\begin{aligned} bat_{N_3} &< \frac{2 \times 1000}{3 \times 1000 + 4 \times 1} \\ bat_{N_3} &< \frac{2000}{3004} \\ bat_{N_3} &< 0,67 \end{aligned} \tag{2.13}$$

Vstavimo še spremenljivki iz drugega primera, kjer je $\alpha = 1$ in $\beta = 1000$. V tem primeru je sprememba pri veliko manj kot enem procentu preostale energije, torej praktično nikoli.

$$\begin{aligned} bat_{N_3} &< \frac{2 \times 1}{3 \times 1 + 4 \times 1000} \\ bat_{N_3} &< \frac{2}{4003} \\ bat_{N_3} &< 0,0005 \end{aligned} \tag{2.14}$$

V poglavju 3.1.3 s pomočjo simulacije primerjamo, kakšen vpliv imata vrednosti α in β na samo delovanje algoritma.

2.4 Algoritem LEACH

Algoritem usmerjanja na principu nizkoenergetske adaptivne hierarhije gruče (ang. *Low-energy adaptive clustering hierarchy* krajše LEACH) [15] za razliko od že predstavljenih algoritmov spada v skupino hierarhičnega usmerjanja. Velja za pionirja hierarhičnega

usmerjanja v brezžičnih senzorskih omrežjih. Osnovna ideja algoritma je delitev vozlišč v manjše gruče, vsaka od teh gruč pa ima vodjo. Člani posamezne gruče svoje podatke pošljejo vodji gruče, do katere imajo neposredno povezavo. Vodje gruče pa nato podatke posredujejo prehodu. Vodja gruče je energijsko bolj potratna vloga kot pa vloga člana v gruči, zato vodja gruče ni vedno isto vozlišče, ampak se vodje menjajo.

Delovanje algoritma LEACH delimo na dve različni fazi. Prva faza je faza vzpostavitve gruč, druga pa normalno pošiljanje podatkov. Faza vzpostavitve gruč se periodično ponavlja glede na poljubno izbrano časovno obdobje. V tej fazi vsako posamezno vozlišče najprej izračuna, če je on sam vodja gruče. Vozlišče generira psevdonaključno število med 0 in 1. Če je število manjše od vrednosti $P_i(t)$, je vozlišče vodja gruče. Vrednost $P_i(t)$ izračuna s pomočjo naslednje enačbe,

$$P_i(t) = \frac{k}{N - k + (r \mod \frac{N}{k})} \tag{2.15}$$

kjer nam N predstavlja število vseh vozlišč, k število pričakovanih gruč in r število krogov pošiljanja od začetka vzpostavitve omrežja. Dodatno velja še pogoj, da vozlišče ni bilo vodja gruče v zadnjih $(r \mod \frac{N}{k})$ krogih.

V primeru, da je vozlišče vodja gruče, pošlje oglaševalno sporočilo, v katerem ostalim vozliščem pove, da je vodja. Vsa ostala vozlišča, ki niso vodje, svojo gručo izberejo glede na najbližjo vodjo gruče. Oddaljenost od vodje vozlišče oceni s pomočjo kvalitete signala RSSI oziroma *received signal strength indicator* oglaševalnega sporočila. RSSI je indikator moči sprejemnega signala. Izbira najbližje vodje je pomembna, ker vozliščem omogoča oddajanje z nižjo oddajno močjo. Vsako vozlišče mora sporočiti vodji, da se priključuje njegovi gruči. Ko algoritem uspešno določi nove vodje in njihove gruče, se začne faza pošiljanja podatkov. V tej fazi vozlišča pošiljajo svoja sporočila vodji, ta pa jih posreduje prehodu.

V članku [16] algoritem LEACH primerjajo s tremi različnimi algoritmi. Prvi, najbolj enostaven algoritem je kar neposredna komunikacija. V tem primeru vsa vozlišča neposredno komunicirajo s prehodom. Drug algoritem je algoritem minimalne energije za pošiljanje (ang. *Minimal transmition energy* krajše MTE). V tem primeru vozlišča oddajajo z minimalno močjo in s tem prihranijo energijo. Sporočilo do prehoda pride po najkrajši možni poti. Avtorji so algoritem LEACH primerjali tudi s statičnimi gručami. Gruče se v tem primeru čez celotno delovanje omrežja ne spreminjajo, kar pa pripelje do hitrejše odpovedi vodij gruč.

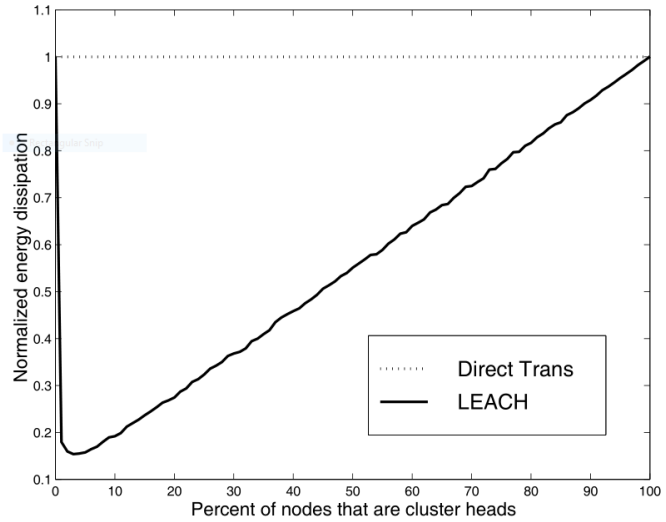
Za simulacijo so avtorji naredili omrežje s 100 naključno razporejenim vozlišči. Njihova maksimalna moč oddajanja je bila $50nJ/bit + 100pJ/bit/m^2$, kjer je $50nJ/bit$ energija vezja med pošiljanjem, ki jo označimo z E_{elec}, $1pJ/bit/m^3$ pa potrebna energija ojačevalnika med pošiljanjem prostem prostoru, ki jo označimo z ϵ_{mp}. Velja naslednja enačba,

$$E_{tx} = l \cdot E_{elec} + l \cdot \epsilon_{fs} \cdot d^2 \tag{2.16}$$

kjer je l število bitov, ki jih pošiljamo ter d razdalja na kateri pošiljamo. Za model omrežja so vzeli prosti prostor, kar pomeni, da energija upada s kvadratom razdalje.

Simulacija je pokazala odlične rezultate za algoritem LEACH. V primerjavi so ugotovili, da je pri algoritmu LEACH čas do odpovedi prvega vozlišča 7-krat daljši od neposredne komunikacije, 78-krat daljši od algoritma MTE in 9-krat daljši od statičnega gručenja. Opazili so tudi, da je poraba energije pri algoritmu LEACH veliko bolj enakomerna, kar vidimo kot bolj skupinsko odpoved vseh vozlišč v kratkem času. Pri algoritmu LEACH je prvo vozlišče odpovedalo po 394 krogih, zadnje pa pri 665 krogih, kar pomeni 65% daljši čas delovanja. Pri algoritmu MTE pa je prvo vozlišče odpovedalo po 5 krogih, zadnje pa po 221 krogih. To pomeni, da je razlika kar 4400% daljše delovanje zadnjega vozlišča.

V opisu smo omenili parameter k, ki nam pove število pričakovanih gruč v omrežju. Avtorji so s pomočjo simulacije ugotovili, da je algoritem najbolj učinkovit, ko je število gruč v omrežju enako 5% vseh vozlišč. Tako nastavitev omrežja so tudi uporabili v simulacijah.



Slika 2.4: Poraba energije v odvisnosti od števila vodij gruč [16]

Iz algoritma LEACH se je razvilo več nadaljnih algoritmov. Ti algoritmi ponavadi delujejo na istem principu, spremenjen je le način izbire vodje gruče. Na primer, pri algoritmu LEACH-C [17], je odločanje o vodjah gruč izvedeno centralizirano s strani prehoda.

V fazi izdelave gruč vsa vozlišča pošljejo svojo lokacijo in preostalo energijo. Na podlagi teh podatkov prehod vozlišča razdeli v gruče. Algoritem lahko tako bolj optimizira gruče, vendar pa pride do večjega stroška dodane komunikacije. Drug primer je algoritem LEACH-F [18], pri katerem se gruče formirajo samo na začetku. V delovanju pa se samo periodično menja vodja gruče med člani specifične gruče. Protokol LEACH-E [19] pa pri odločanju o vodji gruče, kot najbolj pomemben podatek uporabi preostalo energijo posameznega vozlišča. Bolj obsežen pregled izpeljank algoritma LEACH so naredili avtorji v članku [20].

Poglavje 3

Simulacije

Za primerjavo in vrednotenje algoritmov smo naredili simulacije, za katere smo uporabili program OMNet++. Naš cilj je primerjati algoritme v različnih pogojih, zato smo naredili več simulacij. Uporabili smo več različnih topologij omrežja. S tem vidimo, kako uspešno se posamezen algoritem prilagodi različnim topologijam. Za glavno simulacijo smo uporabili topologijo naključno razporejenih 81 vozlišč in topologijo naključno razporejenih 151 vozlišč. Za verifikacijo posameznih algoritmov pa smo uporabili tudi druge topologije kot na primer topologijo enakomerno razporejenih 25 vozlišč.

3.1 OMNeT++

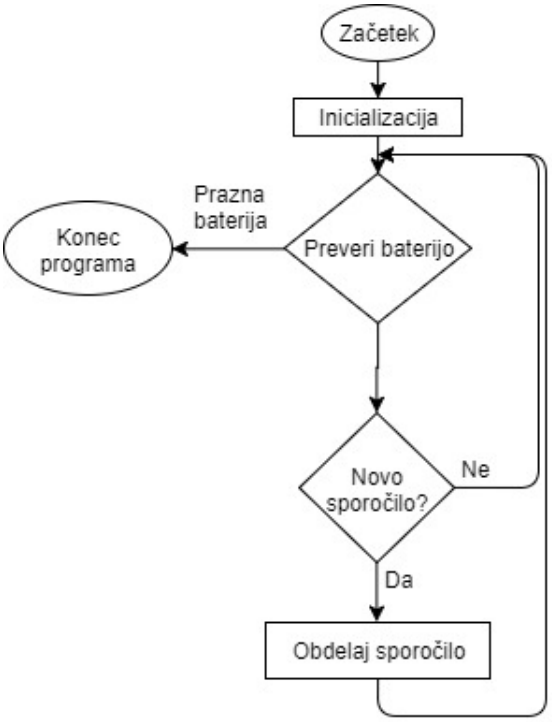
OMNeT++ je programsko ogrodje, ki omogoča simulacijo različnih omrežij [21]. Zgrajen je kot objektno orientiran modularni simulator, ki deluje na principu simulacije dogodkov. OMNeT++ nam zagotovi osnovna orodja oziroma strukturo za izdelavo simulacij. Obnašanje vozlišč pa smo sprogramirali sami v jeziku c++. Za opis omrežja in povezav med napravami OMNeT++ smo uporabili *Network Descripton* oziroma krajše NED. Ker je simulacijsko okolje zelo široko zastavljeno, je primerno za simulacijo različnih stvari od simulacije prometa v omrežjih, simulacije protokolov do simulacije večprocesorskih sistemov in njihove komunikacije. Na splošno torej za vse modele, kjer je simulacija dogodkov primeren način simulacije.

Za uspešno simulacijo najprej potrebujemo model omrežja. Za opis modela uporabimo programski jezik NED. Z njim opišemo povezavo med vozlišči in njene lastnosti. Povezavam lahko nastavimo zakasnitev, hitrost, pogostost napake in podobne parametre, s katerimi se s simulacijo kar najbolj približamo realnemu svetu.

Naš cilj je simuliranje večjih senzorskih omrežij. Ker programski jezik NED od nas zahteva, da vsako povezavo med vozlišči posebej opišemo, smo se odločili, da sprogramiramo Python skripto, ki za nas postavi vozlišča v prostor.

3.1.1 Struktura programa

Kot je že bilo omenjeno, delovanje naših vozlišč in algoritma programiramo v programskem jeziku c++ z uporabo OMNeT++ ogrodja. Naš razred deduje razred *cSimpleModule*, ki predstavlja enostavno napravo. Za vsak algoritem moramo spremeniti obnašanje oziroma procesiranje sporočil. Osnovno ogrodje med njimi pa je enako. Glavna funkcija je funkcija, ki sprejeme in obdela posamezna sporočila. Ta funkcija se spreminja glede na algoritem, ki ga programiramo. Vodimo tudi stanje baterije. Pri vsakem pošiljanju in prejemanju upoštevamo porabo baterije. Ko se baterija izprazni, program neha pošiljati sporočila. Naše ogrodje samodejno zazna, ko ni več sporočil in zaključi simulacijo. Takrat se požene še zaključna funkcija, s katero v datoteke zapišemo vse statistične podatke za nadaljnjo obdelavo. Na sliki 3.1 lahko vidimo poenostavljen prikaz poteka našega programa. Glavni del je procesiranje sporočila, za katerega opis sledi ločeno za vsak posamezen algoritem.



Slika 3.1: Diagram poteka simulacijskega programa

3.1.2 Implementacija algoritma EAR

Algoritem EAR smo implementirali po principu opisanem v poglavju 2.2. Pri izdelavi posredovalne tabele v članku [12] ni omenjenih končnih pogojev, pri katerih rečemo, da imajo vse naprave v omrežju vzpostavljene pravilne posredovalne tabele. Odločili smo se, da uvedemo dodaten parameter, in sicer maksimalno dolžino poti. Pri vzpostavljanju omrežja ponorno vozlišče (ang. *sink node*) najprej pošlje sporočilo, ki ima število skokov nastavljeno na ena. Z vsakim naslednjim poslanim sporočilom to vrednost povečamo, dokler ne dosežemo maksimalne dolžino poti. Takrat je naše omrežje pripravljeno na pošiljanje senzorskih podatkov. S postopnim zviševanjem števila skokov zagotovimo, da se usmerjevalne tabele gradijo postopoma. S tem se izognemo problemu, ko ima oddaljeno vozlišče že zgrajeno svojo usmerjevalno tabelo, bližnje pa še ne. V tem primeru bi morali pri spremembi tabele bližnjega vozlišča, spremeniti tudi vsa nadaljnja vozlišča, kar pa bi pomenilo veliko dvojnega pošiljanja. Zato najprej zgradimo posredovalne tabele napravam, ki so oddaljene en skok, potem dva in tako naprej. V naši simulaciji smo parameter maksimalne dolžine poti nastavili na 25 skokov.

Algoritem EAR za vzpostavitev usmerjevalnih tabel potrebuje dodatno komunikacijo. Vsako sporočilo za postavitev tabel vsebuje 8 bajtov podatkov. In sicer 1 bajt za identifikator sporočila, 1 bajt za identifikator vozlišča, ki je zadnji poslal sporočilo, 1 bajt za število skokov od ponora, 4 bajte za ceno trenutne povezave in 1 bajt za maksimalno dolžino poti. Identifikacija vozlišča je velika samo 1 bajt kar pomeni, da imamo lahko v omrežju največ 255 vozlišč.

Psevdokoda 1 nam prikazuje delovanje vozlišča v času vzpostavitve tabel. Vozlišče najprej preveri, če je prejeto sporočilo že obdelal. Če je preostalo število skokov enako nič, povezavo doda v svojo posredovalno tabelo. V nasprotnem primeru pa sporočilu doda svojo ceno in sporočilo posreduje. Za osvežitev spomina smo še enkrat napisali enačbo (3.1) za izračun poti. Spomnimo se, da enačba vsebuje dve utežni spremenljivki α in β, v naši simulaciji smo uporabili vrednosti $\alpha = 1$ in $\beta = 50$.

$$Metric(N_j, N_i) = e^{\alpha}_{N_i,N_j} R^{\beta}_{N_i} \tag{3.1}$$

Algoritem 1 Psevdokoda funkcije za izgradnjo posredovalnih tabel

```
1: preveriCeJeSporociloPregledano(zacetnoSporocilo)
2: if niPregledano then
3:     if sporocilo.PreostaloSkokov = 0 then
4:         dodajVPosredovalnoTabelo(zacetnoSporocilo.Cena + mojaCena)
5:     else
6:         sporocilo.PreostaloSkokov ← sporocilo.PreostaloSkokov − 1
7:         zacetnoSporocilo.Cena ← zacetnoSporocilo.Cena + mojaCena
8:         posljiSporociloVsem(zacetnoSporocilo)
9:     end if
10: end if
```

Psevdokoda 2 nam prikazuje generiranje sporočil med normalnim delovanjem. Ta funkcija se izvaja periodično, njena naloga pa je enostavna, v sporočilo doda svoje izmerjene podatke, izbere pot, po kateri bo poslala sporočilo, in sporočilo pošlje. Generiranje sporočil se izvaja samo pri vozliščih in ne pri ponoru. Ponor v celotnem času, ko omrežje zaznava, torej po uspešni vzpostavitvi poti, ne pošlje več nobenega sporočila. Tudi poti se pri tem algoritmu vzpostavijo na začetku delovanja omrežja, med samim delovanjem poti ne spreminjamo.

