Kari Lappalainen

Hajautettujen keinotekoisten neuroverkkojen soveltaminen langattomien lähiverkkojen tunkeutumisen havaitsemisjärjestelmiin

Tietotekniikan
(Mobiilijärjestelmät)
pro gradu -tutkielma
15. marraskuuta 2011

JYVÄSKYLÄN YLIOPISTO
TIETOTEKNIIKAN LAITOS

Jyväskylä

Tekijä: Kari Lappalainen
Yhteystiedot: kari.lappalainen@jyu.fi
Työn nimi: Hajautettujen keinotekoisten neuroverkkojen soveltaminen langattomien lähiverkkojen tunkeutumisen havaitsemisjärjestelmiin
Title in English: The Appliance of Artificial Neural Networks for Intrusion Detection Systems of Wireless Local Area Networks
Työ: Tietotekniikan (Mobiilijärjestelmät) pro gradu -tutkielma
Sivumäärä: 125
Tiivistelmä: Langattomien lähiverkkojen yleistyminen tarjoaa mahdollisille tunkeutujille uusia reittejä tietojärjestelmiin. Samaan aikaan tietojärjestelmien merkitys ihmisten ja yhteiskunnan kannalta kasvaa koko ajan. Tietojärjestelmiä turvaamaan tarvitaan tunkeutumisyritykset havaitsevia järjestelmiä. Tietotekniikan arkipäiväistyessä tietojärjestelmien käyttö koskettaa yhä useampia ihmisiä, joten järjestelmien helppokäyttöisyys korostuu entisestään. Tutkimuksessa selvitettiin itseoppiviin hajautettuihin neuroverkkoihin perustuvan tunkeutumisen havaitsemisjärjestelmän toteutusmahdollisuuksia. Tutkimuksen tuloksena esitellään tällaisen järjestelmän arkkitehtuuri. Arkkitehtuurin perusteella toteutettiin esimerkkisovellus, jota kokeiltiin menestyksekkäästi käytännössä.
Abstract: Generalization of the wireless local area networks is providing new opportunities for intruders to find their way in to data systems. At the same time importance of the data systems continues increasing from people and society point of view. There is a need for intrusion detection system for securing these data systems. While using of the data systems becomes a part of everyday life it also concerns more people. This makes easy usability of these systems even more important than before. In this work was studied possibilities to use self learning distributed neural networks to compose an intrusion detection system. As a result of this study an architecture of this kind of system is represented. Also a proof-of-concept application was implemented and successfully tested in practice.
Avainsanat: tietoturva, langattomat lähiverkot, neuroverkot, hajautetut järjestelmät
Keywords: data protection, wireless local area networks, neural networks, distributed systems

Sanasto

AP, Access Point, langattoman verkon tukiasema.
BSS, Basic Service Set, langattoman lähiverkon liityntäpiste.
BPTT, BackProbagation Through Time, rekursiivisien neuroverkkojen opetusmenetelmä.
CFP, Contention-Free Period, aikaväli jossa ei ole kilpailua.
CRC, Cyclic Redundancy Check, syklinen redundanssitarkistus.
DS, Distribution System, erillisiä WLAN-verkkoja yhdistävä verkko, esim. perinteinen lähiverkko.
FCS, Frame Check Sequence, kehyksen tarkistussumma GUID, Globally Unique Identifier, 32 merkkiä pitkä yksilöllinen merkkijono.
HC, Hybrid Coordinator
ICV, Integrity Check Value, kryptografinen sanoman tarkistussumma.
ID, Intrusion Detection, tunkeutumisen havaitseminen.
IDS, Intrusion Detection System, tunkeutumisen havaitsemisjärjestelmä.
IGR Information Gain Ratio
IV, Initialization Vector, WEP-salauksen alustusvektori.
IEEE, Institute for Electrical and Electronics Engineers, kansainvälinen tekniikan alan järjestö.
IBSS, Independent Basic Service Set, langattoman lähiverkon verkkotunnus.
MAC, Media Access Control, verkon varaamisen ja liikennöinnin hoitava osajärjestelmä.
MIB, Management Information Base
MMPDU,MAC Protocol Data Unit
MSDU, MAC Service Data Unit
MIC, Message Integrity Code, kryptografinen sanoman tarkistussumma.
NAV, Network Allocation Vector, langattomien verkkojen virtuaalinen kantoaallon havaitsemisjärjestelmä.
NBA, Network Behavior Analysis, verkkokäyttäytymisanalyysi.
QoS, Quality of Service, palvelun laatu
SOM, Self Orgnized Map, Teuvo Kohosen kehittämä valvomatonta oppimista soveltava neuroverkkotyyppi.
STA, Station, langattoman verkon laite.

TC, Traffic Category, liikenneluokka.
TS, Traffic Stream, liikennevirta.
RTRL, Real-Time Recurrent Learning, rekursiivisien neuroverkkojen opetusmenetelmä.
TKIP, Temporal Key Integrity Protocol, viimeisin langattomien 802.11-verkkojen tietoturvastandardi.
WEP, Wired Equivalent Privacy, ensimmäinen langattomien 802.11-verkkojen tietoturvastandardi.
XML, eXtensible Markup Language, rakenteellinen kuvauskieli.

Sisältö

1 Johdanto

Tunkeutumisen havaitseminen ja estämien tulee koko ajan tärkeämmäksi tietoverkkojen yleistyessä. Yksityisten henkilöiden, yritysten sekä yhteisöjen verkoissa säilytetään yhä enemmän tiedon haltijalle arvokasta tietoa. Tiedon haltija ei usein syystä tai toisesta halua tallentamansa tiedon päätyvän ulkopuolisiin käsiin.

Lisäksi tietoverkkojen ja tietojärjestelmien luotettavuus on koko ajan tärkeämpää. Ihmisten elämä ja koko yhteiskunta nojaa koko ajan enemmän tietojärjestelmien varmaan toimintaan.

Tietojärjestelmiin murtautuminen vaarantaa sekä järjestelmään tallennetun tiedon koskemattomuuden, että järjestelmän virheettömän toiminnan. Murtautuja saattaa varastaa tietoa ja käyttää sitä rikollisesti tai muuten tiedon alkuperäisen haltijan etujen vastaisesti. Murtautuja saattaa myös haluta vahingoittaa järjestelmää ja estää sen toiminnan.

Nopeasti yleistyvät langattomat verkot luovat murtautujalle perinteisiä langallisia verkkoja paljon helpomman reitin edellä mainittuihin tietojärjestelmiin.

Tästä syystä tietojärjestelmissä tarvitaan keino tunnistaa mahdolliset hyökkäykset ja tunkeutumisyritykset. Tätä tarkoitusta varten on olemassa ohjelmistoja ja järjestelmiä, muuta ne saattavat olla varsinkin tavalliselle käyttäjälle vaikeaselkoisia. Lisäksi nykyiset järjestelmät vaativat usein jatkuvaa ylläpitoa.

Tästä syystä haluan tutkia helppokäyttöisen ja tehokkaan langattomien verkkojen tunkeutumisen tunnistusjärjestelmän toteuttamismahdollisuuksia. Parhaassa mahdollisessa tilanteessa järjestelmä ylläpitää itse itseään. Keinoäly ja itseoppivat järjestelmät voivat tarjota ratkaisun ongelmaan.

Siispä perehdyn työssäni hajautettuihin keinotekoisiin neuroverkkoihin, ja niiden käyttöön hyökkäyksen tunnistusjärjestelmän luokittelijana. Työssäni keskitytään nimenomaan langattomien verkkojen tunkeutumisten havaitsemiseen.

1.1 Tutkimusongelma

Tutkimuksen päätavoitteena on selvittää miten hyvin neuroverkko soveltuu hyökkäyksien havaitsemiseen langattomassa mobiilissa ympäristössä. Ongelmaa voidaan tarkastella seuraavien pääkohtien kannalta:

- Miten saavuttaa paras väärien ja oikeiden tunnistusten suhde? Tämä on yleinen ongelma kaikissa ID-järjestelmissä.
- Miten suunnitella järjestelmä, jonka täytyy toimia ympäristössä, jossa laitteet saattavat erota toisistaan huomattavasti?
- Miten suunnitella järjestelmä, jonka täytyy toimia ympäristössä, jossa laitteiden resurssit voivat olla rajalliset?

Edellisten ongelmien ratkaisuksi esitetään neuroverkkojen ja ID-järjestelmän hajauttamista. Tämä puolestaan tuo uusia ongelmia. Siispä työn toissijaisena tavoitteena on selvittää miten voidaan hyväksi käyttää toisten neuroverkkojen oppimaa tietoutta uudelleen vaarantamatta samalla suojauksen tasoa. Tätä ongelmaa tarkastellaan seuraavista näkökulmista:

- Miten neuroverkot voisivat opettaa toisiaan?
- Miten voidaan huomioida käyttäjien yksityisyyden suoja?
- Miten varautua murtautujan yrityksiin ”saastuttaa” järjestelmä virheellisellä tiedolla?

Näiden uusien ongelmien ratkaiseminen toisi käytännössä ratkaisun ID-järjestelmiä yleisesti vaivaaviin ongelmiin, eli miten tehdään järjestelmästä mahdollisimman helppokäyttöinen ylläpitäjän ja käyttäjän kannalta. Tämä on tärkeää, koska perinteisessä tunnisteisiin perustuvissa järjestelmissä (tunnisteiden) ylläpitäjän kuorma saattaa olla suuri. Lisäksi käyttäjä ei välttämättä ole tietotekniikan ammattilainen, joten käyttäjän vaivaaminen hankalilla ilmoituksilla ei aina ole järkevää.

Käytännön osuutena tutkitaan, miten neuroverkkojen hajauttaminen langattomassa ympäristössä käytännössä toimii. Toisin sanoen tutkitaan, miten resurssien jakaminen laitteiden kesken ja kommunikaatio järjestelmän eri osien välillä toimii.

1.2 Aiemmat tutkimukset

Aiheen eri osa-alueita on tutkittu maailmalla kattavasti eri yhteyksissä, mutta kaikki osa-alueet yhteen vetäviä tutkimuksia on vaikea löytää.

Yongguang Zhang, Wenke Lee ja Yi-An Huang käsittelevät langattomien verkkojen ID-järjestelmiä melko kattavasti tutkimuksessaan, mutta he käyttävät järjestelmässään luokittelijoina päätöspuuta vastaavaa RIPPER:ä sekä tukivektorikone SVM

Light:ia [1, sivu 8]. Myös Hongyu Yang, Lixa Xie ja Jizhou Sun esittelevät agentteihin perustuvan WLAN-verkkojen ID-järjestelmän tutkimuksessaan [2]. Tässä tutkimuksessa käytetty päätöksentekomenetelmä ei tosin käy ilmi julkaisusta.

Konsepteja hajautettujen, autonomisista oppivista agenteista koostuvien järjestelmien soveltamisesta ID-järjestelmiin on esitetty useita tutkimuksia ja toteutusmalleja Zhanfei Ma et al.[3], Jia-Jun Xiong et al.[4] ja Jin-Gang Cao et al.[5] toimesta. Olemassa olevat mallit on yleensä tosin kehitetty tavanomaisia, kiinteitä verkkoja silmällä pitäen.

Mouhcine Guennoun ja Khalil El-Khatib tarkastelevat kattavassa katsauksessaan langattoman verkon tunkeutumisentunnistusjärjestelmiä yleisesti ja esittävät keskitettyyn ratkaisuun perustuvaa järjestelmää [6].

Keinotekoisten neuroverkkojen tutkijoilla on luonnollisten neuroverkkojen tutkijoihin verrattuna se etu, että he voivat ottaa verkkojensa pohjalle millaisia ideoita tahansa kunhan ne edistävät verkolle asetetun tehtävän suorittamista. Tämä tietenkin siinä tapauksessa, että neuroverkon ei ole tarkoitus mallintaa luonnollisen neuroverkon toimintaa pikkutarkasti. Toisaalta mikään ei estä ottamasta mallia biologisista järjestelmistä. Siksipä tuntuu luonnolliselta ajatella hajautettua keinotekoista neuroverkkojärjestelmää neuroverkkojen yhteisönä ja ottaa mallia biologisista vastineistaan. Eläinten kulttuurisesta oppimisesta, oppimisen geneettisestä kehittymisestä sekä ohjatusta opettamisesta on olemassa hyödynnettäviä tutkimuksia. Esimerkiksi Telkänrannan eläinten oppimista käsittelevästä artikkelista [7] voi lainata ideoita oppivan hajautetun järjestelmän suunnitteluun.

1.3 Tutkielman rakenne

Tutkielma jakautuu kuuteen osioon, jotka seuraavat loogisesti toinen toisiaan. Luvussa 2 tarkastellaan langattomia lähiverkkoja toimintaympäristönä, langattoman verkon protokollaa sekä langattomaan verkkoon liittyviä haavoittuvuuksia. Luvussa 3 tarkastellaan tunkeutumisen tunnistusjärjestelmiä, niiden perusteita, tyyppejä, tekniikoita sekä haasteita. Luvussa 4 tarkastellaan keinotekoisten neuroverkkomallien käyttöä tunkeutumisen havaitsemiseen, niiden matemaattisia perusteita, opettamista ja alustamista. Luvussa tarkastellaan myös piiteiden valintaa, keinotekoisten neuroverkkojen haasteita sekä keinotekoisten neuroverkkojen hajauttamista. Luvussa 5 esitellään suunniteltu tunkeutumisen havaitsemisjärjestelmä ja sen arkkitehtuuri. Luvussa 6 käydään läpi esimerkkisovelluksen toteutusta, koejärjestely sekä kokeilun tuloksia. Työn lopussa luvussa 7 esitetään yhteenveto työn tuloksista sekä joitakin työhön liittyviä jatkokehitysideoita.

2 Langattomat lähiverkot

Tässä työssä toimintaympäristön muodostavat langattomat lähiverkot (engl. *wireless local area network*, WLAN). Nämä ovat tietokoneiden välisen langattoman kommunikaation mahdollistavia suhteellisen lyhyen kantaman tietoverkkoja. WLAN-verkot perustuvat IEEE 802.11 standardiperheeseen [8]. WLAN lyhenteen tilalla käytetään myös usein myös merkintää Wi-Fi, joka on Wi-Fi Alliancen -tavaramerkki ja tarkoittaa Wi-Fi Alliancen sertifioimia IEEE 802.11-standardin mukaisia laitteita [9].

2.1 IEEE 802.11 -protokolla

Alkuperäinen IEEE 802.11 -standardi julkaistiin vuonna 1999 ja sitä laajennettiin seuraavilla laajennoksilla [8, sivu 7]:

- IEEE 802.11a™-1999 (Laajennos 1)
- IEEE 802.11b™-1999 (Laajennos 2)
- IEEE 802.11b-1999/Korjaus 1-2001
- IEEE 802.11d™-2001 (Laajennos 3)
- IEEE 802.11g™-2003 (Laajennos 4)
- IEEE 802.11h™-2003 (Laajennos 5)
- IEEE 802.11i™-2004 (Laajennos 6)
- IEEE 802.11j™-2004 (Laajennos 7)
- IEEE 802.11e™-2005 (Laajennos 8)

Tällä hetkellä voimassaoleva versio standardista julkaistiin vuonna 2007 ja siihen on tähän mennessä julkaistu seuraavat laajennokset [10]:

- IEEE 802.11k™-2008 (Laajennos 1)
- IEEE 802.11r™-2008 (Laajennos 2)

- IEEE 802.11y™-2008 (Laajennos 3)
- IEEE 802.11w™-2009 (Laajennos 4)
- IEEE 802.11n™-2009 (Laajennos 5)
- IEEE 802.11p™-2010 (Laajennos 6)

Standardit määrittelevät fyysisen ja siirtoyhteyskerroksen toiminnan. Verkon toteutustekniikkana voidaan käyttää infrapunaa tai radioaaltoja. Radioaaltoihin perustuvat verkot toimivat 2,4:n ja 5:n gigahertsin taajuusalueilla [8].

Taajuusalueet ja nopeudet standardeittain löytyvät taulukosta 2.1.

Standardi	Taajuusalue	Maksimi siirtonopeus
802.11a	5 GHz	54 Mbps
802.11b	2,4 GHz	11 Mbps
802.11g	2,4 GHz	54 Mbps
802.11n	2,4 GHz, 5 GHz, 2,4 tai 5 GHz (valittavissa) tai 2.4 ja 5 GHz (rinnakkain)	450 Mbps

Taulukko 2.1: Wi-Fi -standardisukupolvet [9].

WLAN-verkot voivat olla joko avoimia tai suljettuja. Avoimet verkot ovat joko tarkoituksella tai tahattomasti salaamattomaksi jätettyjä verkkoja, joihin kuka tahansa voi liittyä lähes ilman rajoituksia. Pääsyä salaamattomaankin verkkoon voidaan rajoittaa MAC-osoitteen perusteella siten, että vain sallitun MAC-osoitteen (engl. *media access control*) omaavat laitteet voivat liittyä verkkoon. Tämän kaltainen pääsynvalvonta ei tosin käytännössä juuri eroa valvomattomasta ja tunkeutuja voi kiertää eston helposti, kuten Low esittää [11, sivu 11]. Jos verkkoa ei ole tarkoitettu julkiseen käyttöön, salataan se tyypillisesti jollakin kappaleessa 2.1.2 esitellyllä menetelmällä.

2.1.1 Kehysrakenne

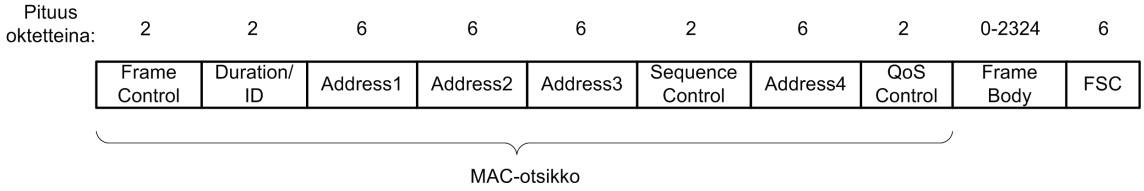
Langattomissa verkoissa toimivat tunkeutumisen havaitsemisjärjestelmät (kappaleessa 3.2) keskittyvät havaitsemaan langattoman verkon protokollaan liittyviä hyökkäyksiä ja väärinkäytöksiä. Eräs keino pyrkiä havaitsemaan edellä mainitut uhat on tutkia MAC-kerroksen tarjoamia signaaleja mahdollisista hyökkäyksistä. Tässä

tutkielmassa keskityn tutkimaan nimenomaan langattoman lähiverkon tunkeutumisenhavaitsemisjärjestelmiä ja tästä syystä lienee paikallaan lyhyesti tutkia IEEE 802.11 standardin kehysrakennetta.

Verkon jokainen kehys koostuu seuraavista peruskomponenteista [8, sivu 59]:

- MAC-otsikosta, joka koostuu kehyksen ohjaus (engl. *frame control*), elinaika (engl. *duration*), osoite (engl. *address*) ja sekvenssin ohjaus -tiedoista (engl. *sequence contrtol*), sekä QoS-kehysten (engl. *Quality of Service*) tapauksessa QoS-tiedosta,
- muuttuvan mittaisesta kehyksen rungosta (engl. *frame body*), joka sisältää kehyksen tyypin ja alityypin mukaista tietoa, sekä
- kehyksen tarkistussummasta (engl. *frame check sequence*, FCS), joka sisältää 32-bittisen CRC:n (engl. *cyclic redundancy check*).

Kuvassa 2.1 esitellään IEEE 802.11 standardin mukainen yleinen MAC-kehys. Jokainen verkossa lähetetty kehys sisältää vähintään ensimmäiset kolme (**Frame Control**, **Duration/ID** ja **Address 1**) ja viimeisen (**FCS**) kenttän. Muut kentät (**Address 2**, **Address 3**, **Sequence Control**, **Address 4**, **QoS Control** ja **Frame Body**) ovat mukana ainoastaan jos kehyksen tyyppi tai alityyppi niin vaatii [8, sivu 60].



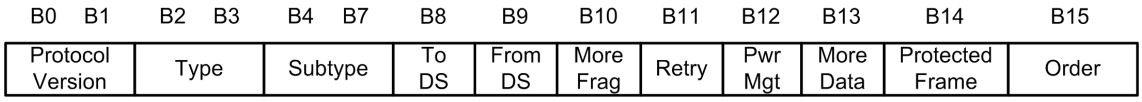
Kuva 2.1: MAC-kehyksen rakenne [8, sivu 60].

Frame Control -kenttä, eli kehyksen ohjauskenttä, koostuu seuraavista alikentistä:

- **Protocol Version** kertoo protokollan version,
- **Type** määrittelee WLAN kehyksen tyypin. Mahdollisia tyyppejä ovet: ohjaus (engl. *control*), data ja ylläpito (engl. *management*),
- **Subtype** määrittelee yhdessä **Type**-kentän kanssa kehyksen tarkan tyypin. Jokaisella kolmesta tyypistä on useita alityyppejä,

- **To DS** ja **From DS** kertovat yhdessä, onko datakehys menossa DS:ään, vai onko se tulossa DS:stä. Ohjaus ja ylläpito -tyyppisissä kehyksissä näitä bittejä ei aseteta,
- **More Fragments** kertoo sisältääkö data- tai ylläpitotyyppinen kehys osan ylemmän kerroksen pakettista ja jatkuuko paketin lähetys seuraavassa kehyksessä,
- **Retry** ilmaisee onko data tai ylläpito -tyyppinen kehys uudelleenlähetetty. Käytettään dublikaattikehysten eliminointiin,
- **Power Management** kertoo kehyksen lähettäneen STA:n siirtyvän tehonhallintatilaan (engl. *power management state*),
- **More Data** käytetään esimerkiksi silloin kun AP haluaa kertoa virransäätötilassa olevalle STA:lle että AP:lla on sille lisää puskuroituja kehyksiä lähetettävänä,
- **Protected Frame** ilmaisee frame body -kentässä lähetetyn datan olevan salattua. Kenttä oli aiemmin nimeltään WEP ja
- **Order**, kertoo ei-QoS data-kehyksen lähetetyn käyttäen StrictlyOrdered -palveluluokkaa.

Frame Control -kentän rakenne on kuvattuna kuvassa 2.2.



Kuva 2.2: Frame Control -kentän rakenne [8, sivu 60].

Duration/ID-kenttä, eli elinaika/tunniste -kentän sisältö riippuu kehyksen tyypistä ja alityypistä ja siitä, onko kehys välitetty CFP:n aikana, eli aikavälillä, jossa ei ole kilpailua (CFP, engl. *contention-free period*). Myös päätelaitteen QoS-ominaisuudet vaikuttavat kentän sisältöön [8, sivu 64].

Address-kenttiä, eli osoitekenttiä käytetään ilmoittamaan BSSID-verkkotunnus (engl. *basic service set identification*), lähdeosoite (engl. *source address, SA*), kohdeosoite (engl. *destination address, DA*), lähettävän päätelaitteen osoite (engl. *transmitting STA address, TA*) ja vastaanottavan päätelaitteen osoite (engl. *receiving STA address, RA*) [8, sivu 65].

Jotkut kehystyypit eivät sisällä kaikkia osoitekenttiä. Osoitekentän käyttö määritellään kentän (1-4) paikan perusteella suhteessa MAC-kehykseen. Tämä tapahtuu riippumatta osoitteen tyypistä, jonka kenttä sisältää. Esimerkiksi vastaanottajan osoitetta verrataan vastaanotetun kehyksen osoitekenttään 1. Toisena esimerkkinä CTS- ja ACK-kehysten vastaanottajan osoite otetaan vastaavan RST- tai kuitattavan kehyksen osoitekentästä 2 [8, sivu 65].

Sequence Control -kenttä on 16 bittiä leveä ja koostuu kahdesta alikentästä: sekvenssinumero (engl. *Sequence Number*) ja pirstalenumero (engl. *Fragment Number*). Kontrollikehykset eivät sisällä Sequence Control -kenttää [8, sivu 66].

Sequence Number -kenttä on 12 bittiä leveä ja sisältää MSDU:n (engl. *MAC service data unit*) tai MMPDU:n (engl. *MAC protocol data unit*) sekvenssinumeron [8, sivu 66].

Fragment Number -kenttä on 4 bittiä leveä ja sisältää MSDU:n tai MMPDU:n pirstaleen numeron. Ensimmäisen tai ainoan pirstaleen numero on nolla ja sitä kasvatetaan jokaisella saman MSDU:n tai MMPDU:n onnistuneella pirstaleen siirrolla. Pirstaleen uudelleenlähetystilanteessa pirstalenumero pysyy muuttumattomana [8, sivu 67].

QoS Control -kenttä on 16 bittiä leveä, ja kertoo TC:n (engl. *traffic category*) tai TS:n (engl. *traffic stream*), johon kehys kuuluu. Kenttä kertoo myös paljon muuta kehyksen tyypin mukaan vaihtuvaa QoS-tietoa. QoS Control -kenttä koostuu viidestä alikentästä, jotka määräytyvät kehyksen tyypin, alityypin ja lähettäjän (HC (engl. *hybrid coordinator*) tai ei-tukiasema) mukaan. Alikenttien käytön voi tarkastaa lähteestä [8, taulukko 7-4]. QoS-kenttä liitetään kaikkiin data-kehyksiin, joiden Subtype-kentän QoS-bitti (bitti 7) on asetettu [8, sivu 67].

Frame Body -kenttä on muuttuvanmittainen ja sisältää kehyksen tyypin ja alityypin mukaan vaihtuvaa tietoa. Data-kehysten hyötykuorma välitetään tässä kentässä [8, sivu 70].

FCS-kenttä on 32 bittiä leveä ja sisältää 32-bittisen CRC:n. FCS lasketaan yli koko MAC-otsikon ja Frame Body -kentän [8, sivu 70].

Kappaleessa 4.5.1 käsitellään lähemmin MAC-kehyksen osien käyttöä tunkeutumisen havaitsemisessa.

2.1.2 Salaus

Langattomissa verkoissa käytetään useita salaus- ja autentikointimenetelmiä. Salauksen tarkoituksena on kätkeä verkossa välitetty tieto ulkopuolisilta autentikoimattomilta laitteilta. Autentikointi taas takaa pääsyn verkkoon ainoastaan sallituil-

le laitteille. Seuraavassa esitellään lyhyesti IEEE 802.11 standardin mukaisissa verkoissa käytettyjä menetelmiä.

WEP (engl. *Wired Equivalent Privacy*) on alkuperäinen vuoden 1999 IEEE 802.11 -standardissa [12] esitelty salausmenetelmä. Ciambriellon ja Ordinen mukaan WEP-salausta ei pitäisi käyttää sen helpon murrettavuutensa takia [13].

WPA (engl. *Wi-Fi Protected Access*) kehitettiin korvaamaan WEP siinä havaittujen heikkouksien vuoksi. WPA on kompromissi, jossa uudelleen käytetään WEP:n RC4-salausalgoritmia ja tällä tavoin se toimii useimmilla vanhemmille standardeille suunnitelluilla laitteilla.

WPA2 (engl. *Wi-Fi Protected Access*) on WPA:n toinen versio ja lisätty standardiin versiossa 802.11i. Se on taaksepäin yhteensopiva WPA:n kanssa.

Taulukossa 2.2 yhteenveto Wi-Fi salaus ja autentikointimenetelmistä.

Standardi	**Autentikointi**	**Salaus**	**Avaimen pituus**
WEP	Open System/Shared Key	RC4	40/104
WAP	PSK/802.1X/EAP	RC4/TKIP/MIC	128
WAP2	PSK/802.1X/EAP	AES-CCMP	128

Taulukko 2.2: Wi-Fi -salausmenetelmät.

Vaikka uusien salausmenetelmien kehittäminen parantaa verkkojen turvallisuutta ei tunkeutumisen havaitseminen ole jatkossakaan turhaa. Verkkojen turvaaminen ja murtaminen on jatkuvaa kilpajuoksua, jossa verkon ylläpitäjän kannattaa olla edellä.

2.2 Verkkomallit

Langaton lähiverkko voidaan toteuttaa kahden erilaisen mallin mukaisesti: infrastruktuuri ja Ad hoc [8, sivu 1169].

Infrastruktuurityyppinen verkko sisältää yhden tai useamman tukiaseman ja tätä kautta mahdollisen yhteyden kiinteään verkkoon.

Ad hoc -verkossa taas ei ole laisinkaan tukiasemia, vaan verkon laitteet kommunikoivat suoraan keskenään.

2.3 Haavoittuvuudet

Teknologiansa vuoksi langaton verkko on alttiimpi hyökkäyksille kuin perinteinen langallinen verkko. Hyökkäystyypit ovat osittain samat molemmissa verkoissa, mutta langallisen verkon uhkien lisäksi langattomissa verkoissa täytyy varautua täysin uudenlaisiin uhkiin.

Vaikka langattomia ja langallisia verkkoja uhkaavat osittain saman tyyppiset hyökkäykset saattaa niiden suhteellinen mahdollisuus kuitenkin vaihdella huomattavasti. Esimerkkinä langattomat hyökkäykset tyypillisesti vaativat, että hyökkääjä tai hyökkääjän laite sijaitsee langattoman verkon välittömässä läheisyydessä. Langallisen verkon hyökkäykset taas voidaan suorittaa etänä mistä tahansa sijainnista. Toisaalta monet langattomat verkot on konfiguroitu siten etteivät ne vaadi minkäänlaista tunnistautumista tai tunnistautuminen on tasoltaan heikko. Tästä johtuen monet hyökkäykset, kuten kolmas mies -hyökkäys (engl. *man-in-the-middle*) hyökkäys, on helpompi toteuttaa langattomassa kuin langallisessa verkossa (katso Scarfone ja Mell [14, sivu 53]).

Suurimpaan osaan langattoman verkon uhista liittyy hyökkääjä, jolla on pääsy langattoman päätelaitteen ja tukiaseman (tai kahden päätelaiteen ad hoc -verkossa) väliseen radioyhteyteen. Monet hyökkäykset perustuvat hyökkääjän mahdollisuuteen kaapata verkkoliikennettä tai lisätä viestejä siihen [14, sivu 53].