Algoritem 2 Psevdokoda funkcije za generiranje sporočil

```
1: vSporociloDodajSvojePodatke(sporocilo)
2: pot ← nakjlucnoIzberiPotIzTableGledeNaVerjetnost()
3: posljiSporociloPoPoti(sporocilo, pot)
```

Psevdokoda 3 pa nam prikazuje obdelavo prejetega sporočila. Vozlišče preveri, če je sporočilo namenjeno njemu, in če je, ga posreduje proti ponoru.

Algoritem 3 Psevdokoda funkcije za preverjanje prejetega sporočila

```
1: if sporocilo.ciljnoVozlisce = mojId then
2:     pot ← nakjlucnoIzberiPotIzTableGledeNaVerjetnost()
3:     posljiSporociloPoPoti(sporocilo, pot)
4: end if
```

3.1.3 Implementacija gradientnega usmerjanja

Glavno vprašanje pri implementaciji gradientnega usmerjanja je uporaba cenovne funkcije. V poglavju 2.3.1 smo opisali našo cenovno funkcijo in njena dva utežna parametra, ki

jih lahko uporabimo za nastavljanje obnašanja naše funkcije. Za osvežitev spomina je predlagana cenovna funkcija ponovno opisana v naslednji enačbi.

$$Metric(N_i, N_j) = \frac{1}{bat} \times \alpha + hop \times \beta \tag{3.2}$$

Primerjali smo dva različna primera. V prvem je $\alpha = 1000$ in $\beta = 1$, v drugem pa ravno obratno $\alpha = 1$ in $\beta = 1000$. Z izračunom smo pokazali, da v prvem primeru do menjave pride pri 67% baterije. V drugem primeru pa ne pride do menjave na daljšo pot. V izračunu smo uporabili poenostavljeno omrežje, simulirali pa smo na bolj realnem omrežju. Testirali smo na enakomernem omrežju z devetimi vozlišči. V testu smo opazovali, kolikokrat so vozlišča izbrala daljšo pot namesto krajše. Za primerjavo uspešnosti smo uporabili povprečno število prejetih sporočil s strani ponora in največje odstopanje od povprečja. S tem lahko ocenimo enakomernost delovanja našega omrežja. Opazovali pa smo tudi čas odpovedi prvega vozlišča, saj je naša želja čim daljši čas delovanja omrežja.

	$\alpha = 1000;\ \beta = 1$	$\alpha = 1;\ \beta = 1000$
število sprememb	194	0
čas odpovedi [s]	585	580
povprečno prejetih sporočil	138(± 43)	139(± 25)

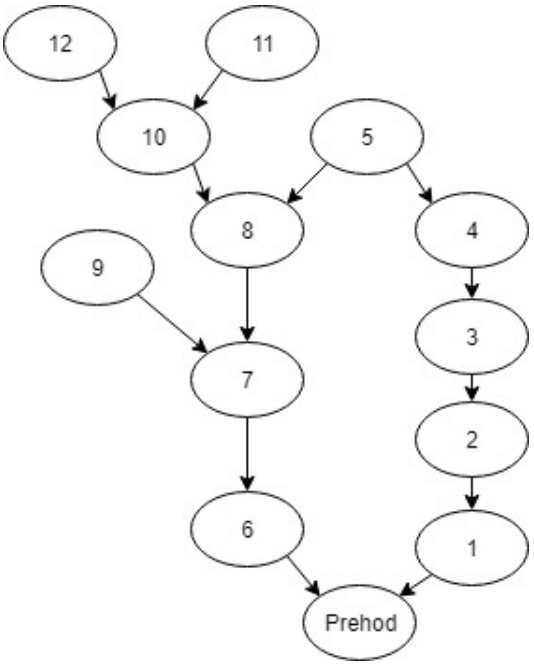
Tabela 3.1: Primerjava vpliva utežnih spremenljivk na enakomernem omrežju

Opazili smo, da se čas odpovedi prvega vozlišča minimalno razlikuje. Prav tako se minimalno razlikuje povprečno število prejetih sporočil na vozlišče. Malo večja razlika je v enakomernosti prejemanja sporočil. Videli smo torej, da je na enakomernem omrežju bolj smiselna druga opcija, pri kateri ne pride do spremembe poti.

Test smo ponovili na topologiji omrežja, ki bolj promovira menjavo na daljšo pot. To smo dosegli tako, da je daljša pot manj obremenjena kot krajša. Topologijo lahko vidimo na sliki 3.2, rezultate pa v tabeli 3.2.

	$\alpha = 1000;\ \beta = 1$	$\alpha = 1;\ \beta = 1000$
število sprememb	346	0
čas odpovedi [s]	535	355
povprečno prejetih sporočil	98(± 66)	97(± 34)

Tabela 3.2: Primerjava vpliva utežnih spremenljivk na neenakomernem omrežju



Slika 3.2: Topologija neenakomerno razporejenih vozlišč

Opazili smo večjo razliko pri času do odpovedi prvega vozlišča. Tokrat je spreminjanje poti pozitivno vplivalo na čas do odpovedi prvega vozlišča. V tem primeru je vozlišče številka pet najbolj podvrženo uporabi daljše poti, saj je pot preko vozlišč štiri, tri, dva in ena veliko manj obremenjena kot pot preko vozlišč osem, sedem, šest. Pri številu prejetih sporočil pa ponovno ni velike razlike. Enakomernost prejemanja je celo manjša pri nastavitvi, ki uporablja tudi daljše poti. To pa se zgodi zato, ker v tem primeru prvi odpovesta vozlišči sedem in dva. S tem odklopita večino omrežja, povezani ostaneta samo še vozlišči šest in ena. Posledično sta vozlišči šest in ena uspeli dostaviti več sporočil in s

tem povečati neenakomernost. Pri nastavitvah, pri katerih se ne uporabljajo daljše poti, pa prvo odpove vozlišče sedem. Za njim odpove vozlišče osem, kar pa pomeni, da vozlišča od ena do šest ostanejo povezana in uspešno pošiljajo sporočila.

Ugotovili smo torej, da je v enakomernem omrežju razlika med obema načinoma zanemarljiva. Pokazali pa smo tudi, da obstaja topologija, na kateri je izbira daljše poti smiselna. V naši simulaciji uporabljamo nastavitev pri kateri ne pride do veliko zamenjav, torej $\alpha = 1$ in $\beta = 1000$.

Za dodatno komunikacijo algoritem potrebuje 5 bajtov, in sicer 4 bajte za ceno povezave in 1 bajt za identifikacijo vozlišča, ki je poslalo ceno. Ponovno predvidevamo največ 255 vozlišč v omrežju.

Psevdokoda 4 nam prikazuje generiranje sporočil med normalnim delovanjem. Ta funkcija se izvaja periodično, njena naloga pa je enostavna, v sporočilo doda svoje izmerjene podatke, izbere pot, po kateri bo poslala sporočilo, in sporočilo pošlje. Vozlišče pa vsakih *PONOVNOPOSLJICENO* generiranih sporočil v sporočilo doda svojo posodobljeno ceno. Cena se konstantno spreminja zaradi porabe energije vozlišča. *PONOVNOPOSLJICENO* je konstanta, ki določi, kako pogosto se pošlje cena vozlišča. V naši simulaciji je konstanta nastavljena na 10 sporočil. Tudi pri algoritmu gradientnega usmerjanja pa velja, da ponor med delovanjem ne pošilja sporočil.

Algoritem 4 Psevdokoda funkcije za generiranje sporočil

```
1: stevecPonovnegaPosiljanjaCene ← stevecPonovnegaPosiljanjaCene + 1
2: if stevecPonovnegaPosiljanjaCene = PONOVNOPOSLJICENO then
3:     stevecPonovnegaPosiljanjaCene ← 0
4:     izracunajNovoCeno()
5:     vSporociloDodajSvojoCeno(sporocilo)
6: end if
7: vSporociloDodajSvojePodatke(sporocilo)
8: pot ← izberiNajboljsoPot()
9: poslji SporociloPoPoti(sporocilo, pot)
```

Psevdokoda 5 pa nam prikazuje obdelavo prejetega sporočila. Vozlišče preveri, če je sporočilo namenjeno njemu, in če je, ga posreduje proti ponoru.

Algoritem 5 Psevdokoda funkcije za preverjanje prejetega sporočila

```
1: if sporocilo.posodobiCeno = True then
2:     posodobiGradientnoTabelo(sporocilo)
3: end if
4: if sporocilo.ciljnoVozlisce = mojId then
5:     pot ← izberiNajboljsoPot()
6:     posljiSporociloPoPoti(sporocilo, pot)
7: end if
```

3.1.4 Implementacija algoritma LEACH

LEACH je edini izmed algoritmov, ki med delovanjem omogoča spreminjanje moči oddajanja. Ob formiranju gruče lahko člani gruče s pomočjo moči signala ugotovijo, kolikšna minimalna moč oddajanja je še potrebna, da sporočilo doseže vodjo gruče. S tem vozlišča prihranijo veliko energije. V simulaciji smo tako s pomočjo natančnih lokacij posameznih vozlišč implementirali tudi izbiro moči oddajanja. Oddajanje je tako mogoče s petimi različnimi stopnjami.

Na podlagi ugotovitev avtorjev v članku [16], smo za pričakovano število vodij gruč vzeli 5% vseh vozlišč v omrežju. Gruče pa spreminjamo vsakih pet poslanih izvornih sporočil. S tem zagotovimo dovolj pogosto menjavo gruč.

Posamezno vozlišče pri vzpostavljanju gruč potrebuje dve sporočili, če je vodja, in eno sporočilo, če je suženj. Prvo sporočilo vodje je veliko 1 bajt, ki vsebuje identifikator vozlišča, ki je vodja. Suženj odgovori s sporočilom, ki je veliko 2 bajta, in vsebuje identifikacijo vodje, ki se mu pridružuje v gručo in svoj identifikator. Na koncu vodja odgovori vsem vozliščem v njegovi gruči. V najslabšem primeru je to torej toliko bajtov, kolikor je vseh vozlišč. To se zgodi v primeru, če ni nobene vodje, in tako ponor prevzame vlogo vodje za vsa vozlišča.

Psevdokoda 6 nam prikazuje delovanje vozlišča v času izdelave gruč. Vsako vozlišče glede na naključno generirano število izračuna, ali je vodja gruče ali ne. Vozlišča, ki so vodje gruče, to naznanijo ostalim vozliščem. Takrat ostala vozlišča pošljejo sporočilo, kateri gruči bi se pridružili, vodja pa potem še potrdi vsa pridružena vozlišča.

Algoritem 6 Psevdokoda funkcije za izgradnjo gruč

```
1: preveriCeSemVodja()
2: if semVodja then
3:     posljiSporociloVodja()
4:     cakajNaSporociloSuznjev()
5:     posljiSporociloPotrdiloSprejetihSuznjev()
6: else
7:     cakajNaSporociloVodje()
8:     odgovoriNajbljiziVodji()
9:     cakajNaPotrditevSprejemaVGruco()
10: end if
```

Psevdokoda 7 pa nam prikazuje obdelavo prejetega sporočila. V primeru, da je vozlišče vodja gruče, si sporočilo shrani za kasnejše pošiljanje ponoru, to torej pomeni da vodja združi vsa sporočila in proti ponoru pošlje samo eno veliko sporočilo. V nasprotnem primeru sporočilo vozlišče zavrže.

Algoritem 7 Psevdokoda funkcije za preverjanje prejetega sporočila

```
1: if semVodja then
2:     shraniSporocilo(sporocilo)
3: end if
```

Psevdokoda 8 nam prikazuje generiranje sporočil med normalnim delovanjem. Vozlišče, ki je vodja gruče najprej počaka, da prejme sporočila od ostalih vozlišč v gruči. Nato vsa sporočila združi in pošlje ponoru. Ostala vozlišča pa v tej funkciji pošljejo svoje podatke vodji gruče.

Algoritem 8 Psevdokoda funkcije za generiranje sporočil

```
1: vSporociloDodajSvojePodatke(sporocilo)
2: if semVodja then
3:     pocakajNaVsaSporocilaSuznjev()
4:     zdruziPodatke(sporocilo)
5:     posljiSporociloPonoru(sporocilo)
6: else
7:     posljiSporociloVodji(sporocilo)
8: end if
```

3.1.5 Zbiranje statističnih podatkov

Za analizo algoritmov vsake simulacije potrebujemo določene podatke. Naš cilj je analizirati energijo vozlišč, med drugim, kdaj se zgodi odpoved prvega vozlišča in koliko vozlišč je povezanih v odvisnosti od časa. Zanima pa nas tudi, koliko sporočil je posamezno vozlišče uspelo poslati in koliko sporočil je ponor dejansko prejel. Vzrok izgube sporočila je odpoved vozlišča na poti do ponora. Različni algoritmi različno prilagodijo pot ob odpovedi omrežja. Zanima nas tudi, koliko sporočil je poslalo vsako vozlišče in kolikšen delež teh sporočil so podatki vozlišča in kolikšen podatki ostalih vozlišč, ki jih posredujemo. Zaradi potrebe opisanih podatkov smo morali implementirati dodatna sporočila, ki v resnici ne obstajajo. Uporabljamo jih samo za zbiranje statističnih informacij in ne vplivajo na rezultat. Primer takih sporočil je direktno potrjevanje prejema sporočila iz strani ponora. Ponor smo za ta namen povezali z vsemi vozlišči. Te poti se uporabljajo samo za potrjevanje prejema sporočil, zato da vemo, kdaj je vozlišče povezano in kdaj ne.

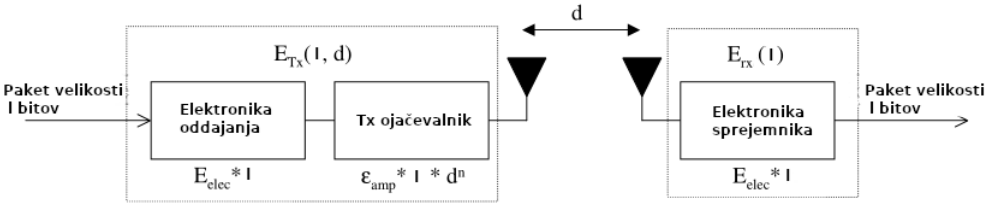
3.2 Okolje simulacije

3.2.1 Radijski model

Zelo pomemben del simulacije je okolje, v katerem simuliramo. Glavna parametra sta predvsem poraba baterije ter doseg brezžične komunikacije. Pri porabi baterije imamo porabo v stanju prejemanja in oddajanja sporočila, porabo med obdelavo podatkov in porabo med spanjem vozlišča. Poraba med obdelavo podatkov in med spanjem vozlišča je veliko manjša kot poraba med prejemanjem in oddajanjem sporočil. Zato smo se odločili, da porabo med obdelavo podatkov in spanjem zanemarimo. Na tem mestu je vredno poudariti, da smo upoštevali energijo prejemanja vsakega sporočila, ki je bilo poslano v dosegu posameznega vozlišča. To pomeni, da upoštevamo tudi energijo za sporočilo, ki ni bilo namenjeno posameznemu vozlišču a ga je le-ta vseeno prejel. Tako je tudi v realnem svetu.

Doseg brezžične komunikacije oziroma energija potrebna za oddajanje na določeni razdalji je odvisna od okolja. V urbanem okolju bo zaradi motenj potrebno več energije za enako razdaljo kot na odprtem prostoru, kjer motenj ni. Pri algoritmu LEACH so avtorji [15] poenostavili radijski model in za simulacijo uporabili kar odprt prostor (ang. *free space*). Pri članku o algoritmu EAR [12] so avtorji uporabili večsmerni prostor v katerem energija upada z kubom razdalje. Za našo simulacijo smo uporabili splošno uveljavljen model, pri katerem upoštevamo tako porabo enegije v odprtem prostoru kot tudi porabo enegije pri večsmernem prostoru. Radijski model smo povzeli po doktorski na-

logi [22], uporabili pa so ga tudi avtorji članka [23] v katerem avtorji predlagajo izboljšave za algoritem LEACH. Slika 3.3 nam predstavlja potrebno energijo za prenos sporočila.