Langattomien verkkojen turvaamisen kannalta suurin ero verrattuna langallisiin verkkoihin on suhteellinen helppous päästä käsiksi ja korvata verkkoliikennettä. Langallisessa verkossa hyökkääjän täytyy muodostaa fyysinen yhteys verkkoon tai avata pääsy verkkoon etänä. Langattomassa verkossa riittää pääsy verkon kantaman sisäpuolelle [14, sivu 53].

Langattoman verkon haavoittuvuuksia ja hyökkäyksiä on luokiteltu useissa lähteissä (katso CSEC [15], Antoniewicz [16], Hussain [17] ja Jungwoo Ryoo et al. [18]). Hyökkäysten luokitteluun ei ole mitään standardia menetelmää, eikä mikään staattinen lista pystyisikään kuvaamaan kaikkia hyökkäyksiä nyt ja tulevaisuudessa: hyökkääjät löytävät ja oppivat koko ajan uusia keinoja hyödyntää protokollan, salauksen sekä ohjelmistojen haavoittuvuuksia. Toinen luokitteluun liittyvä ongelma on hyökkäysten nimeäminen: jokainen voi nimittää hyökkäyksiä miten haluaa. Tästä johtuen täytyy määritellä millaista toimenpidettä tarkoitetaan puhuttaessa jostakin tietystä hyökkäyksestä.

Tässä työssä keskitytään kolmeen erilaiseen hyökkäystyyppiin:

- De-authentication-hyökkäys,

- ChopChop-hyökkäys,
- Fragmentation-hyökkäys ja
- Duration-hyökkäys.

Yllä listatut hyökkäystyypit valikoituivat kappaleessa 4.5.2 valittujen piirteiden kautta. Syy juuri näiden neljän hyökkäystyypin valintaan on selvä: käyttäessäni Guennoun et al. tutkimuksessaan [19] valitsemia, juuri näiden hyökkäystyyppien löytämiseen optimoituja piirteitä, on myös tässä tutkimuksessa turvallisinta pitäytyä samoissa hyökkäyksissä. Joudutaan siis tekemään kompromissi tunnistettavien hyökkäysten määrän ja piirteiden valintaan käytetyn työajan välillä. Mutta koska tässä työssä painopiste on enemmän neuroverkkojen hajauttamisen tutkimuksessa kuin optimaalisten piirteiden tutkimuksessa on tällainen kompromissi mielestäni hyväksyttävissä.

De-authentication -hyökkäyksessä hyökkääjä väärentää autentikaation purku -paketin (engl. *de-authentication frame*) siten kuin se olisi lähetetty tukiasemasta. Vastaanottaessaan kyseisen paketin laite katkaisee yhteytensä ja yrittää liittyä uudelle tukiasemaan. Tätä toimenpidettä jatkamalla voidaan estää laitetta liittymästä takaisin tukiasemaan. Halutessaan hyökkääjä voi kohdistaa hyökkäyksensä kaikkiin tukiasemaan liittyneisiin laitteisiin käyttämällä vastaanottajan osoitteena yleislähetysosoitetta (engl. *broadcast address*). On kuitenkin huomattu, että jotkin verkkosovittimet hylkäävät tämän tyyppiset autentikaation purku -kehykset [19].

ChopChop-hyökkäyksessä hyökkääjä kaappaa salatun paketin ja käyttää tukiasemaa arvatakseen sen salaamattoman sisällön. Hyökkäys suoritetaan siten, että kaapatusta paketista poistetaan viimeinen tavu. Tästä paketista hyökkääjä muodostaa uuden, yhtä tavua lyhemmän paketin. Muodostaakseen uudelle paketille käyvän ICV:n (engl. *Integrity Check Value*) hyökkääjä yrittää arvata poistetun tavun salaamattoman sisällön. Tarkistaakseen arvauksensa oikeellisuuden hyökkääjä lähettää luomansa paketin tukiasemalle käyttäen vastaanottajana monilähetysosoitetta (engl. *multicast address*). Jos kehyksen ICV ei ollut käypä, eli arvaus ei osunut oikeaan, hylkää tukiasema paketin ilmoittamatta. Jos taas arvaus osui oikeaan lähettää tukiasema paketin takaisin verkkoon. Tällä tavoin hyökkääjä voi tarkastaa tekemänsä arvauksen. Toimenpidettä jatketaan kunnes paketin jokaisen tavun salaamaton sisältö on selvitetty [19].

Fragmentation-hyökkäyksessä hyökkääjä lähettää paketin palasina (engl. *fragment*). Tukiasema koostaa näistä palasista uuden paketin ja lähettää sen takaisin langattomaan verkkoon. Koska hyökkääjä tietää paketin salaamattoman sisällön voi

hän selvittää paketin salaamiseen käytetyn avainjonon (engl. *key stream*). Toimenpidettä toistetaan kunnes hyökkääjällä on 1500 avainta pitkä jono. Hyökkääjä voi tätä avainjonoa käyttäen salata tai avata samaa IV:tä (engl. *Initialization Vector*) käyttäviä paketteja. Toimenpidettä voidaan periaatteessa jatkaa niin kauan, että hyökkääjällä on hallussaan avainjonotaulukko kaikille mahdollisille IV:lle. Tällainen taulukko vaatii 23 gigatavua muistia [19].

Duration-hyökkäyksessä hyökkääjä lähettää paketteja, joiden duration-kenttä on asetettu maksimiarvoonsa (32767ms) ja bitti 15 on asettamatta. Tämä asettaa laitteiden NAVin (engl. *network allocation vector*) kyseiseen arvoon ja estää laitetta käyttämästä jaettua mediaa ennen kuin NAV-ajastin on saavuttanut arvon nolla. Hyökkääjä toistaa toimenpiteen ennen laskurin nollautumista. Toistamalla toimenpidettä hyökkääjä voi estää laitetta pääsemästä langattomaan verkkoon [19].

Vaikka tarkoituksena oli kokeilla valmista järjestelmää kaikilla neljällä hyökkäystyypillä, kävi työn edetessä selväksi, että ainoastaan fragmentation ja De-authentication -hyökkäykset ovat testattavissa mielekkäällä työmäärällä. Lisää hyökkäyksistä ja niiden toteuttamisesta kappaleessa 6.2.3.

3 Tunkeutumisen havaitsemisjärjärjestelmät

Tässä luvussa tarkastellaan tunkeutumisen tunnistusjärjestelmiä, niiden perusteita, tyyppejä, tekniikoita sekä haasteita.

3.1 Tunkeutumisen havaitsemisjärjärjestelmien perusteet

Scarfonen ja Mellin mukaan tunkeutumisen havaitsemisella tarkoitetaan prosessia, jossa tarkkaillaan tietokonejärjestelmän tai -verkon tapahtumia ja analysoidaan niitä. Tapahtumista etsitään merkkejä mahdollisista epäilyttävistä toimista, jotka voivat loukata tai viitata mahdolliseen uhkaan turvallisuuspolitiikkoja, hyväksyttäviä käyttöpolitiikkoja tai standardeja käytäntöjä kohtaan. Tunkeutumisen tunnistus- ja estojärjestelmät on suunnattu pääasiassa epäilyttävien tapahtumien havaitsemiseen, niihin liittyvän tiedon tallentamiseen sekä niiden lopettamiseen ja niistä järjestelmän ylläpitäjälle ilmoittamiseen [14, sivu 21].

Tyypillisesti IDS:t tallentavat havaittuihin tapahtumiin liittyvän informaation, hälyttävät tietoturvavastaavan ja luovat raportin tapahtuneesta. Monet IDS:t voivat myös vastata havaittuun uhkaan yrittämällä torjua se. Järjestelmät käyttävät useita tekniikoita, jotka käsittävät itse hyökkäyksen pysäyttämisen, turvallisuusympäristön muuttamisen (esimerkiksi palomuurin asetusten muuttaminen) tai hyökkäyksen sisällön muuttamisen [14, sivu 21].

3.2 IDS-tyypit

IDS-tekniikoita on olemassa useita erilaisia tyyppejä, jotka eroavat toisistaan pääasiassa havaitsemiensa tapahtumien ja epäilyttävien tapahtumien havaitsemiseen käyttämiensä metodien osalta. Eräitä IDS-tyyppejä ovat esimerkiksi seuraavat [14, sivu 21]:

- **Verkkoperusteinen**. Valvoo verkkoliikennettä tiettyihin verkkosegmentteihin tai -laitteisiin ja analysoi liikennettä verkko- tai sovellusprotokollatasolla tunnistaakseen epäilyttävät toimet.
- **Langaton**. Valvoo langattoman verkon liikennettä ja analysoi sitä tunnistaakseen langattoman verkon protokolliin itseensä liittyvät epäilyttävät toimet.

- **Verkon käyttäytymisen analyysi** (engl. *Network Behavior Analysis, NBA*). Tutkii verkkoliikennettä tunnistaakseen uhat, jotka tuottavat poikkeavia liikennemääriä, kuten DDoS-hyökkäys (engl. *Distributed Denial of Service*), verkon skannaus tai tietyntyyppiset haittaohjelmat.

- **Isäntäperusteinen**. Valvoo yhden isäntälaitteen tunnusmerkkejä ja tapahtumia tunnistaakseen epäilyttävät toimet.

Kuten johdannossa mainittiin, keskitytään tässä työssä lähinnä langattomaan tyyppiin, mutta käytännössä IDS voi sisältää piirteitä kaikista edellä mainituista tyypeistä. Langattomissa verkoissa voisi kysymykseen tulla esimerkiksi kombinaatio, jossa käytettäisiin sekä langattoman verkon että isäntäperusteista mallia. Tällä tavoin voitaisiin ikään kuin muodostaa kaksi sisäkkäistä suojausvyöhykettä, joista ensimmäinen tunnistaisi tunkeutumisen verkkoon ja jälkimmäinen tunkeutumisen itse laitteeseen.

3.3 Tunnistusmenetelmät

Monet IDS:t käyttävät useita tunnistusmenetelmiä, joko yhdessä tai erikseen, tarkoituksenaan tarjota kattavampi ja tarkempi havaitseminen. Seuraavassa Scarfonen ja Mellin mukaan tärkeimmät tunnistusmenetelmien luokat [14, sivu 21]:

- **Tunnusmerkkiperusteinen**. Tunnistaa haitalliset tapahtumat vertaamalla tapahtumia tunnettujen uhkien tunnusmerkkeihin. Tämä menetelmä on hyvin tehokas tunnistettaessa tunnettuja uhkia, mutta hyvin tehoton tunnistettaessa uhkia, joita ei vielä tunneta tai jotka ovat vanhojen tunnettujen muunnoksia. Tunnusmerkkiperusteinen tunnistaminen ei pysty seuraamaan ja ymmärtämään monimutkaisen viestinnän tilaa, joten se ei yleensä pysty havaitsemaan hyökkäyksiä, jotka koostuvat useista tapahtumista.

- **Poikkeamaperusteinen**. Tunnistaa huomattavasti poikkeavat tapahtumat vertaamalla havaittuja tapahtumia normaaliksi määriteltyyn toimintaan. Tämä menetelmä käyttää profiileja, jotka on luotu seuraamalla tyypillisen toiminnan tunnusmerkkejä tietyltä ajalta. IDS vertaa toimintoja profiilin määrittelemiin kynnysarvoihin. Poikkeamaperusteinen menetelmä on erittäin tehokas, kun pyritään havaitsemaan ennestään tuntemattomia uhkia. Yleisimmät poikkeamaperusteiseen tunnistamiseen liittyvät ongelmat ovat tahaton haitallisen toiminnan sisällyttäminen profiiliin, todellisen maailman toimintaa kuvaamatto-

mien, liian yksinkertaisten profiilien luominen sekä suuri virheellisten positiivisten tunnistusten määrä.

- **Tilaperusteinen protokolla-analyysi**. Tunnistaa poikkeamat vertaamalla jokaista protokollan tilaa ennalta määriteltyihin yleisesti hyväksyttyihin profiileihin, jotka kuvaavat protokollien ei-vihamielistä käyttöä. Toisin kuin poikkeamaperusteinen tunnistaminen, joka käyttää isäntä- tai verkkoriippuvaisia profiileja, tilaperusteinen protokolla-analyysi nojaa valmistajakohtaisiin yleisiin profiileihin. Nämä profiilit määrittelevät, miten tiettyä protokollaa pitää ja ei pidä käyttää. Menetelmä pystyy käsittelemään ja seuraamaan tila-käsitteen omaavien protokollien tilaa. Tämä mahdollistaa monien sellaisten hyökkäysten havaitsemisen, joita muut menetelmät eivät pysty havaitsemaan. Tilaperusteisen protokolla-analyysin ongelmia ovat täysin kattavan ja tarkan protokolla-mallin kehittämisen mahdottomuus, suuri resurssivaativuus sekä se, ettei malli pysty tunnistamaan hyökkäystä, joka ei riko protokollan normaalin käytön tunnusmerkkejä.

Koska tässä työssä sovelletaan neuroverkkoja tunnistavana elementtinä, saattavat kaikki edellä luetellut menetelmät tulla kyseeseen. Kuitenkin käytettävyyden kannalta erityisen käyttökelpoiselta vaikuttaa poikkeamaperusteinen menetelmä.

3.4 IDS:n haasteet

IDS:t eivät voi tarjota täysin tarkkaa tunnistusta vaan ne kaikki antavat vääriä negatiivisia ja positiivisia tunnistuksia, eli eivät tunnista haitallista toimintaa tai tulkitsevat normaalin toiminnan haitalliseksi. Monet organisaatiot säätävät IDS:t siten, että ne tuottavat pikemminkin vääriä positiivisia kuin vääriä negatiivisia tunnistuksia. Tämä vaatii Scarfonen ja Mellin mukaan ylimääräisiä resursseja väärien positiivisten erottamiseksi todellisista haitallisista tapahtumista [14, sivu 22].

Eräs toinen IDS:n liittyvä huomioitava seikka on yksityisyyden suoja: IDS ei saisi paljastaa ylimääräistä henkilökohtaista tietoa järjestelmän käyttäjistä.

Edelliset seikat koskevat kaikkia IDS:iä, ja vaikka kiinteille verkoille on kehitetty suuri määrä erilaisia tunkeutumisen tunnistus- ja estojärjestelmiä, ei niitä voida suoraan soveltaa langattomiin järjestelmiin. Tämä johtuu kiinteiden ja langattomien verkkojen perustavaa laatua olevista eroista. Seuraavaksi selvitetään, millaisia erityishaasteita langattomat verkot tuovat IDS:n näkökulmasta katsottuna.

3.4.1 Avoin infrastruktuuri

Kuten luvussa 2.3 mainittiin, on suurin ero kiinteiden ja langattomien verkkojen välillä se, ettei langattomalla verkolla ole kiinteää infrastruktuuria. Tästä johtuen kiinteiden verkkojen reaaliaikaiseen liikenteen analysointiin nojaavat verkkoperusteiset IDS:t eivät sellaisenaan toimi kovin hyvin langattomassa ympäristössä. Kiinteässä verkossa liikenteen valvonta suoritetaan yleensä kytkimissä, reitittimissä tai yhdyskäytävissä. Langattomassa verkossa tällaista liikenteen solmukohtaa, jossa IDS voisi valvottavaa aineistoa koko verkon osalta kerätä, ei ole. Tästä johtuen valvonta rajoittuu liikenteeseen, joka tapahtuu radion kantaman sisäpuolella, kuten Yongguang Zhang et al. huomauttaa [1, sivu 547].

Langattoman verkon avoin infrastruktuuri tuo mukanaan sen, ettei verkkoon liitettyihin laitteisiin voi välttämättä luottaa. Kiinteässä verkossa vihamielisten laitteiden liittyminen verkkoon voidaan yleensä estää fyysisesti, esimerkiksi tietoverkon sisältävän rakennuksen kulunvalvonnalla. Langattoman verkon kanssa näin ei ole, vaan langattoman verkon laitteiden tulee varautua siihen, että mikä tahansa muu verkkoon liittyvä laite saattaa olla vihamielinen. Tämän lisäksi, jos langaton verkko kytketään osaksi kiinteää verkkoa, täytyy mahdollisesti suljettu kiinteä verkon osa suojella potentiaalisesti vihamielisiltä, langattomaan verkon osaan liittyviltä laitteilta.

3.4.2 Monimuotoisuus

Langattomassa verkossa laitteiden kirjo saattaa myös aiheuttaa omia haasteitaan verrattuna kiinteään verkkoon. Tämä korostuu erityisesti tarkasteltaessa hajautettua langatonta IDS:ää, koska jokainen verkon laite osallistuu jollakin tasolla järjestelmän toimintaan. Tästä syystä järjestelmän tulee tukea useita, hyvinkin eritasoisia laitealustoja.

Langattoman verkon laitteissa ei yleensä ole käytettävissä yhtä paljon resursseja kuin kiinteän verkon laitteissa. Tämä asettaa järjestelmälle tiettyjä lisävaatimuksia. Resursseja ovat muun muassa

- keskusmuisti,
- massamuistitila,
- laskentakapasiteetti,
- tietoliikennenopeus,

- käyttöenergia (akku) ja
- kyky vastaanottaa liikennettä.

Kyky vastaanottaa liikennettä tarkoittaa tässä yhteydessä laitteen kykyä kuunnella liikennettä esimerkiksi useammalta kanavalta yhtä aikaa, kykyä kuunnella kaikkea kanavan liikennettä tai kykyä välittää MAC-kerroksen otsikkotietoja ylemmälle tasolle. Suurin osa laitteista ja käyttöjärjestelmistä ei edellä mainittuihin toimintoihin pysty. Tästä syystä langattomien verkkojen ISD:ssä suositaan erillisiä laitteita, jotka kuuntelevat liikennettä. Nämä niin sanotut anturit(engl. *sensor*) voivat olla myös osa jotakin toista langattoman verkon laitetta.

3.4.3 Kommunikaatiomalli

Toinen huomattava ero on kommunikaatiomalli: langattomissa verkoissa laitteet ja käyttäjät yrittävät kommunikoida mahdollisimman vähän ja siksi omaksuvat uusia toimintatiloja, kuten ei-kytkeytyneen toimintamallin. Tästä johtuen langallisten verkkojen poikkeamaperusteisia malleja ei voi Yongguang Zhang et al. mukaan soveltaa langattomiin verkkoihin sellaisenaan [1, sivu 547].

3.4.4 Ylläpito

Järjestelmän käyttö ja ylläpito on haaste, joka aiheuttaa ongelmia kaikille tietojärjestelmille, eikä IDS ole tästä poikkeus. Järjestelmiä käyttävät usein henkilöt, joille ei ole mahdollisuutta antaa mittavaa koulutusta järjestelmän käyttöön. Tästä syystä järjestelmän pitäisi olla käyttäjän näkökulmasta mahdollisimman läpinäkyvä. Lisäksi, jos langattoman verkon IDS perustuu hajautettuun malliin, asettaa järjestelmän ylläpito uusia haasteita. Tämä siksi, että laitteilla ei välttämättä ole jatkuvaa yhteyttä mihinkään keskitettyyn palvelimeen tai muuhun hallintajärjestelmään.

Monissa langattoman verkon laitteissa myös käyttöliittymä on rajoittuneempi verrattaessa sitä kiinteän verkon laitteiden käyttöliittymiin. Tämä aiheuttaa sen, että monimutkaisten järjestelmien ylläpito ja konfigurointi vaikeutuu.

3.4.5 Laaja ”hyökkäyspinta-ala”

Langattoman verkon luonteesta johtuen se tarjoaa mahdolliselle tunkeutujalle paljon hyökkäys- ja piiloutumismahdollisuuksia. Tämä on Bury et al. mukaan suora seuraus langattoman verkon turvattomasta infrastruktuurista, rajallisesta käy-

tönvalvonnasta, paikallisesta monimuotoisuudesta sekä lyhyistä kytkeytymisajoista [20].

Turvaton infrastruktuuri tarkoittaa Fluhrer et al. mukaan rajallisia salausmenetelmiä [21] ja verkon avointa topologiaa, kuten luvussa 3.4.1 kerrottiin.

Paikallinen monimuotoisuus tarkoittaa tässä sitä, että mahdollinen hyökkääjä saattaa fyysisesti piileskellä tai jopa liikkua missä tahansa verkon kantaman sisäpuolella, koska verkko ei rajoita kytkeytymispistettä millään tavoin. Lyhyt kytkeytymisaika taas vähentää tunkeutujan tunnistamis- ja kiinni jäämismahdollisuutta.

Rajallinen käytönvalvonta tulee ajankohtaiseksi esimerkiksi salaamattomissa avoimissa langattomissa verkoissa, esimerkiksi lentokentillä ja hotelleissa. Myös muut edellä mainitut seikat korostuvat avoimissa verkoissa.

3.4.6 Muut haasteet

Langattoman verkon IDS:n ollessa poikkeamaperusteinen pätevät siihen samat ongelmat kuin langallisen verkon vastaavaan systeemiin.

Poikkeavan käyttäytymisen toteaminen haitalliseksi on vaikeaa, koska verkon käyttäytyminen on luonteeltaan erittäin vaihtelevaa.

Järjestelmän pitää Bury et al. mukaan pystyä tunnistamaan, mitkä muutokset johtuvat hyökkäyksistä ja mikä on haitallista [20].

4 Keinotekoiset neuroverkot

Tässä luvussa esitellään keinotekoisten neuroverkkojen soveltamista tunkeutumisen havaitsemiseen sekä keinotekoisten neuroverkkojen teoreettista taustaa.

4.1 Neuroverkot tunkeutumisen havaitsemisessa

Perustuipa IDS mihin tahansa kappaleessa 3.3 mainittuun malliin täytyy tunnusmerkkien, poikkeamien tai tilojen luokitteluun ja havaitsemiseen käyttää jotakin sopivaa menetelmää.

Erilaiset tekoälyjärjestelmät ja luokittelijat soveltuvat melko hyvin tämäntapaisiin tehtäviin, ja erityisesti keinotekoiset neuroverkot ovat Sandarenun mukaan osoittautuneet hyviksi ratkaisuiksi vastaavissa vaikeissa ongelmissa [22], kuten hahmon tunnistuksessa (hahmontunnistuksesta Bishopin kirjassa [23]). Hahmon tunnistaminen kuvasta vastaa tässä tapauksessa hyökkäyksen ”hahmon” tai tunnusmerkin tunnistamista langattoman verkon liikenteestä.

Neuroverkkojen käyttö poikkeamien tunnistamisessa perustuu päinvastaiseen ajatteluun kuin tunnusmerkkien tunnistaminen. Poikkeamaperusteisessa järjestelmässä neuroverkolle opetetaan, millainen on normaalia, hyökkäyksetöntä liikennettä. Havaitessaan liikennettä, jota neuroverkko ei tunnista, tulkitaan se epänormaaliksi ja mahdollisesti haitalliseksi sekä suoritetaan hälytys.

Kiinteiden verkkojen poikkeamaperusteisissa IDS:issä neuroverkkoja on jo jonkin aikaa käytetty tunnistamaan poikkeavaa käytöstä, kuten Ryan et al. toteaa [24]. Langattomissa verkoissa neuroverkkojen soveltaminen ei ole vielä saavuttanut kovin suurta suosiota.

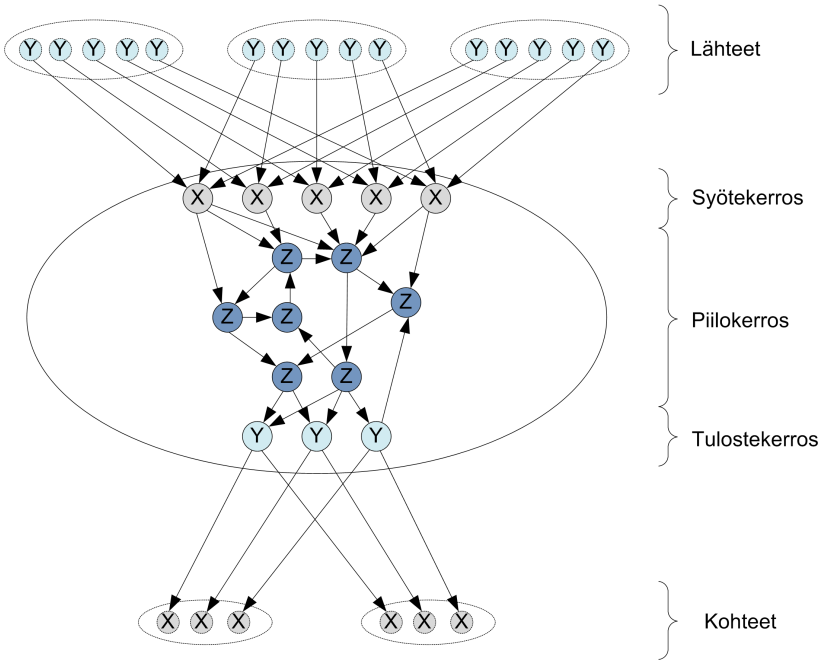
Protokollan tilojen tunnistamista liikenteestä voidaan verrata vaikkapa käsin kirjoitetun tekstin tunnistamiseen. Protokollan erillisillä tiloilla ei ole merkitystä vaan vasta tilojen sarja tunnistetaan joko normaaliksi tai epänormaaliksi. Tällöin neuroverkon kunkin hetkinen tila riippuu paitsi tämän hetkisestä syötteestä, myös aikaisemmista verkon tiloista. Kappaleessa 4.2.6 kuvattua LSTM-verkkoa on käytetty Graves et al. tutkimuksessa juuri käsin kirjoitetun tekstin tunnistamiseen erittäin hyvin tuloksin [25].

4.2 Matemaattinen perusta

Vuosien saatossa on kehitetty lukematon määrä erilaisia keinotekoisia neuroverkkomalleja. Uudet mallit pyrkivät yleensä ratkaisemaan jonkin tietyn tyyppisen ongelman paremmin kuin edeltäjänsä. Tästä syystä tietyntyyppiset verkot soveltuvat erityyppisten ongelmien ratkaisemiseen paremmin kun toiset.

Mallien kirjosta huolimatta useiden keinotekoisten neuroverkkojen taustalla ovat edelleen samat periaatteet, jotka Warren McCulloc ja Walter Pitts esittelivät 1943 (lisää aiheesta Fausettin kirjassa [26, sivu 22] ja Riosis [27] verkkosivuilla). McCulloc ja Pitts ehdottivat, että yhdistämällä useita yksinkertaisia operaatioita suorittavia laskentayksiköitä toisiinsa voitaisiin laskentavoimaa lisätä. Verkon yhteyksien painokertoimia muuttamalla saadaan se suorittamaan erilaisia ja mutkikkaampia operaatioita kuin laskentayksiköt yksinään pystyvät. Malli pyrkii jäljittelemään biologisten neuroverkkojen toimintaa.

Laskentayksiköitä kutsutaan yleisesti neuroneiksi biologisten esikuviensa, hermosolujen, mukaan. Laskentayksiöitä yhdistäviä yhteyksiä kutsutaan samasta syystä usein synapseiksi [26, sivu 5].



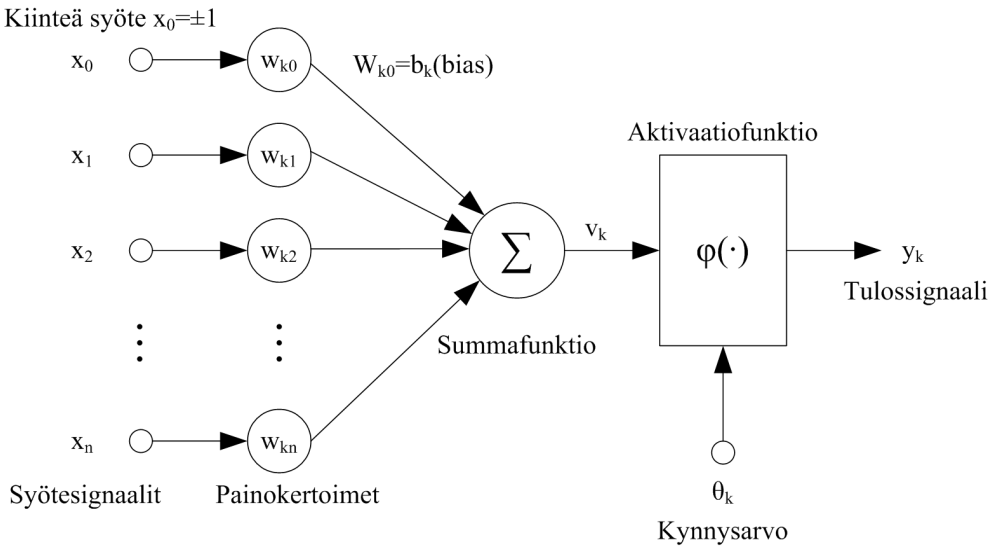
Kuva 4.1: Verkon rakenne ja suhde muihin verkkoihin.

Kuvassa 4.1 nähdään erään neuroverkon periaatteellinen rakenne. Kuvan neuroverkko on monikerroksinen verkko. Verkon kerroksia nimitetään joko syöte-, piilo- tai tuloskerroksiksi riippuen niiden näkyvyydestä verkon ulkopuolelle. Kerrosten

määrä ja toiminta riippuvat verkon tyypistä. Syötekerroksen neuronit saavat syötteensä erilaisista lähteistä, kuten esimerkiksi toisista neuroverkoista. Piilokerroksen neuronit saavat syötteensä joko syötekerrokselta tai piilokerrokselta. Piilokerroksia voi olla useampia. Syötekerroksen neuronit saavat syötteensä aiemmin mainituilta kerroksilta ja siirtävät tuloksensa eteenpäin jollekin kohteelle. Tämä kohde voi jälleen olla toinen neuroverkko. Kyseinen neuroverkko on selvästi rekursiivinen, koska piilokerroksen neuronit muodostavat silmukoita (katso kappale 4.2.5).

4.2.1 Neuroni

Kuvassa 4.2 esitetään biologiseen neuroniin perustuva matemaattinen malli.



Kuva 4.2: Neuronin matemaattinen malli (katso Rios [27]).

Malli koostuu kolmesta peruskomponentista: painokertoimista, summafunktiosta ja aktivaatiofunktiosta. Painokertoimet vastaavat biologisen neuronin synapseja eli neuroneiden välisiä yhteyksiä ja niiden voimakkuutta. Negatiivinen painokerroin tarkoittaa vaimentavaa (engl. *inhipitory*) yhteyttä. Positiivinen painokerroin taas tarkoittaa kiihdyttävää (engl. *excitatory*) yhteyttä. Painokertoimella painotetut syötesignaalit lasketaan yhteen summafunktiossa. Tämä edustaa syötteiden lineaarista yhdistämistä. Summafunktion tulos syötetään aktivaatiofunktiolle (engl. *activation function*), joka määrittelee neuronin tulossignaalin arvon. Aktivaatiofunktiosta riippuen tulosignaali vaihtelee yleensä välillä $[0, 1]$ tai $[-1, 1]$.