Slika 3.3: Radijski model

Enačba (3.3) nam torej prikazuje potrebno energijo za pošiljanje l bitov dolgega sporočila. Če je razdalja med dvema vozliščema manjša od mejne vrednosti d_0, ki jo imenujemo tudi križna razdalja (ang. *crossover distance*), potem velja model odprtega prostora. Drugače velja enačba za večpotni prostor.

$$E_{tx} = \begin{cases} l \cdot E_{elec} + l \cdot \epsilon_{fs} \cdot d^2; & d < d_0 \\ l \cdot E_{elec} + l \cdot \epsilon_{mp} \cdot d^4; & d \geq d_0 \end{cases} \tag{3.3}$$

E_{elec} označuje energijo vezja med pošiljanjem, ϵ_{fs} je energija ojačevalnika med pošiljanjem v odprtem prostoru, ϵ_{mp} pa je enegija pri pošiljanju v večpotnem prostoru. d predstavlja razdaljo med oddajnikom in sprejemnikom.

Energijo za sprejem pa izračunamo po naslednji formuli:

$$E_{rx} = l \cdot E_{elec} \tag{3.4}$$

V naši simulaciji smo uporabili naslednje vrednosti, ki smo jih povzeli po doktorski nalogi [22].

$$\epsilon_{fs} = 10 \frac{pJ}{bit \cdot m^2} \tag{3.5}$$

$$\epsilon_{mp} = 0,0013 \frac{pJ}{bit \cdot m^4} \tag{3.6}$$

$$E_{elec} = 50 \frac{nJ}{bit} \tag{3.7}$$

$$l = 80bit \tag{3.8}$$

$$d_0 = 86m \tag{3.9}$$

V simulaciji smo omejili maksimalno moč oddajanja na $170mJ$, kar pomeni domet 200 metrov. Algoritem LEACH za delovanje zahteva, da je mogoče spreminjati moč oddajanja,

saj ravno z zmanjševanjem moči dosega dobre rezultate. V ta namen smo izbrali 5 različnih moči za oddajanje. Stopnje moči in njihov domet lahko vidimo v tabeli 3.3.

Moč oddajanja [mJ]	Domet [m]
4	25
6	50
14	100
56	150
179	200

Tabela 3.3: Moč oddajanja in domet

3.2.2 Topologija omrežja

Za uspešnost našega omrežja je zelo pomembna tudi topologija omrežja. Za simulacijo smo izbrali prostor velikosti 200 metrov krat 200 metrov. Primerjave med algoritmi smo izvedli na dveh različnih velikostih omrežja. Prvo omrežje vsebuje 80 vozlišč in en ponor, skupaj torej 81 naprav. Drugo omrežje pa vsebuje 150 vozlišč in en ponor, skupaj torej 151 naprav. V obeh primerih smo ponor postavili točno na sredino, torej na koordinato (100, 100). Ostala vozlišča smo po prostoru razvrstili naključno saj smo se s tem hoteli približati realnem svetu, kjer enakomerna postavitev ni pogosta. Vsa vozlišča imajo enako funkcijo delovanja, razen seveda ponora, ki skrbi za zbiranje vseh sporočil.

Simulacija med svojim delovanjem uporablja naključni generator za proženje različnih dogodkov med delovanjem omrežja. Preverili smo, da naši rezultati niso odvisni od naključnega generatorja. To smo potrdili z 10 ponovitvami simulacij in preverili, če vedno dobimo primerljive rezultate.

Kot smo že omenili posamezne topologije generiramo naključno. Simulacija na samo eni naključno generirani topologiji, bi lahko pokazala nepravilne rezultate zaradi značilnosti tiste topologije. Vsako simulacijo smo zato ponovili na 10 različnih naključnih topologijah.

Skupaj imamo torej 10 simulacij na 81 napravah z konstantno topologijo in 10 simulacij na 81 napravah, kjer vedno zamenjamo naključno topologijo. To smo ponovili tudi na 151 napravah, skupaj imamo torej 40 različnih simulacij.

3.2.3 Ostale značilnosti simulacijskega modela

V simulacijskem modelu smo predvidevali periodično pošiljanje sporočil. Gre torej za simulacijo brezžičnega senzorskega omrežja, katerega naloga je kontinuirano zaznavanje in posredovanje zaznanih vrednosti ponoru. Perioda zajemanja podatkov je lahko različna in je odvisna predvsem od primera. V naši simulaciji smo zajem podatkov in pošiljanje sporočila naredili enkrat na minuto. Da pa bi v model vnesli še nekaj ne-determinističnosti, smo za generiranje dogodkov uporabili Poissonovo porazdelitev, kjer je lambda 60 sekund. Poenostavili pa smo značilnosti povezave oziroma medija. V realnem svetu je brezžični medij nezanesljiv, mi pa smo predpostavili idealni medij, torej medij ki vedno dostavi sporočilo. Predpostavili smo tudi to, da na našem omrežju ni kolizij paketov. V realnem svetu v primeru istočasnega oddajanja dveh vozlišč pride do kolizije v katerem sta oba paketa izgubljena, v naši simulaciji tega ni. Vsako poslano sporočilo je veliko 80 bitov. Vsa vozlišča imajo enako osnovno vlogo, torej pošiljanje zajetih podatkov enkrat na minuto. Ponor pri vseh sporočilih oddaja samo za namene grajenja omrežja. Med normalnim oddajanjem ni oddajanja sporočil, kar nam prihrani energijo bližnjih vozlišč saj vsako sprejemanje sporočila porablja energijo.

Definirali smo tudi kapaciteto baterije. Kapaciteta posamezne baterije vsakega vozlišča je $250J$. To je primerljivo eni desetini kapacitete gumbne baterije cr2032 z nazivno napetostjo $3V$ in kapaciteto $235mAh$.

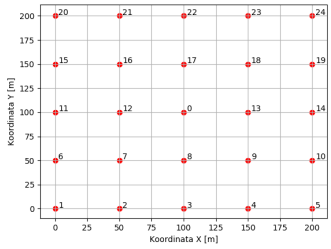
Pravilnost celotnega modela simulacije in pravilnost delovanja posameznih algoritmov smo preverili s pomočjo simulacije na manjšem omrežju. Na manjšem omrežju smo lahko simulaciji sledili po korakih in sproti preverjali pravilnost delovanja. Validacija pravilnosti modela je bila narejena s primerjavo z rezultati iz obstoječe literature. Direktne primerjave niso bile mogoče predvsem zaradi manjših razlik med simulacijskimi modeli. Boljšo validacijo bi lahko izvedli z izbiro brezžične tehnologije, s katero bi lahko dodelali naš model z bolj natančnimi podatki o porabi med pošiljanjem in sprejemanjem.

3.3 Verifikacija delovanja algoritmov

Za preverjanje pravilnosti naše implementacije algoritmov smo naredili verfikacijo algoritmov. Verifikacijo algoritmov smo izvedli na manjših primerih zaradi lažje podrobne analize rezultatov. Uporabili smo tudi manj energije in sicer samo $10J$.

3.3.1 Verifikacija algoritma EAR

Verifikacijo algoritma EAR smo naredili na omrežju s 25 vozlišči. Vozlišča so enakomerno razporejena na območju velikem 200 krat 200 metrov, kot vidimo na sliki 3.4.



Slika 3.4: Enakomerna razporeditev 25 vozlišč v prostor velikosti 200 metrov krat 200 metrov.

Rezultati simulacije so nam pokazali, da po približno 7700 sekundah odpovejo prva štiri vozlišča in sicer vozlišča 7, 9, 16 in 18. Zanimivo to niso vozlišča, ki so neposredno povezana s ponorom. Pričakovali bi, da najprej odpovejo vozlišča, ki so neposredno povezana s ponorom saj preko njih potujejo vsa sporočila. Vendar ta vozlišča, torej vozlišča 8, 12, 13 in 17 odpovejo šele po okoli 10500 sekundah. Ob odpovedi teh štirih vozlišč nam odpove tudi celotno omrežje, saj so preostala vozlišča preveč oddaljena za neposredno komunikacijo s ponorom.

V tabeli 3.4 vidimo koliko izvornih, posredovanih in prejetih sporočil ima vsako izmed vozlišč. Opazimo, da ima skupina vozlišč, ki odpove prva veliko več prejetih sporočil kot druga skupina vozlišč. Do tega pride, ker imajo vozlišča 7, 9, 16 in 18 vsako po štiri sosedna vozlišča, ki aktivno pošiljajo svoja sporočila. Vozlišča 8, 12, 13 in 17 pa imajo samo tri vozlišča, ki aktivno pošiljajo, četrti sosed pa je ponor, ki samo prejema podatke. Kot že omenjeno za vsak prejem sporočila vozlišče porabi $4mJ$ energije ne glede na to ali je bilo vozlišče cilj tega sporočila ali ne.

Preverimo še smiselnost razmerja med številom izvornih, posredovanih in prejetih sporočil. Dejstvo je, da vsa sporočila v zunanjem krogu (na robu prostora) pošljejo svoja sporočila preko notranjega kroga vozlišč. Imamo 16 vozlišč v zunanjem krogu in 8 vozlišč v notranjem krogu, iz tega lahko izračunamo, da bo posamezno vozlišče v notranjem krogu povprečno posredovala dve sporočili na eno svoje izvorno sporočilo. Razmerje izvornih in posredovanih sporočil v prvi skupini je res približno faktor dve kot smo predvidevali. Podobno lahko izračunamo za drugo skupino, v tem primeru je 20 vozlišč, ki vsa posredujejo sporočila preko 4 vozlišč. Posredovanih sporočil je v tem primeru povprečno torej 5 krat več kot izvornih. V tabeli vidimo, da je faktor nekoliko nižji to pa zato, ker pri našem izračunu nismo upoštevali, da 4 vozlišča prej odpovejo, kar pomeni, da je povprečno število vseh sporočil vsak krog manj kot dvajset.

Pri številu prejetih sporočil lahko izračunamo, da vsak krog celotno omrežje pošlje 60 sporočil, in sicer imamo 4 vozlišča, ki imajo pot dolgo 4 skoke, kar pomeni 16 oddajanj. Imamo 8 vozlišč s potjo dolgo 3 skoke torej 24 oddajan, prav tako imamo 8 vozlišč s potjo dolgo 2 skoka torej 16 oddajan in pa 4 vozlišča z potjo dolgo 1 skok torej 4 oddajanja. Vsa štiri vozlišča, ki so najbolj obremenjena bodo prejela vsa ta sporočila, zato lahko pri njih pričakujemo povprečno 15 prejetih sporočil na vozlišče. Točno tako razmerje tudi vidimo v tabeli.

ID vozlišča	Št. izvornih sporočil	Št. posredovanih sporočil	Št. prejetih sporočil
7	126	247	1906
9	126	265	1880
16	126	261	1885
18	128	256	1891
8	168	735	1110
12	173	730	1110
13	175	723	1117
17	172	728	1114

Tabela 3.4: Število izvornih, posredovanih in prejetih sporočil za posamezno vozlišče.

3.3.2 Verifikacija algoritma gradientnega usmerjanja

Podobno kot pri verifikaciji algoritma EAR smo verifikacijo gradientnega usmerjanja naredili na omrežju s 25 vozlišči. Vozlišča so enakomerno razporejena na območju velikem 200 krat 200 metrov, kot vidimo na sliki 3.4.

Tudi pri gradientnem usmerjanju najprej odpovejo vozlišča 7, 9, 16 in 18, tako kot pri algoritmu EAR. Prvo med temi vozlišči odpove vozlišče 16 po 7866 sekundah, zadnje pa vozlišče 18 po 8320 sekundah. Vidimo torej, da so ta vozlišča odpovedala manj enakomerno kot pri EAR algoritmu. V tabeli 3.5 preverimo razliko pri številu sporočil pri vozlišču 16 in 18. Vidimo lahko, da je vozlišče 18 posredovalo opazno manj sporočil kot pa vozlišče 16, ki je zato tudi prej odpovedalo. Pričakujemo lahko torej bolj neenakomerno obnašanje omrežja v primerjavi z algoritmom EAR.

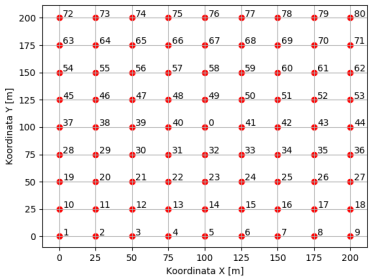
V prejšnjem poglavju smo ugotovili, da pri vozliščih v notranjem krogu za vsako izvorno sporočilo pričakujemo dve posredovani sporočili. Vidimo pa, da to pri gradientnem usmerjanju ne velja povsem, to pa zato, ker se poti sproti prilagajajo. Zato torej bolj obremenjena vozlišča 7, 9, 16 in 18 posredujejo manj sporočil kot pri algoritmu EAR, kjer se poti vzpostavijo na začetku in se skozi celotno delovanje algoritma ne spreminjajo. V primeru gradientnega usmerjanja dodatno breme prevzamejo vozlišča 3, 11, 14 in 22. Tako lahko na primer vozlišče 2 raje uporabi pot skozi vozlišča 3 in 8 kot pa pot skozi vozlišča 7 in 8.

ID vozlišča	Št. izvornih sporočil	Št. posredovanih sporočil	Št. prejetih sporočil
7	137	261	1903
9	138	236	1939
16	130	190	2020
18	138	79	2175
8	164	661	1264
12	161	756	1125
17	160	836	1006
13	160	835	1009

Tabela 3.5: Število izvornih, posredovanih in prejetih sporočil za posamezno vozlišče.

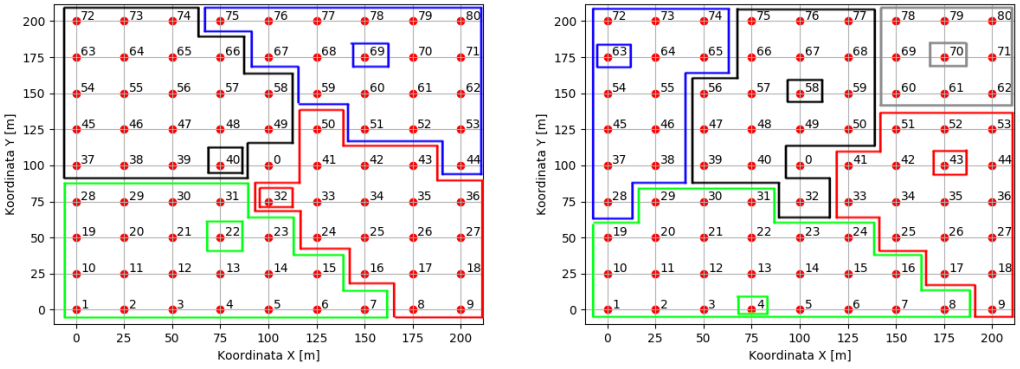
3.3.3 Verifikacija algoritma LEACH

Za verifikacijo algoritma LEACH smo uporabili omrežje z večjim številom vozlišč. Algoritem LEACH vozlišča razvrsti v različne gruče, zato uporaba na manjših omrežjih ni smiselna. Pri verifikaciji uporabljamo nastavitev, kjer je pričakovano število gruč vsak krog 5% vseh vozlišč, to pa pomeni, da bi v primeru 25 vozlišč pričakovali samo $1,25$ gruče. Zato smo pri verifikaciji algoritma LEACH obdržali velikost prostora 200 metrov krat 200, povečali pa smo število vozlišč na 81 vozlišč. Vozlišča so še vedno enakomerno razporejena po prostoru, kot vidimo na sliki 3.5.



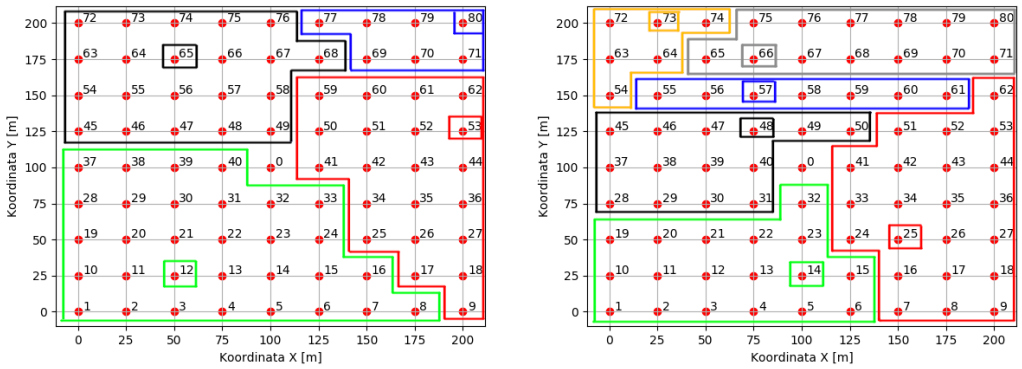
Slika 3.5: Enakomerna razporeditev 81 vozlišč v prostor velikosti 200 metrov krat 200 metrov.

Poglejmo si torej kako se ustvarjajo različne gruče med delovanjem omrežja. Imamo 81 vozlišč, vsak krog pa pričakujemo, da bo 5% teh vozlišč vodja gruče, pričakujemo torej 4 gruče na krog. Na sliki 3.6 lahko vidimo razporeditev gruč v prvih dveh krogih. Razporeditev gruč je v prvem krogu dokaj enakomerna. Zelena gruča vsebuje 21 vozlišč, črna 20 vozlišč, modra 18 in rdeča 17 vozlišč. Vidimo pa lahko, da vodje treh gruč, torej vozlišča 22, 32 in 40 niso najbolje postavljena saj so vsa zelo skupaj. V drugem krogu je omrežje razdeljeno na 5 gruč. Tokrat so vodje lepše postavljeni po prostoru, gruče pa so malo manj enakomerno porazdeljene, predvsem siva gruča z vodjo 70 ima v gruči samo 8 drugih vozlišč.



Slika 3.6: Levo vidimo razporeditev gruč v prvem krogu. Desno vidimo razporeditev gruč v drugem krogu.

Na sliki 3.7 vidimo gruče v tretjem in četrtem krogu. V tretjem krogu imamo ponovno 4 različne gruče, v četrtem krogu pa imamo kar 6 različnih gruč. Ponovno so slabo izbrane vodje 48, 57 in 66, ki so popolnoma drug zraven druge. V tem krogu imamo kar dve gruči z manjšim številom vozlišč in sicer gručo z vodjo 73, ki ima samo 5 vozlišč in gručo z vodjo 57, ki ima 6 vozlišč. Vidimo lahko torej, da naključna izbira vodji gruč lahko vodi do nekaj slabše postavljenih gruč.



Slika 3.7: Levo vidimo razporeditev gruč v tretjem krogu. Desno vidimo razporeditev gruč v četrtem krogu.

Preverimo še pravilnost delovanja porabe energije v teh nekaj prvih krogih. V tabeli 3.6 vidimo podatke za vozlišče 22 po prvem krogu. V prvem krogu vozlišče pošlje 5 svojih sporočil, 105 sporočil pa je tistih, ki jih posreduje, saj ima v svoji gruči 21 vozlišč. Vozlišče v celotnem krogu prejme 242 sporočil, konec kroga pa mu preostane $7368,8mJ$ energije.

ID vozlišča	Št. izvornih sporočil	Št. posredovanih sporočil	Št. prejetih sporočil	Preostala energija
22	5	105	242	$7368,8mJ$
32	5	85	271	$8455,2mJ$
40	5	100	266	$8398,4mJ$
69	5	90	197	$7775,6mJ$

Tabela 3.6: Število izvornih, posredovanih in prejetih sporočil za vsako od vodji v prvem krogu.

Za preverjanje podatkov uporabimo enačbo (3.10), kjer je E_p preostala energija, E_z začetna energija, E_{rx} energija porabljena za sprejem sporočil, E_{txm} energija porabljena za pošiljanje izvornih sporočil, E_{txr} energija porabljena za pošiljanje posredovanih sporočil, E_{vod} energija porabljena za vzpostavljanje gruč, če je vozlišče vodja in E_{sl} energija porabljena za vzpostavljanje gruč, če vozlišče ni vodja.

$$E_p = E_z - E_{rx} - E_{txm} - E_{txr} - Evod - Esl \tag{3.10}$$

Definirali smo, da vsako vozlišče začne z $10J$ začetne energije, velja torej

$$E_z = 10000mJ \tag{3.11}$$

Vozlišče v celotnem krogu prejme 242 sporočil, vsako prejemanje pa porabi $4mJ$ energije skupaj je to $968mJ$

$$\begin{aligned} E_{rx} &= 242 * 4mJ \\ &= 968mJ \end{aligned} \tag{3.12}$$

Vozlišče 22 je od ponora oddaljeno 56 metrov, kar pomeni da za oddajanje sporočil uporabi moč $14mJ$, kjer imamo domet 100 metrov. Vozlišče pošlje 5 izvornih sporočil in 5 posredovanih sporočil za vsakega od 21 članov svoje gruče.

$$\begin{aligned} E_{txm} &= 5 * 14mJ \\ &= 70mJ \end{aligned} \tag{3.13}$$

$$\begin{aligned} E_{txr} &= 5 * 21 * 14mJ \\ &= 1470mJ \end{aligned} \tag{3.14}$$

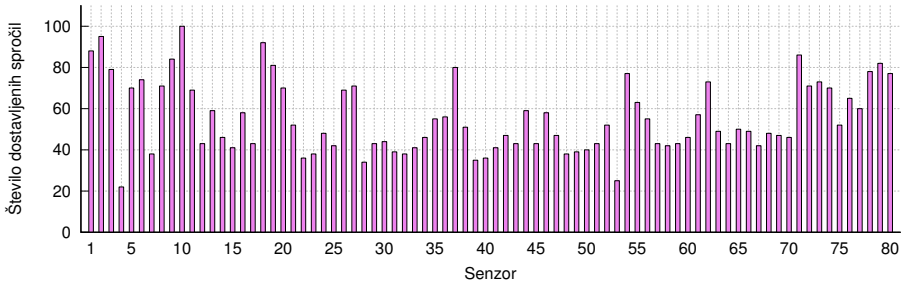
Preostane še energija, ki je potrebna za vzpostavitev gruče. V primeru vodje je torej pomembna $E_{sl} = 0$, izračunati pa je potrebno E_{vod}. Za vzpostavitev vodje gruče najprej pošlje sporočilo pri moči $56mJ$ vendar pa je sporočilo veliko eno desetino običajnega sporočila, tako da vozlišče porabi samo $5,6mJ$. Ob uspešni vzpostavitvi gruče, vodja pošlje še sporočilo s seznamom vseh svojih članov gruče, za to pa potrebuje $5,6mJ$ krat število članov.

$$\begin{aligned} E_{vod} &= 5,6mJ + 5,6mJ * 21 \\ &= 123,2mJ \end{aligned} \tag{3.15}$$

Če to vstavimo v enačbo (3.10) dobimo, da je preostale energije $7368,8mJ$ kar je točno toliko kot nam sporoči simulacija.

$$\begin{aligned} E_p &= 10000mJ - 968mJ - 70mJ - 1470mJ - 123,2mJ \\ &= 7368,8mJ \end{aligned} \tag{3.16}$$

Na sliki 3.8 vidimo koliko sporočil je prejel ponor za posamezno vozlišče. Opazimo lahko, da ponor najmanj sporočil prejme od vozlišča 4, preverimo zakaj je temu tako.



Slika 3.8: Število prejetih sporočil za posamezno vozlišče pri algoritmu LEACH na topologiji s 81 naključno razporejenimi vozlišči.

Vozlišče je v prvem krogu priključeno gruči, katere vodja je vozlišče 22, od njega je torej oddaljen 100 metrov, kar pomeni da oddaja z močjo $6mJ$. V naslednjem krogu je vozlišče 4 vodja gruče, to je tudi krog v katerem se porabi največ energije. Gruča je v tem primeru kar velika saj ima kar 23 članov. Od ponora je vozlišče 4 oddaljeno 103 metre, kar zahteva oddajanje z močjo $56mJ$. V tem krogu torej vozlišče pošlje 5 svojih izvornih sporočil in 5 sporočil za vsakega od 23 članov gruče. V tretjem krogu vozlišče spada pod gručo z vodjo 12, v četrtem krogu pa v gručo z vodjo 14. V obeh primerih torej potrebuje

$6mJ$ za komunikacijo. V zadnjem krogu pa spada pod gručo kjer je vodja vozlišče 29 in tako za pošiljanje potrebuje $14mJ$ energije. V tem krogu vozlišču zmanjka energije po dveh poslanih sporočilih. V celotnem času delovanja vozlišče prejme 760 sporočil. V tabeli 3.7 vidimo natančne izračune porabe energije za pošiljanje. K temu prištejemo še energijo prejemanja in dobimo končni rezultat porabljene energije $10004,4mJ$. Vidimo lahko torej, da je več kot polovico energije vozlišče porabilo za posredovanje sporočil, saj je vozlišče zelo oddaljeno od ponora, prav tako pa je bila gruča, katere je bil vodja v drugem krogu, zelo velika.

$$\begin{aligned} E &= E_{rx} + E_{txm} + E_{txr} + Evod + Esl \\ &= 760 * 4mJ + 384mJ + 6440mJ + 134,4mJ + 6mJ \\ &= 10004,4mJ \end{aligned} \tag{3.17}$$

Krog	E_{txm}	E_{txr}	E_{vod}	E_{sl}
1	$5 * 6mJ$	0	0	$1,2mJ$
2	$5 * 56mJ$	$5,6 * 56mJ * 23$	$5,6mJ + 5,6mJ * 23$	0
3	$5 * 6mJ$	0	0	$1,2mJ$
4	$5 * 6mJ$	0	0	$1,2mJ$
5	$2 * 14mJ$	0	0	$2,4mJ$
total	$= 384mJ$	$= 6440mJ$	$= 134,4mJ$	$= 6mJ$

Tabela 3.7: Porabljena energija za vozlišče 4 po krogih.

3.4 Primerjava algoritmov

Kot že omenjeno nam algoritem LEACH omogoča spreminjanje moči oddajanja, pri ostalih algoritmih pa je moč oddajanja stalna. Za algoritma EAR in gradientno usmerjanje pa smo uporabili minimalno moč, ki še vedno zagotavlja, da je celotno omrežje povezano. Najmanjšo še zadovoljivo moč smo našli s pomočjo simulacije. Simulirali smo 1000 naključno postavljenih omrežij in pri vsakem preverili, če so vsa vozlišča povezana. Pri omrežju z 81 vozlišči pri dometu 25 metrov, omrežje nikoli ni v celoti povezano. Pri dometu 50 metrov imamo 97,5% povezanost. Za 100% povezanost pa potrebujemo domet 100 metrov. Pri 151 vozliščih pa z dometom 50 metrov že zagotovimo 100% povezanost v omrežju. Za simulacijo smo v obeh primerih izbrali domet 50 metrov.

Sledi primerjava vseh algoritmov na posameznih topologijah in vrednotenje, kje je kateri algoritem boljši.

3.4.1 Kriteriji za primerjavo

Za ocenjevanje algoritmov smo uporabili več kriterijev. Prvi kriterij je čas do odpovedi prvega vozlišča. Ta kriterij je predvsem pomemben, če je namembnost omrežja taka, da mora biti celotno omrežje aktivno in si ne moremo privoščiti izpada vozlišča. Obstajajo tudi okolja, ki ne zahtevajo 100% delovanja omrežja, torej izpad večih vozlišč ni nujno problematičen. Tako smo nadzorovali tudi čas do odpovedi 90% in 50% vozlišč. S tem, da v tem primeru odpoved definiramo kot nepovezanost vozlišča s prehodom. To pa zato, ker lahko pride do situacije, kjer enemu vozlišču zmanjka energije, zaradi tega pa ostane več vozlišč nepovezanih s prehodom.

Za ocenjevanje enakomernosti porabe energije pa merimo tudi število prejetih sporočil posameznega vozlišča. V idealnem primeru hočemo, da vsa vozlišča odpovejo istočasno. To nam zagotavlja enakomerno porabo energije in tudi enako število prejetih sporočil na vozlišče.

3.4.2 Predpostavke in poenostavitve simulacije

V prejšnjih poglavjih smo že omenjali predpostavke in poenostavitve, ki smo jih upoštevali v naši simulaciji. Vse te predpostavke in poenostavitve smo zbrali na enem mestu. Splošne predpostavke, ki veljajo za vse simulacije.

- Energijska enačba

$$E_{tx} = \begin{cases} l \cdot E_{elec} + l \cdot \epsilon_{fs} \cdot d^2; & d < d_0 \\ l \cdot E_{elec} + l \cdot \epsilon_{mp} \cdot d^4; & d \geq d_0 \end{cases} \tag{3.18}$$

- Pogoji radijskega modela

$$\epsilon_{fs} = 10 \frac{pJ}{bit \cdot m^2} \tag{3.19}$$

$$\epsilon_{mp} = 0,0013 \frac{pJ}{bit \cdot m^4} \tag{3.20}$$

$$E_{elec} = 50 \frac{nJ}{bit} \tag{3.21}$$

$$l = 80bit \tag{3.22}$$

$$d_0 = 86m \tag{3.23}$$

- Uporabljamo naslednje moči oddajanja:

Moč oddajanja [mJ]	Domet [m]
4	25
6	50
14	100
56	150
179	200

Tabela 3.8: Moč oddajanja in domet

- Sporočilo pošljemo enkrat na minuto s Poissonovo porazdelitvijo.
- Uporabljamo popolno povezavo, kjer predvidevamo, da ne pride do napak in kolizij med sporočili.
- Ponor med delovanjem samo sprejema sporočila in jih ne oddaja. Ponor oddaja sporočila samo med vzpostavitvijo omrežja, če je to potrebno.
- Kapacitete baterije je $250J$.
- Prostor simulacije je velik 200 metrov krat 200 metrov.
- Prva topologija vsebuje 80 naključno razporejenih vozlišč in en ponor na sredini.
- Druga topologija vsebuje 150 naključno razporejenih vozlišč in en ponor na sredini.

Predpostavke pri algoritmu EAR:

- V cenovni funkciji uporabljamo utežni spremenljivki $\alpha = 1$ in $\beta = 50$.
- Pri vzpostavitvi omrežja dovoljujemo maksimalno 25 skokov.
- Algoritem EAR tabel za usmerjanje ne posodablja med delovanjem omrežja.

Predpostavke pri algoritmu gradientnega usmerjanja:

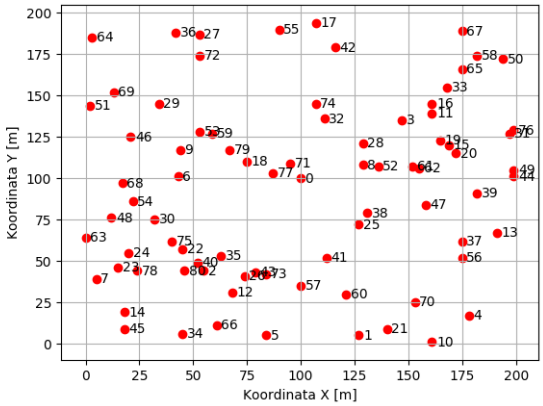
- V cenovni funkciji uporabljamo utežni spremenljivki $\alpha = 1$ in $\beta = 1000$.
- Algoritem gradientnega usmerjanja tabele za usmerjanje posodablja vsakih deset izvornih sporočil, torej vsakih deset minut.

Predpostavke pri algoritmu LEACH:

- Vodja gruče združi sporočila vseh svojih članov.
- Gruče menjamo vsakih 5 krogov, torej vsakih 5 minut.
- Število pričakovanih gruč je 5% vseh vozlišč.
- Oddajna moč pri izdelavi gruč je $56mJ$ kar pomeni domet 150 metrov.

3.4.3 Primerjava na omrežju z 81 naključno razporejenimi vozlišči

Začeli smo s simulacijami na naključni topologiji z 81 vozlišči, na kateri smo simulacijo ponovili 10 krat. Topologijo vidimo na sliki 3.9.

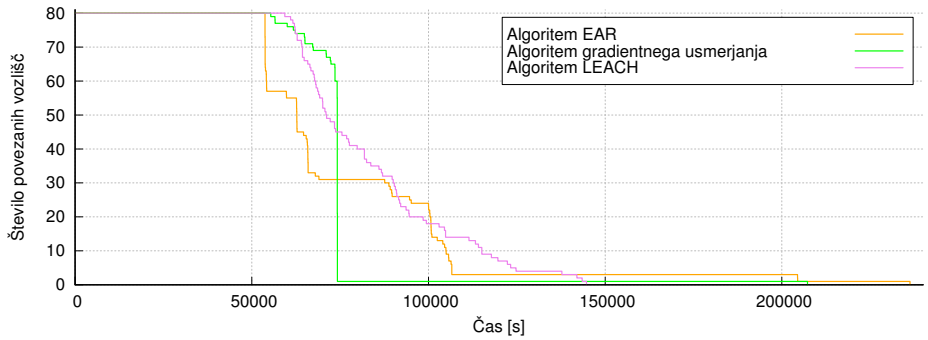


Slika 3.9: Naključna topologija 81 vozlišč.