Kuvasta 4.2 voidaan johtaa neuronin aktiivisuuden välitulokselle (neuronin nettosyöte) kaava 4.1.

$$v_k = \sum_{j=1}^{n} w_{kj} x_j \tag{4.1}$$

Neuronin ulostulo y_k on siis aktivaatiofunktion tulos arvolla v_k kuten kaavassa 4.2 on esitetty.

$$y_k = \varphi(v_k) \tag{4.2}$$

Aktivaatiofunktioita käsitellään enemmän kappaleessa 4.2.2.

4.2.2 Aktivaatiofunktio

Kappaleessa 4.2.1 mainittiin aktivaatiofunktio ($\varphi(\cdot)$), josta käytetään joskus nimitystä siirtofunktio. Samassa verkossa voidaan käyttää useita erilaisia aktivaatiofunktioita, mutta yleensä niin, että samalla kerroksella sijaitsevilla neuroneilla on sama aktivaatiofunktio. Useimmiten aktivaatiofunktio on epälineaarinen, varsinkin monikerroksisissa verkoissa. Monikerroksisuuden hyödyntämiseksi tarvitaan epälineaarisuutta, sillä lineaarisilla aktivaatiofunktioilla varustettu monikerrosverkko vastaa Fausettin mukaan laskentavoimaltaan vastaavaa yksikerroksista verkkoa [26, sivu 17].

Erilaisia epälineaarisia aktivaatiofunktioita on Riosin mikaan käytännössä kolmenlaisia: **kynnysfunktio** (engl. *threshold function*), **paloittain lineaarinen -funktio** (engl. *piecewise-linear function*) ja **sigmoidifunktio** (engl. *sigmoid function*) [27].

Kaavassa 4.3 nähdään esimerkki kynnysfunktiosta, joka antaa arvon 1 kun syöte v on suurempi tai yhtä suuri kuin kynnysarvo θ, ja arvon 0 kun syöte on pienempi kuin kynnysarvo. Kuvan 4.3 kuvaaja **D** esittää **porrasfunktiota**, joka on kynnysfunktio kynnysarvolla 0.

$$\varphi(v) = \begin{cases} 1 & \text{jos } v \geq \theta \\ 0 & \text{jos } v < \theta \end{cases} \tag{4.3}$$

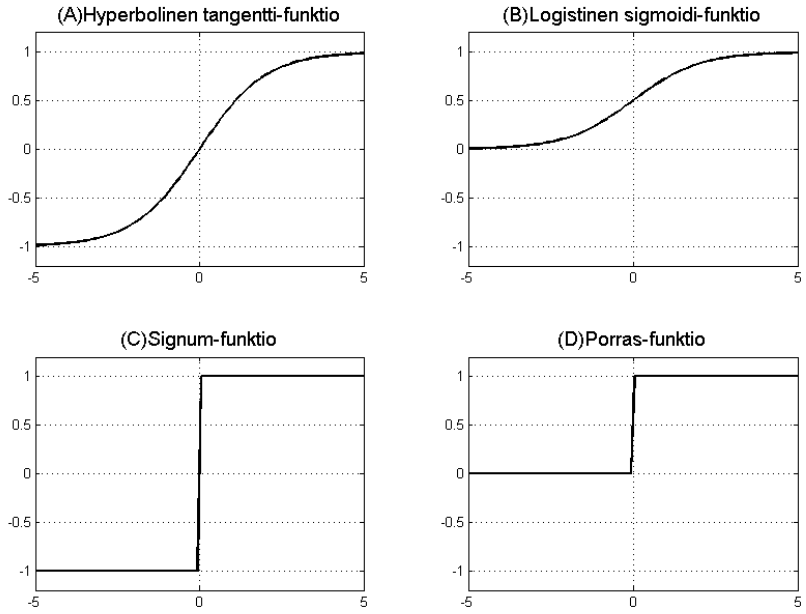
Toinen esimerkki kynnysfunktiosta on kaavassa 4.4 esitetty **signum-funktio**, joka antaa arvon 1 kun syöte v on suurempi kuin 0, arvon 0 kun syöte v on 0 ja arvon -1 kun syöte v on pienempi kuin 0. Kuvan 4.3 kuvaaja **C** esittää signum-funktiota.

$$\varphi(v) = \begin{cases} 1 & \text{jos } v > 0 \\ 0 & \text{jos } v = 0 \\ -1 & \text{jos } v < 0 \end{cases} \tag{4.4}$$

Kaavassa 4.5 nähdään esimerkki **paloittain lineaarisesta -funktiosta**, joka antaa edellisen funktion tavoin arvon 1 kun syöte v on suurempi tai yhtä suuri kuin kynnysarvo θ_{hi}, ja arvon 0 kun syöte on pienempi tai yhtä suuri kuin kynnysarvo θ_{low}. Näiden kynnysarvojen välillä funktion arvo riippuu suoraan syötteestä v.

$$\varphi(v) = \begin{cases} 1 & \text{jos } v \geq \theta_{hi} \\ v & \text{jos } \theta_{low} > v > \theta_{hi} \\ 0 & \text{jos } v \leq \theta_{low} \end{cases} \tag{4.5}$$

Kolmas ryhmä, eli **sigmoidifunktiot**, ovat jatkuvina ja derivoituvina hyvin kiinnostavia erityisesti backpropagation-algoritmien kannalta, kuten kappaleessa 4.3.3 kerrotaan.



Kuva 4.3: Neuroverkoissa Riosin mikaan yleisimmin käytettyjen epälineaaristen aktivaatiofunktioiden kuvaajat arvoalueella $[-5, 5]$ [27].

Kaavassa 4.6 nähdään esimerkki **hyperbolisesta tangentti -funktiosta**, joka saa arvoja väliltä $[-1, 1]$. Kuvan 4.3 kuvaaja **A** esittää kyseistä funktiosta.

$$\varphi(v) = \frac{1 - e^{(-2v)}}{1 + e^{(-2v)}} \tag{4.6}$$

Kaavassa 4.7 nähdään esimerkki **logistisesta sigmoidi -funktiosta**, joka saa arvoja väliltä $[0, 1]$. Käyrän jyrkkyyttä voidaan säätää parametria σ. Kuvan 4.3 kuvaaja **B** esittää kyseistä funktiota σ:n arvolla 1.

$$\varphi(v) = \frac{1}{1 + e^{(-\sigma v)}} \tag{4.7}$$

Epälineaaristen aktivaatiofunktioiden lisäksi käytetään lineaarista, niin kutsuttua identiteettifunktiota (engl. *identity function*). Tätä kaavan 4.8 kuvaamaa funktiota käytetään yleensä syötekerroksen aktivaatiofunktiona.

$$\varphi(v) = v \tag{4.8}$$

4.2.3 Aktivaatiofunktioiden derivaatat

Tässä työssä käytetään kappaleen 4.2.6 mukaista LSTM-verkkoa, jonka yksiköiden aktivaatiofunktioina toimivat logistinen sigmoidi -funktio, hyperbolinen tangentti -funktio sekä identiteettifunktio. Näiden lisäksi kertolaskuyksiköissä suoritetaan usean muuttujan kertolasku, jota voidaan rinnastaa aktivaatiofunktioon. Nämä kaikki ovat derivoituvia.

Logistisen sigmoidi -funktion derivaatta voidaan laskea kaavalla 4.9.

$$\varphi(v)' = \sigma\varphi(v)(1 - \varphi(v)) \tag{4.9}$$

Hyperbolisen tangentti -funktion derivaatta voidaan laskea kaavalla 4.10.

$$\varphi(v)' = (1 + \varphi(v))(1 - \varphi(v)) \tag{4.10}$$

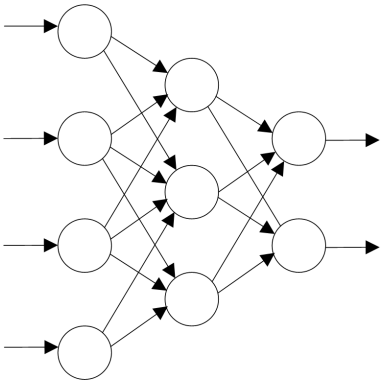
Identiteettifunktion derivaatta on 1, joten usean muuttujan kertolaskun derivaatta voidaan laskea kaavalla 4.11.

$$\frac{d\varphi(v_0, v_1, v_2, ..., v_i, ..., v_n)}{dv_i} = \varphi(v_i)' \cdot v_0 \cdot v_1 \cdot v_2 \cdot ... \cdot v_n = v_0 \cdot v_1 \cdot v_2 \cdot ... \cdot v_n \tag{4.11}$$

4.2.4 Feedforward-verkot

Myötäkytkentäverkko (engl. *feedforward network*) tarkoittaa neuroverkkotyyppiä, jossa ei esiinny silmukoita. Verkko voi olla yksi, kaksi tai useampikerroksinen. Monet verkkotyypit koostuvat syöte-, tulos- ja yhdestä tai useammasta piilokerroksesta. Kerrokset koostuvat kappaleessa 4.2 kuvatuista neuroneista. Kullakin kerroksella sijaitsevat neuronit voivat liittyä ainoastaan seuraavilla kerroksilla sijaitseviin, mutta ei samalla tai edellisellä kerroksella sijaitseviin, neuroneihin. Feedforward-verkkoja on monia eri tyyppisiä. Esimerkiksi logistista sigmoidi- ja porrasfunktiota käyttäviä

monikerroksisia feedforward-verkkoja, joiden opettamisessa käytetään kappaleessa 4.3.2 kuvattua perceptron-sääntöä, kutsutaan yleisesti monikerrosperseptroneiksi. Kuvassa 4.4 esimerkki yksinkertaisesta feedforward verkosta.



Kuva 4.4: Esimerkki yksinkertaisesta kolmikerroksisesta feedforward-verkosta.

Verkon piilokerrosten koko ja määrä vaikuttavat verkon kykyyn oppia. Kahdella tai useammalla painotetulla kerroksella varustettua feedforward-verkkoa voidaan Bishopin mukaan käyttää universaalina approksimaattorina, eli toisin sanoen se voi arvioida mielivaltaisella tarkkuudella minkä tahansa jatkuvan funktion [23]. Tätä ominaisuutta hyväksi käyttäen voidaan feedforward-verkolla luokitella aineistoa, jota esitetään verkolle sen syöteneuronien kautta. Verkkoa opettamalla, eli verkon panoja muuttamalla, saadaan verkko antamaan tietyllä syötteellä haluttu tulos. Verkon opettamisesta kerrotaan lisää kappaleessa 4.3.

Verkolle syötettävää tietoa kutsutaan ”piirteiksi”. Piirteet ovat yleensä tunnistettavasta aineistosta erotettua, yksinkertaistettua tietoa. Lisää piirteistä ja niiden erottamisesta kappaleessa 4.5. Jokaista piirrettä kohti tarvitaan yksi syöteneuroni verkon syötekerrokselle. Piirrejoukon kasvaessa verkon koko, ja tätä kautta verkon laskentaan tarvittava kapasiteetti, kasvaa. Tästä syystä piirrejoukon koko olisi pyrittävä pitämään mahdollisimman pienenä, mutta kuitenkin tarpeeksi isona, jotta verkolla on tarpeeksi tietoa eri hahmojen tai kuvioiden erottamiseen toisistaan.

Kun pakettien sisällöstä tai otsikkotiedoista erotetaan piirteitä voidaan niiden perusteella tunnistaa millainen paketti on kyseessä, kuten kappaleessa 4.1 kerrottiin. Feedforward-verkko ei kuitenkaan pysty tunnistamaan näiden yksittäisten pakettien muodostamia sekvenssejä, vaan siihen tarvitaan kappaleessa 4.2.5 kuvailtava rekursiivinen neuroverkko.

4.2.5 Rekursiiviset neuroverkot

Fausettin mukaan rekursiiviset neuroverkot (engl. *Recurrent Neural Networks*) ovat neuroverkkoja, joissa esiintyy silmukoita [26, sivu 102]. Rekursiivisia neuroverkkoja on kehitetty, koska perinteiset liukuvalla aikaikkunalla varustetut feedforward-verkot eivät kykene tunnistamaan kovin pitkiä ajallisia sekvenssejä. Sekvenssin pituutta rajoittaa aikaikkunan koko, joka pysyy vakiona ja jonka optimaalista pituutta on vaikea määritellä (katso lisää Frank et al. tutkimuksesta [28]). Rekursiivisissa verkoissa edelliset tilat säilötään syöteneuroneihin liitetyn liukuvan aikaikkunan sijaan viivästettyihin takaisi kytkettyihin yhteyksiin. Tällä tavoin verkon neuronien edelliset tilat toimivat yhdessä verkon ulkopuolelta tulevien syötteiden kanssa määriteltäessä verkon seuraavaa tilaa. Liukuvan aikaikkunan tapauksessa verkon kunkin hetkiseen tilaan vaikuttaa ainoastaan verkon ulkopuolinen syöte eikä sillä ole sisäistä muistia.

Rekursiivinen neuroverkko on tunkeutumisen havaitsemisen kannalta mielenkiintoinen tekniikka juuri sen vuoksi, että sillä pystytään tunnistamaan monimutkaisia ajallisia sekvenssejä. Sekvenssien tunnistaminen on mielenkiintoista siksi, että tietoliikenne protokollat määrittelevät ja käyttävät yleensä tarkasti määriteltyjä pakettisekvenssejä, joissa tietyntyyppisille paketeille on määritelty tietyntyyppinen vastaus paketti. Esimerkiksi avoimen IEEE 802.11 -verkon autentikointisekvenssi [8, sivu 161] koostuu yhdestä pyyntö ja vastaussanomasta, joilla on tarkkaan määritelty sisältö.

Käytettäessä rekursiivista neuroverkkoa voidaan tunnistaa periaatteessa kahdenlaisia protokolliin liittyviä väärinkäytöksiä:

- väärin tai poikkeavasti koostettuja paketteja ja/tai
- väärään tai poikkeavaan aikaan lähetettyjä paketteja.

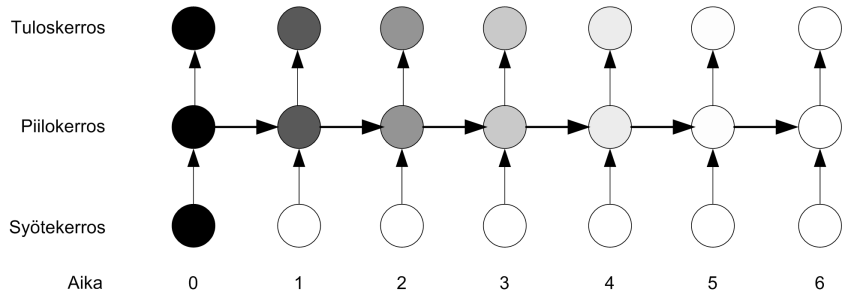
Rekursiivinen neuroverkko voi siis muodostaa eräänlaisen tilakoneen, joka seuraa verkon tapahtumia ja ilmoittaa havaitessaan tunnistettuja tai tunnistamattomia kuvioita liikenteessä. Hyötynä etukäteen ohjelmoituun tilakoneeseen verrattuna on se, että neuroverkko voi oppia tunnistettavat sekvenssit itse. Uusien protokollien tai hyökkäysten lisääminen vaatii ainoastaan uuden opetusjakson.

4.2.6 Long Shor Term Memory, LSTM

LSTM on eräs rekursiivinen neuroverkkomalli, jonka kehittivät Sepp Hochreiter ja Jürgen Schmidhuber 1990-luvun alussa. LSTM pyrkii ratkaisemaan perinteisiä re-

kursiivisia verkkoja vaivaavan niin kutsutun ”katoavan gradientin ongelman” (katso lisätietoja Hochreiter et al. tutkimuksesta [29]), joka tekee vaikeaksi yhdistää yli kymmenen aika-askeleen päässä sijaitsevat syötteet ja mallit [30].

Kuvassa 4.5 nähdään tavallinen rekursiivinen neuroverkko (esimerkiksi kappaleessa 4.3.3 kuvattu BPTT-verkko) kuvattuna ajassa. Verkko koostuu yhdestä syöte-, piilo- ja tuloskerroksen neuronista. Piilokerroksen neuroni on liitetty rekursiivisesti itseensä. Neuronien väri kuvaa niiden herkkyyttä ajanhetkellä 0 esitetylle syötteelle. Musta väri kuvaa suurta herkkyyttä ja valkoinen väri pientä herkkyyttä. Kuvasta voidaan nähdä miten ensimmäisen syötteen vaikutus vähenee eksponentiaalisesti ajan kuluessa (katso lisätietoja Graves et al. tutkimuksesta [25, sivu 5]).

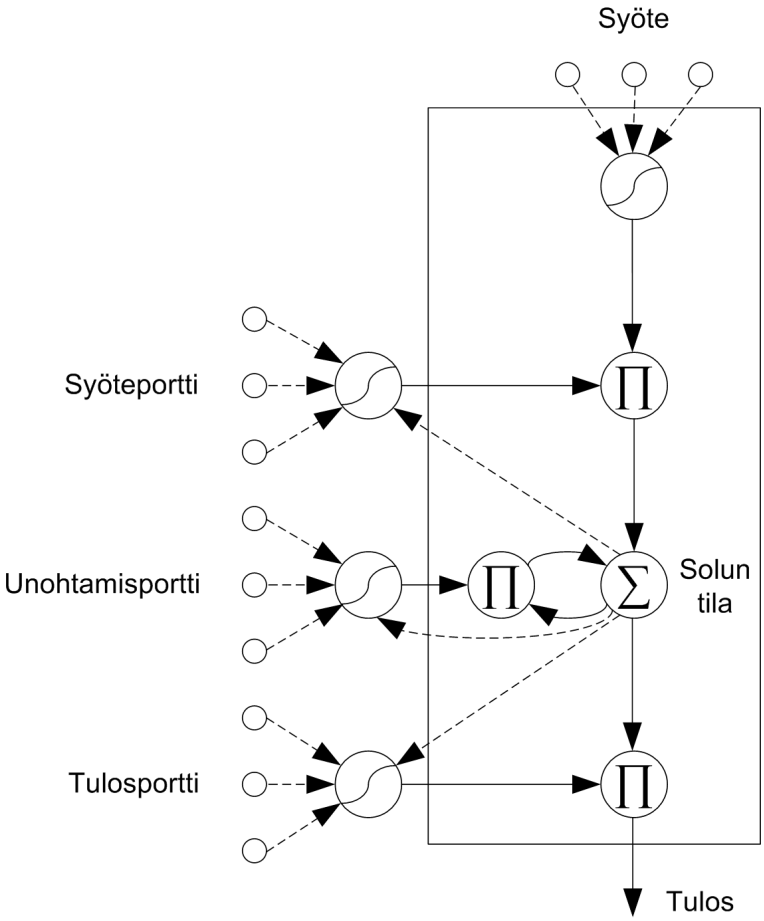


Kuva 4.5: Havainnollistus gradientin häipymisestä normaalissa rekursiivisessa verkossa (katso Graves et al. tutkimus [25, kuva 10])

Kuvassa 4.6 on esimerkkinä yleisesti käytetty LSTM-lohko. Esimerkkinä käytetty Gers et al. tutkimuksessaan esittelemä LSTM-lohko sisältää neljä epälineaarista yksikköä: syöte-, syöteportti-, unohtamisportti- ja tulosporttiyksikkö. Jokainen näistä yksiköistä vastaa kappaleessa 4.2.1 kuvattua yksittäistä neuronia. Yksikön aktivaatiofunktiona voidaan käyttää esimerkiksi kappaleessa 4.2.2 mainittua logistista sigmoidifunktiota [31].

Esimerkkilohko sisältää yhden muistisolun, joka kuvassa sijaitsee suorakulmion sisällä. Muistisolu koostuu edellä mainitusta syöteyksiköstä ja lineaarisesta yksiköstä. Lineaarista yksikköä kutsutaan myös solun tilaksi. Yksi LSTM-lohko voi sisältää useampia muistisoluja. Lineaarisen yksikön oma tulossignaali summataan sen syötesignaaliin. Tätä kutsutaan Gers et al. mukaan ”jatkuvaksi virhekaruselliksi” (engl. *Constant Error Carousel, CEC*) [31].

Lohko sisältää myös kolme kertojayksikköä. Alimmassa kertojayksikössä kerrotaan keskenään syöte- ja syöteporttiyksiköiden tulokset. Keskimmäisessä kertojassa kerrotaan keskenään aiemmin mainittu lineaarisen yksikön yhden aika-askeleen verran viivästetty CEC-takaisinkytkentä ja Gers et al. toisessa tutkimuksessaan esittelemän unohtamisportin tulos [32]. Ylimmässä kertojassa kerrotaan keskenään li-



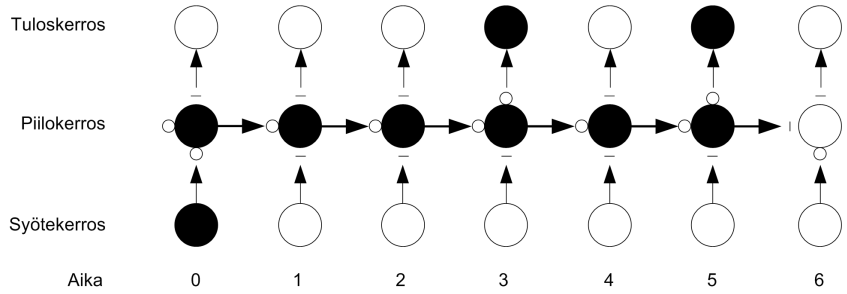
Kuva 4.6: Esimerkki LSTM-lohkosta (katso Gers et al. tutkimus [31, sivu 121])

neaarisen yksikön ja tulosportin tulokset. Tällä tavoin kertojien välityksellä portit ohjaavat syötteen päätymistä lineaariseen yksikköön, lineaarisen yksikön takaisinkytkennän voimakkuutta ja lineaarisen yksikön tuloksen päätymistä lohkon tulokseksi [31].

Katkoviivalla merkityt yhteydet tarkoittavat epälineaaristen yksiköiden painotettuja syötteitä. Verkko oppii näiden syötteiden painokertoimia muuttamalla, kuten kappaleessa 4.3 kerrotaan. CEC:stä portteihin johtavat muuttuvat yhteydet ovat niin kutsuttuja ”kurkistusyhteyksiä” (engl. *peephole connection*). LSTM-lohkon suorityskyky paranee Gers et al. mukaan huomattavasti kun se varustetaan kyseisillä yhteyksillä [31].

Jatkuvalla viivalla kuvatut yhteydet ovat kertoimella 1 varustettuja vakioyhteyksiä, jotka eivät muutu opetuksessa.

Kuvassa 4.7 nähdään esimerkki siitä, miten LSTM-verkko suojelee signaalia sulkemalla ja avaamalla edellä esitettyjä LSTM-lohkon portteja tarpeen mukaan. Verkko koostuu kuvan 4.5 tapaan yhdestä syöte- ja tuloskerroksen neuronista. Piilokerros koostuu yhdestä LSTM-lohkosta. Selvyyden vuoksi portit ovat joko kokonaan auki (o) tai kokonaan kiinni (-). Lohkon sisällä sijaitseva CEC säilyttää tilansa jos lohkon syöteportti on kiinni ja unohtamisportti on auki. Lohkon tila ei näy tuloskerroksella jos tulosportti on kiinni.



Kuva 4.7: Havainnollistus gradientin säilymisestä LSTM-verkossa (katso Graves et al. tutkimus [25, kuva 12])

4.3 Neuroverkon opettaminen

Verkon opettaminen, eli verkon solmujen välisten painokertoimien muuttaminen on neuroverkkoihin liittyvä keskeinen ongelma. Neuroverkkoa opettamalla pyritään kyseinen neuroverkko muokkaamaan sellaiseksi, että se tietyn syötteen saadessaan tuottaa halutun tuloksen. Lisäksi neuroverkon pitäisi pystyä approksimoimaan opetettujen syötteiden välille jääviä tuloksia siten, että verkolle tuntematto-

mat syötteet antavat mielekkäitä tuloksia. Tällä tavoin neuroverkkoa voidaan käyttää luokittelijana, kuten kappaleen 4 alussa mainittiin. Tarkoitukseen on kehitetty monia erilaisia menetelmiä, sääntöjä ja algoritmeja.

4.3.1 Opetusmenetelmät

Neuroverkkojen opettamiseen käytetään periaatteessa kahta erilaista menetelmää: ohjattu oppiminen (engl. *supervised learning*) ja ohjaamaton oppiminen (engl. *unsupervised learning*). Lisäksi lähteestä riippuen saatetaan mainita muitakin menetelmiä, kuten esimerkiksi vahvistettu oppiminen (engl. *reinforcement learning*).

Ohjattu oppiminen tarkoittaa Fausettin mukaan opetusmenetelmää, jossa verkkoa opetetaan käyttäen hyväksi verkon tuottamaa tulosta. Opettaminen tapahtuu käytännössä esittämällä verkolle sarja syötevektoreita tai -kuvioita sekä niitä vastaavia opetusvektoreita. Opetusvektorista ja verkon tulosvektorista lasketaan virhe, jota käytetään verkon painojen muuttamiseen. Opetusvektori voi sisältää esimerkiksi useita binäärisiä arvoja väliltä $[0, 1]$ tai $[-1, 1]$, riippuen verkosta. Nämä yksittäiset vektorin alkiot voivat edustaa esimerkiksi luokkia, joihin verkon halutaan luokittelevan sille esitetty aineisto. Jos verkon halutaan luokittelevan tietty syöte johonkin luokkaan, asetetaan kyseistä luokkaa edustava alkio arvoon 1 ja muut arvoon 0 tai -1 samaan aikaan kun syöte esitetään. Vaikka opetusvektorissa on ainoastaan yksi luokka kerrallaan aktiivisena, ei verkon syöte luokittelutilanteessa välttämättä tarjoa vastaavia tuloksia. Verkko saattaa esimerkiksi luokitella syötteen useampaan eri luokkaan, tai tuloskerroksen aktivaatiofunktiosta riippuen, hieman useisiin luokkiin. Ohjatulla oppimisella opetettu monikerroksinen verkko pystyy periaatteessa kuvaamaan mielivaltaisen moniulotteisen avaruuden toiseksi mielivaltaisen moniulotteiseksi avaruudeksi riippuen syöte- ja tulosvektorien pituuksista [26, sivut 15–16].

Haittana ohjatussa oppimisessa on opetusaineiston tarve. Verkon opettamiseen tarvitaan ennalta valmisteltua ehdottoman paikkansapitävää aineistoa.

Ohjaamaton oppiminen tarkoittaa Fausettin mukaan opetusmenetelmää, jossa verkkoa opetetaan ainoastaan käyttäen hyväksi opetusvektoreita tai syötevektoreita. Tätä opetusmenetelmää käyttäviä verkkoja kutsutaan itseorganisoituviksi ja ne kykenevät ryhmittelemään toisiaan muistuttavat, eli syöteavaruudessa toisiaan lähellä olevat, syötevektorit ryhmiin tai klustereihin. Ehkä tunnetuin tämän tyyppinen verkko on Teuvo Kohosen kehittämä SOM (engl. *Self Orgnized Map*) [26, sivu 16].

Toisin kuin ohjatussa oppimisessa ei ohjaamattomassa oppimisessa ole ennalta määriteltyjä tiukkoja luokkia tai ryhmiä, joihin syötteet luokitellaan. Verkon täytyy

itse ryhmitellä syötteet ja luoda niille kuvaus.

Vahvistettu oppiminen voidaan Riosin mukaan ajatella kahden edellä esitetyn opetustavan yhdistelmäksi. Mallissa opettaminen tapahtuu ohjaamattoman oppimisen tapaan, mutta ohjaamista hienosäädetään verkolle annetulla positiivisella ja negatiivisella palautteella. Verkko saattaa olla kytketty johonkin järjestelmään, joka tuottaa edellä mainitun palautteen verkon järjestelmään aiheuttaman vaikutuksen perusteella. Verkkoa ikään kuin palkitaan tai rangaistaan sen tekemien päätösten perusteella [27].

Opetusmenetelmät voidaan jakaa vielä suoraan oppimiseen (engl. *on-line learning*) ja erilliseen oppimiseen (engl. *off-line learning*) sen mukaan, suoritetaanko verkon opettaminen erillisellä opetusjaksolla vai oppiiko verkko käytön aikana.

4.3.2 Opetussäännöt

Psykologi Donald Hebb esitteli 1940-luvulla ensimmäisen säännön kuvaamaan neuroneiden välisten liitosten painokertoimien muutoksia. Sääntöä käytetään Hebb net -verkoissa. Säännön perusideana on, että neuroneiden välinen yhteys vahvistuu jos neuronit aktivoituvat samanaikaisesti ja heikkenee jos neuronit aktivoituvat erikseen. Edellisen toiminnan toistaminen johtaa ennen pitkää merkityksellisten yhteyksien vahvistumiseen ja merkityksettömien yhteyksien heikkenemiseen. Yhteyden vahvistuminen tarkoittaa tässä tapauksessa kertoimen kasvamista heikkeneminen kertoimen lähestymistä nollaa (katso lisätietoja Fausettin kirjasta [26, sivu 48]).

Alkuperäinen sääntö ei ottanut kantaa siihen mitä tapahtuu kun neuronit ovat samanaikaisesti ei-aktiivisia. Laajennettu Hebb:n sääntö voimistaa oppimista vahvistamalla yhteyttä myös tässä tapauksessa. Kaavassa 4.12 nähdään laajennettu Hebb:n sääntö. Sääntö toimii tapauksissa, joissa tieto on esitetty bipolaarisesti, eli välillä $[-1, 1]$ [26, sivu 49].

$$w_{newj} = w_{oldj} + x_j y \tag{4.12}$$

Kaavassa 4.12 j merkitsee yhteyden indeksiä ja w yhteyden voimakkuutta (vertaa kuva 4.2). Painokertoimen muutostermiä $x_j y$ merkitään usein käyttäen merkintää Δw kuten kaavassa 4.13.

$$\Delta w_j = x_j y \tag{4.13}$$

Nykyistä ajanhetkeä voidaan merkitä kirjaimella t. Tällöin painokertoimen uusi arvo voidaan laskea kaavalla 4.14.

$$w_j^t = w_j^{t-1} + \Delta w_j \tag{4.14}$$

Perceptron-sääntöä voidaan pitää Hebb:n säännön seuraavana kehitysaskeleena. Perceptron-sääntöä käytetään perceptroneissa, jotka ovat itse asiassa kokonainen neuroverkkojen luokka, jota useat eri tutkijat ovat kehittäneet [26, sivu 23].