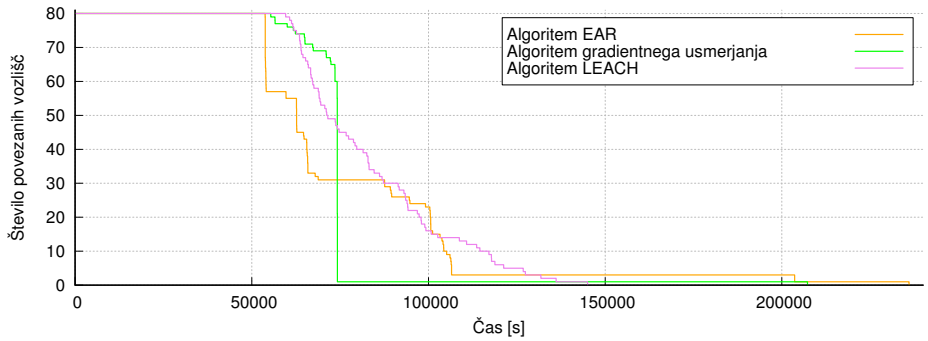
Na grafu 3.10 vidimo število povezanih vozlišč v odvisnosti od časa v primeru prve simulacije. Na grafu 3.11 vidimo povprečen čas odpovedi posameznega senzorja v 10 simulacijah. Oba grafa sta si zelo podobna. Iz tega lahko sklepamo, da naša simulacija ni odvisna od naključnega generatorja sporočil.

Pri 100% delovanju omrežja se najbolje odreže algoritem LEACH, sledi mu algoritem gradientnega usmerjanja, najslabši pa je algoritem EAR. Pri času ob 90% in 50% delujočega omrežja opazimo drugačen način odpovedi omrežja pri različnih algoritmih. Pri algoritmu EAR in gradientnem usmerjanju vidimo več navpičnih delov, kjer se v nekem trenutku odklopi več vozlišč. Vzrok za to je, odpoved določenega vozlišča, ki ga več vozlišč uporablja na poti do ponora. Algoritem LEACH omogoča prilagajanje oddajne moči, zato je vsako vozlišče vedno lahko povezano z ponorom. To vidimo kot počasen

odklop vozlišč od ponora. Pri 90% delujočega omrežja se algoritma LEACH in gradientno usmerjanje zamenjata, najboljši je torej gradientno usmerjanje. Pri 50% pa je ponovno najboljši algoritem LEACH. Točne čase odpovedi vidimo v tabeli 3.9.



Slika 3.10: Število povezanih vozlišč v odvisnosti od časa pri prvi simulaciji. Primerjava različnih algoritmov na topologiji s 81 naključno razporejenimi vozlišči.



Slika 3.11: Število povezanih vozlišč v odvisnosti od časa na povprečju 10 simulacij. Primerjava različnih algoritmov na topologiji s 81 naključno razporejenimi vozlišči.

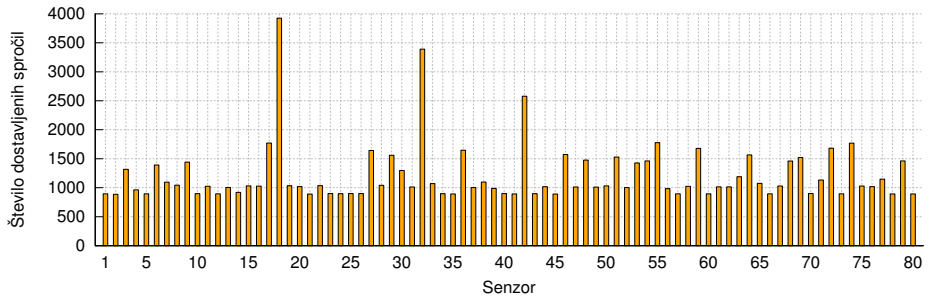
Odstotek delujočih vozlišč	EAR [s]	Gradientno usmerjanje [s]	LEACH [s]
100%	53783	55360	59583
90%	53810	65044	63703
50%	65634	74194	81497

Tabela 3.9: Čas do delovanja določenih odstotkov omrežja.

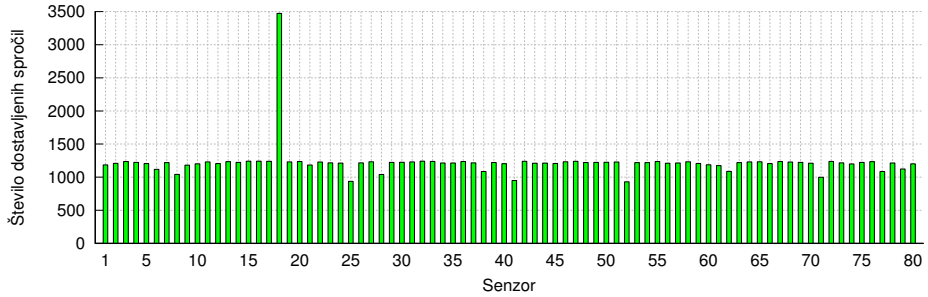
Sledijo še grafi s številom uspešno dostavljenih izvornih sporočil za vsako vozlišče posebej. Ponovno imamo podatke, ki so povprečje 10 simulacij. Na sliki 3.12 vidimo podatke za algoritem EAR. Opazimo povečano število sporočil pri vozliščih 18, 32 in 42, drugače pa so sporočila relativno enakomerno razporejena. Pri algoritmu gradientnega usmerjanja, slika 3.13, opazimo zelo enakomerno število sporočil razen za vozlišče 18. Vozlišče 18 pošlje opazno več sporočil, ker pride do situacije v kateri ostane edino povezano vozlišče. Na grafu, ki nam prikazuje število povezanih vozlišč lahko vidimo, da je ob času 74229 povezano samo še eno vozlišče. Ob tem času vidimo navpično črto, ki prikazuje padec iz 60 povezanih na eno povezano vozlišče, to se zgodi zaradi odpovedi enega vozlišča, ki je povezovalo celoten preostal del omrežja. V tem primeru so vsa ostala vozlišča odpovedala ali pa ostala brez povezave. Takšnih problemov pri algoritmu LEACH nimamo, saj posamezno vozlišče prilagodi z večjo oddajno močjo. Na sliki 3.14 vidimo, da pri algoritmu LEACH ni posameznega vozlišča, ki bi tako odstopal. Opazimo pa lahko, da je število sporočil bolj neenakomerno. V tabeli 3.10 vidimo še natančno število vseh prejetih sporočil, povprečno prejetih sporočil za posamezno vozlišče in največje odstopanje od povprečja.

	EAR	Gradientno	LEACH
Vsa prejeta sporočila	97107	97838	112873
Povprečno število sporočil na vozlišče	1213	1222	1410
Največje odstopanje od povprečja	2712	2251	1066

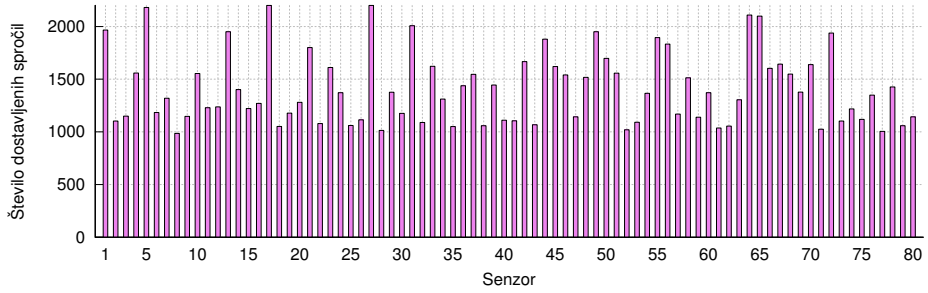
Tabela 3.10: Podatki o prejetih sporočilih na topologiji z 81 naključno razporejenimi vozlišči.



Slika 3.12: Povprečno število prejetih sporočil za posamezno vozlišče pri algoritmu EAR na topologiji s 81 naključno razporejenimi vozlišči.



Slika 3.13: Povprečno število prejetih sporočil za posamezno vozlišče pri algoritmu gradientnega usmerjanja na topologiji s 81 naključno razporejenimi vozlišči.



Slika 3.14: Povprečno število prejetih sporočil za posamezno vozlišče pri algoritmu LEACH na topologiji s 81 naključno razporejenimi vozlišči.

Simulacijo na tej topologiji 81 naključno razporejenih vozlišč smo torej izvedli deset krat. Naš cilj je pokazati, da naključni generator, ki ga uporabljamo, ne vpliva na rezultate. Uporabili smo statistično metodo Kolmogorov-Smirnov [24]. Kolmogorov-Smirnov test kvantificira razdaljo med empiričnima distribucijama dveh vzorcev. Rezultat testa je p vrednost. Ničelna hipoteza testa pravi, da sta distribuciji obeh vzorcev enaki. Ničelno hipotezo lahko zavrnemo, če je p vrednost manjša od 0,05. To pomeni, da če je p vrednost dveh vzorcev večja od 0,05 ne moremo zavrniti ničelne hipoteze. V našem primeru imamo več kot dve distribuciji, zato smo paroma primerjali rezultate vseh desetih simulacij. V tabeli 3.11 vidimo rezultate testa Kolmogorov-Smirnov med posameznimi pari. Podobno vidimo v tabeli 3.12 rezultate za algoritem gradientnega usmerjanja in v tabeli 3.13 za algoritem LEACH. Najnižje rezultate dobimo pri algoritmu EAR, še vedno pa so vsi testi večji od 0,05.

Simulacija / Simulacija	2	3	4	5	6	7	8	9	10
1	0,819	0,154	0,082	0,120	0,082	0,154	0,120	0,182	0,181
2	1	0,135	0,182	0,241	0,182	0,135	0,120	0,182	0,182
3	/	1	0,113	0,135	0,120	0,919	0,154	0,182	0,182
4	/	/	1	0,182	0,692	0,113	0,182	0,819	0,182
5	/	/	/	1	0,182	0,135	0,120	0,182	0,436
6	/	/	/	/	1	0,120	0,182	0,692	0,182
7	/	/	/	/	/	1	0,154	0,182	0,182
8	/	/	/	/	/	/	1	0,182	0,692
9	/	/	/	/	/	/	/	1	0,182

Tabela 3.11: Rezultati testa Kolmogorov-Smirnova pri algoritmu EAR.

Simulacija / Simulacija	2	3	4	5	6	7	8	9	10
1	0,677	0,807	0,975	0,975	0,229	0,149	0,543	0,543	0,112
2	1	0,229	0,975	0,543	0,975	0,420	0,912	0,112	0,677
3	/	1	0,543	0,912	0,229	0,149	0,912	0,912	0,162
4	/	/	1	0,912	0,807	0,229	0,997	0,314	0,543
5	/	/	/	1	0,420	0,112	0,677	0,677	0,162
6	/	/	/	/	1	0,807	0,807	0,175	0,975
7	/	/	/	/	/	1	0,229	0,107	0,912
8	/	/	/	/	/	/	1	0,677	0,543
9	/	/	/	/	/	/	/	1	0,131

Tabela 3.12: Rezultati testa Kolmogorov-Smirnova pri algoritmu gradientnega usmerjanja.

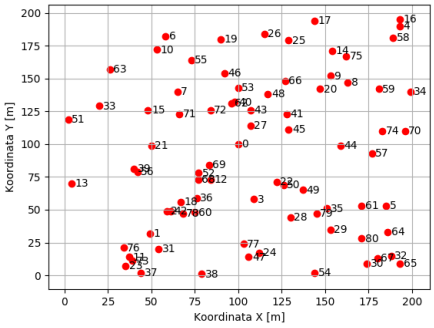
Simulacija / Simulacija	2	3	4	5	6	7	8	9	10
1	0,819	0,898	0,819	0,898	0,900	0,878	0,819	0,898	0,898
2	1	0,878	0,898	0,819	0,878	0,878	0,819	0,900	0,819
3	/	1	0,898	0,878	0,898	0,878	0,878	0,900	0,900
4	/	/	1	0,819	0,900	0,819	0,898	0,898	0,819
5	/	/	/	1	0,878	0,900	0,819	0,878	0,898
6	/	/	/	/	1	0,878	0,878	0,898	0,898
7	/	/	/	/	/	1	0,819	0,898	0,900
8	/	/	/	/	/	/	1	0,898	0,819
9	/	/	/	/	/	/	/	1	0,898

Tabela 3.13: Rezultati testa Kolmogorov-Smirnova pri algoritmu LEACH.

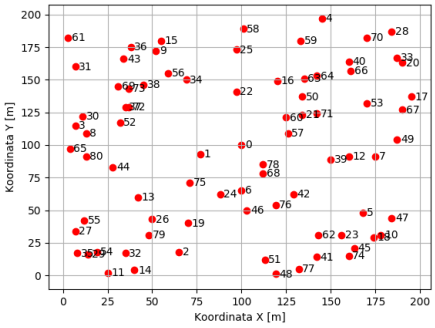
Pri primerjavi na omrežju z 81 naključno razporejenimi vozlišči se je najbolje odrezal algoritem LEACH. Bil je najboljši pri prvi odpovedi senzorja in pri polovici delujočega omrežja. Algoritem gradientnega usmerjanja je boljši samo pri odpovedi 10% omrežja.

3.4.4 Primerjava na različnih topologijah 81 naključno razporejenih vozlišč

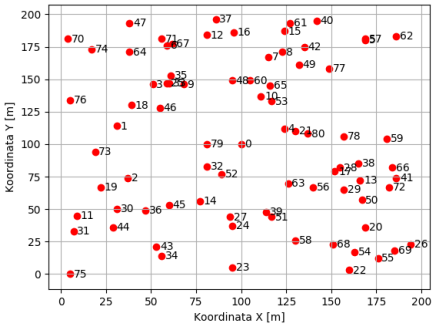
Zanima nas tudi, kako na rezultate vplivajo različne naključne topologije. Generirali smo deset topologij z 81 naključno razporejenimi vozlišči, ki jih vidimo na naslednjih slikah. Vidimo lahko, da so si topologije med sabo kar različne.