Perceptron-säännön ideana on käyttää percepronin tuottaman tuloksen virhettä laskettaessa uusia painokertoimia. Jokaista perceptronin syötevektoria vastaa perceptronin tuottama tulos. Tätä tulosta opetusarvoon (malliin) vertaamalla lasketaan perceptronin virhe. Alkuperäisessä perceptronissa käytettiin aktivaatiofunktiona kynnysfunktiota (katso kappale 4.2.2), joka antaa arvon -1, 0 tai $+1$. Perceptron-sääntö huomioi ainoastaan virheen etumerkin, ei sen suuruutta. Täten virheet, joista toinen laskettiin vertaamalla perceptronin antamaa tulosta 0 opetusarvoon -1, ja toinen laskettiin vertaamalla tulosta $+1$ opetusarvoon -1, tuottavat säännön kannalta saman tuloksen. Molemmissa tapauksissa ainoastaan virheen etumerkki kertoo, että painoja täytyy muuttaa opetusvektorin arvon ilmoittamaan suuntaan. Virheen ollessa 0 ei muutosta tarvitse laskea. Toinen huomioitava seikka on, että ainoastaan yhteydet, jotka syöttivät nollasta poikkeavia arvoja, osallistuivat virheen tuottamiseen. Näin ollen ainoastaan näiden yhteyksien painoja täytyy muuttaa. Edellä esitettyä sääntö kuvaava muutostermi voidaan esittää kaavalla 4.15 [26, sivu 59].

$$\Delta w_j = \alpha x_j t \tag{4.15}$$

Kaavassa 4.15 t esittää opetusarvoa, joka voi saada arvon -1 tai $+1$. α on oppimiskerroin, joka vaikuttaa painoihin tehtävien muutosten suuruuteen ja sitä kautta oppimisnopeuteen. Oppimiskerroin täytyy olla tarpeeksi suuri jotta oppiminen tapahtuisi mielekkäässä ajassa, mutta ei kuitenkaan liian suuri, koska se johtaisi Bishopin mukaan kertoimien oskilloimiseen [23, sivu 95]. Muutoskerrointa ei lasketa jos virhettä ei ole, eli $t = y$.

Bernard Widrow ja Mercian Hoff esittelivät 1960 läheisesti perceptron-sääntöön liittyvän säännön, jota nimitetään joko Widrow-Hoff:n, pienimmän keskineliövirheen (engl. *least mean square error*) tai delta-säännöksi [26, sivu 23].

Ideana delta-säännössä on muuttaa verkon painoja siten, että tulosneuronien nettosyötteen v ja opetusarvon t välinen erotus minimoituu. Tarkoituksena on minimoida erotus yli koko opetusjoukon, eli kaikkien opetuskuvioiden (engl. *training pattern*). Tähän tulokseen päästään kuitenkin pienentämällä virhettä yksi opetuskuvio kerrallaan. Virheen korjaukset voidaan myös haluttaessa antaa kumuloitua yli

useamman opetuskuvion. Jatkossa käytetään merkintää J erottamaan derivoinnissa käytetty painon w indeksi nettosyötteen kaavassa 4.1 esiintyvästä painon indeksistä j. Kaksikerroksisessa verkossa, jossa on n kappaletta syöteneuroneita ja yksi tulosneuroni delta-sääntö painon w_J muuttamiseen per opetuskuvio voidaan esittää kaavalla 4.16 [26, sivu 86].

$$\Delta w_J = \alpha x_J (t - v) \tag{4.16}$$

Kaava 4.16 voidaan johtaa seuraavasti:

x esittää $n + 1$ alkioista tulosneuronin syötevektoria, jossa kukin alkio kuvaa yhden syöteneuronin aktivaatiota y. v on tulosneuronin nettosyöte, ja se lasketaan kaavan 4.1 mukaisesti. t esittää verkon opetusarvoa.

Neliövirhe tietylle opetuskuviolle lasketaan kaavalla 4.17.

$$E = \frac{1}{2}(t - v)^2 \tag{4.17}$$

E on kaikkien painojen w_j ($j \in \{0, 1, ..., n\}$) funktio. E:n gradientti taas on vektori, joka koostuu E:n osittaisderivaatoista näiden painojen suhteen. Gradientti antaa suunnan, jossa E kasvaa nopeimmin. Tällöin E luonnollisesti pienenee nopeimmin vastakkaisessa suunnassa. Virhettä voidaan siis pienentää muuttamalla painoa w_J $-\frac{dE}{dw_J}$:n osoittamassa suunnassa [26, sivu 86].

Derivoimalla kaava 4.17 saadaan

$$\frac{dE}{dw_J} = \frac{-2}{2}\frac{dv}{dw_J}(t - v) = x_J(t - v), \tag{4.18}$$

joten paikallinen virhe pienenee nopeimmin muuttamalla painoja kaavan 4.16 mukaisesti [26, sivu 87].

Verkolle, jossa on enemmän kuin yksi tulosneuroni, deltasääntö voidaan esittää kaavan 4.19 mukaisesti. K esittää tulosneuronin indeksiä. Verkossa on edelleen kaksi kerrosta kuten edellä kuvattiin [26, sivu 87].

$$\Delta w_{KJ} = \alpha x_J (t_K - v_K) \tag{4.19}$$

Kaava 4.19 saadaan edellä kuvatulla tavalla derivoimalla kun neliövirhe korvataan kaavassa 4.20 kuvatulla neliövirheiden summalla, jossa k on tulosneuronin indeksi [26, sivu 87].

$$E = \frac{1}{2}\sum_{k=0}^{m}(t_k - v_k)^2 \tag{4.20}$$

Edellä esitetty delta-sääntö koski ainoastaan identiteettifunktiolla (katso kaava 4.8) varustettuja tulosneuroneita. Sääntö voidaan kuitenkin laajentaa koskemaan kaikkia derivoituvalla aktivaatiofunktiolla varustettuja tulosneuroneita. Tämä laajennettu sääntö painolle w_{KJ} voidaan esittää kaavalla 4.21 [26, sivu 106].

$$\Delta w_{KJ} = \alpha x_J (t_K - y_K) \varphi'(v_K) \tag{4.21}$$

Kaava 4.21 saadaan edelleen edellä kuvatulla tavalla derivoimalla kun neliövirhe korvataan kaavassa 4.22 kuvatulla neliövirheiden summalla, jossa k on tulosneuronin indeksi ja y lasketaan kaavalla 4.2.

$$E = \frac{1}{2} \sum_{k=0}^{m} (t_k - y_k)^2 \tag{4.22}$$

Kaavasta 4.16 on olemassa useita muunnoksia joilla pyritään parantamaan kappaleen 4.3.3 backpropagation-menetelmän toimintaa, kuten Fausett kirjassaan mainitsee [26, sivu 305]. Tässä työssä käytetään niin kutsuttua momenttia (engl. *momentum*), jonka tehtävä on vähentää muusta aineistosta suuresti poikkeavan opetusmateriaalin vaikutusta opetukseen. Momentti yrittää nimensä mukaisesti muuttaa painokerrointa samaan suuntaa johon se aikaisemmin oli muuttumassa.

$$\Delta w_J^t = \alpha x_J (t - v) + \eta \Delta w_J^{t-1} \tag{4.23}$$

Kaavassa 4.23 nähdään momentilla laajennettu deltasääntö. Kaavassa η on momentti, jolla kerrottu edellisen opetuskerroksen muutos lisätään menossa olevan kierroksen muutokseen.

Kaavan 4.16 oppimiskerroin α voidaan tehdä myös dynaamiseksi. Tästä voi olla hyötyä esimerkiksi kappaleen 5.5.9 mukaisessa järjestelmässä. Oppimiskertoimen muuttaminen voisi toimia siten, että aluksi oppimiskerroin on suuri ja painokertoimet muuttuvat paljon. Ajan kuluessa oppimiskerrointa pienennetään, jolloin painokertoimista tulee pysyvämpiä eivätkä ne muutu enää kovin herkästi. Tästä on se etu, ettei jo opittua tietoa enää hukata myöhemmin. Tällaisesta toiminnasta on viitteitä myös luonnosta, kuten Gan et al. tutkimuksesta selviää [33]. Ominaisuudesta on hyötyä varsinkin silloin, kun useita eritasoisia verkkoja opetetaan ja käytetään ryhmässä. Uusien verkkojen täytyy oppia nopeasti, mutta vanhojen vain sen verran, että ne mukautuvat toimimaan uusien verkkojen kanssa. Tästä lisää kappaleessa 5.5.3 ja 5.5.9.

4.3.3 Backpropagation

Backpropagation-opetusmenetelmä tarkoittaa kappaleessa 4.3.2 kuvatun laajennetun delta-säännön soveltamista kappaleessa 4.2.4 kuvattujen monikerroksisten feedforward verkkojen opetukseen. Fausettin mukaan menetelmä on geneerisen luonteensa vuoksi hyvin käytetty monissa eri sovelluksissa [26, sivu 289].

Menetelmä sisältää kolme toisistaan erotettavaa vaihetta: syötteen eteenpäin siirtäminen (engl. *feedforward*), virheen laskeminen ja taaksepäin levittäminen (engl. *backprobagation*) sekä painojen muuttaminen. Opetusvaiheen jälkeen suoritetaan ainoastaan ensimmäistä vaihetta. Tällöin, vaikka verkon opettaminen olisi hidasta, toimii opetettu verkko suhteellisen nopeasti.

Eteenpäin-vaihe ei eroa muiden feedforward-tyyppisten verkkojen vastaavasta toiminnasta. Varsinainen ero, kuten menetelmän nimikin vihjaa, piilee taaksepäin-vaiheessa. Kuten kappaleessa 4.3.2 mainittiin, on tavoitteenamme minimoida verkon virhe E, joka on kaikkien opetus-tulos -vektoriparien virheiden summa:

$$E = \sum_n E^n \tag{4.24}$$

Tarkoituksenamme on minimoida tämä virhe muuttamalla verkon painokertoimia w. Kuten kappaleessa 4.3.2 mainittiin, voimme tehdä sen muuttamalla painokertoimia gradientin vastakkaisessa suunnassa aina opetus-tulos -vektoripari kerrallaan. Voimme huomata, että E^n riippuu painosta w_{kj} ainoastaan neuronin k yhteenlasketun nettosyötteen v_k kautta (katso kaava 4.1). Näin ollen voimme Bishopin mukaan soveltaa gradientin osittaisderivaattoihin ketjusääntöä ja kirjoittaa gradientin kaavan 4.25 [23, kappale 4.8]:

$$\frac{dE^n}{dw_{kj}} = \frac{dE^n}{dv_k}\frac{dv_k}{dw_{kj}} \tag{4.25}$$

Seuraavaksi laajennamme jälleen kerran deltasääntöä esittelemällä neuronin virhe δ, jota levitetään verkossa vastasuuntaan alkaen tuloskerrokselta ja päätyen syötekerrokselle. Laskemme seuraavassa virheen yhdelle opetus-tulos -vektoriparille n. Merkitsemme siis kaavan 4.25 oikean puolen ensimmäistä tekijää kirjaimella δ_j, joka ilmoittaa neuronin j virheen:

$$\delta_k = \frac{dE}{dv_k} \tag{4.26}$$

joka voidaan edelleen ketjusääntöä hyväksi käyttäen kirjoittaa muotoon

$$\frac{dE}{dv_j} = \sum_k \frac{dE}{dv_k} \frac{dv_k}{dv_j}, \tag{4.27}$$

jossa summa käy yli kaikkien neuronien k joihin neuronista j on syöttävä yhteys.

Virhefunktiona E voidaan käyttää esimerkiksi hyvin yleisesti käytettyä kaavan 4.22 neliövirheiden summaa. Oletuksena siis on, että tuloskerroksen neuronista (k) ei lähde yhteyksiä muihin neuroneihin. Näillä tiedoilla saamme kaavasta 4.27 tuloskerroksen k:nen neuronin virheelle yksinkertaisen kaavan 4.28.

$$\delta_k = t_k - y_k \tag{4.28}$$

Muiden kerrosten neuroneille (j) virhettä laskettaessa täytyy ottaa huomioon neuronit (k), joiden virheen aiheuttamiseen kyseinen neuroni on osallistunut, eli ne neuronit, joihin kyseinen neuroni eteenpäin-vaiheessa oman tuloksensa (y_j) syöttää. Tällöin kaavasta 4.27 saadaan virheelle kaava 4.29, jossa j on syöttävän ja k on syötettävän neuronin indeksi.

$$\delta_j = \varphi'(v_j) \sum_k w_{kj} \delta_k \tag{4.29}$$

Derivoimalla kaava 4.1 w_{kj}:n suhteen saamme kaavan 4.30.

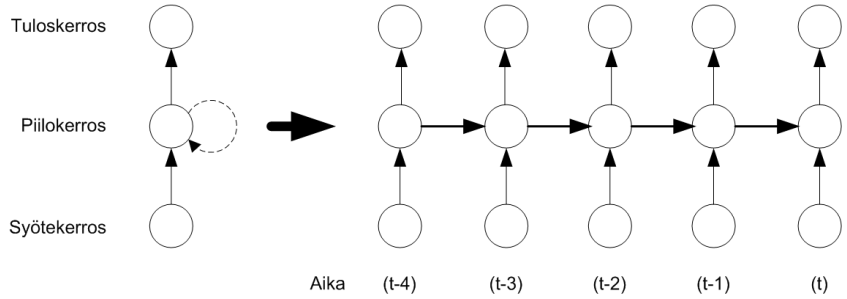
$$\frac{dv_k}{dw_kj} = x_j \tag{4.30}$$

Sijoittamalla kaavat 4.26 ja 4.30 kaavaan 4.25 saamme gradientille kaavan 4.31 mukaisen ratkaisun:

$$\frac{dE}{dw_{kj}} = \delta_k x_j \tag{4.31}$$

Backpropagation-algoritmin soveltamista rekursiivisiin verkkoihin kutsutaan nimellä BPTT (engl. *backpropagation through time*). Rekursiivinen verkko voidaan ajatella ryhmäksi alkuperäisen verkon kopioita, joista on poistettu alkuperäisen verkon silmukat. Nämä kopiot kommunikoitavat keskenään taaksepäin suuntautuneet viivästetyt silmukkayhteydet korvaavien tavallisten yhteyksien välityksellä. BPTT-algoritmin soveltamisessa ensimmäinen vaihe onkin purkaa rekursiivinen verkko kuvan 4.8 mukaisesti siten, että verkossa ei esiinny katkoviivalla merkittyjä viivästettyjä silmukoita (katso lisätietoja Bodén oppaasta [34]).

Tässä työssä käsitetään LSTM-verkkoa, jonka lohkot muodostuvat kappaleen 4.2.6 mukaisesti pienemmistä yksiköitä. Lohkoja voidaan siis pitää pieninä verkkoina, joiden solmujen aktivaatiofunktiot ovat derivoituvia kuten kappaleessa 4.2.3



Kuva 4.8: Yksinkertaisen rekursiivisen verkon purkaminen viiteen aika-askeleeseen.

nähtiin. Tästä syystä niiden opetuksessa voidaan käyttää edellä kuvattua backpropagation algoritmia.

Käytetään siis erästä backpropagation-algoritmin muunnosta, niin kutsuttua katkaistua BPTT:tä (engl. *truncated BPTT*)(katso lisätietoja Bodén oppaasta [34]). Katkaistu BPTT tarkoittaa LSTM-verkon kannalta sitä, että virhettä ei kuljeteta lohkon sisääntuloista (syöte-, syöteportti-, unohtamisportti- ja tulosporttiyksiköt) edellisen lohkon ulostuloon. Lohkon tuloksen aiheuttamaa virhettä käytetään kyllä näiden painokertoimien muuttamiseen. Lohkon sisällä CEC kuljettaa virhettä ajassa taaksepäin tarvittaessa rajoittamattoman ajan kuten kappaleessa 4.2.6 kerrottiin. Katso lisätietoja Sepp Hochreiterin ja Jürgen Schmidhuberin tutkimuksesta [30].

Sepp Hochreiter ja Jürgen Schmidhuber ovat kehittäneet LSTM-verkoille tehokkaamman, katkaistun BPTT:n ja RTRL:n (engl. *Real-Time Recurrent Learning*) yhdistävä, opetusmenetelmän. Merkittävin ero tämän ja katkaistun BPTT:n välillä on se, että verkolle ei opetettaessa tarvitse laskea virhettä jokaiselle aika-askeleella. Tämä johtuu siitä, että painojen muuttamiseen tarvittavat osittaisderivaatat lasketaan aina eteenpäin-syklin aikana. Osittaisderivaatat ilmaisevat, miten paljon kyseinen liitäntä (painokerroin) on osallistunut CEC:n arvon, eli solun tilan, muodostamiseen. Siten millä tahansa ajan hetkellä laskettuun virheeseen perustuva painokertoimien korjaus ottaa huomioon painokertoimen ”historian”. Virhettä ei siis tarvitse laskea jokaiselle askeleelle kuten normaalin backpropagation-algoritmin tapauksessa [30].

BPTT on mielestäni kuitenkin parempi valinta yleistettävyytensä vuoksi. Hochreiterin ja Schmidhuberin kehittämä tehokkaampi LSTM-opetusalgoritmi vaatii nimittäin, että kappaleessa 4.2.6 mainitun CEC:n tila tiedetään syöte-, syöteportti-, unohtamisporttiyksiköissä. Tilaa tarvitaan laskettaessa kyseisten yksiköiden edellä mainittuja osittaisderivaattoja. Halusin kuitenkin luoda järjestelmän, jossa koko neuroverkko LSTM-lohkoista lähtien on muokattavissa. Backpropagation-algoritmissa verkon yksiköt voidaan käsitellä topologisessa järjestyksessä yksitellen, ja painokertoimen laskemiseksi tarvitaan ainoastaan painokertoimen yhdistämien yk-

siköiden tila. Verkon täydellinen muokattavuus yhdistettynä kappaleessa mainittuihin 4.4 geneettisiin menetelmiin saattaa tuottaa todella tehokkaita verkkoja. Justin Bayer et al. päätyi käyttämään menestyksekkäästi samaa menetelmää vastaavassa järjestelmässään yrittäessään luoda uudenlaisia versioita LSTM-lohkoista [35].

Sovellan edellä saatuja tuloksia suoraan liitteestä D.2 löytyvässä taaksepäin suuntautuvassa algoritmissa (katso algoritmit 5 ja 7).

Opetukseen liittyen on paikallan mainita, että opetukseen tarvitaan kahdenlaista opetusdataa: normaalia liikennettä sekä näytteitä hyökkäyksistä. Normaalia liikennettä käytetään opetettaessa verkkoja tunnistamaan normaali toiminta. Hyökkäysnäytteitä voidaan käyttää opetettaessa verkkoja tunnistamaan tiettyjä hyökkäystyyppejä.

4.4 Neuroverkon alustaminen

Ennen opettamisen aloittamista verkko täytyy alustaa johonkin alkutilaan. Tämä tarkoittaa sitä, että verkon neuronien välisille yhteyksille annetaan jokin alkuarvo. Lisäksi tietyn tyyppisten verkkojen kohdalla voidaan määritellä verkolle topologia.

Verkon alkuarvojen valinnalla on ratkaiseva vaikutus siihen miten hyvin verkko tulee annetusta tehtävästään suoriutumaan, eli miten hyvin se oppii sille opetetut hahmot. Huonosti valitut alkuarvot voivat johtaa siihen ettei verkko opi annettuja hahmoja järkevässä ajassa ollenkaan. Lisäksi esimerkiksi LSTM-verkon topologia (kappale 4.2.6) vaikuttaa verkon soveltuvuuteen tietyn tyyppisten ongelmien ratkaisemisessa, kuten Tiflin ja Omlin tutkimuksessaan toteavat [36]. Ongelmana kuitenkin on, ettei optimaalisia painokertoimia ja topologiaa useinkaan tunneta. Tämä korostuu erityisesti poikkeamaperusteista järjestelmää suunniteltaessa, koska siinä verkko pyritään muodostamaan siten, että se tunnistaisi ennalta tuntemattomia hahmoja.

Uuden verkon painokertoimien alustamisessa voidaan käyttää useita tapoja. Hyvin yleinen ja yksinkertainen tapa alustaa verkko on valita neuronien väliset painokertoimet sattumanvaraisesti. Toinen varteenotettava vaihtoehto on käyttää geneettistä algoritmia verkon alustamiseen. Tällöin verkot paranevat sukupolvi sukupolvelta ja oppivat ratkaistavan ongelman paremmin, kuten Tiflin ja Omlin [36] sekä Salakoski [37] tutkimuksisaan esittävät. Pelkän keinotekoisen evoluutionkaan, ilman mukautuvia neuroverkkoja, soveltaminen hyökkäysten havaitsemiseen ei ole uutta, kuten Zhou tutkimuksessaan esittää [38]. Bayer et al. on tutkinut LSTM-verkkojen evoluutionaalista alustamista ja luomista [35]. Hänen mukaansa LSTM-lohkosta voi geneettisen algoritmin avustuksella muodostua hyvin erilaisia ja erilai-

siin tehtäviin soveltuvia muunnoksia.

Tarkasteltaessa näitä kahta mallia, satunnaista ja geneettistä alustamista, on lähestymistapa opetettaessa useita verkkoja samanaikaisesti melko samankaltainen. Aluksi luodaan useita uusia verkkoja, joita opetetaan joukkona ja valitaan lopuksi parhaiten menestyneet käyttöön. Menetelmien ero tulee esiin ensimmäisessä vaiheessa, jossa satunnaisessa mallissa mitään tietoa edellisten sukupolvien menestymisestä ei käytetä, kun taas geneettisessä mallissa voidaan valita edellisen sukupolven parhaat ja luoda uusi sukupolvi näitä risteyttämällä. Satunnaisen menetelmän hyvänä puolena on yksinkertaisuus, kun taas huonona puolena voi nähdä ongelman ratkaisun kannalta käyttökelpoisen tiedon käyttämättä jättämisen. Geneettisen mallin huonona puolena taas voi nähdä suhteellisen monimutkaisuuden satunnaiseen malliin verrattuna. Geneettisen mallin hyvänä puolena voidaan nähdä se, että edellisen sukupolven tehokkuudesta saatua tietoa voidaan käyttää seuraavan sukupolven parantamiseen, jolloin järjestelmä paranee kokonaisuudessaan askel askeleelta. Geneettisellä algoritmilla voidaan vaikuttaa esimerkiksi siihen, miten nopeasti verkot käsiteltävän ongelman oppivat. Tämä voidaan saavuttaa asettamalla arviointiperusteet siten, että seuraavan sukupolven ”vanhemmiksi” valitaan sellaisia verkkoja, jotka opetusvaiheessa oppivat ongelman nopeiten. Tämä perustuu siihen, että verkkoon voi ajan ja sukupolvien mittaan kehittyä hyvinkin monimutkaisia rakenteita, jotka erikoistuvat tietyn tyyppisten hahmojen tunnistamiseen.

Edellä kuvatut menetelmät voitaisiin myös yhdistää siten, että verkon perusrakenne luodaan geneettisesti, mutta sitä täydennetään vielä satunnaisesti luoduilla yhteyksillä. Satunnaiset yhteydet voisivat olla hieman heikoimpia kuin geneettisesti luodut. Tällä tavoin verkoihin voisi geneettisesti kehittyä tiettyjä, jokaisessa samaan tarkoitukseen käytetyssä järjestelmässä tarvittavia, peruominaisuuksia. Verkot osaisivat nämä taidot jo ennen opettamista, ja niitä vain täydennettäisiin ja hienosäädettäisiin opettamalla. Tätä voidaan jopa verrata biologisten neuroverkkojen perittyihin ja opittuihin ominaisuuksiin.

4.5 Piirteet

Jotta neuroverkkoa voidaan käyttää, täytyy sille olla jokin syöte. Neuroverkon syöte esitetään tyypillisesti vektorina, jonka sisältää kohdejärjestelmästä tuotettuja numeroarvoja. Nämä numeroarvot kuvaavat järjestelmästä saatavia erilaisia muuttuvia tietoja ja niitä kutsutaan piirteiksi (engl. *features*). Näiden piirteiden valintaa ja tuottamista kaikesta järjestelmästä saatavasta tiedosta kutsutaan piirteiden valitsemiseksi (engl. *feature selection*) tai erottamiseksi (engl. *feature extraction*). Molemmilla

termeillä viitataan tekniikoihin, joilla edellä mainitun syötteen ulottuvuuksia pyritään vähentämään erilaisin keinoin (esimerkkejä Addison et al. tutkimuksessa [39]), mutta ensimmäinen viittaa enemmän tekniikoihin, joilla piirteistä valitaan oleellisimmat ja jälkimmäinen tekniikoihin joilla matemaattisesti pienennetään piirrevektorin dimensioita.

Edellisen lisäksi piirrevektorille täytyy tehdä arvoalueiden normalisointi ja mahdollinen koodaus.

Kaikkea edellä mainittua kutsutaan Bishopin mukaan tiedon esikäsittelyksi (engl. *pre-prosessing*) [23, luku 8].

Piirteiden esikäsittelyn käytännön toteutusta käsitellään kappaleessa 5.2.1.

4.5.1 Piirteiden erottaminen

Järjestelmästä saatua tietoa voidaan käyttää sellaisenaan, mutta usein tämä ei ole järkevää vaan tiedosta täytyy erottaa käyttökelpoisempia piirteitä. Esimerkkinä piirteiden erottamisesta voidaan ottaa vaikkapa hahmon tunnistus bittikarttakuvasta: esimerkiksi yleisesti käytetyn 24 bittisen väriavaruuden omaavasta 100*100 pikselin kokoisesta bittikarttakuvasta saadaan 3*100*100=30000 kappaletta kahdeksanbittisiä lukuja (katso bittikarttakuvan formaatti Bourken sivustolta [40]). Tällaisen piirremäärän käsitteleminen neuroverkolla ei ole järkevää jo pelkästään tarvittavan neuroverkon koon takia. Neuroverkon koko kasvaa piirteiden lisääntyessä koska jokainen piirre vaatii oman syöteneuroninsa, kuten kappaleessa 4.2.4 kerrotaan. Piirteiden erottaminen tässä tapauksessa voisi tarkoittaa esimerkiksi kuvioiden reunaviivojen hakemista jollakin metodilla ja näiden muuttamista numeeriseen muotoon.

IDS:n kannalta ehkä tärkein piiteiden erottamismenetelmä on pakettien tilastointi: voidaan esimerkiksi laskea sekunnin aikana lähetettyjen tai vastaanotettujen tietyn tyyppisten tai tietyn ominaisuuden omaavien pakettien määrä.

4.5.2 Piirteiden valinta

Piirteiden erottamisen jälkeen on yleensä syytä tehdä vielä piirteiden valinta, jotta järjestelmän suorituskyvyn kannalta tarpeettomat tai jopa haitalliset piirteet voidaan jättää pois prosessoinnista ja käyttää ainoastaan tarpeellisia.

Piirteiden valinta yksinään on haastava ja keskeinen osa-alue sovellettaessa neuroverkkoja. Liian suuri tai huonosti valittu piirrevalikoima voi kostautua verkon hitaana oppimisena, tuhlattuina resursseina tai huonona onnistumisasteena. Tästä syystä oikeiden piirteiden valinta on koko järjestelmän suorituskyvyn kannalta erittäin tärkeää, kuten Nambiar et al. [41] ja Zhang et al. [1, sivu 550] tutkimuksissaan

toteavat.

Langattoman verkon tapauksessa mahdollisesti hyödynnettävien piirteiden joukko vaihtelee itse langattoman verkon MAC-tason paketeista saatavista tiedoista (katso kappale 2.1.1), paketeista kerättyyn tilastotietoon sekä ylemmän tason protokollista erotettuun vastaaviin tietoihin.

Zhong et al. on tutkimuksessaan listannut [42, taulukko 2] mahdollisesti käyttökelpoisia mitattavia ominaisuuksia, joista Athira.M.Nambiar et al. on edelleen työssään [41, taulukko 1] valinnut kymmenen langattomien IDS:en kannalta käyttökelpoisinta piirteinä. Nämä piirteet on kuvattuna taulukossa 4.1. Jotkin mainituista piireistä ovat luonteeltaan tilastollisia (katso Balazinskan ja Castron tutkimus [43]). Mainituissa tutkimuksissa käytetty aineisto on kerätty 177:ltä Cisco Aironet 350 -tukiasemalta kerran viidessä minuutissa käyttäen SNMP-protokollaa ja se on ladattavissa internetistä [1].

Guennoun et al. on omassa tutkimuksessaan [19] yhdistänyt DARPA99(KDD 99) datasetistä erotettuja (tilastollisia ja ylemmän tason protokollien) piirteitä (katso kuvaus Stolfo sivustolta [44]) sekä edellä mainittuja MAC-tason paketeista erotettuja piirteitä [19, taulukko 1] ja näistä sekä MAC-paketeista prosessoimalla johdetuista piirteistä [19, taulukko 2] IGR (engl. *information gain ratio*) menetelmällä valinnut kymmenen parasta piirrettä [19, taulukko 3]. Edelleen K-means luokittelijalla kokeilemalla on etsitty optimaalinen määrä piirteitä. Optimaaliseksi määräksi osoittautui kahdeksan. Kaikki kymmenen piirrettä on listattuna selityksineen taulukossa 4.2. Piirteet ovat listassa käyttökelpoisuutensa mukaan siten, että käyttökelpoisin on listassa ylimmäisenä ja optimaalisen joukon muodostavat kahdeksan käyttökelpoisinta piirrettä.