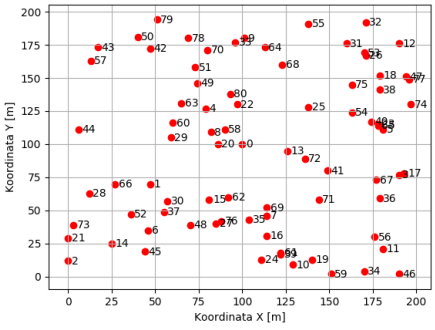
Slika 3.15: Topologija 1



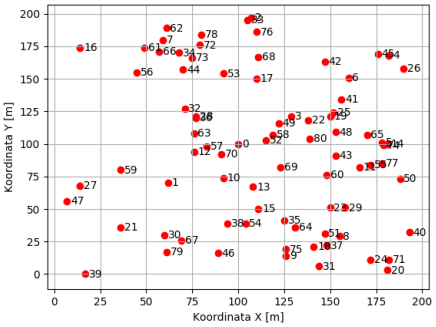
Slika 3.16: Topologija 2



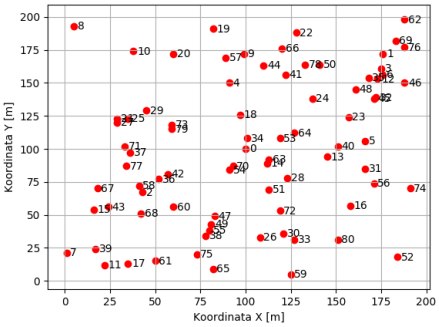
Slika 3.17: Topologija 3



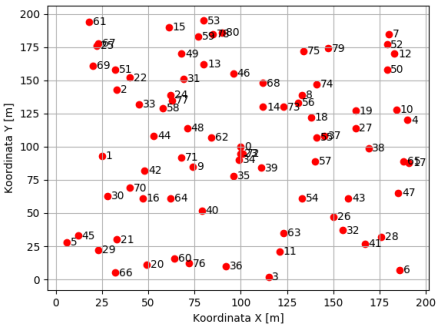
Slika 3.18: Topologija 4



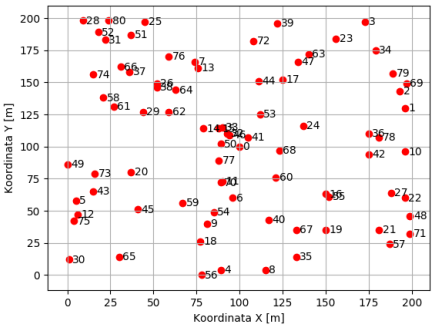
Slika 3.19: Topologija 5



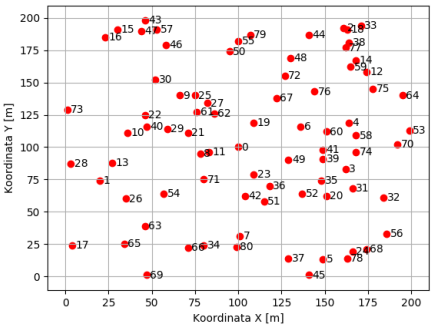
Slika 3.20: Topologija 6



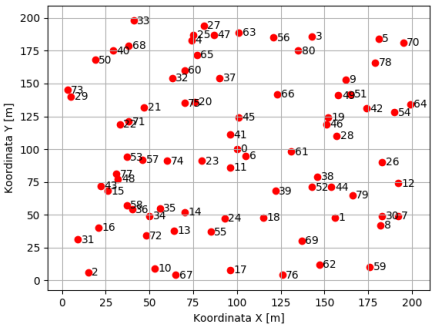
Slika 3.21: Topologija 7



Slika 3.22: Topologija 8



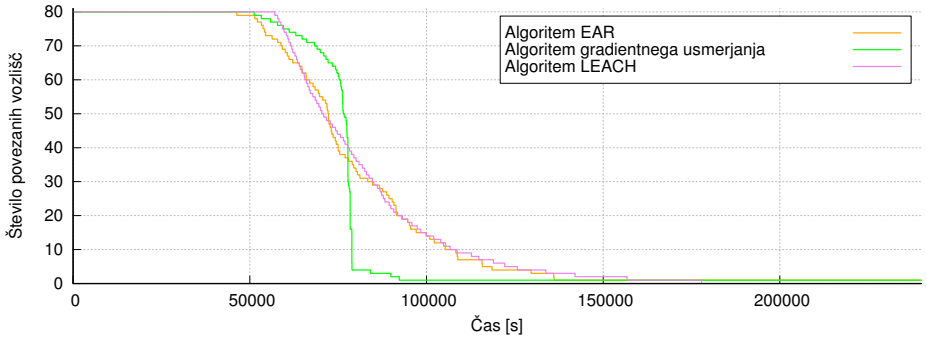
Slika 3.23: Topologija 9



Slika 3.24: Topologija 10

Na grafu 3.25 vidimo povprečen čas odpovedi vozlišč. Iz grafa lahko vidimo, da se najbolje odreže algoritem LEACH, sledi mu algoritem gradientnega usmerjanja, najslabši

pa je algoritem EAR. Preverili smo, če to drži tudi na vseh posameznih primerih. V tabeli 3.14 vidimo kolikokrat je bil posamezen algoritem najboljši pri posameznem kriteriju ocenjevanja. Pri prvi odpovedi vozlišča je bil devet krat najboljši algoritem LEACH enkrat pa algoritem gradientnega usmerjanja. Pri odpovedi 10% vozlišč je boljše gradientno usmerjanje. Pri 50% delujočega omrežja sta gradientno usmerjanje in algoritem LEACH primerljiva, enkrat pa je celo najboljši algoritem EAR. Pri primerjavi grafa 3.25 tabele 3.14 vidimo, da se rezultati ujemajo.



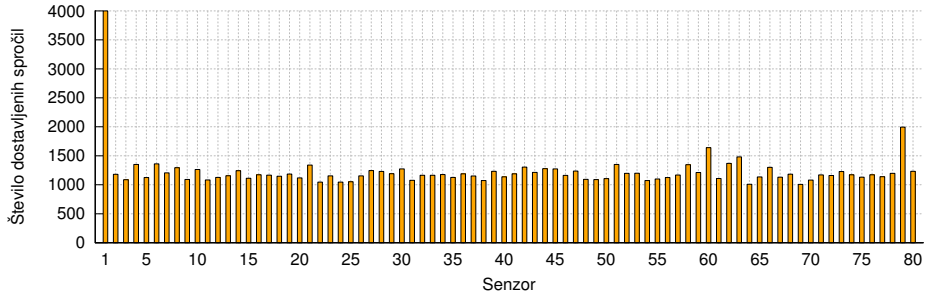
Slika 3.25: Število povezanih vozlišč v odvisnosti od časa na povprečju 10 simulacij. Primerjava različnih algoritmov na različnih naključnih topologijah s 81 naključno razporejenimi vozlišči.

Št. povezanih vozlišč	100%	90%	50%
EAR	0	0	1
Gradientno usmerjanje	1	8	5
LEACH	9	2	4

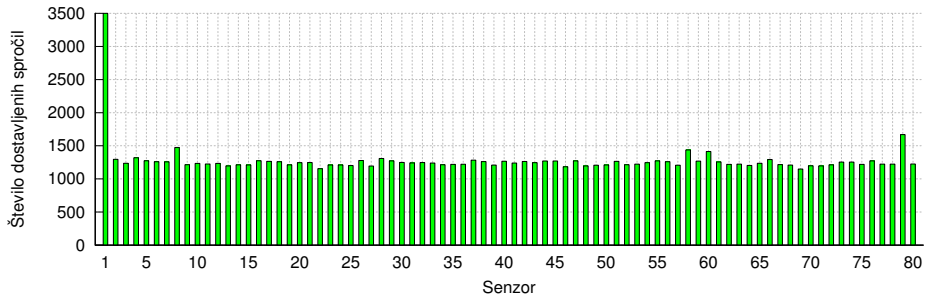
Tabela 3.14: Število zmag za vsak algoritem pri posameznih naključnih topologijah

Na slikah 3.26, 3.27 in 3.28 vidimo povprečno število prejetih sporočil za posamezno vozlišče. Pri algoritmu EAR in gradientnem usmerjanju vidimo, da ima vozlišče 1 opazno več prejetih sporočil. Po pregledu rezultatov smo ugotovili, da je pri drugi topologiji vozlišče 1 v neposredni bližini ponora. V tem primeru tako vozlišče 1 uspešno dostavi veliko več sporočil, v primeru algoritma EAR kar 17182, v primeru gradientnega usmerjanja

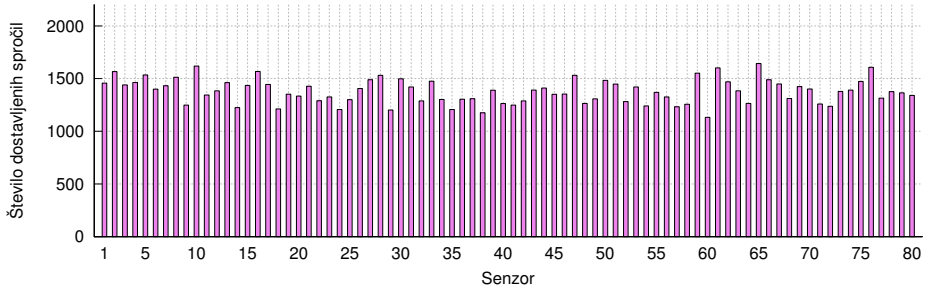
pa 14808. Povprečno število prejetih sporočil je na tej topologiji 1404 prejetih sporočil pri algoritmu EAR in 1425 pri algoritmu gradientnega usmerjanja. Vidimo torej, da ta anomalija povzroči tako veliko število povprečnih sporočil za vozlišče 1. Pri algoritmu LEACH do tega problema na pride.



Slika 3.26: Povprečno število prejetih sporočil za posamezno vozlišče pri algoritmu EAR na topologiji s 81 naključno razporejenimi vozlišči.

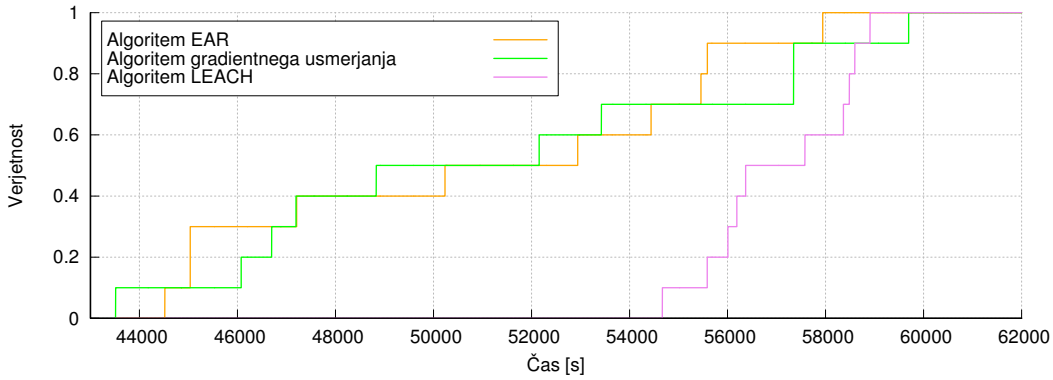


Slika 3.27: Povprečno število prejetih sporočil za posamezno vozlišče pri algoritmu gradientnega usmerjanja na topologiji s 81 naključno razporejenimi vozlišči.

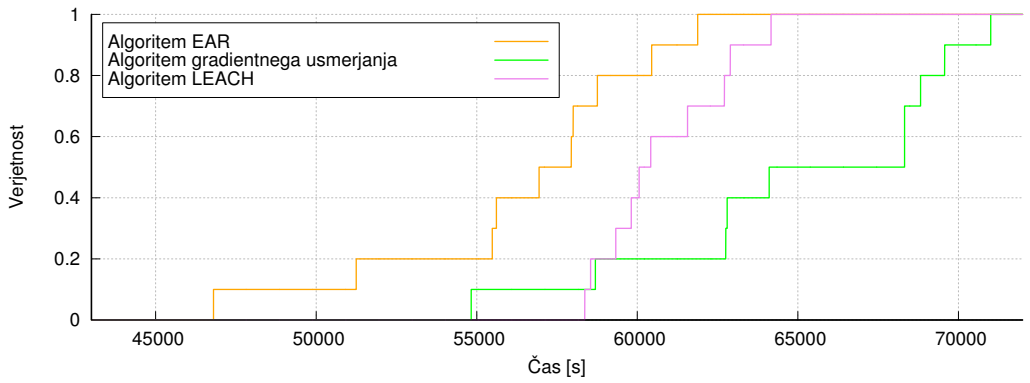


Slika 3.28: Povprečno število prejetih sporočil za posamezno vozlišče pri algoritmu LEACH na topologiji s 81 naključno razporejenimi vozlišči.

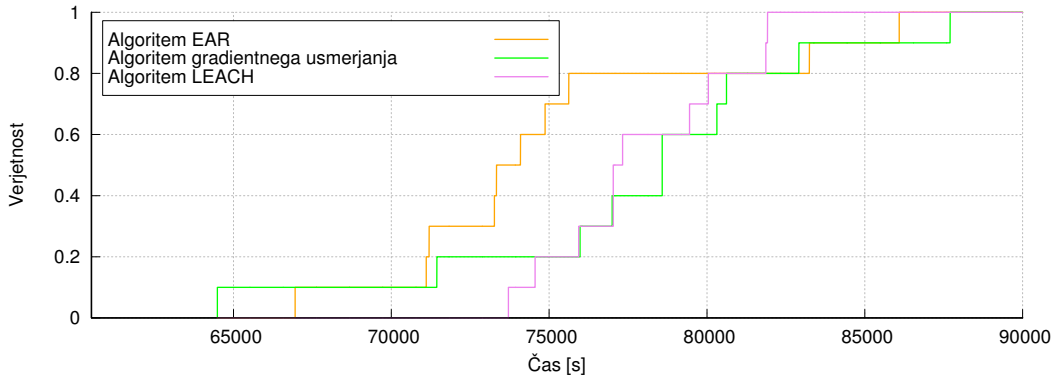
Na naslednjih grafih smo prikazali kumulativno porazdelitveno funkcijo. Graf 3.29 nam prikazuje kakšna je verjetnost odpovedi prvega vozlišča v odvisnosti od časa. Za primer poglejmo čas 56000 sekund, kjer vidimo, da je verjetnost odpovedi prvega senzorja pri algoritmu EAR že 90%, pri algoritmu gradientnega usmerjanja 70% in pri algoritmu LEACH samo 30%. Graf 3.30 predstavlja verjetnost odpovedi 10% omrežja in graf 3.31 predstavlja verjetnost odpovedi polovice omrežja. Na grafih potrdimo to, kar vidimo tudi v tabeli 3.14, pri prvi odpovedi je najboljši algoritem LEACH, pri 10% odpovedi omrežja je najboljši algoritem gradientnega usmerjanja, pri 50% pa sta algoritma LEACH in gradientno usmerjanje primerljiva. V vseh primerih pa je algoritem EAR najslabši.



Slika 3.29: Verjetnost prve odpovedi v odvisnosti od časa za posamezen algoritem.



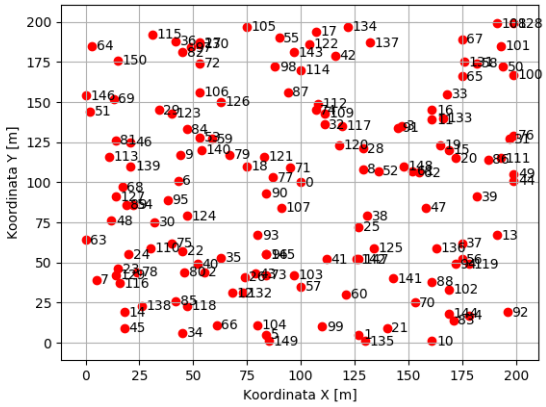
Slika 3.30: Verjetnost odpovedi 10% omrežja v odvisnosti od časa za posamezen algoritem.



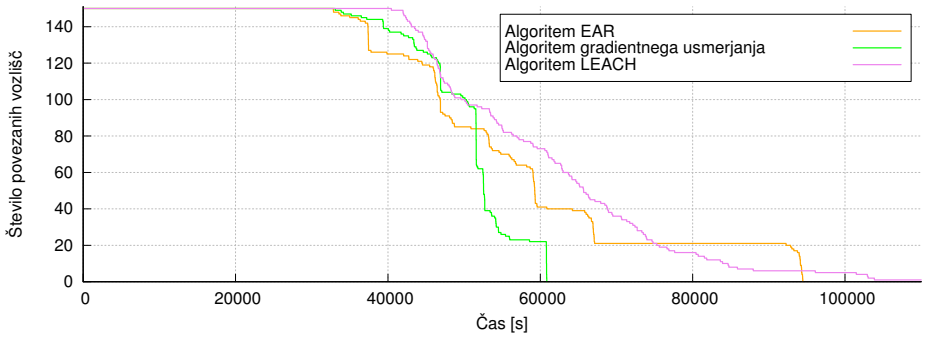
Slika 3.31: Verjetnost odpovedi polovice omrežja v odvisnosti od časa za posamezen algoritem.

3.4.5 Primerjava na omrežju s 151 naključno razporejenimi vozlišči

Sledi še primerjava algoritmov na omrežju s 151 vozlišči. Ponovno smo naredili 10 ponovitev na isti topologiji, ki jo vidimo na sliki 3.32. Na sliki 3.33 vidimo število povezanih vozlišč v odvisnosti od časa. Vidimo lahko, da prvi odpove algoritem EAR, sledita pa mu gradientno usmerjanje in algoritem LEACH. Algoritem LEACH je najboljši tako v prvi odpovedi, kot tudi pri 90% in 50% delujočega omrežja. V tabeli 3.15 vidimo točne čase odpovedi za posamezne algoritme.



Slika 3.32: Naključna topologija 151 vozlišč.



Slika 3.33: Število povezanih vozlišč v odvisnosti od časa. Primerjava različnih algoritmov na topologiji s 151 naključno razporejenimi vozlišči.

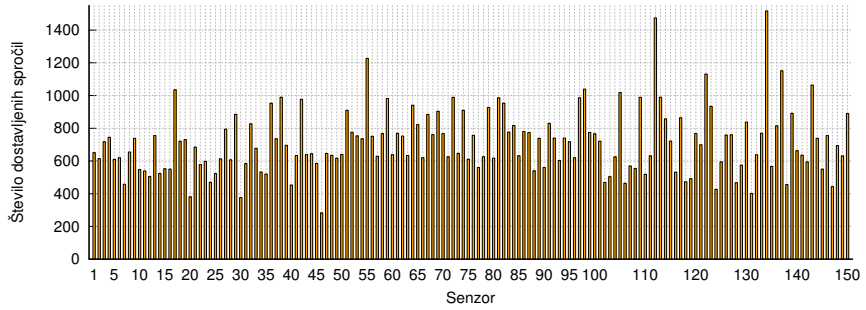
Odstotek delujočih vozlišč	EAR [s]	Gradientno usmerjanje [s]	LEACH [s]
100%	32833	33081	40448
90%	36107	37221	42416
50%	53326	51586	59070

Tabela 3.15: Čas do delovanja določenih odstotkov omrežja.

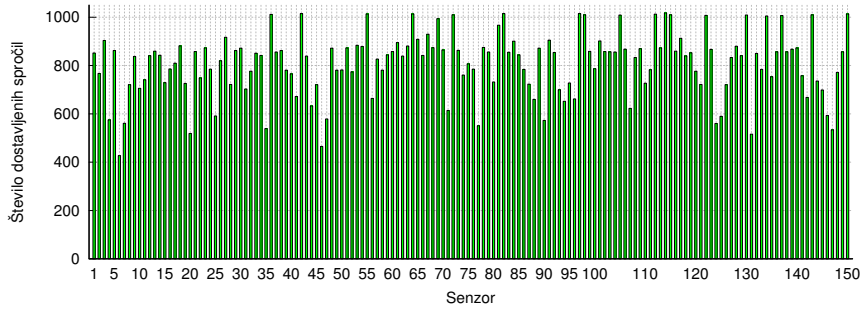
Poglejmo si še število prejetih sporočil. V tabeli 3.16 vidimo, da je največ dostavljenih sporočil pri algoritmu LEACH, ponovno pa je najbolj enakomerno gradientno usmerjanje. Na sliki 3.34 vidimo, da imajo največ sporočil vozlišča 17, 55, 112 in 134. To so vozlišča, ki so postavljena skupaj v bližini in sicer okoli koordinate (100, 200). V tem delu je veliko vozlišč, zato ta del omrežja preživi dalj časa in s tem pošlje več sporočil. Predvsem je za to pomemben tudi del v okolici koordinate (110, 130). Tam vidimo veliko vozlišč, ki so povezana s ponorom in služijo kot pot sporočilom od vozlišč z največ sporočili. Pri gradientnem usmerjanju na sliki 3.35 opazimo podoben pojav in sicer več poslanih sporočil iz tega dela omrežja. Razlika je, da je zaradi posodabljanja poti ob delovanju v tem primeru več vozlišč, ki imajo več poslanih sporočil, vendar pa število sporočil ni tako veliko večje. Vozlišča, ki pošljejo več kot 1000 sporočil so 36, 42, 55, 64, 72, 82, 97, 98, 105, 112, 114, 115, 122, 130, 134, 137, 143 in 150. Vidimo, da so to pretežno vozlišča iz zgornjega srednjega in zgornjega levega dela. Kot vidimo na sliki 3.36 pri algoritmu LEACH največ sporočil pošljeta vozlišči 64 in 128 in sicer 1694 in 1979. Vozlišče 64 se nahaja v levem zgornjem kotu, vozlišče 128 pa v desnem zgornjem kotu. Glavni razlog zakaj sta te dve vozlišči bolj neobremenjeni, je to da prejemata manj sporočil. V celotnem omrežju posamezno vozlišče prejme povprečno 55 tisoč sporočil, vsa ta sporočila vozlišču porabljajo energijo. Vozlišče 64 prejme 40 tisoč sporočil, vozlišče 128 pa 37 tisoč sporočil.

	EAR	Gradientno	LEACH
Vsa prejeta sporočila	107038	121394	150488
Povprečno število sporočil na vozlišče	713	809	1003
Največje odstopanje od povprečja	803	382	957

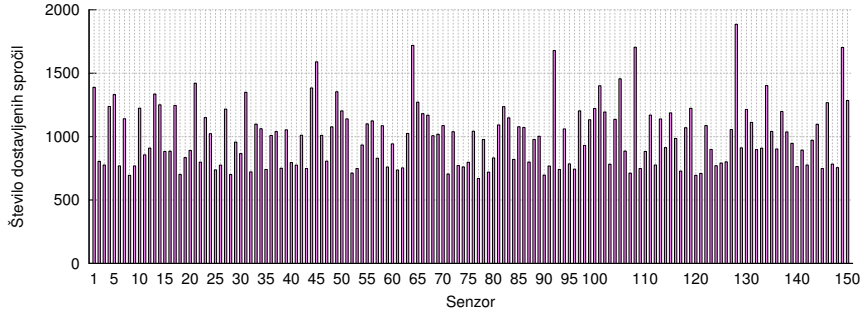
Tabela 3.16: Podatki o prejetih sporočilih na topologiji z 151 naključno razporejenimi vozlišči.



Slika 3.34: Število prejetih sporočil za posamezno vozlišče pri algoritmu EAR na topologiji s 151 naključno razporejenimi vozlišči.



Slika 3.35: Število prejetih sporočil za posamezno vozlišče pri algoritmu gradientnega usmerjanja na topologiji s 151 naključno razporejenimi vozlišči.



Slika 3.36: Število prejetih sporočil za posamezno vozlišče pri algoritmu LEACH na topologiji s 151 naključno razporejenimi vozlišči.

Ponovno smo uporabili test Kolmogorov-Smirnova za preverjanje podobnosti rezultatov. V tabeli 3.17 vidimo rezultate za algoritem EAR, kjer so vrednosti ponovno najnižje a še vedno večje od mejne vrednosti 0,05. Sledi še tabela 3.18 z rezultati za algoritem gradientnega usmerjanja in tabela 3.19 ta algoritem LEACH.

Simulacija / Simulacija	2	3	4	5	6	7	8	9	10
1	0,437	0,227	0,136	0,287	0,104	0,287	0,177	0,227	0,104
2	1	0,227	0,136	0,227	0,136	0,287	0,136	0,136	0,136
3	/	1	0,177	0,227	0,104	0,437	0,136	0,227	0,104
4	/	/	1	0,178	0,287	0,357	0,437	0,437	0,287
5	/	/	/	1	0,104	0,357	0,104	0,104	0,104
6	/	/	/	/	1	0,104	0,437	0,287	0,357
7	/	/	/	/	/	1	0,177	0,287	0,104
8	/	/	/	/	/	/	1	0,526	0,357
9	/	/	/	/	/	/	/	1	0,227

Tabela 3.17: Rezultati testa Kolmogorov-Smirnova pri algoritmu EAR.

Simulacija / Simulacija	2	3	4	5	6	7	8	9	10
1	0,437	0,949	0,812	0,983	0,719	0,812	0,287	0,357	0,983
2	1	0,437	0,983	0,949	0,997	0,997	0,997	0,890	0,890
3	/	1	0,719	1,000	0,812	0,949	0,357	0,437	0,983
4	/	/	1	0,983	0,949	0,890	0,812	0,890	0,949
5	/	/	/	1	0,997	0,997	0,812	0,719	0,949
6	/	/	/	/	1	0,949	0,949	0,719	0,890
7	/	/	/	/	/	1	0,719	0,812	0,997
8	/	/	/	/	/	/	1	0,983	0,437
9	/	/	/	/	/	/	/	1	0,890

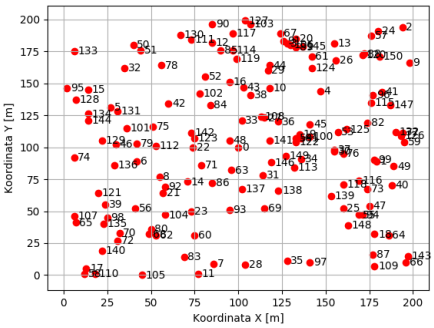
Tabela 3.18: Rezultati testa Kolmogorov-Smirnova pri algoritmu gradientnega usmerjanja.

Simulacija / Simulacija	2	3	4	5	6	7	8	9	10
1	0,847	0,882	0,618	0,808	0,888	0,882	0,618	0,888	0,847
2	1	0,900	0,900	0,900	0,888	0,900	0,882	0,888	0,888
3	/	1	0,896	0,896	0,847	0,900	0,882	0,900	0,715
4	/	/	1	0,900	0,808	0,896	0,900	0,888	0,522
5	/	/	/	1	0,888	0,896	0,896	0,900	0,522
6	/	/	/	/	1	0,896	0,808	0,847	0,896
7	/	/	/	/	/	1	0,896	0,896	0,808
8	/	/	/	/	/	/	1	0,882	0,618
9	/	/	/	/	/	/	/	1	0,888

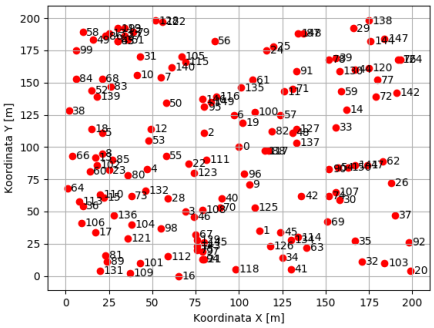
Tabela 3.19: Rezultati testa Kolmogorov-Smirnova pri algoritmu LEACH.

3.4.6 Primerjava na različnih topologijah 151 naključno razporejenih vozlišč

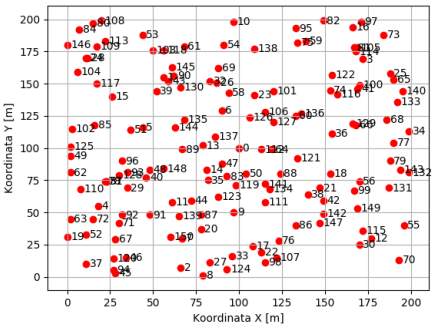
Simulacijo na desetih različnih topologijah smo ponovili tudi pri 151 vozliščih. Topologije vidimo na naslednjih slikah.



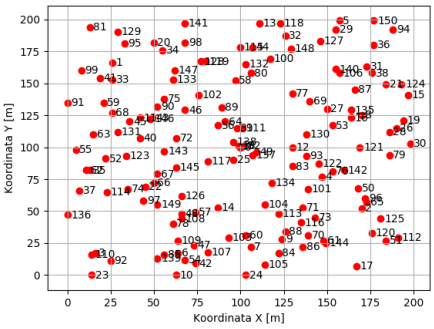
Slika 3.37: Topologija 1



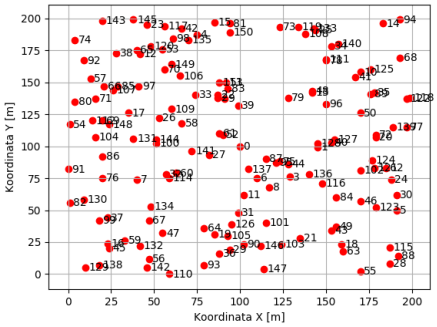
Slika 3.38: Topologija 2



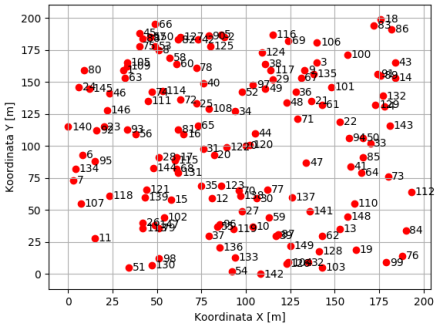
Slika 3.39: Topologija 3



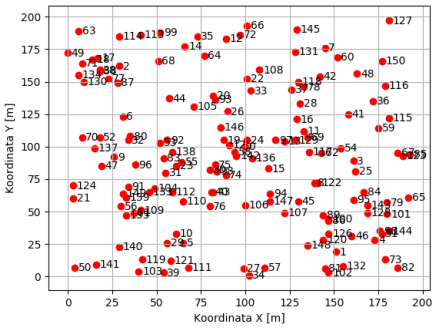
Slika 3.40: Topologija 4



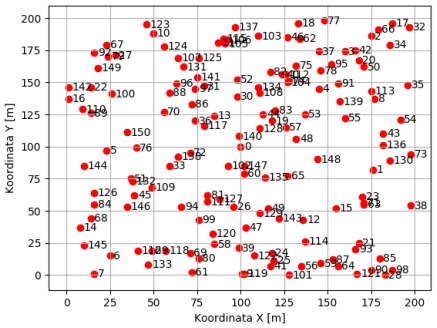
Slika 3.41: Topologija 5



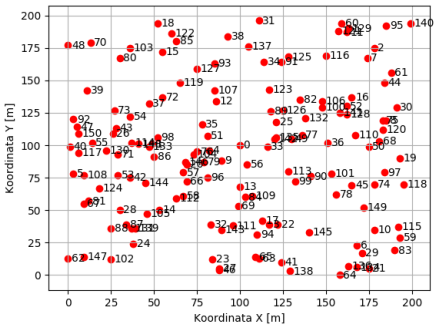
Slika 3.42: Topologija 6



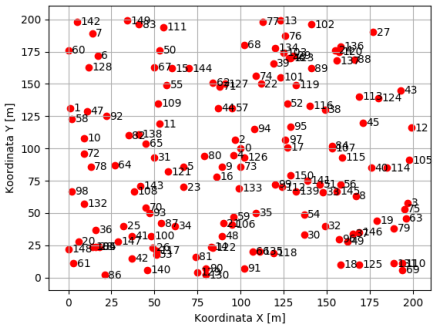
Slika 3.43: Topologija 7



Slika 3.44: Topologija 8



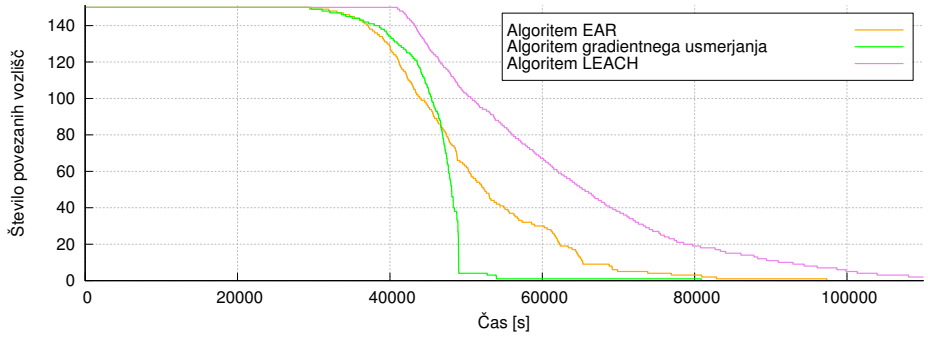
Slika 3.45: Topologija 9



Slika 3.46: Topologija 10

Na sliki 3.47 vidimo povprečen čas odpovedi. Brez dvoma je na topologiji v vseh primerih najboljši algoritem LEACH. V tabeli 3.20 vidimo, da je algoritem LEACH vedno

najboljši v vseh kriterijih razen enkrat, ko je algoritem gradientnega usmerjanja boljši pri odpovedi 10% omrežja.



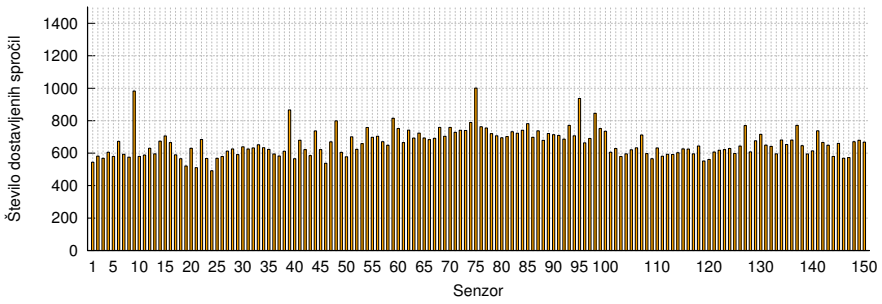
Slika 3.47: Število povezanih vozlišč v odvisnosti od časa na povprečju 10 simulacij. Primerjava različnih algoritmov na različnih naključnih topologijah s 151 naključno razporejenimi vozlišči.

Št. povezanih vozlišč	100%	90%	50%
EAR	0	0	0
Gradientno usmerjanje	0	1	0
LEACH	10	9	10

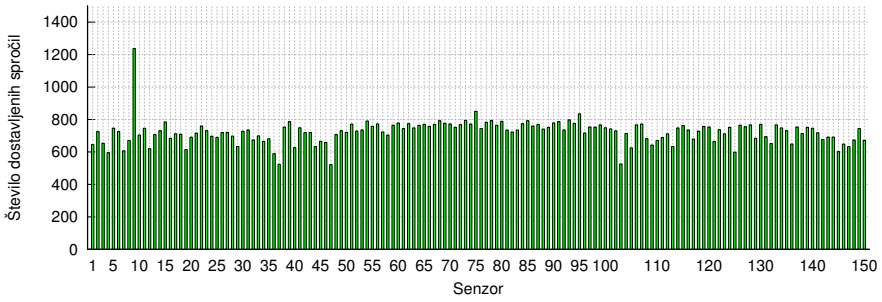
Tabela 3.20: Število zmag za vsak algoritem pri posameznih naključnih topologijah

Na grafih 3.48 in 3.49 opazimo, da pri algoritmu EAR in gradientnem usmerjanju vozlišče 9 dobi povprečno več sporočil. Pri analizi podatkov smo ugotovili, da je razlog za to topologija številka 5, v kateri vozlišče 9 (na koordinati (91,137)) dostavi kar 4964 sporočil pri algoritmu EAR in 6173 sporočil pri gradientnem usmerjanju. Drugače je povprečno število prejetih sporočil na vozlišče nekje med 600-800 sporočili.