Guennoun et al. tutkimuksesta nähdään, että MAC-kehyksistä prosessoimalla saadut piirteet (IsWepValid, DurationRange ja Casting) näyttävät soveltuvan parhaiten langattoman verkon tunkeutumisen havaitsemiseen. Toiseksi parhaiten näyttävät soveltuvan suoraan MAC-kehyksestä erotetut piirteet (More, WEP, Type, SubType, Retry ja From). Muut suoraan MAC-kehyksestä ja DARPA99 datasetistä erotetut piirteet eivät valikoituneet kärkikymmenikköön. Tästä voidaan päätellä, että MAC-kehyksistä prosessoimalla saatavien piirteiden piirteiden kehittelyyn kannattaisi panostaa lisää aikaa.

Toinen Guennoun et al. tutkimukseen liittyvä jatkokehityskohde liittyy tutkimuksessa käytettyyn IEEE 802.11 -standardin versioon: tutkimuksessa käytetty versio on IEEE 802.11-1999[12]. Tällä hetkellä uusin versio on IEEE Std 802.11-2007[8],

[1] http://nms.csail.mit.edu/ mbalazin/wireless/

Piirre	Selitys
ShortRet	Kaikista tälle päätelaitteelle suuntautuneista lähetysyrityksistä aiheutuneiden lyhyiden uusintalähetysten (engl. *short retries*) (RTS-tyyppisten) kokonaismäärä.
LongRet	Kaikista tälle päätelaitteelle suuntautuneista lähetysyrityksistä aiheutuneiden pitkien uusintalähetysten (engl. *long retries*) (data-tyyppisten) kokonaismäärä.
Quality	Laiteriippuvainen arvo, joka ilmaisee tältä päätelaitteelta viimeksi vastaanotetun paketin signaalin laadun.
Strength	Laiteriippuvainen arvo, joka ilmaisee tältä päätelaitteelta viimeksi vastaanotetun paketin signaalin voimakkuuden. Voi olla normalisoitu tai normalisoimaton.
srcPkts	Niiden havaittujen pakettien määrä, joiden lähettäjänä tämä päätelaite on.
srcErrPkts	Niiden havaittujen viallisten pakettien määrä, joiden lähettäjänä tämä päätelaite on.
dstErrPkts	Niiden havaittujen viallisten pakettien määrä, joiden vastaanottajana tämä päätelaite on.
destMaxRetryErr	Niiden havaittujen viallisten max-retry pakettien määrä, joiden lähettäjänä tämä päätelaite on.
dstPkts	Niiden havaittujen pakettien määrä, joiden vastaanottajana tämä päätelaite on.
srcErrOct	Niiden havaittujen viallisten oktettien määrä, joiden lähettäjänä tämä päätelaite on.

Taulukko 4.1: Nambiar et al. mukaan parhaat piirteet.

Piirre	Selitys
IsWepValid	Ilmaisee onko WEP:n eheystarkistus läpäisty.
DurationRange	Ilmaisee onko duration-arvo alhainen (<5ms), keskitasoa (5-20ms) vai korkea (>20ms).
More_Frag	Ilmaisee onko kehys viimeinen fragmentti vai ei.
To_DS	Ilmaisee onko kehys menossa DS:ään.
WEP	Ilmaisee onko kehys salattu vai ei.
Casting_Type	Ilmaisee onko vastaanotto-osoite unicast, multicast vai broadcast -tyyppinen.
Type	Ilmaisee kehyksen tyypin (Management, Control tai Data).
SubType	Ilmaisee kyhyksen alityypin.
Retry	Ilmaisee onko kehys uudellenlähetetty.
From_DS	Ilmaisee onko kehys tulossa DS:stä.

Taulukko 4.2: Guennoun et al. mukaan parhaat piirteet.

johon on julkaistu useita laajennoksia [2]. Vanhentunut standardiversio on mielenkiintoinen, koska uusi standardi on tuonut mukanaan uusia kenttiä MAC-kehykseen 2.1.1. Esimerkiksi QoS-kenttä voisi olla hyödyllinen lisä piirteiden joukossa.

Tässä työssä valittiin Guennoun et al. esittämät kahdeksan optimaalista piirrettä niiden helpommasta tuotettavuudesta johtuen. Nämä piirteet on mahdollista erottaa missä tahansa järjestelmässä kunhan kappaleessa 3.4.2 mainittu ehto IEEE 802.11 -otsikkotietojen saatavuudesta täytyy. Nambiar et al. esittämät piirteet taas vaativat järjestelmältä tietyn tyyppisiä tukiasemia, koska osa niistä on kerätty Cisco Aironet Access Point MIB:stä (engl. *Management Information Base*) [43]. Kaikki tukiasemat eivät tue tiedon keräämiseen vaadittua SNMP-protokollaa.

On mahdollista, että yhdistämällä molempia edellä mainittuja piirrejoukkoja olisi mahdollista saavuttaa vielä parempia tuloksia. Lisäksi on huomattava, että nyt käytetty luokittelija on eri kuin Guennoun et al. työssään [19] käyttämä eivätkä saavutetut tulokset siksi välttämättä täysin ennusta nyt toteutettavan järjestelmän suorituskykyä. Nyt toteutettavalle järjestelmälle tulisi suorittaa vastaavan tyyppinen piirteiden valintaprosessi tai kehittää valintaprosessia automaattisemmaksi. Automaattisessa piirteiden valinnassa voitaisiin käyttää oppimista: järjestelmälle tarjottaisiin kaikki mahdolliset piirteet, joista turhilta vaikuttavat karsiutuisivat käyttämättöminä pois kuten kappaleessa 4.6.1 kuvataan. Toinen vaihtoehto voisi olla hyö-

[2] http://standards.ieee.org/about/get/802/802.11.html

dyllisten piirteiden hakeminen geneettisellä algoritmilla.

Toinen valittuihin piirteisiin liittyvä huomioitava seikka on tehokkaasti tunnistettavat hyökkäystyypit. Valitut piirteet vaikuttavat järjestelmän kykyyn tunnistaa erilaisia hyökkäystyyppejä kuten kappaleessa 2.3 kerrotaan.

4.5.3 Piirteiden normalisointi

Kun valitut piirteet on erotettu järjestelmän tarjoamasta tiedosta täytyy ne saattaa neuroverkon käsittelemään muotoon. Tämä tarkoittaa yleensä piirteiden normalisointia. Normalisoinnissa piirteiden madollisesti toisistaan eroavat arvoalueet saatetaan yhteneväisiksi keskenään ja neuroverkon kanssa. Valittu arvoalue riippuu neuroverkon syöteneuronien arvoalueesta, eli käytännössä valitusta aktivaatiofunktiosta. Jatkuvan arvoalueen piirteiden lisäksi myös epäjatkuvat ja binääriset piirteet koodataan samalle arvoalueelle (katso lisää Bishopin kirjasta [23, sivut 298–300]).

Jatkuvat muuttujat voidaan skaalata samalle arvoalueelle useammalla eri tavalla: arvot voidaan skaalata niiden keskiarvon ja varianssin (Bishop [23, sivu 298]), tai arvoalueen perusteella (Klevecka ja Lelis [45, sivu 171]).

Valitsemani piirteet eivät ole luonteeltaan ”luonnollisia piirteitä”, kuten lämpötila tai paine, vaan binäärisiä tai tavun mittaisia lukuja kuten kappaleessa 4.5.2 kerrottiin. Tästä johtuen katson, että jatkuvat arvot kannattaa skaalata pelkästään arvoalueensa perusteella käyttäen kaavaa 4.32.

$$\overline{x}_t = \frac{(x_t - x_{t,min})(\overline{x}_{t,max} - \overline{x}_{t,min})}{(x_{t,max} - x_{t,min})} + \overline{x}_{t,min} \tag{4.32}$$

Kaavassa 4.32 $x_{t,min}$ ja $x_{t,max}$ ovat skaalaamattoman muuttujan x_t minimi- ja maksimiarvot. $\overline{x}_{t,min}$ ja $\overline{x}_{t,max}$ ovat aktivaatiofunktiosta riippuvat skaalatun muuttujan $\overline{x}_t$ minimi- ja maksimiarvot [45, sivu 171].

Binääriset muuttuja skaalataan siten, että tosi arvo saa aktivaatiofunktion maksimi- ja epätosi miniarvon.

4.5.4 Puuttuvat arvot

Piirrevektorista puuttuvat arvot voivat aiheuttaa ongelmia varsinkin tapauksissa, joissa ainoastaan osa arvoista puuttuu. Tämä voi Bishopin mukaan aiheuttaa tulosten vääristymistä [23, sivu 301].

Tämän työn kohdalla voisi olettaa piirrevektorien olevan aina täydellisiä: jos paketti saadaan kaapattua pystytään kaikki sen sisältämä tieto lukemaan kokonaan tai ei ollenkaan. Tästä syystä voisi edelleen olettaa, että puuttuvien arvojen käsittelyyn

ei tarvitse kiinnittää huomiota. Tämä on totta, mutta ainoastaan kappaleessa 4.5.2 valittujen piirteiden ansiosta!

Kuten kappaleessa 4.5.2 mainittiin, valittiin käytettäviksi piirteiksi Guennoun et al. valikoimat 8 piirrettä. Guennoun et al. ei kuitenkaan mainitse työssään [19] miten kehyksistä puuttuvat kentätä on huomioitu piireitä erotettaessa. Kuten kappaleessa 2.1.1 mainitaan, eivät kaikki kehykset sisällä kaikkia kenttiä. Onneksi kappaleessa 4.5.2 mainittujen kymmenen käyttökelpoisimman piirteen joukkoon ei valikoitunut yhtäkään tällaista piirrettä. Voimme vain arvailla johtuuko tämä ehkä huonosti valitusta tavasta käsitellä puuttuvia arvoja.

Toinen syy voi olla se, että pitkät MAC-kehyksen kentät (MAC address, FCS ja Duration) käsiteltiin useampana kahdeksanbittisenä piirteenä eikä kokonaisuuksina. Mutta kuten kappaleessa 4.5.2 mainitsin, valittujen piirteiden jatkojalostus voisi olla hyödyllistä. Piirteiden valinta on kuitenkin jo yksinään niin iso osa-alue ettei siihen ole mahdollisuuksia panostaa tämän työn puitteissa.

4.6 Keinotekoisten neuroverkkojen haasteet

Keinotekoisten neuroverkkojen soveltaminen käytäntöön ei ole täysin ongelmatonta. Niihin sisältyy muutamia haittatekijöitä, jotka täytyy järjestelmää suunniteltaessa ottaa huomioon.

4.6.1 Rajalliset resurssit

Eräs neuroverkkojen soveltamista rajoittava tekijä on vuosien saatossa ollut niiden suuri resurssien tarve: keinotekoiset neuroverkot vaativat erityisen paljon laskentatehoa. Resurssien tarve korostuu entisestään mobiilissa ympäristössä, joten siihen täytyy kiinnittää erityistä huomiota.

Eräs keino vähentää laskentatehon tarvetta verkon mobiileissa laitteissa on suorittaa verkkojen opettaminen tehokkaammissa koneissa. Tällöin vähemmän suorituskykyisissä laitteissa verkkoja ajetaan ainoastaan tunnistustarkoituksessa. Tällaista opettamista kutsutaan off-line -opettamiseksi.

Usein keinotekoinen neuroverkko toteutetaan siten, että verkon kerrokset ovat täysin kytkettyjä. Tämä tarkoittaa sitä, että kerroksen jokainen neuroni on kytketty kaikkiin muihin saman kerroksen neuroneihin. Tällöin jokainen verkon liitos lasketaan jokaisella kierroksella sen painokertoimesta huolimatta. Tämä kuluttaa laskentatehoa, joka varsinkin mobiililaitteiden näkökulmasta on yksi kriittisistä resursseista 3.4.2.

Verkon toimintaa voidaan Strömin mukaan tehostaa karsimalla heikoimmat kytkennät pois opettamisen jälkeen ennen verkon ottamista aktiiviseen käyttöön [46]. Näin ilmeisesti tapahtuu myös ihmisen aivoissa teini-iässä, kun aivot "valmistuvat" käytettäväksi itsenäisesti ilman aikuisten ohjausta [47]. Samaa periaatetta soveltamalla voidaan keinotekoisesta neuroverkosta tehdä harva versio, jolloin tarvittavan laskennan määrä vähenee. Aluksi verkko kannattaa olla tiheä, jolloin verkon neuroneilla on enemmän mahdollisuuksia havaita toisten neuronien samanaikainen aktivoituminen ja verkolla on paljon vaihtoehtoisia kehityssuuntia. Jos opetus suoritetaan off-line -opetuksena tehokkaassa ympäristössä ei ylimääräisistä yhteyksistä ei ole kovin suurta haittaa. Oppimisen edetessä useat yhteydet käyvät tarpeettomiksi ja niiden painokerroin lähestyy nollaa. Tällaisia yhteyksiä ei kannata laskea koska niiden vaikutus lopputulokseen on häviävän pieni. Kun karsiminen tehdään vähitellen, voivat jäljelle jääneet yhteydet korvata poistetut yhteydet mukautumalla uuteen tilanteeseen. Karsiminen suoritetaan ennen verkon siirtämistä aktiiviseen käyttöön. Jos karsinta tehdään kerralla on vaarana ettei karsitun verkon vaste vastaa lähellekään alkuperäisen karsimattoman verkon vastetta. Siksi myös verkkojen validointi täytyy tehdä karsimisen päätyttyä.

Kappaleessa 4.4 mainituilla geneettisellä algoritmilla voidaan mahdollisesti saada aikaan kevyempiä verkkoja. Tämä edellyttää, että arvioinnissa käytetyssä niin kutsutussa hyvyysfunktiossa (engl. *fitness-function*) otetaan yhtenä kriteerinä huomioon myös verkon vaatima laskentakapasiteetti.

Kuten kappaleessa 4.5.2 mainitaan, voitaisiin geneettisiä algoritmejä ehkä soveltaa myös optimaalisen piirrejoukon valintaan. Aluksi järjestelmässä voisivat olla tarjolla kaikki mahdolliset erotettavissa olevat piirteet, mutta verkot käyttäisivät niistä vain osaa. Geneettinen algoritmi aiheuttaisi vahitelua valittuihin piirteisiin. Tällöin hyvyysfunktiossa täytyisi huomioida myös käytettyjen piirteiden määrä verrattuna verkon suorituskykyyn, eli tunnistus tarkkuuteen. Tällöin piirteet, jotka eivät paranna verkon tunnistusominaisuuksia karsiutuvat ennenpitkään pois ja järjestelmä toimii tehokkaammin.

Optimoitu tehokkuus ei ole tärkeää pelkästään mobiililaitteissa, vaan siitä on hyötyä myös kiinteissä laitteissa ajettaessa useita verkkoja rinnakkain. Varsinkin suuressa mittakaavassa tehonkulutus nousee merkittäväksi tekijäksi.

4.6.2 Optimaalisen verkkomallin puute

Eräs yleisesti neuroverkkoihin liittyvä haaste on optimaalisen verkkomallin puute. Kullekkin ratkaistavalle ongelmalle täytyisi löytää juuri oikeanlainen verkkomalli.

Hyökkäysten havaitsemisjärjestelmän käsittelemä aineisto on täysin erilaista kuin esimerkiksi kuvantunnistusjärjestelmän. Liun mukaan verkon solmujen määrä riippuu tutkittavasta aineistosta [48]. Liian yksinkertainen verkko ei ole tarpeeksi mukautuvainen eikä pysty oppimaan syötteen pienimpiä yksityiskohtia. Liian monimutkainen verkko taas mukautuu liian hyvin syötteen pienimpiin vaihteluihin, josta seuraa niin kutsuttua ylioppimista (engl. *over fitting*), kuten Bishop kirjassaan kertoo [23, sivut 371–372].

Tätä haastetta voidaan lähestyä kappaleessa 4.4 mainittujen geneettisten algoritmien kautta. Algoritmeilla voidaan luoda verkkoja, joiden topologia sopii mahdollisimman hyvin ratkaistavaan ongelmaan.

Optimaalisten neuroverkkomallien etsimisestä voi lukea lisää Bebis ja Georgiopoulosin [49] sekä Wang et al. [50] tutkimuksista.

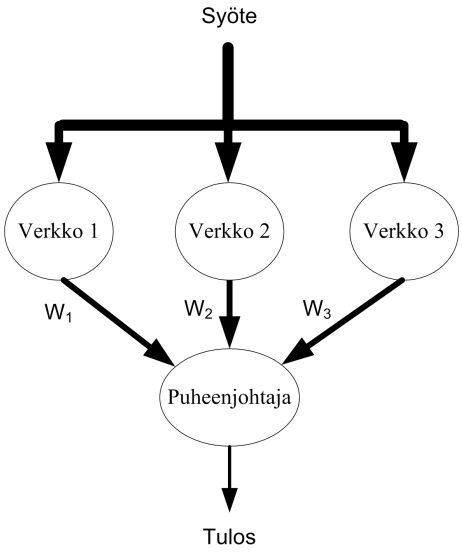
4.7 Keinotekoisten neuroverkkojen hajauttaminen

Eräs keino kappaleessa 4.6.1 esitettyjen haasteiden hallintaan voisi olla neuroverkkojen hajauttaminen. Tällöin neuroverkkoja voitaisiin ajaa laitteilla, joilla on riittävästi resursseja.

Toinen haaste, jonka käsittelemisessä hajauttamisesta voisi olla hyötyä, on kappaleessa 3.4 mainittujen väärien positiivisten ja negatiivisten tunnistusten minimoiminen käyttäen kappaleissa 4.7.1 ja 4.7.2 kuvattuja tekniikoita.

4.7.1 Verkkojen komitea

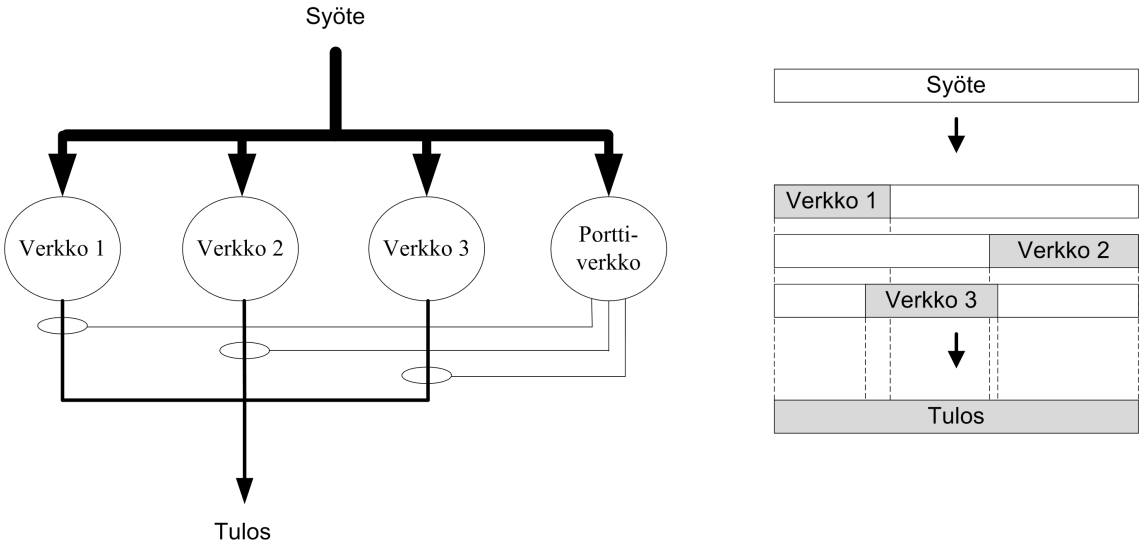
Hajauttamisella voidaan vähentää virheiden määrää. Tämä perustuu äänestämiseen, joka toimii eräänlaisena verkkojen ehdotusten keskiarvoistamisena. Tällaista Bishopin kirjassaan kuvaamaa menetelmää kutsutaan verkkojen komiteaksi (engl. *committee of networks* tai *committee of machines*) [23, sivut 364–369]. Komitean jäsenet antavat äänensä puheenjohtajalle, joka näiden äänien perusteella laskee äänestyksen tuloksen (painotetun keskiarvon), ja antaa lopullisen päätöksen. Tässäkin tapauksessa pätevät solmujen väliset painot siten, että luotettavat solmut saavat päätöksessä suuremman painoarvon, kun taas epäluotettavat ja uudet solmut osallistuvat ainoastaan pienellä painolla. Jos uusi solmu ei eroa kovin paljon yleisestä linjasta sen paino arvo nousee ja se todetaan luotettavammaksi. Esimerkki verkkojen komiteasta kuvassa 4.9.



Kuva 4.9: Verkkojen komitea.

4.7.2 Asiantuntijaverkko

Asiantuntijaverkko (engl. *mixture of experts*) on eräänlainen hajautettu, modulaarinen verkko. Bishopin mukaan ajatuksena on, että ongelma voidaan jakaa useampaan osa-alueeseen, joiden ratkaiseminen yksinään on helpompaa. Asiantuntijaverkossa ongelma ratkaistaan siis useamman eri osa-alueeseen erikoistuneen verkon (engl. *expert network*) yhteistyönä. Myös tämä malli perustuu kappaleessa 4.7.1 mainitun verkkojen komitean tapaan kilpailuun. Siispä tässäkin mallissa tarvitaan ulkoinen taho, joka päättää kilpailun voittajan. Tämä taho voi olla niin kutsuttu porttiverkko (engl. *gating network*), joka syötteen perusteella oppii mikä verkko ratkaisee minkäkin tyyppisen osaongelman parhaiten ja tämän perusteella antaa vastausvuoron aina oikealle verkolle [23, sivut 369–371]. Esimerkki asiantuntijaverkosta kuvassa 4.10.



Kuva 4.10: Experttiverkko ja miten lopullinen tulos muodostuu verkkojen yhteistyönä.

5 Järjestelmän arkkitehtuuri

Tässä luvussa kuvataan luvuissa 3 ja 4 esitetyn teoriataustan perusteella suunniteltu tunkeutumisen havaitsemisjärjestelmä ja sen arkkitehtuuri.

5.1 Vaatimukset

Ennen arkkitehtuurin suunnittelemista asetetaan sille joitakin vaatimuksia. Vaatimuslista voidaan laatia kappaleessa 3.4 esitettyjen langattomien verkkojen ja niiden tunkeutumisen havaitsemisjärjestelmien haasteiden ja reunaehtojen perusteella. Näitä vaatimuksia ovat esimerkiksi seuraavat:

- Hyökkäysten havaitseminen tulee toteuttaa keinotekoisella neuroverkolla.
- Järjestelmän tulee olla hajautettu siten, että eri laitteet kommunikoivat keskenään.
- Järjestelmän täytyy toimia langattomassa ympäristössä.
- Järjestelmän täytyy toimia ainakin kahdessa erilaisessa ympäristössä. Langattomiin verkkoihin yhdistetyissä laitteissa mahdollisia ympäristöjä ovat esimerkiksi seuraavat:
  - Windows (erityyppiset laitteet), [1]
  - Linux (erityyppiset laitteet), [2]
  - MeeGo (ajoneuvot, matkapuhelimet, tablettietokoneet ja muut medialaitteet), [3]
  - Symbian (matkapuhelimet), [4]
  - Android (tablettietokoneet, matkapuhelimet ja PC-tietokoneet) ja [5]
  - Applen iOS (tablettietokoneet, matkapuhelimet ja mediasoittimet). [6]

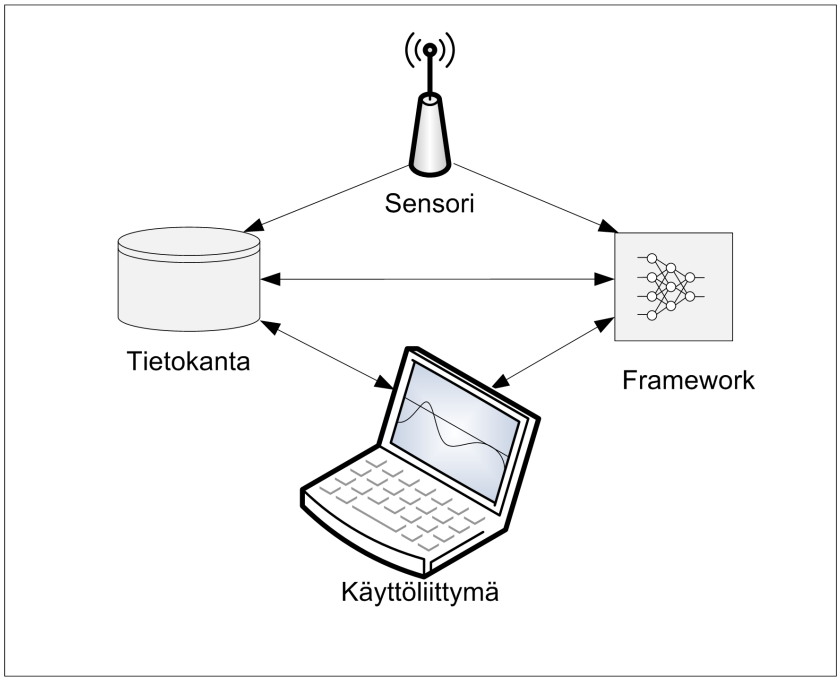
[1] http://www.windowsfordevices.com
[2] http://www.linuxfordevices.com
[3] http://meego.com/devices
[4] http://www.allaboutsymbian.com/devices
[5] http://android-devices.net
[6] http://developer.apple.com/technologies/ios

- Järjestelmän täytyy säästää energiaa mobiileissa laitteissa.

5.2 Järjestelmän osat

Tyypillinen tunkeutumisen havaitsemisjärjestelmä sisältää Scarfonen ja Peter Mellin mukaan neljä pääkomponenttia: sensori tai agentti, hallintapalvelin (engl. *management server*), tietokantapalvelin (engl. *Database Server*) ja käyttöliittymä (engl. *Console*). Sensori tai agentti on tarkoitettu tiedon keräämiseen järjestelmästä ja sen analysoimiseen. Ylläpitopalvelin on keskitetty laite, jonka tarkoituksena on kerätä sensorien ja agenttien lähettämää tietoa. Sitä käytetään myös sensorien ja agenttien hallintaan sekä sellaisten toimintojen toteuttamiseen, joita sensorit tai agentit eivät pysty itse suorittamaan. Tietokantapalvelinta käytetään järjestelmän tuottaman tiedon, kuten järjestelmän tilamuutosten, tallentamiseen. Käyttöliittymän avulla ylläpitäjä ja käyttäjä voivat kommunikoida järjestelmän kanssa [14].

Tässä työssä suunniteltu arkkitehtuuri koostuu hieman vastaavista komponenteista: sensori, framework, tietokanta sekä käyttöliittymä. Kuvassa 5.1 kuvataan kyseiset järjestelmät ja niiden väliset loogiset yhteydet. Kuvassa jokaisesta komponentista on ainoastaan yksi ilmentymä. Todellisuudessa tarkoituksena olisi kutenkin luoda järjestelmä, jossa komponenttien määrää ei ole rajoitettu.



Kuva 5.1: Järjestelmän osat.

5.2.1 Sensori

802.11 -verkon fyysisen kerroksen paketteja kuunnellaan ja kaapataan edelleen lähetettäviksi sensoreissa. Jotkin verkon laitteista toimivat sensoreina. Paketit täytyy kaapata tarkoitukseen soveltuvalla verkkosovittimella, joka pystyy toimimaan niin sanotussa promiscuous-tilassa erotettaessa piirteitä 802.11 -pakettien hyötykuormasta, ja monitor-tilassa erotettaessa piirteitä pakettien 802.11 -otsikoista. Ensimmäisessä tilassa sovitin normaalista poiketen vastaanottaa kaiken verkossa havaitsemansa liikenteen ja välittää sen kaappausohjelmalle. Myös jälkimmäisessä tilassa sovitin vastaanottaa kaiken havaitsemansa liikenteen, mutta välittää sen käsittelemättömässä muodossa juuri sellaisena kuin se ilmarajapinnassa 802.11 -otsikkotietoineen ja mahdollisesti salattuna liikkuu. Normaalissa toimintotilassa ainoastaan verkkosovittimelle itselleen tarkoitettu liikenne huomioidaan: salaus puretaan, 802.11 -otsikkotiedot poistetaan ja paketti välitetään eteenpäin.

Koska langattoman lähiverkon sovittimet ovat käytännössä radiolähettimiä/-vastaanottimia täytyy jokaista valvottavaa kanavaa kohti olla oma sovitin.

Kuten kappaleessa 4.5.1 kerrottiin, pitää verkkoliikenteestä erottaa piirteitä. Tämä tapahtuu osana sensorin toimintaa. Sensorin kaappaamista paketeista muodostetaan piirrevektoreita. Piirrevektorit normalisoidaan ja lähetetään eteenpäin niiden tilaajille.

5.2.2 Framework

Framework on komponentti, joka toimii neuroverkkojen suoritusalustana ja mahdollistaa yhtenäisellä kuvauskielellä kuvattujen neuroverkkojen suorittamisen ja opettamisen kappaleessa 5.1 kuvatuissa erilaisissa ympäristöissä. Framework voidaan ajatella järjestelmän tärkeimpänä komponenttina, johon muut komponentit liittyvät kappaleessa 5.5.7 kuvatulla protokollalla.

Frameworkista kannattaa pyrkiä tekemään mahdollisimman yleiskäyttöinen, jolloin sitä voidaan uudelleen käyttää myös muunlaisissa kuin tässä työssä kuvatussa tunkeutumisen tunnistusjärjestelmässä.

Frameworkin tarkempi toiminnallinen kuvaus löytyy kappaleesta 5.5.1.

5.2.3 Tietokanta

Järjestelmässä tarvitaan komponentti tiedon tallentamiseen. Tämä komponentti on esitetty kuvassa 5.1 nimellä ”tietokanta” (engl. *database*). Tietokanta tässä tapauksessa tarkoittaa mitä tahansa keinoa tallentaa järjestelmän toiminnan kannalta tärkeää

tietoa. Tämä tieto voi olla esimerkiksi:

- verkosta kaapattua liikennettä,
- tietoa hyökkäyksistä sekä
- itse neuroverkot.