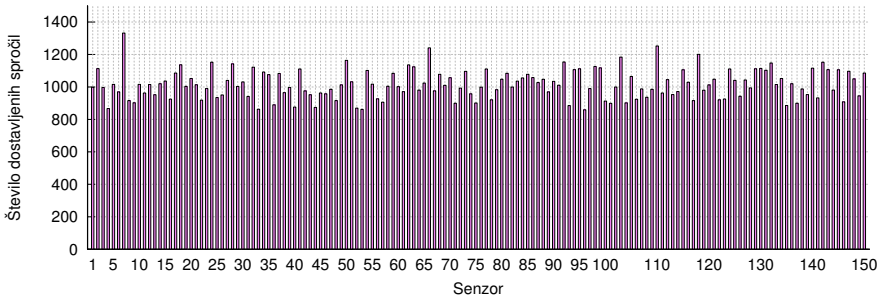
Kot vidimo na grafu 3.50 je malo bolj izrazito vozlišče 7. Ugotovili smo, da pri topologijah 6, 8 in 10 to vozlišče dobi malo več sporočil (1950, 2135 in 1605 sporočil). Povprečno vozlišče pa pri tem algoritmu dobi okoli 1000 sporočil.



Slika 3.48: Povprečno število prejetih sporočil za posamezno vozlišče pri algoritmu EAR na topologiji s 151 naključno razporejenimi vozlišči.

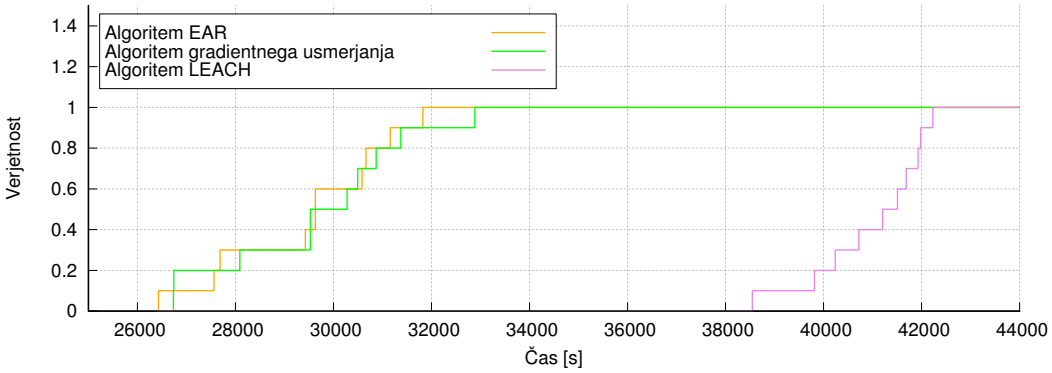


Slika 3.49: Povprečno število prejetih sporočil za posamezno vozlišče pri algoritmu gradientnega usmerjanja na topologiji s 151 naključno razporejenimi vozlišči.

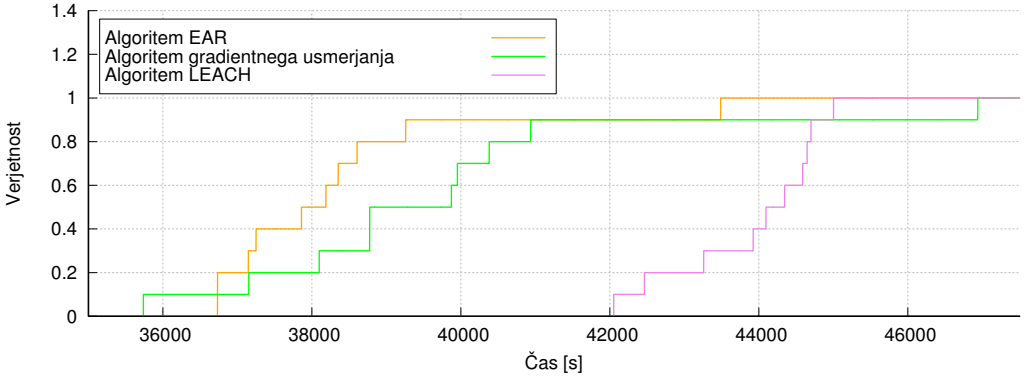


Slika 3.50: Povprečno število prejetih sporočil za posamezno vozlišče pri algoritmu LEACH na topologiji s 151 naključno razporejenimi vozlišči.

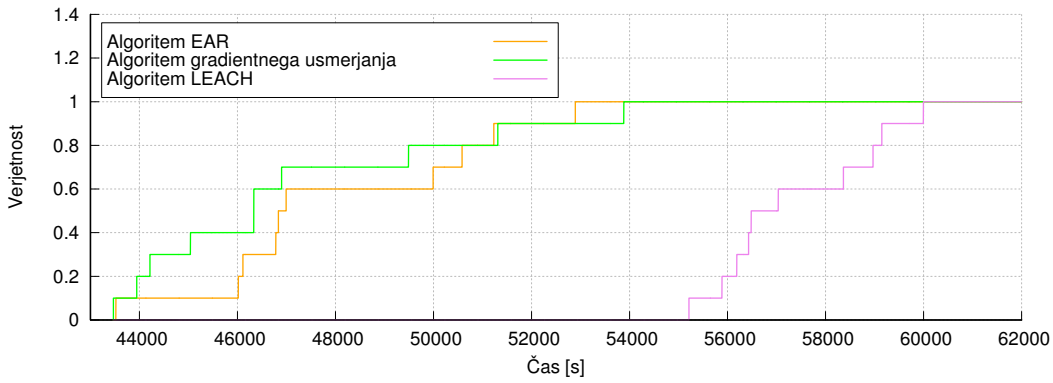
Poglejmo si še kumulativno porazdelitveno funkcijo. Graf 3.51 nam prikazuje kakšna je verjetnost odpovedi prvega vozlišča v odvisnosti od časa. Algoritem LEACH je opazno boljši, algoritma EAR in gradientno usmerjanje pa sta zelo izenačena. Graf 3.52 predstavlja verjetnost odpovedi 10% omrežja in graf 3.53 predstavlja verjetnost odpovedi polovice omrežja. Tudi tu je v obeh primerih algoritem LEACH najboljši. Pri odpovedi 10% omrežja mu sledi algoritem gradientnega usmerjanja, najslabši pa je algoritem EAR. Pri odpovedi 50% omrežja sta algoritma EAR in gradientno usmerjanje primerljiva, malce je boljši algoritem EAR.



Slika 3.51: Verjetnost prve odpovedi v odvisnosti od časa za posamezen algoritem.



Slika 3.52: Verjetnost odpovedi 10% omrežja v odvisnosti od časa za posamezen algoritem.



Slika 3.53: Verjetnost odpovedi polovice omrežja v odvisnosti od časa za posamezen algoritem.

3.5 Sklep simulacij

Cilj naših simulacij je bila primerjava različnih algoritmov na različnih topologijah. Za izključitev odvisnosti od generatorja naključnih števil smo najprej simulacijo na eni topologiji ponovili deset krat. Preverili pa smo tudi kako si razlikujejo različne naključne topologije. Tudi tukaj smo naredili deset različnih simulacij. Ugotovili smo, da so rezultati simulacij med sabo zelo podobni. Še največ razlik smo imeli pri testiranju različnih topologij z 81 vozlišči. Vseeno pa so tudi v tem primeru rezultati konstantni pri primerjanju časa do prve odpovedi in do odpovedi 10% omrežja. Pri času odpovedi polovice omrežja pa smo imeli bolj ne konstantne rezultate.

Glede na rezultate simulacij lahko rečemo, da se je v naših pogojih najbolje odrezal algoritem LEACH. V nekaterih primerih je bil algoritem gradientnega usmerjanja boljši, predvsem pri času do odpovedi 10% omrežja pri topologijah z 81 vozlišči. Malo boljši pa je algoritem gradientnega usmerjanja v enakomernosti dostavljenih sporočil. To lepo vidimo tudi na grafih, ki prikazujejo število povezanih vozlišč, saj je krivulja ponavadi bolj navpična, kar pomeni, da v nekem trenutku izgubi povezavo več vozlišč.

Menimo, da je glavna prednost algoritma LEACH možnost spreminjanja oddajne moči in s tem dosega. Pri drugih dveh algoritmih je moč oddajanja konstantna. Možnost spreminjanja oddajne moči ima dve glavni prednosti. Prva je ta, da vedno uporabimo kar se da malo moči za dostavljanje sporočila, kar pomeni, da lahko vozlišče prihrani veliko energije, če pošilja sporočilo bližnjemu sosedu. Ob enem pa manjša oddajna moč pomeni tudi, da manj vozlišč prejme sporočilo in zato se tudi ostalim vozliščem manj troši energija. Druga prednost spreminjanja oddajne moči pa je to, da posamezno vozlišče nikoli ni brez poti

do ponora. V primeru, da so ostala vozlišča odpovedala ima vozlišče še vedno možnost uporabe višje moči in s tem neposredno komunikacijo s ponorom. Kot nadaljevanje raziskave bi bilo torej zanimivo implementirati nekakšen način izbire spreminjanja oddajne moči tudi pri algoritmu gradientnega usmerjanja.

Poglavje 4

Zaključek

V magistrski nalogi smo si ogledali aktualno področje brezžičnih senzorskih omrežij. V delu smo se osredotočili predvsem na različne načine energetsko učinkovitega usmerjanja med vozlišči. Primerjali smo tri algoritme, in sicer algoritem energetsko učinkovitega usmerjanja (krajše EAR), algoritem gradientnega usmerjanja in algoritem usmerjanja na principu nizkoenergetske adaptivne hierarhije gruče (krajše LEACH). V simulacijah se je najbolje odrezal algoritem LEACH, sledi mu algoritem gradientnega usmerjanja, najslabši pa je algoritem EAR. Velika pomanjkljivost algoritma LEACH je to, da za pravilno delovanje zahteva neposredno komunikacijo vseh vozlišč s prehodom. To pa pomeni, da nam velikost nadzorovanega območja omejuje brezžična tehnologija in njen domet. Pri algoritmu EAR in algoritmu gradientnega usmerjanja pa nismo omejeni z velikostjo nadzorovanega območja, saj je lahko vozlišče poljubno oddaljeno od prehoda. Zagotoviti moramo le dovolj vmesnih vozlišč, da obstaja pot, po kateri lahko pošljemo sporočilo do prehoda.

Imamo tudi nekaj idej za nadaljevanje raziskovanja na tem področju. Zanimiva bi bila analiza še kakšne druge topologije omrežja, kot je na primer uporaba omrežja z večimi prehodi. V simulaciji smo večkrat opazili, da so najbolj obremenjena vozlišča tista, ki so v neposredni bližini prehoda. Algoritem LEACH to pomanjkljivost poizkuša odpraviti z adaptivno močjo oddajanja zato ne potujejo vsa sporočila preko vozlišč, ki so v neposredni bližini. S pomočjo večih prehodov bi se izognili tem vročim točkam omrežja. Druga ideja za nadaljevanje pa je izboljšava algoritma gradientnega usmerjanja s pomočjo spreminjanja oddajne moči med delovanjem.

Literatura

[1] J. Gutiérrez, J. F. Villa-Medina, A. Nieto-Garibay, M. Á. Porta-Gándara, Automated irrigation system using a wireless sensor network and gprs module, IEEE transactions on instrumentation and measurement 63 (1) (2014) 166–176.

[2] Y. E. Aslan, I. Korpeoglu, Ö. Ulusoy, A framework for use of wireless sensor networks in forest fire detection and monitoring, Computers, Environment and Urban Systems 36 (6) (2012) 614–625.

[3] B. Son, Y.-s. Her, J.-G. Kim, A design and implementation of forest-fires surveillance system based on wireless sensor networks for south korea mountains, International Journal of Computer Science and Network Security (IJCSNS) 6 (9) (2006) 124–130.

[4] K. K. Khedo, R. Perseedoss, A. Mungur, et al., A wireless sensor network air pollution monitoring system, arXiv preprint arXiv:1005.1737 2 (2) (2010) 31–45.

[5] S. Misra, S. Goswami, Exterior gateway protocol, Network Routing: Fundamentals, Applications, and Emerging Technologies (2017) 159–193.

[6] C. L. Hedrick, Routing information protocol, Tech. rep. (1988).

[7] J. Yan, M. Zhou, Z. Ding, Recent advances in energy-efficient routing protocols for wireless sensor networks: A review, IEEE Access 4 (2016) 5673–5686.

[8] N. Balhara, T. Malik, Review paper on integration of robust different hierarchical routing protocol of wireless sensor network, International Journal of Advance Research , Ideas and Innovations in Technology 3 (2017) 854–858.

[9] H. Echoukairi, K. Bourgba, M. Ouzzif, A survey on flat routing protocols in wireless sensor networks, in: Advances in Ubiquitous Networking, Springer, 2016, pp. 311–324.

[10] P. Ran, M.-h. Sun, Y.-m. Zou, Zigbee routing selection strategy based on data services and energy-balanced zigbee routing, in: Services Computing, 2006. APSCC'06. IEEE Asia-Pacific Conference on, IEEE, 2006, pp. 400–404.

[11] A. Tanenbaum, D. Wetherall, Computer Networks, Pearson custom library, Pearson, 2013.
URL https://books.google.si/books?id=w_d5ngEACAAJ

[12] R. C. Shah, J. M. Rabaey, Energy aware routing for low energy ad hoc sensor networks, in: Wireless Communications and Networking Conference, 2002. WCNC2002. 2002 IEEE, Vol. 1, IEEE, 2002, pp. 350–355.

[13] S. Yessad, L. Bouallouche-Medjkoune, D. Aissani, Proposition and evaluation of a novel routing protocol for wireless sensor networks, in: Proceedings of the Fifth international conference on verification and evaluation of computer and communication systems, 2011, pp. 1–9.

[14] T. Watteyne, K. Pister, D. Barthel, M. Dohler, I. Auge-Blum, Implementation of gradient routing in wireless sensor networks, in: Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE, IEEE, 2009, pp. 1–6.

[15] G. Hou, K. W. Tang, E. Noel, Implementation and analysis of the leach protocol on the tinyos platform, in: ICT Convergence (ICTC), 2013 International Conference on, IEEE, 2013, pp. 918–923.

[16] W. R. Heinzelman, A. Chandrakasan, H. Balakrishnan, Energy-efficient communication protocol for wireless microsensor networks, in: System sciences, 2000. Proceedings of the 33rd annual Hawaii international conference on, IEEE, 2000, pp. 10–pp.

[17] W. Xinhua, W. Sheng, Performance comparison of leach and leach-c protocols by ns2, in: Distributed Computing and Applications to Business Engineering and Science (DCABES), 2010 Ninth International Symposium on, IEEE, 2010, pp. 254–258.

[18] K. Pawar, V. Pawar, T. Sharma, Enhancement of leach protocol using energy heterogeneity concept, Internal Journal of Emerging Trends and Technology in Computer Science 2 (1) (2013) 49–56.

[19] J. Xu, N. Jin, X. Lou, T. Peng, Q. Zhou, Y. Chen, Improvement of leach protocol for wsn, in: Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on, IEEE, 2012, pp. 2174–2177.

[20] G. R. S. Reddy, S. Balaji, A review on different types of leach protocol for wireless sensor networks 2 (4) (2017) 840–844.

[21] A. Varga, R. Hornig, An overview of the omnet++ simulation environment, in: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 60.

[22] W. B. Heinzelman, Application-specific protocol architectures for wireless networks, Ph.D. thesis, Cambridge, MA, USA, aAI0801929 (2000).

[23] L. Tang, S. Liu, Improvement on leach routing algorithm for wireless sensor networks, in: 2011 International Conference on Internet Computing and Information Services, 2011, pp. 199–202. doi:10.1109/ICICIS.2011.57.

[24] M. Hazewinkel, Kolmogorov-smirnov test, Encyclopedia of Mathematics, Springer, ISBN (2001) 978–1.