Verkosta kaapattua liikennettä käytetään verkkojen opetuksessa, kuten kappaleessa 4.3 kerrotaan. Lisäksi liikennettä käytetään liikenteen arviointiin käyttäjän toimesta. Käyttäjä tai verkon ylläpitäjä voi hälytyksen saatuaan tutkia liikennettä ja arvioida onko hälytys aiheellinen vai ei. Tästä syystä tallennetun liikenteen täytyy sisältää itse piirteet, aika sekä jokin tunniste lähteelle tai sensorille.

Järjestelmän tulee tallentaa myös tietoa hyökkäyksistä. Tämän tiedon täytyy sisältää kellonajat ja tunnisteet lähteille tai sensoreille. Tällöin hyökkäykset voidaan liittää tiettyyn järjestelmän, ja sitä kautta liikenteen, osaan.

Itse neuroverkkoja täytyy pystyä tallentamaan esimerkiksi virtakatkosten aikana. Toisaalta tallentamista vaaditaan jos verkkoja halutaan kappaleen 5.5.9 tapaan siirtää järjestelmän osasta tai järjestelmästä toiseen.

5.2.4 Käyttöliittymä

Käyttöliittymä (engl. *console*) toimii järjestelmän rajapintana käyttäjän suuntaan.

Käyttöliittymä voisi näyttää mittarilla tai liikennevaloilla vallitsevan tilanteen. Esimerkiksi vihreä väri tarkoittaa normaalia tilannetta, keltainen epävarmaa tilannetta ja punainen varmaa hyökkäystä. Myös jonkinlainen trendi-käyrä voisi olla hyödyllinen.

Järjestelmän ylläpitäjällä tai edistyneellä käyttäjällä voisi olla erilainen, laajennettu käyttöliittymä. Tästä laajemmasta käyttöliittymästä voisi olla olla pääsy hyökkäyksestä tallennetun tiedon selaamiseen. Epäselvissä tilanteissa ylläpitäjän tai käyttäjän olisi tehtävä päätös hyökkäyksen olemassaolosta. Edistyneessä käyttöliittymässä voisi olla myös mahdollisuus verkkojen hallinnointiin: uuden verkon lisääminen, verkkojen poistaminen ja muita ylläpitotoimenpiteitä.

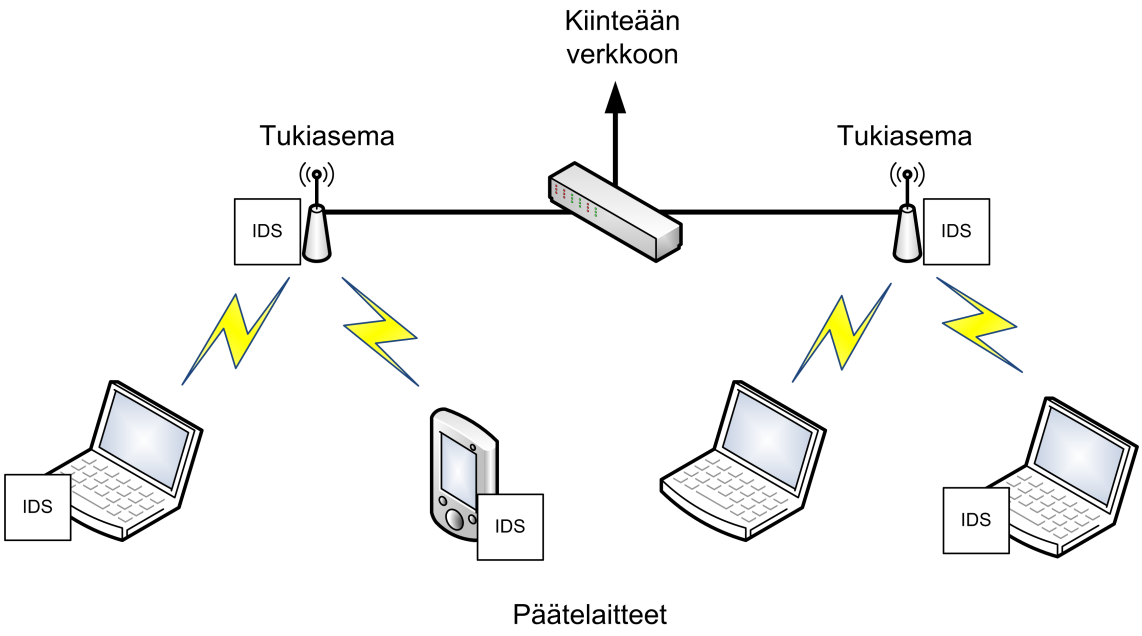
5.3 Järjestelmän hajauttaminen

Yhtenä johtavana ajatuksena järjestelmää suunniteltaessa on pidetty sen hajauttamista verkon (langattoman ja kiinteän) laitteille.

Koska pyrkimyksenä on, että mahdollisimman moni langattoman verkon laite toimii osana IDS:ää, on se tällöin hajautettu jossakin muodossa lähes kaikille verkon laitteille. Tämä ei tarkoita sitä, että laite ei voisi liittyä verkkoon, jos se ei ole osa IDS:ää, mutta siinä tapauksessa laitteen oma suojaus jää puutteelliseksi eikä se lisää järjestelmän kokonaisturvallisuutta.

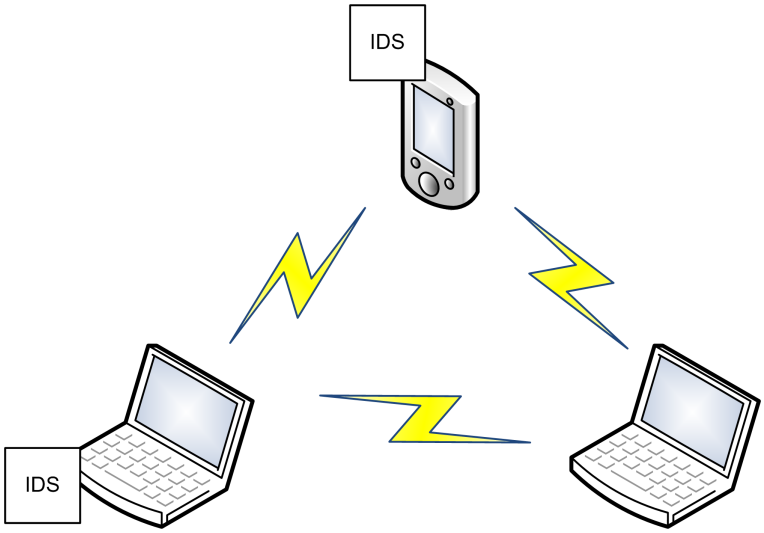
Hajauttaminen toteutetaan siten, että jokaisella verkon laitteella toimii yksi tai useampia kappaleessa 5.2 kuvatuista komponenteista. Erilaisille laitteille asennettavat komponentit kannattaa valita siten, että laitteen resurssit käytetään oikein. Laitteiden välisestä työnjaosta lisää kappaleessa 5.4.

Kuvassa 5.2 esitetään infrastruktuurityyppinen esimerkkiverkko, jossa langattomat laitteet voivat liittyä verkkoon kahden tukiaseman kautta. Kuvan verkon kolmella päätelaitteella ja molemmilla tukiasemilla toimivat IDS-agentit, jotka kuuntelevat liikennettä.



Kuva 5.2: Esimerkki infrastruktuuriverkosta.

Kuvassa 5.3 esitetään toinen, Ad hoc -tyyppinen, WLAN-verkko. Ad hoc -tyyppisessä verkossa päätelaitteet muodostavat verkon ilman tukiasemia ja kommunikoivat suoraan toistensa kanssa. Tällaisessa verkossa IDS-agentit toimivat päätelaitteilla.



Kuva 5.3: Esimerkki Ad hoc -verkosta.

5.4 Työnjako

Hajautettua järjestelmää ja sen arkkitehtuuria suunniteltaessa tulee miettiä järjestelmän komponenttien sijoittelemista. Järjestelmän kannalta voi olla hyödyllistä jakaa järjestelmän erilaisia komponentteja järjestelmän eri laitteille. Kaikkia komponentteja ei välttämättä ole järkevää suorittaa jokaisessa laitteessa. Suoritettava komponentti kannattaa valita laitteen resurssien ja käyttötarkoituksen mukaan.

Esimerkiksi kuvan 5.2 infrastruktuurityyppisessä verkossa framework voitaisiin suorittaa jollakin kiinteään verkkoon liitetyllä palvelimella. Tällä tavoin voitaisiin säästää mobiililaitteiden resursseja ja energiaa. Erityisesti uusien neuroverkkojen opettaminen kannattaa ehdottomasti suorittaa erillisellä palvelimella, jolla tämänkaltainen intensiivinen laskentakapasiteettia vaativa toimenpide ei rasita mobiililaitteen vähäisiä resursseja.

Toisaalta sensori voisi olla järkevää liittää osaksi tukiasemaa, jolloin saavutettaisiin laaja peitto ja nopea yhteys mahdollisesti samassa kiinteässä verkossa sijaitsevaan frameworkiin. Lisäksi jokainen laite ei voi toimia sensorina jo siitä syystä, että kaikki verkkosovittimet eivät kykene kaappaamaan muille tarkoitettua liikennettä ja/tai ne eivät tarjoa rajapintaa raa'an, ilmarajapinnassa kulkevan liikenteen, vastaanottamiseen.

Myös tietokanta kannattaisi infrastruktuurityyppisessä verkossa sijoittaa kiinteän verkon palvelimelle. Mobiililaitteiden tallennuskapasiteetti on usein rajallinen verrattuna kiinteän verkon laitteisiin.

Käyttöliittymä kannattaa luonnollisesti sijoittaa laitteelle, jossa on jokin käyttä-

järajapinta, esimerkiksi näyttö ja näppäimistö. Käyttöliittymää ei luonnollisestikaan kannata sijoittaa tukiasemaan, mutta mobiililaitteeseen sijoitettu käyttöliittymä toimittaa ilmoituksen mahdollisesta tunkeutumisyrityksestä suoraan käyttäjälle. Parhaassa mahdollisessa tilanteessa järjestelmä osaa suodattaa eri laitteille juuri kyseistä laitetta koskevat varoitukset.

Kuvan 5.3 ad hoc -tyyppinen verkko on työnjaon kannalta infrastruktuurityyppistä verkkoa huomattavasti haastavampi tapaus. Verkossa ei ole mitään keskitettyä, pysyvää laitetta, jonka saavutettavuuteen muut verkon laitteet voisivat luottaa. Verkko saattaa pienimmillään koostua yhdestä laitteesta.

Miten sitten yhdistää näiden kahden erilaisen verkkotyypin vaatimukset samaan järjestelmään? Ongelmaa kannattanee lähestyä ensin edellä mainitun pienimmän mahdollisen verkon näkökulmasta. Jotta verkon ainoa laite voidaan suojata, täytyy järjestelmän jokaisen komponentin toimia kyseisellä laitteella. Sama lähestymistapa pätee kaikille ad hoc -tyyppisille verkoille, joiden konfiguraatio saattaa nimensä mukaisesi varoittamatta muuttua laitteiden liittyessä siihen tai poistuessa siitä. Tästä johtuen järjestelmän eri komponentteja kannattaa suorittaa jokaisella laitteella, jotta jonkin laitteen poistuessa verkosta ei äkkiarvaamatta menetetä jotakin arvokasta resurssia, kuten esimerkiksi tietokantaa tai sensoria.

Toisaalta, kuten kappaleessa 4.6.1 mainittiin, kannattaa mobiilien laitteiden kohdalla pyrkiä mahdollisimman vähäiseen resurssien kulutukseen. Siispä järjestelmään tarvitaan jonkin tyyppinen mahdollisuus säätää käytettävien resurssien määrää. Yksinkertaisimmillaan resurssien säätämien voisi tapahtua manuaalisesti asettamalla jotkin prosentuaaliset rajat eri resurssien kulutukselle. Monimutkaisempi vaihtoehto on säätää järjestelmän käyttöön annettavia resursseja automaattisesti olosuhteiden mukaan jonkin algoritmin ohjaamana.

Resursseja voitaisiin säästää esimerkiksi edellä mainitulla käsiteltävien pakettien suodattamisella, jolloin ainoastaan laitteelle tarkoitetut paketit käsiteltäisiin. Tallennustilaa taas voitaisiin säästää pitämällä tallennetun liikenteen määrä mahdollisimman pienenä. Energiaa säästyy kytkemällä järjestelmä käytöstä verkkoyhteyden ollessa poissa käytöstä.

Tällaiset ad hoc -verkkoon suunnitellut itsenäiset järjestelmät sopivat käytettäväksi myös infrastruktuuriverkossa. Vaikka jokainen laite tutkisi ainoastaan itseensä suuntautuvaa liikennettä täydentävät yksittäiset laitteet toisiaan niin, että tuloksena on verkon täydellinen peitto niiltä osin kun järjestelmä laitteille on asennettu. Verkkoon liittyvät laitteet jotka eivät sisällä järjestelmän osia eivät tässä tapauksessa hyödy järjestelmästä. Siispä edellä esitetty tukiasemaan sijoitettu sensori täydentää verkon peittoa myös näiden laitteiden osalta.

Kuvattu eri komponenttien yhteistoiminta täytyy tapahtua automaattisesti ilman käyttäjän toimenpiteitä. Järjestelmän eri osien täytyy tunnistaa ja liittyä toisiinsa ilman käyttäjän apua. Kuvaan erityisesti kappaleissa 5.5.6 ja 5.5.7 ehdottamani automaattisen komponenttien välisen yhteistoiminnan yksityiskohtia.

5.5 Järjestelmän rakenne ja toiminta

Kuvissa 4.2 ja 4.1 voidaan nähdä samankaltaisuutta. Molemmat voidaan ajatella verkon solmuina. Solmuilla taas on sisäänmeneviä syötteitä ja ulostulevia tuloksia. Näillä yhteyksillä on myös painokertoimet kuten kappaleissa 4.2.1 ja 4.7.1 mainittiin.

Edellä esitetyn perusteella hahmottelin mallin järjestelmästä, joka koostuu sisäkkäisistä rakenteista. Nämä rakenteet ovat pohjimmiltaan samanlaisia tasosta riippumatta: solmuja ja niitä yhdistäviä painotettuja yhteyksiä.

Kuvassa 5.4 nähdään yleiskuva järjestelmän sisäkkäisistä rakenteista.

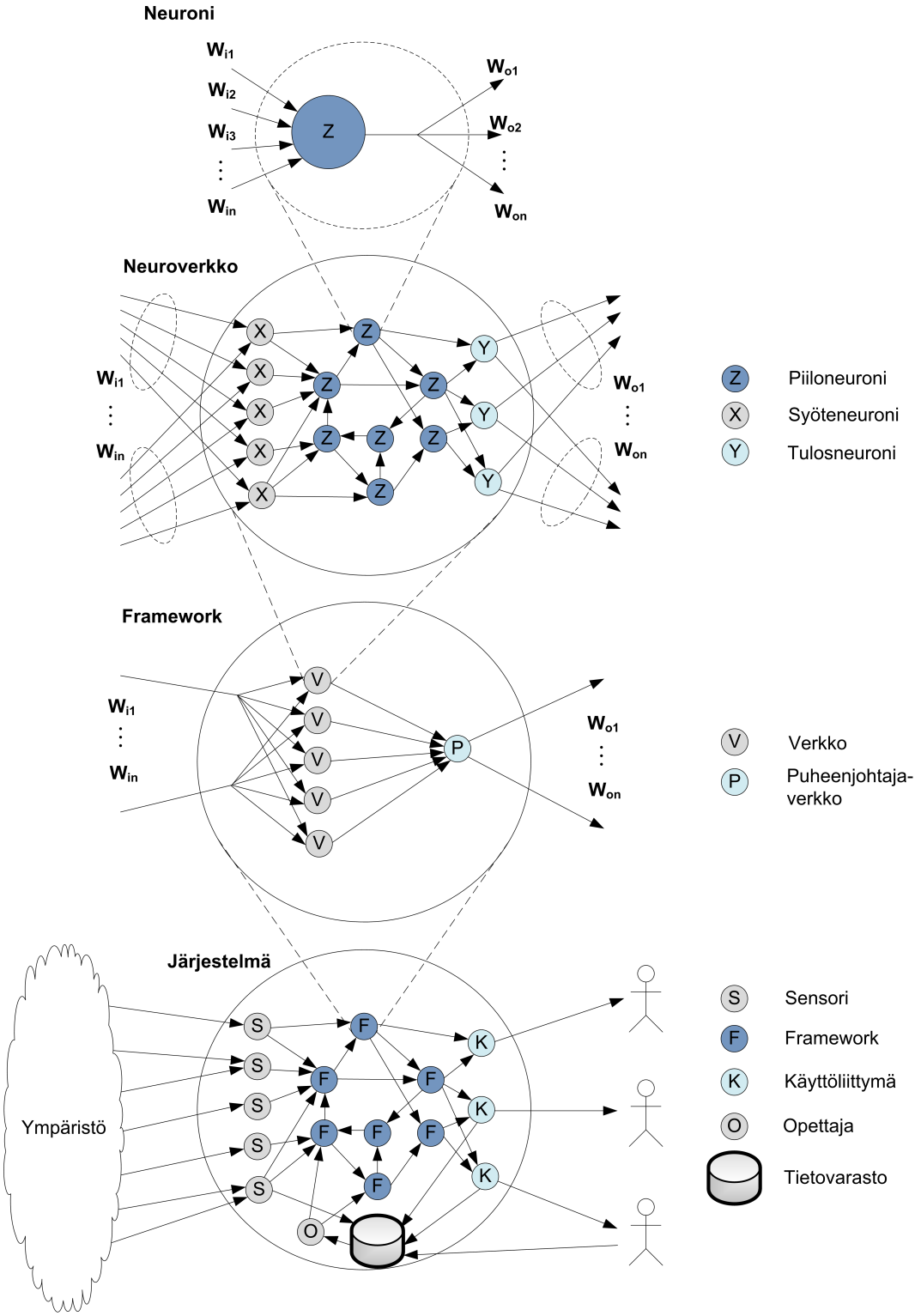
5.5.1 Frameworkin toiminta

Framework on geneerinen suoritusympäristö (engl. *framework*) keinotekoisten neuroverkkojen suorittamiseen. Frameworkin tarkoituksena on mahdollistaa XML-kuvauskielellä kuvattujen verkkojen käyttäminen, opettaminen ja tallentaminen.

Framework koostuu useista, mahdollisesti sisäkkäisistä ryhmistä. Tällä hetkellä framework voi sisältää tavallisia ryhmiä ja yhden erityisen ”external”-nimisen ryhmän. Normaaleilla ryhmillä voidaan ryhmitellä verkkoja käyttäjän haluamalla tavalla. Varsinainen hyöty tästä saavutetaan määriteltäessä erilaisia näkyvyyksiä eri ryhmille: ryhmälle voidaan visible -ominaisuudella kertoa minkä ryhmän, oman tai toisen, verkot voivat muodostaa yhteyksiä ryhmän sisällä oleviin verkkoihin. Ilman näkyvyysrajoituksia kaikkien verkkojen kaikki syöte ja tulos -neuronit yritetään yhdistää kappaleessa 5.5.2 kuvatulla tavalla.

Erityinen ”external”-ryhmä sisältää framewokin ulkopuolisia verkkoja edustavia virtuaalisia verkkoja. Virtuaalisten verkkojen toimintaa kuvataan tarkemmin kappaleessa 5.5.6.

Näiden verkkotyyppien lisäksi tarvittaisiin mahdollisesti vielä yksi ryhmä sellaisten verkkojen tallentamiseen, joiden tarkoituksena olisi näkyä muille frameworkeille ja mahdollistaa kommunikaatio eri frameworkien välillä. Tähän ryhmään lisättyjä verkkoja mainostettaisiin toisille frameworkeille ja niiden virtuaalinen vastine ilmestyisi frameworkin tavoitettavissa olevien muiden frameworkien ”external”-



Kuva 5.4: Järjestelmän sisäkkäiset rakenteet.

listalle muiden ulkoisten verkkojen tapaan. Tällainen ryhmä esitellään kuvassa 5.12 nimellä ”share”.

Framework on periaatteessa tilakone, jonka tilat voidaan esittää kuvan 5.6 tilakaaviona. Framework on suunniteltu alunperin toimimaan kappaleessa 4.3.3 kuvatun backprobacation menetelmän mukaisesti. Tästä syystä tilakaavio jakautuu kahteen päätilaan: ”tunnistus” ja ”opetus”. Käytetyn backprobacation algoritmin eteenpäin suuntautuva osa löytyy liittestä D.1 ja taaksepäin suuntautuva osa liitteestä D.2.

Usein verkot tallennetaan tietokoneen muistiin matriisimuodossa siten, että yksi matriisi kuvaa yhtä verkon kerrosta ja yksi matriisin sarake yhtä kerroksen neuronia. Matriisin alkiot sisältävät yhteyksien painokertoimet. Tämä kuvaustapa toimii erityisen tehokkaasti täysin kytkettyjen verkkojen kanssa. Sen on yleensä myös hyvin tehokas ajatellen algoritmien suorittamista.

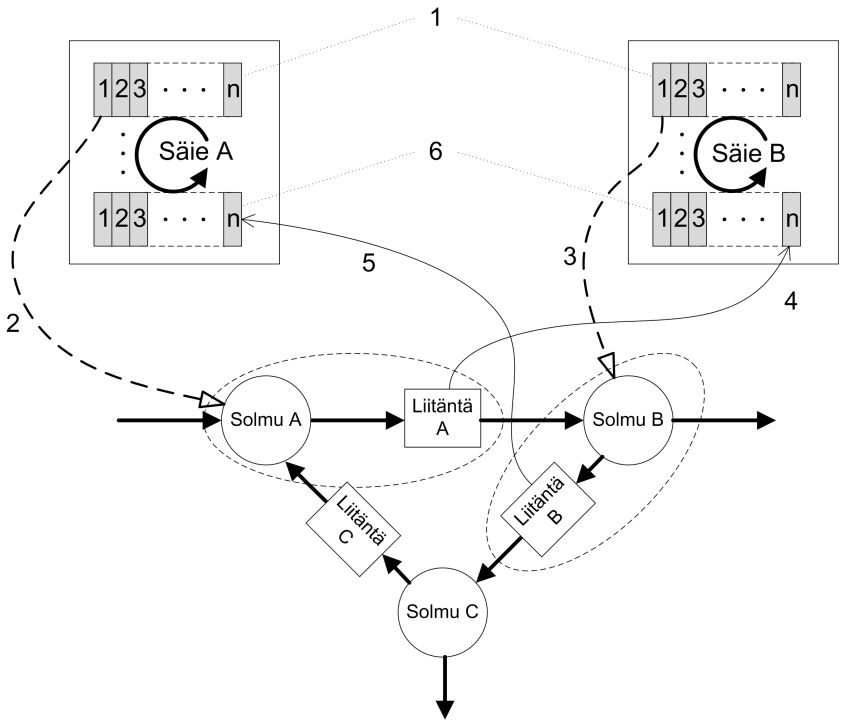
Päätin kuitenkin toteuttaa järjestelmäni mahdollisimman olio-orientoituneesti, ja määrittelin jokaisen synapsin yhdeksi olioksi jolla on linkki sekä syöttävään että syötettävään neuroniin. Lisäksi synapsi pitää sisällään painoarvon ja muuta tarpeellista tietoa. Tällaiseen lähestymistapaan päädyin lähinnä siitä syystä, että halusin välttää turhaa tilanvaraamista jota syntyy kappaleessa 4.6.1 kuvattujen niin kutsututtujen karsittujen, osittain kytkettyjen, verkkojen tallentamisesta matriisiin. LSTM-verkon lohkot eivät ole täysin kytkettyjä.

Toinen etu tällaisesta toteutuksesta oli se, että pystyin jakamaan algoritmin suorituksen liitäntöjen ja solmujen välillä kahteen osaan. Tämä on erityisen hyödyllinen ominaisuus mietittäessä moniprosessorisia ympäristöjä ja poissulkemisongelmaa. Frameworkin suoritus on nimittäin säikeistetty, ja jaettu tasan yhtä monen säikeen kesken kuin suorittavassa ympäristössä on prosessoriytimiä. Verkkojen solmut, eli neuronit, on jaettu tasan kaikkien säikeiden kesken. Ongelmaksi muodostuvat tällaisessa tapauksessa erityisesti juuri painokertoimet, joita esimerkiksi opetustilanteessa saattaa toinen säie käyttää laskentaan kuin toinen muuttaa sen suuruutta. Toinen ongelma syntyy neuronien summaa laskettaessa: kaikkien syötteiden täytyy olla valmiina ennen kuin summa voidaan laskea. Tästä syystä sekä suoritus, että tiedon päivittäminen on jaettu kahteen osaa. Solmut suoritetaan aina saman, verkkoa lisättäessä määrätyn, säikeen toimesta. Solmuja yhdistävän liitännän suorittava säie saattaa muuttua, jos liitännän alku solmu ja loppusolmu suoritetaan eri säikeissä. Tällöin liitäntää suorittava säie on sama kuin päivityksen vastaanottava solmu. Toisin sanoen eteenpäin syklissä liitännän koodin suorittaa syötettävän solmun säie ja taaksepäin syklissä syöttävän solmun säie. Tämä johtuu neuronin syötteen ja virheen summausfunktioista, jotka on toteutettu liitännän koodissa. Tällä tavoin voi-

daan taata ilman poissulkemista, että monta liitäntää ei voi samaan aikaan lisätä solmun nettosyötettä tai nettovirhettä koska saman säikeen sisällä suoritusjärjestys on varmasti peräkkäinen. Tämä lisää tehokkuutta.

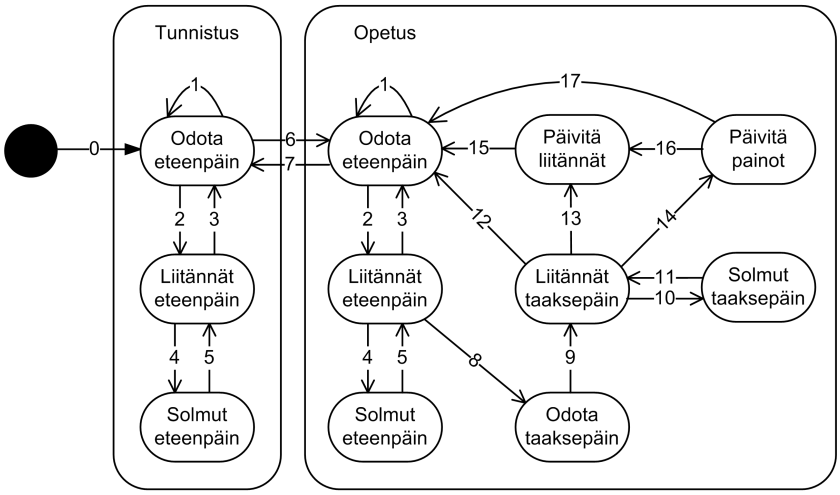
Säikeillä on erityiset työjonot, joihin toiset säikeet voivat lisätä päivittämiään solmuja ja liitäntöjä odottamaan suorittamistaan. Työjonoja on yhteensä seitsemän: eteenpäin-syklin solmuille, taaksepäin-syklin solmuille, eteenpäin-syklin liitännöille, taaksepäin-syklin liitännöille, eteenpäin-syklin viivästetyille liitännöille, taaksepäin-syklin viivästetyille liitännöille ja kertoimien päivittämistä odottaville liitännöille. Jokainen säie käy oman, frameworkin tilasta riippuvan, työjononsa niin nopeasti läpi kuin mahdollista. Samalla se täyttää omaa ja toisten säikeiden työjonoja, jotka vastaavasti suoritetaan kun frameworkin tila on sopiva.

Kuvassa 5.5 esimerkki kahden säikeen järjestelmästä, jossa säie A ”omistaa” solmut A ja C ja säie B solmun B. Aluksi molemmat säikeet (säie A ja B) ottavat ensimmäisen solmun eteenpäin-syklin työjonostaan (numero 1). Säikeet suorittavat solmujen A ja B eteenpäin-syklin koodia (numerot 2 ja 3). Somu A lisää liitännän A säikeen B työjonoon viimeiseksi (numero 4) ja solmu B lisää liitännän B säikeen A työjonoon (numero 5) viimeiseksi. Kun molempien säikeiden somuja sisältävä työjono (numero 1) on käsitelty siirtyvät molemmat säikeet suorittamaan liitäntöjä sisätävää työjonoaan (numero 6).



Kuva 5.5: Esimerkki kahdesta säikeestä lisäämässä liitäntöjä toistensa työjonoon.

Käydään kuvan 5.6 tilakaavio läpi vaihe vaiheelta. Normaalissa käytössä framework toimii ”tunnistus”-tilassa ja tietoa siirretään ainoastaan eteenpäin syöttäviltä solmuilta syötettäville solmuille. Framework käynnistyy (0) aina tähän tilaan ja jää odottamaan ”forward”-sanomia verkon ulkopuolelta (kappale 5.5.7). Verkon ei kannata tehdä mitään ilman ulkopuolista syötettä, joten tässä tilassa odotetaan kunnes joku ulkopuolinen taho lähettää kyseisen sanoman, tai joku muu, verkkoihin liittyvä sanoma vastaanotetaan ja käsitellään (1). Jos framework vastaanottaa yhden tai useamman ”forward”-sanoman kirjoittaa se sanoman sisältämän tiedon liitäntää vastaaviin virtuaalisiin synapseihin (kappale 5.5.6), lisää liitännän siihen liittyvän syötettävän neuronin ”omistavan” säikeen työjonoon ja siirtyy sitten suorittamaan eteenpäin suuntautuvia liitäntöjä (2). Liitännän suorittaminen tarkoittaa liitännän sisältämän syötteen lisäämistä syötettävän solmun nettosyötteeseen ja solmun lisäämistä työjonoon. Kun kaikki työjonoissa odottaneet liitännät on suoritettu siirtyy framework uuteen tilaan (4). Tässä tilassa solmujen nettosyöte on selvillä ja säikeet suorittavat solmujen tyypistä riippuvan aktivaatiofunktion. Aktivaatiofunktion tulos, joka siis on solmun tulos, päivitetään kaikkiin lähteviin liitäntöihin, jonka jälkeen ne siirretään työjonoon odottamaan suorittamistaan. Kun kaikki odottavat solmut on käsitelty ja syötteet päivitetty liitäntöihin voidaan siirtyä takaisin suorittamaan liitäntöjä (5). Liitäntöjä ja solmuja suoritetaan vuorotellen niin kauan kunnes viimeinenkin liitäntä on suoritettu eikä työjonoissa ole enää muuta suoritettavaa. Viimeinen suoritettava liitäntä on virtuaalinen synapsi (kappale 5.5.6) joka lähettää ”forward”-sanoman seuraavalle komponentille. Tämän jälkeen framework siirtyy takaisin odottamaan vastaanotettuja ”forward”-sanomia (3).



Kuva 5.6: Verkon tilakaavio.

Verkkoja opetettaessa framework toimii ”opetus”-tilassa. Opetus tila koostuu,

kuten edellä mainittiin, kahdesta eri osasta: eteenpäin ja taaksepäin suuntautuvasta syklistä. Eteenpäin suuntautuva sykli on täysin samanlainen kuin edellä kuvatussa ”tunnistus”-tilassa. Ainoa ero on, että viimeisen eteenpäin suuntautuvan liitännän käsiteltyään framework siirtyy odottamaan ”backward”-sanomia samoista lähteistä joihin se hetkeä aikaisemmin lähetti ”forward”-sanoman (8). Tämä lähde on luultavimmin kappaleessa 5.5.3 kuvattu ulkoinen opettaja, joka laskee ”forward”-sanoman virheen ja palauttaa sen ”backward”-sanomassa. Hieman kuten eteenpäinsyklissä, kun framework vastaanottaa yhden tai useamman ”backward” sanoman kirjoittaa se sanoman sisältämän tiedon liitäntää vastaaviin virtuaalisiin synapseihin (kappale 5.5.6), lisää liitännän siihen liittyvän syötävän neuronin ”omistavan” säikeen työjonoon ja siirtyy sitten suorittamaan taaksepäin suuntautuvia liitäntöjä (9). Liitännän suorittaminen tarkoittaa liitännän sisältämän virheen lisäämistä syöttävän solmun nettovirheeseen ja solmun lisäämistä työjonoon. Kun kaikki työjonoissa odottaneet liitännät on suoritettu siirtyy framework uuteen tilaan (10). Tässä tilassa solmujen nettovirhe on selvillä ja säikeet suorittavat solmujen tyypistä riippuvan aktivaatiofunktion derivaatan laskemisen. Aktivaatiofunktion derivaatan tulos, joka siis on solmun lopputulokseen aiheuttama virhe, päivitetään kaikkiin tuleviin liitäntöihin, jonka jälkeen ne siirretään työjonoon odottamaan suorittamistaan. Kun kaikki odottavat solmut on käsitelty ja virheet päivitetty liitäntöihin voidaan siirtyä takaisin suorittamaan liitäntöjä (11). Liitäntöjä ja solmuja suoritetaan jälleen vuorotellen niin kauan kunnes viimeinenkin liitäntä on suoritettu eikä työjonoissa ole enää muuta suoritettavaa. Viimeinen suoritettava liitäntä on virtuaalinen synapsi (kappale 5.5.6) joka lähettää ”backward”-sanoman edelliselle komponentille. Tämän jälkeen framework tutkii, onko kertoimien päivittämistä odottavien liitäntöjen työjonossa suoritettavaa. Jos on, siirrytään laskemaan ja päivittämään odottavien liitäntöjen kertoimet (14). Tämän jälkeen framework siirtyy takaisin odottamaan vastaanotettuja ”forward”-sanomia (17). Jos kertoimien päivittämistä odottavien liitäntöjen ei ollut, siirtyy framework suoraan takaisin odottamaan vastaanotettuja ”forward”-sanomia (12).

Kuvassa 5.6 on ylimääräinen ”Päivitä liitännät” -tila, jota ei edellisessä kuvauksessa käsitelty. Tämä tila on varattu mahdolliseen verkon topologian päivittämiseen. Täysin kytketyssä verkossa topologian päivittämistä ei tarvita, koska kaikki solmut ovat aina yhteydessä toisiinsa ja ainoastaan painojen päivittäminen riittää. Osittain kytketyssä verkossa kytkentöjen muuttaminen jonkin algoritmin perusteella voi olla osa opetusvaihetta. Kyseisen algoritmin kehittäminen rajataan kuitenkin tämän työn ulkopuolelle. Kyseistä vaihetta voisi verrata esimerkiksi nisäkkäiden aivoissa unen aikana tapahtuvaan toimintaan, jossa aivot järjestäytyvät uudelleen osana

oppimisprosessia, kuten Wilhelm artikkelissaan kertoo [51].

Edellä kuvattu työjonojen täyttäminen dynaamisesti ajon aikana ei välttämättä ole tehokkain tapa, mutta monikäyttöisyyden kannalta katsottuna se puolustaa paikkaansa. Ajatellaan esimerkiksi tapausta, jossa aktivaatiofunktiona on kynnysfunktio (kappale 4.2.2). Kynnysfunktio voi toimia siten, että seuraavaa neuronia ei aktivoida, jos kynnysarvo ei ylity. Tässä tapauksessa staattinen työlista ei toimi, koska neuroni löytyy työjonosta riippumatta siitä, onko se aktivoitu vai ei. Tietysti voidaan lisätä esimerkiksi lippu, joka kertoo aktivaation tilasta. Tässäkin tapauksessa neuronin koodi joudutaan kuitenkin suorittamaan aina ainakin osittain, ja osa saavutetusta hyödystä menetetään. Staattinen työlista voitaisiin luoda samalla kun verkko käydään läpi solmujen suoritusjärjestyksen selvittämiseksi kappaleessa 5.5.4 kuvatulla tavalla.

5.5.2 Neuroverkko

Verkko koostuu frameworkin tapaan ryhmistä. Verkossa voi olla kaksi erityistä ulkoiseen kommunikaatioon tarkoitettua ryhmää: ”input” ja ”output”. Nämä ryhmät sisältävät verkon syöte ja tulos -neuronit, ja näin ollen määrittelevät verkon ulkoisen rajapinnan. Tämä tarkoittaa sitä, että ”input”-ryhmän neuronien nimet kertovat suoraan, mitä piirteitä verkko käsittelee. Vastaavasti ”output” -ryhmän neuronien nimet kertovat minkä nimisiä syötteitä verkko tarjoaa. Framework kytkee näiden ryhmien neuronit automaattisesti kaikkien muiden verkkojen vastaaviin ryhmiin uutta verkkoa lisättäessä. Liittäminen tapahtuu nimien perusteella siten, että toisen verkon ”output”-ryhmän verkosta muodostetaan synaptinen yhteys toisen verkon saman nimiseen ”input”-ryhmän neuroniin. Näiden ryhmien neuronien nimet siis vastaavat eri piirteiden nimiä. Tästä syystä piirteille pitää kehittää yksilöllisiä nimiä, jotka pysyvät aina samoina jos kyseessä on sama piirre. Esimerkiksi liitteessä B esitetyssä esimerkkiverkossa erään syöteneuronin nimi on ”Type”. Vastaavasti sensoria vastaavassa virtuaalisessa verkossa on tulosneuroni nimeltään ”Type”. Nämä neuronit kytketään aina yhteen frameworkin tasolla. Jokainen järjestelmän komponentti olettaa, että ”Type” niminen neuroni (tai piirre) ilmoittaa kaapatun paketin tyypin kuten kappaleessa 4.5.2 kerrotaan.

Näiden erityisryhmien lisäksi verkko voi sisältää lukuisia piilokerroksia tai -ryhmiä. Piilokerrosten neuronit eivät näy ulospäin ja voivat liittyä ainoastaan oman tai toisen piilokerroksen, ”input” tai ”output”-ryhmän neuroniin näkyvyyssääntöjen salliessa. Sisäiset neuronit voivat olla joko ennalta tai dynaamisesti ajon aikana nimettyjä.

Ryhmä vastaa tässä tapauksessa siis monikerroksisen verkon kerrosta, mutta on monikäyttöisempi. Ryhmillä voidaan ryhmitellä neuroneita, kuten kuvattaessa kappaleessa 4.2.6 esiteltyä LSTM-verkon lohkoa. Liitteessä B nähdään ryhmä, jonka nimeksi on annettu ”LSTM”. Tämä ryhmä kuvaa LSTM-lohkoista muodostuvaa piilokerrosta. Kyseisen ryhmän sisältä löytyy kolme aliryhmää: ”Block1”, ”Block2” ja ”Block3”. Nämä ryhmät kuvaavat kukin yhden LSTM-lohkon.

Tässä työssä käytettiin kuvan 6.3 mukaista testiverkkoa. Testiverkko on samanlainen kuin Gers et. al. tutkimuksessaan [32] käyttämä.

Syöteneuronin aktivaatiofunktiona käytettiin hyperbolista tangenttia. Porttien aktivaatiofunktiona käytettiin logistista sigmoidi-funktiota. Lisää aktivaatiofunktioista kappaleessa 4.2.2. Portit ja tulosneuronit biasoitiin kytkemällä ne erityiseen framework kohtaiseen bias-neuroniin. Bias-neuroni on neuroni, jonka tulos on aina 1. Bias-yhteyden tarkoitus on muuttaa neuronin perustilaa (katso lisätietoja Fausettin kirjasta [26, sivu 41]).

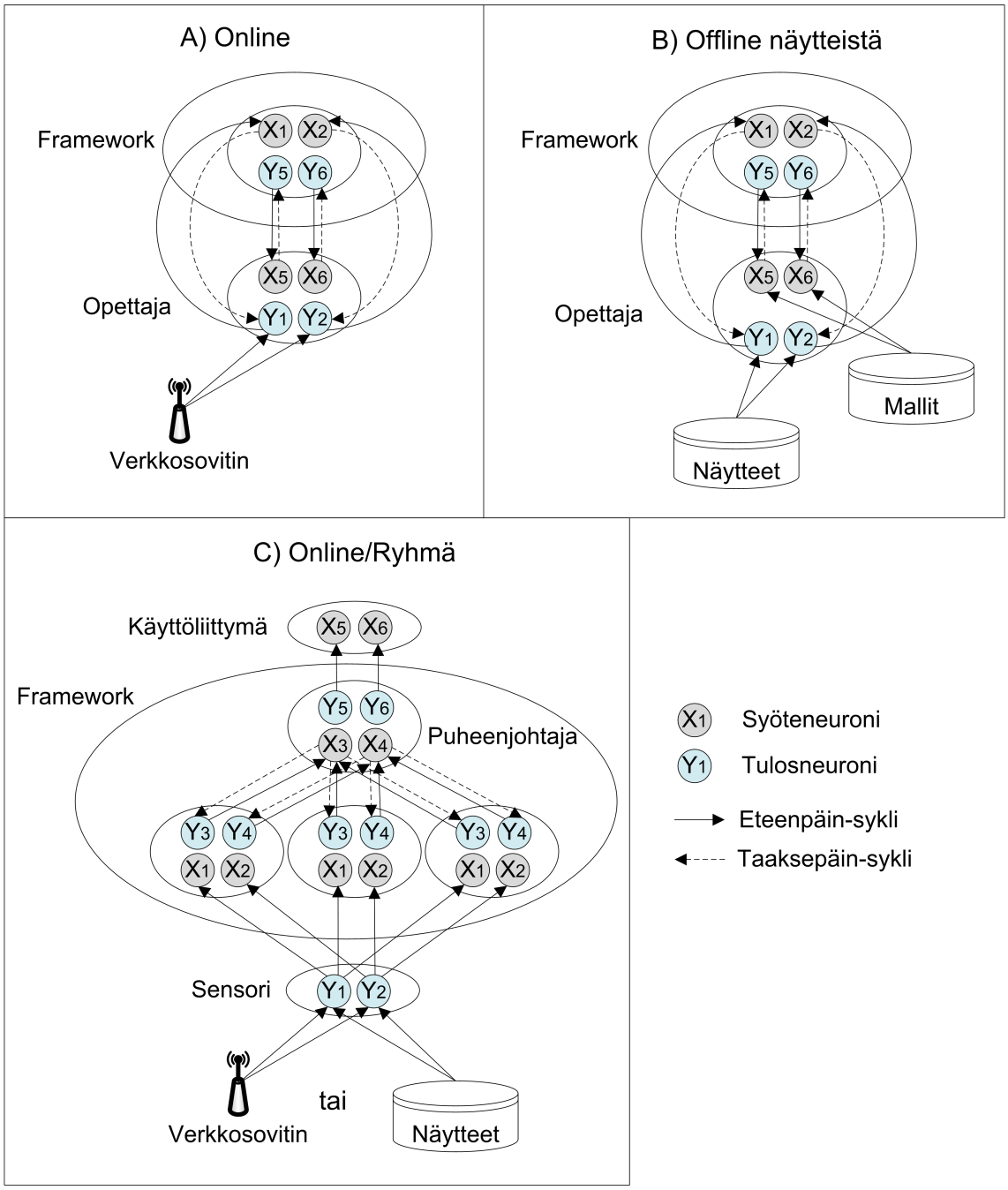
5.5.3 Opettaja

Opettajalla tarkoitetaan tässä työssä joko erityistä tätä tehtävää varten suunniteltua neuroverkkoa tai frameworkin ulkopuolista komponenttia. Opettajan tarkoitus on opettaa frameworkissa käytettäviä neuroverkkoja ennen niiden soveltamista luokittelu ja tunnistustehtäviin. Opettaminen tapahtuu kappaleessa 4.3 kuvattujen menetelmien mukaisesti.

Kuvassa 5.7 esitellään kolme erilaista esimerkkiä siitä, miten verkkoja voitaisiin mahdollisesti opettaa. Vaikka tapauksissa **A** ja **B** kuvassa näkyy ainoastaa yksi verkko, voi frameworkissa todellisuudessa olla useita rinnakkaisia verkkoja tapauksen **C** tapaan. Tulosten keskiarvoistamisen ansiosta verkkojen suuri määrä suojelee yksittäisiltä huonosti käyttäytyviltä verkoilta.

Kuvan 4.3 **A**-tapaus esittelee sensoriin integroidun opettajan. Opettaja opettaa verkkoja suoraan verkosta kaapatulla liikenteellä, joten kyse on suorasta, eli on line -oppimisesta. Opetuksessa oletetaan liikenteen olevan peräisin normaalista toiminnasta, joten opettaja laskee virheen tämän oletuksen perusteella. Menetelmä sopii poikkeamaperusteisen tunkeutumisen tunnistusjärjestelmän opettamiseen.

Kuvan 4.3 **B**-tapaus esittelee valmiiden näytteiden kautta opettavan opettajan. Opettajalla on näytteitä normaalista liikenteestä, jolla verkolle opetetaan tapaus A:n tavoin mikä on normaalia liikennettä. Lisäksi opettajalla on mahdollisuus käyttää näytteitä tunnetuista hyökkäyksistä, jolloin järjestelmälle voidaan opettaa myös varmoja hyökkäyksiä. Tämän pitäisi parantaa järjestelmän kykyä tunnistaa ennalta tun-



Kuva 5.7: Esimerkkejä vaihtoehtoisista opetusmalleista.

nettuja uhkia ja täydentää poikkeamaperusteista tunnistusta. Opettajan tietokantaan on tallennettu näytteitä, sekä näitä vastaavia malleja. Mallit ovat käytännössä aikaleimattuja opetusvektoreita. Näytteet ja mallit täytyy synkronoida keskenään näiden aikaleimojen perusteella. Näyte-malli -pareja voidaan tuottaa esimerkiksi poikkeamaperusteisen tunnistamisen sivutuotteena: havaittu poikkeama todetaan hyökkäykseksi, kaapattu liikenne tallennetaan tietokantaa näytteenä ja luodaan hyökkäystä vastaava malli.

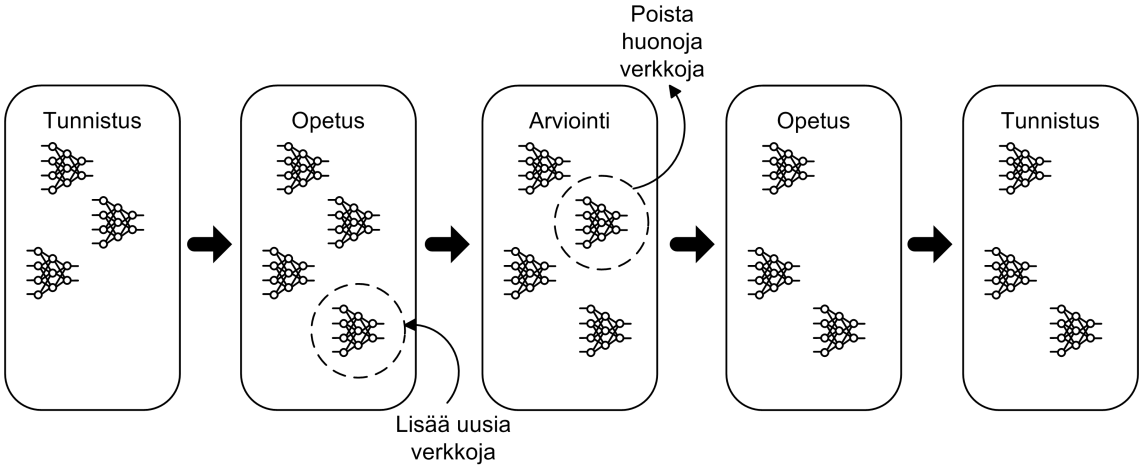
Kuvan 4.3 **C**-tapaus esittelee hieman poikkeavan opettajatyypin. Opettaja on tässä tapauksessa erityinen verkko eikä ulkoinen komponentti. Nimitän verkkoa tässä yhteydessä nimellä ”puheejohtaja”. Tässä mallissa opetusdata voi tulla A-tapauksen tapaan suoraan verkosta tai B-tapauksen mukaan näytteistä. Menetelmä perustuu kappaleessa 4.7.1 esiteltyyn verkkojen komiteaan. Puheenjohtaja laskee painotetun summan verkkojen ”äänestyksestä”. Luotettavasti, eli yleisen linjan mukaan äänestävät verkot, saavat suuremman painokertoimen kuin epäluotettavat verkot. Myös kaikki uudet tai järjestelmään tuodut, vielä opettamattomat ja testaamattomat, verkot saavat pienemmän painokertoimen. Tällä minimoidaan mahdollisesti virheellisesti äänestävien verkkojen äänestystulokseen aiheuttama virhe. Opetus ja painokertoimien muuttaminen voi tapahtua kappaleen 4.3.3 backpropagation-algoritmin mukaisesti. Virhe lasketaan joko puheenjohtajan syöte tai tulos -neuroneissa ilman opetusvektoria. Opetusvektorina toimii äänestystulos, jota vasten virhe lasketaan. Jos tarkoituksena on luokitella syötteet tiukasti eri luokkiin, täytyy puheenjohtajan verrata tulosvektorin kaikkia alkioita toisiinsa ja valita ”voittanut” äänestystulos eli luokka ja asettaa sen arvoksi 1. Muut luokat tulkitaan arvoon 0 tai -1 riippuen verkkojen aktivaatiofunktiosta. Näin saatua opetusvektoria käytettään nyt kaavaan 4.28.

Esimerkissä **C** kerrottiin uusista ja järjestelmään ulkopuolelta tuodusta verkoista. Uudet verkot tarkoittavat täysin uusia, kappaleen 4.4 mukaisesti luotuja verkkoja. Järjestelmän ulkopuolelta tuoduita verkoista kerrotaan tarkemmin tietämyksen jakamisen yhteydessä kappaleessa 5.5.9. Opetuksen kannalta nämä verkot ovat erilaisesta alkuperästään riippumatta samassa asemassa.

Kuvassa 5.6 esitettyä tilakaaviota voidaan laajentaa edelleen uudella tilalla ”arviointi”. Arviointi tilassa opetettuja verkkoja verrataan keskenään jollakin menetelmällä. Esimerkin **C**-tapauksessa menetelmä voi olla yksinkertaisimmillaan ”puheenjohtajan” ja opetettavien verkkojen välisten painokertoimien vertailu. Pienimmän painokertoimen omaava verkko tai verkot poistetaan. Käytännössä kannattanee vertailuun kuitenkin ottaa mukaan myös verkon ikä. Ilman verkon iän huomioimista uudet verkot siivottaisiin todennäköisesti pois ennen kuin ne ehtisivät oppia

ja kasvattaa kerrointaan. Toisaalta optimaalisen verkkokonfiguraation kannalta voi olla hyödyllistä luopua vanhoista dominoivista verkoista aina silloin tällöin ja antaa tilaa uusille ”näkökulmille”.

Kuvassa 5.8 nähdään miten frameworkin tilat seuraavat toinen toisiaan järjestyksessä: Tunnistus –> Opetus –> Arviointi –> Opetus –> Tunnistus.

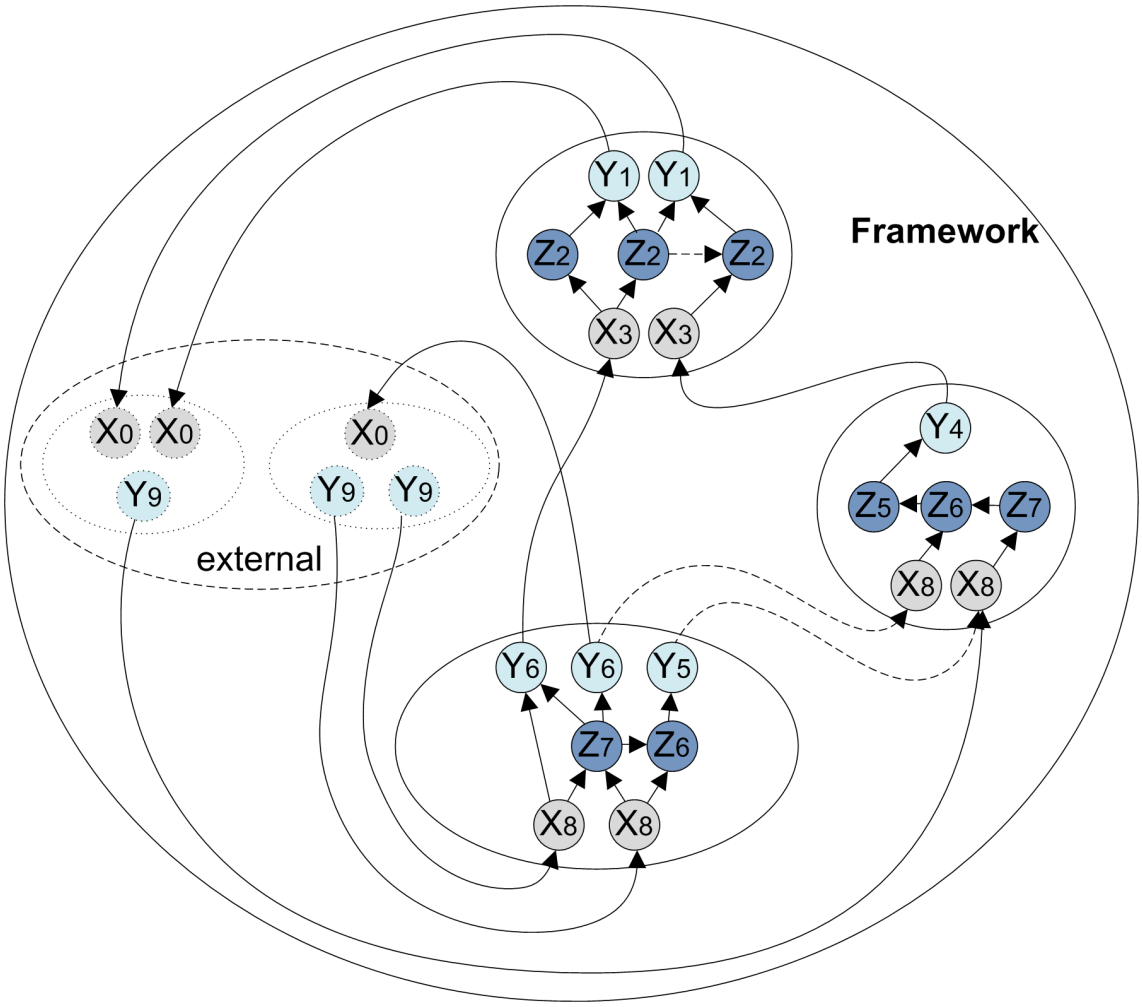


Kuva 5.8: Verkkojen kierto.

Opetustilassa edellisen kierroksen parhaat verkot pääsevät opettamaan uusia. Ensimmäisessä opetustilassa lisätään uusia verkkoja ja opetetaan kaikki edellä kuvatulla tavalla. Arviointivaiheessa poistetaan huonoimmat verkot niin ikään edellä kuvatun tavan mukaisesti. Verkkojen poistamisen jälkeen tarvitaan toinen opetusjakso, jotta jäljelle jääneet verkot saadaan vastaamaan uutta konfiguraatiota. Menetelmä vastaa siis kappaleen 4.6.1 verkon karsintaa, jossa verkon pienimmillä painokertoimilla varustetut yhteydet poistetaan opetuksen edetessä.

Opetuksessa voidaan soveltaa myös kappaleessa 4.7.2 kuvattua tekniikkaa ja opetetaan verkkoja useissa eri ryhmissä. Eri ryhmille asetetaan erilaiset tavoitteet. Toisin sanoen verkot yrittävät oppia eri asioita. Yksi ryhmä voi esimerkiksi opetella tunnistamaan normaalin liikenteen kun taas muut ryhmät opettelevat kukin tunnistamaan erilaisia tunnettuja hyökkäyksiä.

Vanhojen ja uusien verkkojen mukautumista täytyy säädellä siten, että vanhat jo käytössä hyväksi havaitut verkot eivät oppisi uusilta verkoilta huonoja taitoja, vaan uudet verkot oppisivat vanhoilta verkoilta hyviä taitoja. Tästä oppimisen hidastumisesta lisää kappaleessa 4.3.2.



Kuva 5.9: Esimerkki verkkojen topologiseksi järjestys.

5.5.4 Suoritusjärjestys

Oikean suoritusjärjestyksen selvittämiseksi frameworkin kaikki solmut täytyy järjestää topologiseksi puurakenteeksi siten, että juurisolmuna toimivat kappaleessa 5.5.6 kuvatussa ”external”-ryhmässä sijaitsevien verkkojen kaikki syöteneuronit ja lehtisolmuna ”external”-ryhmän verkkojen kaikki tulosneuronit. Solmut järjestetään puuksi käyttäen muokattua Dijkstran algoritmia määritellen jokaisen solmun, eli viime kädessä neuronin, etäisyys edellä mainitusta juurisolmusta. Kuvassa 5.9 esitetään kuvitteellista järjestelmää, jonka solmut on järjestetty siten, että ne suoritetaan sellaisessa järjestyksessä, jossa taaksepäin suuntautuvien viivästettyjen yhteyksien määrä jää mahdollisimman pieneksi. Kuvassa X kuvaa syöteneuronia, Y tulosneuronia ja Z piiloneuronia. Katkoviivalla merkityt yhteydet ovat viivästettyjä. Solmujen perässä oleva luku kuvaa tasoa, jolla kyseinen neuroni sijaitsee. Tasoa

käytetään selvitettäessä neuronien suoritusjärjestys.

Kuten edellä mainittiin, käytetään neuronien suoritusjärjestyksen määrittelemiseen Dijkstran algoritmia. Päädyin tähän algoritmiin siksi, että sitä käyttäen voidaan selvittää missä järjestyksessä suunnatun verkon solmut ovat toisiinsa nähden. Dijkstran algoritmi suoritetaan käänteisessä järjestyksessä, verkon yhteyksien suuntaa kuvaavia nuolia vastaan. Algoritmin lähtösolmuna toimii kuvitteellinen solmu, jonka muodostavat, kuten edellä mainittiin, kaikki "external"-ryhmän verkkojen tulosneuronit. Tämä tapahtuu asettamalla kaikkien näiden neuronien etäisyydeksi 0. Verkon suoritusjärjestyksessä käytetyt algoritmit löytyvät liitteestä D.3.

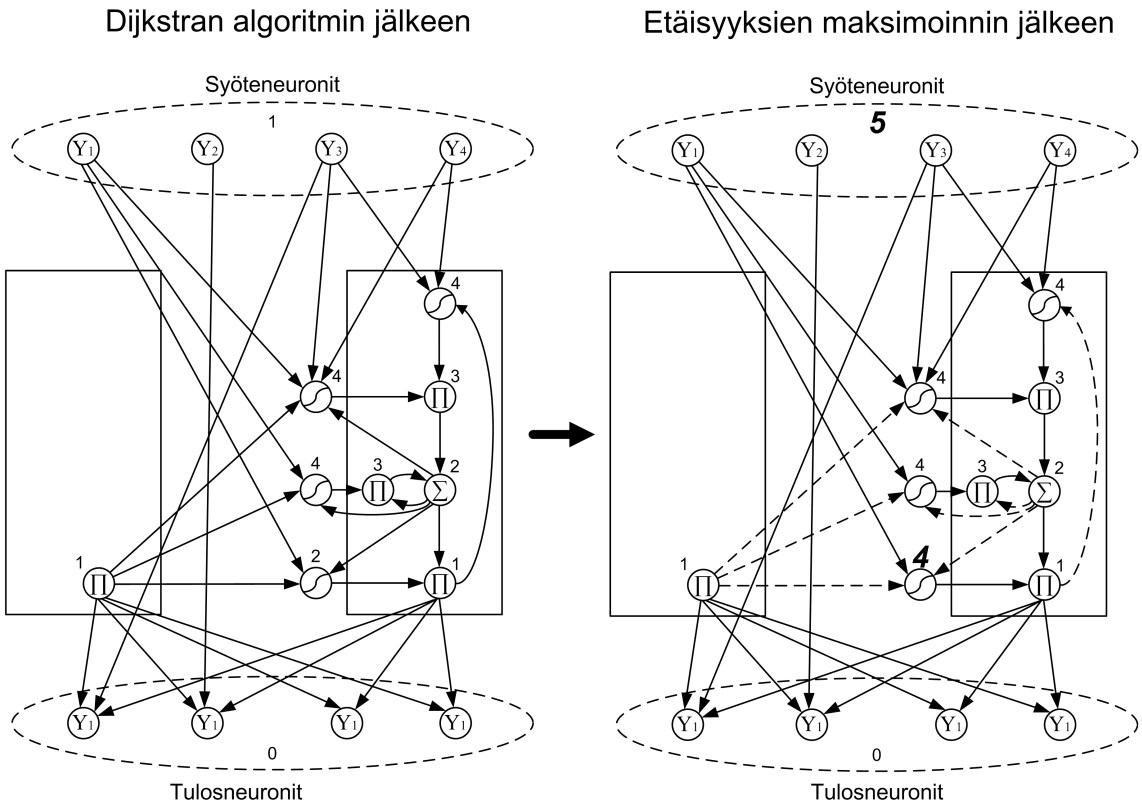
Dijkstran algoritmin (lagoritmi 10) kannalta frameworkin sisältämät verkot ja neuronit ovat kaikki verkon solmuja, joilla on pienin ja suurin etäisyys juuresta. Koska verkoilla on sisäinen rakenne, pienin ja suurin etäisyys ovat eri suuruiset, ja eroavat toisistaan kyseisen solmun sisältämän lyhimpien polkujen puun korkeuden verran. Solmun pienin etäisyys siis kertoo verkon tulosneuronien (sisäinen juuri) etäisyyden koko järjestelmän juuresta, ja suurin etäisyys verkon syöteneuronien etäisyyden järjestelmän juuresta. Neuroneilla ei ole sisäistä rakennetta, joten pienin ja suurin etäisyys on sama.

Kuvassa 5.10 esitetään LSTM-lohkon tasot kaksivaiheisen algoritmin (algoritmi 8) eri vaiheiden jälkeen. Vasemmanpuoleisessa verkossa tulosneuronit muodostavat yhdessä lähtösolmun, jonka etäisyys globaalista lähtösolmusta on annettu ylemmän tason algoritmissa. Kaikkien solmujen etäisyys on laskettu takaperin nuolia vastaan. Kuvasta voidaan helposti nähdä, että pelkkä neuronien etäisyyden määritteleminen ei riitä.

Kuvan 5.10 oikeanpuoleisessa verkossa nähdään (korostetut numerot) miten algoritmin toisen vaiheen (algoritmi 11) jälkeen syöteneuronien ja tulosportin etäisyys tulosneuroneista on muuttunut. Tällä tavoin algoritmien 2 ja 5 suoritus etenee topologisesti oikeassa järjestyksessä. Ilman maksimointia osa neuroneista suoritettaisiin ennen kuin niiden syötteet olisivat valmiina.

Algoritmissa 10 rivillä 9 tai 7 kiinnitetään solmun etäisyys juurisolmusta, joka verkon tapauksessa tarkoittaa algoritmin 8 rekursiivista suorittamista kyseiselle verkolle. Jos solmu on neuroni, ei sillä, kuten edellä mainittiin, ole sisäistä rakennetta. Siitä syystä etäisyyden asettaminen tarkoittaa ainoastaan solmun etäisyyden tallettamista tietorakenteeseen. Tällä tavoin Dijkstran algoritmia hyväksi käyttäen voidaan määritellä verkon solmujen suoritusjärjestys, joka "forward" -vaiheessa (algoritmi 2) kulkee suurimmasta etäisyydestä pienimpään ja "backward" -vaiheessa (algoritmi 5) pienimmästä suurimpaan.

Sen lisäksi, että algoritmi 11 aikaistaa neuronien suoritushetkeä mahdollisim-



Kuva 5.10: Verkon solmujen suoritusjärjestyksen määritteleminen kaksivaiheisella algoritmilla.

man paljon, hoitaa se myös taaksepäin suuntautuvien yhteyksien tunnistamisen. Tämä on tärkeää siksi, että ilman viivettä verkkoon muodostuisi ikuisia suoritussilmukoita. Silmukoiden katkaisemiseksi algoritmi asettaa matalammalta tasolta korkeammalle tasolle (eli taaksepäin) suuntautuvalle yhteydelle viivelipun. Viivelippu asetetaan myös samalla tasolla olevien solmujen välisille yhteyksille. Viivelippu aiheuttaa suoritettaessa sen, että kyseinen yhteys välittää siihen kirjoitetun tiedon eteen- tai taaksepäin vasta seuraavalla suoritusaskeleella. Jos solmu sijaitsee algoritmin 10 jälkeen niin kutsutulla kriittisellä polulla ei sen suoritushetki muutu algoritmia 11 suoritettaessa.

Eräänä huomiona täytyy mainita, että algoritmien ei tarvitse huomioida yhteyksiä, joiden viivelippu on asetettu edeltä käsin käyttäjän toimesta. Tämä siksi, että niiden suorittaminen ei ole kriittistä saman aika-askeleen aikana.

Tässä kappaleessa esitelty tapa ei ehkä ole täysin optimaalinen, koska verkko täytyy käydä läpi kahteen kertaa lopusta alkuun ja alusta loppuun. Nopeamman algoritmin kehittäminen voidaan lisätä tulevaisuuden kehityskohteisiin.

Lisäksi edellä esitetyista algoritmeista täytyy mainita se, ettei niissä huomioi-

da ”external”-ryhmän verkkojen erilaista käsittelyä. Kuten kuvasta 5.9 nähdään, ”external”-ryhmän verkot pitää käsitellä täysin eri tavalla. Tämän ryhmän verkot näyttäytyvät algoritmille täysin päinvastaisina ”input” ja ”output” -ryhmien kannalta. Lisäksi ”external”-verkoilla ei ole sisäistä rakennetta.

5.5.5 Nimeäminen

Neuronien ja verkkojen nimet voidaan määritellä joko verkkoa kuvaavassa XML-tiedostossa, tai ne voidaan nimetä dynaamisesti. Dynaaminen nimeäminen tapahtuu esimerkiksi tilanteessa, jossa verkko luodaan niin sanotusta ”aihiosta” (engl. *template*) jolloin verkolle ei ole annettu nimeä etukäteen. Tämä helpottaa uusien verkkojen luomista, koska käyttäjä voi lisätä useita saman tyyppisiä verkkoja keksimättä jokaiselle verkolle uutta yksilöllistä nimeään.

Jokaisella solmulla, eli verkolla, ryhmällä tai neuronilla, täytyy olla ryhmänsä sisällä yksilöllinen nimi. Tätä sääntöä noudattamalla voimme yksilöidä järjestelmän jokaisen solmun pystyen silti nimeämään samaa tehtävää suorittavat solmut samalla nimellä. Tämä on tärkeää esimerkiksi liitettäessä verkkoja aiemmin kuvatulla tavalla toisiinsa.

Verkon eri solmuihin voi viitata käyttäen pisteellä (.) tai kauttaviivalla (/) erotettuja solmunimiä. Erottimista ja solmunimistä koostuvaa merkkijonoa voidaan kutsua poluksi. Jokainen polun alussa oleva erotin kertoo, että osoituksen on tarkoitus alkaa yhtä tasoa ylempänä kuin missä solmu itse sijaitsee. ’.’-erotin tarkoittaa ryhmää ja ’/’-erotin solmua. Solmu taas tarkoittaa joko verkkoa tai frameworkia.

Esimerkiksi polku ”*./.joku.toinen*” tarkoittaa, että solmusta, jossa osoitus tehdään, noustaan ensin oman ryhmän tasolle. Omasta ryhmästä taas noustaan oman verkon tasolle, josta noustaan vielä verkon sisältämän ryhmän tasolle. Tästä ryhmästä etsitään ”joku” niminen solmu, jonka alta etsitään lopulta ”toinen” niminen solmu. Itse asiassa pelkän ’.’ merkin käyttö riittää, muutta jos halutaan varmistaa, että verkon rakenne on oikein voi käyttää ’/’ merkkiä merkitsemään verkkoa tai frameworkia.

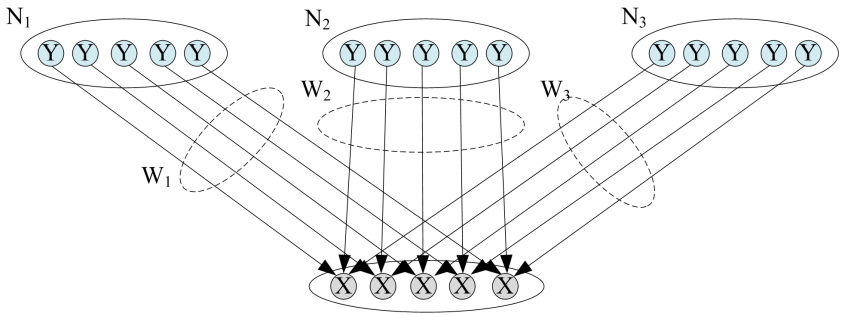
Toisena esimerkkinä polulla ”*/kolmas/neljäs.viides*” voidaan osoittaa frameworkin sisältämän verkon ”kolmas” ryhmässä ”neljäs” sijaitsevaa solmua ”viides”.

Ilman etumerkkejä haku aloitetaan suoraan solmusta, jossa haku suoritetaan.

Esimerkkejä polkujen käytöstä on liitteessä B. Selvitys liitteessä käytetyn XML-kielen kieliopista löytyy kappaleesta 6.1.3.

5.5.6 Verkkojen välinen kommunikointi

Verkkojen välisissä yhteyksissä painoarvon toteuttamisella on kaksi vaihtoehtoa: jokaisella synapsilla on oma painoarvonsa tai yhteydellä on yksi yhteinen painoarvonsa. Ensimmäisessä tapauksessakin on hyödyllistä laskea yhteydelle yksi painoarvo, esimerkiksi synapsien painoarvojen keskiarvo, jota voidaan käyttää arvioitaessa yhteyden hyödyllisyyttä. Kuvassa 5.11 esitellään vaihtoehto, jossa ylhäällä olevien verkkojen N_1, N_2 ja N_3 tulosneuronit (Y) liittyvät alhaalla olevan verkon syöteneuroneihin (X). Jokaisella verkkojen välisessä yhteydellä on yksi paino (W).



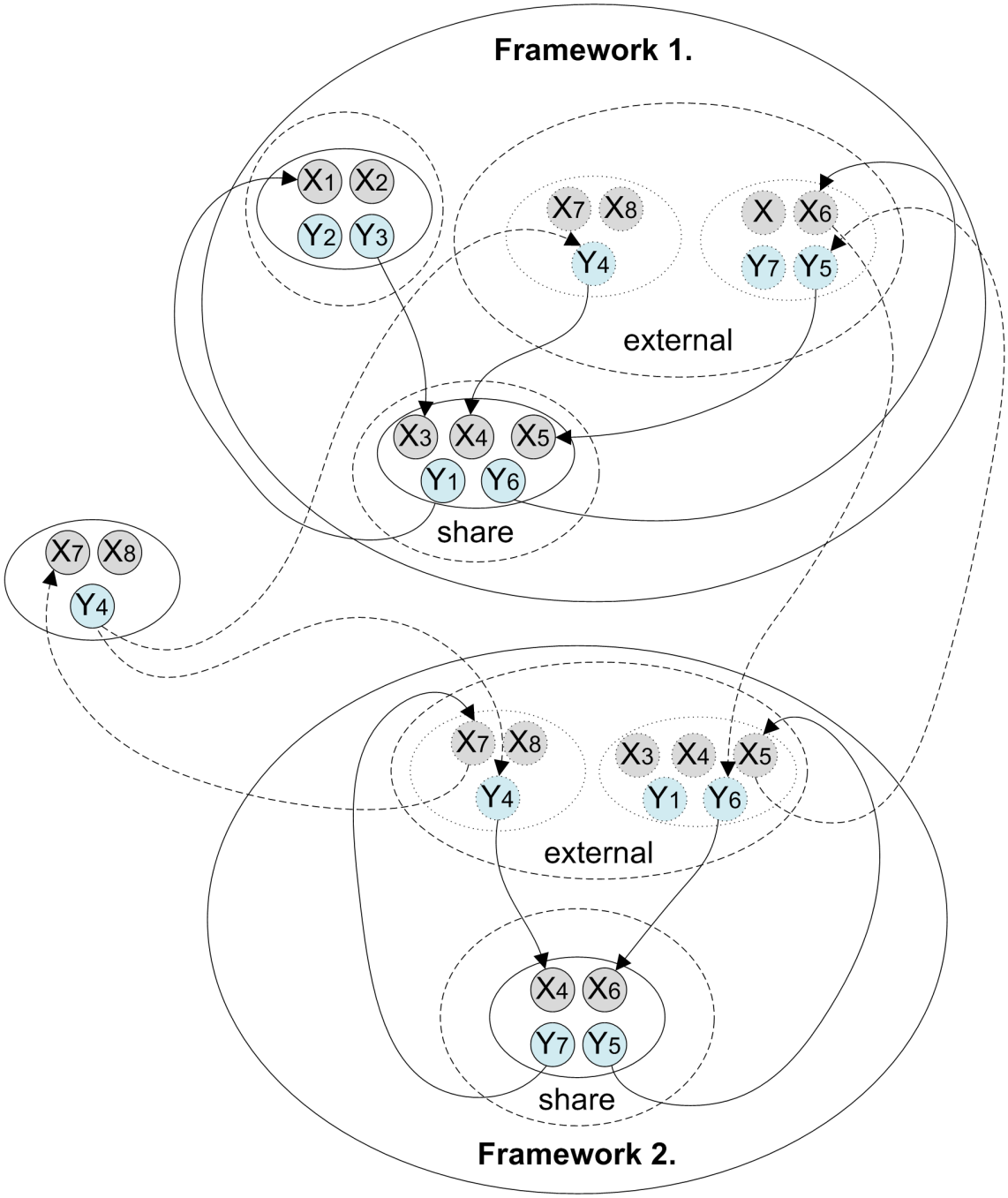
Kuva 5.11: Verkkojen suhteet.

Tässä harjoitustyössä toteutettiin tavallaan osittain molemmat. Käytännössä verkkoja yhdistävät yksittäiset neuronit ylläpitävät omia painoarvojaan, mutta tallennettaessa ja ladattaessa verkkoja käytetään yksittäisten painojen keskiarvoa.

Ulkopuoliset verkot näkyvät erityisessä ryhmässään, joka on nimetty ”external”. Tässä ryhmässä olevat verkot sijaitsevat siis frameworkin ulkopuolella, ja ryhmässä olevat verkot ovat ikään kuin näiden verkkojen ”kuvia”. Tästä syystä käytän niistä nimitystä virtuaalinen verkko (engl. *virtual network*). Virtuaalisen verkon tehtävänä on huolehtia pakallisten ja ulkoisten verkkojen välisestä kommunikoinnista. Tässä tehtävässä sen tulee kytkeytyä paikallisiin verkkoihin synapsien välityksellä siten, että paikallinen verkko ei tiedä, onko se liittynyt paikalliseen vai virtuaaliseen verkkoon. Jokaisen ”forward” ja ”backward” -syklin jälkeen virtuaaliverkko lähettää kappaleessa 5.5.7 mainittuja ”forward” ja ”backward” -paketteja toisissa frameworkeissa sijaitseviin virtuaalisiin tai täysin ulkoisiin verkkoihin.

Ulkoinen verkko voi tarkoittaa myös jotakin järjestelmän komponenttia, kuten käyttöliittymää. Tässä harjoitustyössä käyttöliittymä (kappale 5.2.4), sensori (kappale 5.2.1) ja opettaja (kappale 5.5.3) ovat ulkoisia verkkoja ja näkyvät frameworkille (kappale 5.5.1) virtuaalisena verkkona ”external”-ryhmässä.

Esimerkki ulkoisten verkkojen liittämisestä frameworkiin kuvassa 5.12.



Kuva 5.12: Vasemmalla näkyvä ulkoinen verkko on liitetty kahteen eri frameworkiin. Katkoviivaiset nuolet kuvaavat virtuaalisia synapseja. Pisteviivaiset verkot kuvaavat virtuaalisia verkkoja.

Ulkoisten verkkojen painot täytyi tässä harjoitustyössä asettaa kiinteiksi. Tämä siksi, että toteutuksesta puuttui keino ulkoisten verkkojen tunnistetietojen tallennukseen. Tämä taas johti vaikeuksiin palauttaa vanhan yhteyden painoarvot takaisin tilanteessa, jossa framework välillä tallennetaan levylle ja ladataan sieltä myöhemmin uudelleen. Jos painoarvot muuttuvat, pitää ne palauttaa täysin samalla tavalla takaisin tai verkkojen opitut painot eivät enää vastaa uutta tilannetta.

Jotta edellä kuvattu toiminnallisuus voitaisiin siis toteuttaa ulkoisille verkoille olisi ulkoiset verkot ja muut lähteet tunnistettava joka kerta luotettavasti, kuten kappaleessa 5.5.7 kerrotaan.

Ulkoisten verkkojen kommunikointiin tarvitaan erityinen protokolla, joka kuvataan kappaleessa 5.5.7.

5.5.7 Protokolla

Järjestelmän eri komponenttien keskinäiseen kommunikaatioon tarvitaan erityinen kommunikaatioprotokolla. Protokolla toteutetaan käyttäen hyväksi Qt:n sarjallistamis- ja rinnakkaistamisoperaatioita. Tällä tavoin pystyn toteuttamaan alustariippumattoman ja tehokkaan binääriprotokollan melko vähällä vaivalla.

Jokainen järjestelmän komponentti kuuntelee järjestelmässä yleislähetykseen käytettyä porttia 50000. Tätä porttia komponentit käyttävät toistensa löytämiseen. Tämän lisäksi jokainen komponentti varaa itselleen oman, laitteen sisällä yksilöllisen, porttinsa. Tätä porttia komponentti käyttää ottaessaan yhteyttä toisiin komponentteihin. Yksilöllinen portti varataan alueelta 50001 – 50100. Protokolla toimii saman aliverkon sisällä. Samalla laitteella voi toimia 100 eri komponenttia.

Protokollan jokainen paketti, eli sanoma sisältää vähintään ”MsgHeader”-tietotyyppiä olevan otsikon. Otsikko sisältää protokollapakettien tunnistamiseen ennalta määritellyn numeroarvon kentässä nimeltään ”magic”. Otsikko sisältää myös sanoman tyypin kentässä ”type”. Otsikko sisältää myös lähde- ja kohdeosoitteet kentissä ”source” ja ”destination”. Näillä kentillä voidaan sanoman tyypistä riippuvalla tavalla ilmaista mikä on sanoman sisältämän tiedon lähde ja kohde. Lähde ja kohde osoite ilmoitetaan kappaleessa 5.5.5 kuvatulla tavalla. Sanomien muu sisältö riippuu edellä mainitusta sanoman tyypistä.

Tällä hetkellä protokolla sisältää sanomat seuraaviin toimenpiteisiin:

- sanoma verkon lisäämiseen frameworkiin (**put**),
- sanoma verkon hakemiseen frameworkista (**get**),
- sanoma verkon poistamiseen frameworkista (**remove**),

- sanoma komponenttien etsimiseen (**probe**),
- sanoma komponenttien mainostamiseen (**advertise**),
- sanoma tulosvektorin lähettämiseen (**forward**) ja
- sanoma virhevektorin lähettämiseen (**backward**).

Kuvaus edellä lueteltujen sanomien rakenteesta ja tietotyypeistä löytyy liitteestä E.

”NodeMsg”-tietotyypistä muodostuvaa **put**-sanomaa käytetään siis verkon lisäämiseen tiettyyn ”destination”-kentässä kerrottuun osoitteeseen. ”Source”-kentän osoite riippuu sanoman lähettäjästä. Itse verkon kuvaus löytyy kappaleen 6.1.3 mukaan XML-muotoisena kuvattuna kentästä ”node”. Sanomaan saadaan ”AckMsg”-tyyppinen vastaussanoma, jonka ”status”-kentässä on toiminnon onnistumisesta tai epäonnistumisesta kertova tilakoodi. Vastaus sanoman ”source” sisältää lisätyn verkon absoluuttisen osoitteen frameworkin sisällä. ”Destination”-kentässä on ”put”-sanoman ”source”-kentän sisältö. Kappaleessa 5.5.6 mainitut virtuaaliset verkot luodaan lisäämällä verkko ”external”-ryhmään tätä sanomaa käyttäen. Tätä sanomaa ei huomioida jos se vastaanotetaan yleislähetysportista.

Pelkästä ”MsgHeader”-tietotyypistä muodostuvalla **get**-sanomalla saadaan noudettua frameworkista tietty, ”destination”-kenttään merkitty verkko. Vastaanottaja voi merkitä ”source”-kenttään haluamansa osoitteen. Tämä tulee löytymään ”GetAckMsg”-tyyppisen vastaussanoman ”destination”-kentästä ja helpottaa näin ollen vastaanotetun verkon mahdollista tallentamista oikeaan paikkaan. ”Source”-kentässä on luonnollisesti noudetun verkon osoite. Itse verkon kuvaus löytyy kappaleen 6.1.3 mukaan XML-muotoisena kuvattuna kentästä ”node”. ”status”-kentässä on toiminnon onnistumisesta tai epäonnistumisesta kertova tilakoodi. Tätä sanomaa ei huomioida jos se vastaanotetaan yleislähetysportista.

Pelkästä ”MsgHeader”-tietotyypistä muodostuvalla **remove**-sanomalla voidaan poistaa frameworkista tietty verkko. Poistettavan verkon osoite ilmoitetaan ”destination”-kentässä. ”Source”-kentän osoite riippuu sanoman lähettäjästä. Sanomaan saadaan ”AckMsg”-tyyppinen vastaussanoma, jonka ”status”-kentässä on toiminnon onnistumisesta tai epäonnistumisesta kertova tilakoodi. Vastaus sanoman ”source” sisältää poistetun verkon absoluuttisen osoitteen frameworkin sisällä. ”Destination”-kentässä on ”remove”-sanoman ”source”-kentän sisältö. Tätä sanomaa ei huomioida jos se vastaanotetaan yleislähetysportista.

Pelkästä ”MsgHeader”-tietotyypistä muodostuvalla **probe**-sanomalla voidaan kartoittaa samasta aliverkosta löytyviä järjestelmän komponentteja.

”Source” ja ”destination”-kenttien osoitteet riippuvat sanoman lähettäjästä. Sanomaan saadaan ”NodeMsg”-tyyppinen vastaussanoma, jonka ”node”-kentässä on sanomaan vastaavan komponentin ylätason rekenne XML-muotoisena. Frameworkille se tarkoittaa kappaleessa 6.1.3 kuvattua ”.nfx”-tiedostoa vastaavaa sisältöä. Vastaus sanoman ”source”-kentässä on ”probe”-sanoman ”destination”-kentän sisältö ja ”destination”-kentässä on ”probe”-sanoman ”source”-kentän sisältö. Tähän sanomaan vastataan sekä yleislähetys että yksilöllisestä portista.

”NodeMsg”-tietotyypistä muodostuvaa **advertise**-sanomaa käytettään verkkojen mainostamiseen. ”Source”-kentän osoite kertoo alkuperäisen verkon suhteellisen osoitteen (nimen) sanoman lähettäjän sisällä. ”Destination”-kentän osoite riippuu sanoman lähettäjästä. Itse verkon kuvaus löytyy kappaleen 6.1.3 mukaan XML-muotoisena kuvattuna kentästä ”node”. Sanomaan saadaan ”AckMsg”-tyyppinen vastaussanoma sinä tapauksessa, että sanoman vastaanottaja päättää lisätä mainostetun verkon itseensä. ”Status”-kentässä on toiminnon onnistumisesta tai epäonnistumisesta kertova tilakoodi. Vastaus sanoman ”source” sisältää lisätyn verkon absoluuttisen osoitteen komponentin sisällä. ”Destination”-kentässä on ”advertise”-sanoman ”source”-kentän sisältö. Tämä sanoma huomioidaan sekä yleislähetys että yksilöllisestä portista.

”SynapseMsg”-tietotyypistä muodostuvaa **forward**-sanomaa käytetään verkossa eteenpäin lähetettävän synaptisen tiedon välittämiseen. Eri komponenteissa, kuten esimerkiksi frameworkissa tai niiden ulkopuolella, sijaitsevat verkot lähettävät tätä sanomaa käyttäen omien tulosneuroniensa muodostamia tulosvektoreita toisten verkkojen syöteneuroneille. Lisää tietoa verkkojen välisestä kommunikoinnista kappaleessa 5.5.6. Sanoman sisältämä tieto, eli piirteet, on tallennettu ”features”-tyyppiseen ”data”-kenttään. ”Features”-tietotyyppi sisältää aikaleiman (katso kappale 5.5.8) ja matriisimuotoon tallennetut piirrevektorit. Matriisin sarakkeet edustavat erillisiä, tulos-syöte -neuroniparia edustavia, piirteitä. Matriisin rivit taas edustavat erillisiä, samaan aikaan samaan kohteeseen lähettäviä verkkoja edustavia, piirrevektoreita. Piirteiden tulee olla piirrevektorissa samassa järjestyksessä kuin tulosneuronit olivat **put** -sanomassa koska vastaanottaja päättelee neuronin, jolle kyseinen piirre välitetään, sen paikasta vektorissa. Useiden piirrevektorien lähettäminen samassa sanomassa vähentää komponenttien välistä liikennettä. Sanoman ”source”-kentän osoite kertoo lähettävän verkon osoitteen ja ”destination”-kenttä vastaanottavan verkon osoitteen. Tälle sanomalle ei ole vastaussanomaa eikä sitä huomioida jos se vastaanotetaan yleislähetysportista.

”SynapseMsg”-tietotyypistä muodostuvaa **backward**-sanomaa käytetään verkossa taaksepäin lähetettävän synaptisen tiedon välittämiseen. Se toimii muu-

ten täysin samalla tavalla kuin **forward**-sanoma, mutta se sisältää virhevektoreita joiden lähettäjänä toimivat **forward**-sanoman vastaanottaneen verkon syöteneuronit ja vastaanottajana **forward**-sanoman lähettäneet tulosneuronit. Tätä sanomaa käyttäen verkko voi opetustilanteessa kertoa vastaanottamansa **forward**-sanoman virheen.

Varsinkin ”SynapseMsg”-sanomat aiheuttavat verkkoon suuren kuorman. Tämä ilmenee varsinkin kappaleessa 6.2 kuvatun kaltaisessa järjestelmässä, jossa tarkoituksena on valvoa samaa langatonta verkkoa jota laitteet käyttävät kommunikoidakseen. Tästä aiheutuu noidankehä: sensori kaappaa langattomasta verkosta paketin, muodostaa siitä piirrevektori ja lähettää sen mahdollisesti eteenpäin käyttäen samaa langatonta verkkoa. Voidaan helposti nähdä, että jokainen lähetetty paketti generoi vähintään yhden uuden paketin jos järjestelmän komponentit sijaitsevat eri laitteilla.

Eräs vaihtoehto torjua edellä mainittua noidankehää on suodattaa järjestelmän itsensä lähettämät paketit siten, että niistä ei lasketa piirteitä eikä niitä lähetetä eteenpäin. Tämä tosin saattaa avata mahdolliselle hyökkääjälle mahdollisuuden käyttää suodatusta hyväkseen ja välttää hyökkäyspakettien kaappaaminen. Lisäksi järjestelmän lähettämien pakettien erottaminen muista paketeista on erittäin hankalaa, ja vaatii käytännössä kaapattujen pakettien salauksen purkamista sisällön tutkimiseksi. Toinen vaihtoehto on pyrkiä järjestelmän lähettämien pakettien määrän minimoimiseen. Tämä lähestymistapa sopii hyvin langattomiin järjestelmiin. Verkon kapasiteettia säästyy muihin tarkoituksiin ja langattomien laitteiden tehon kulutus vähenee.

Toinen vaihtoehto on kerätä yhteen pakettiin piirrevektoreita useammalta ajanhetkeltä ja lähettää ne käyttöliittymälle tai muulle asiakkaalle vain kun tilanne muuttuu yli asetetun kynnysarvon.

Ensimmäinen vaihtoehdoista on vaikea toteuttaa, koska varsinkin opetusvaiheessa arkkitehtuuri vaatii, että yhtä ”forward”-sykliä vastaa yksi ”backward”-sykli, ja näin ollen tietyllä aikaleimalla opettajalle lähetettyä ”forward”-pakettia vastaa samalla aikaleimalla varustettu ”backward”-paketti. Tilanteen muuttaminen vaatii lisää älykkyyttä ”VirtualNetwork”-oliolta, jotta se pystyy puskuroimaan opettajan lähettämiä paketteja molemmista suunnista.

Myös jälkimmäinen vaihtoehto vaatii lisäominaisuuksia ”VirtualNetwork”-olioon: olion täytyy laskea tulosvektori ennen pakettien lähettämistä ja verrata tulosta kynnysarvoon. Tämä vaihtoehto toisi suuremman hyödyn ja olisi helpompi toteuttaa. Välillisenä hyötynä verkon kuormituksen laskemisesta olisi myös mobiili laitteiden kuormituksen väheneminen, joka oletettavasti näkyisi jonkin verran myös

tehon kulutuksen vähenemisenä.

Koska kyseessä on hajautettu järjestelmä, jossa vastaanotettuun tietoon pitää voida luottaa, on sanoman lähettäjä voitava tunnistaa luotettavasti. Sanomien lähettäjästä täytyy olla täysi varmuus siksi, ettei järjestelmän saastuttaminen väärällä tiedolla onnistu, ja että järjestelmä kykenee arvioimaan oikeaa sanomien lähdettä. Siksi sanomien välittämisessä pitäisi käyttää jotakin protokollaa joka tarjoaa sanoman lähettäjän todentamisen, kuten esimerkiksi SLT-protokollaa [7]. Yksityisyyden suojan vuoksi lähettäjän tunnistaminen ei kuitenkaan saa paljastaa lähettäjän todellista identiteettiä. Lähteiden tunnistus ja yhteyden salaus rajattiin kuitenkin työn ulkopuolelle.

Hajautuksen ja tietoliikenneverkkojen luonteen vuoksi tarvitaan sanomien vastaanottoon jonkinlainen puskurointi ja uudelleen järjestämistoiminto. On tärkeää, että samanaikaisesti verkosta erotetut piirteet saapuvat samaan aikaan verkkojen analysoitavaksi. Tätä tukemaan tarvitaan aikaleimajärjestelmä ja solmujen välinen synkronointi, jotta piirresanomien oikea-aikaisuus voidaan taata kuten kappaleessa 5.5.8 kerrotaan. Aikaleimasta on hyötyä myös hyökkäystapahtumien ja tallennettujen piirteiden yhdistämisessä ja analysoinnissa jälkeenpäin kuten kappaleessa 5.2.3 kerrotaan.

5.5.8 Synkronointi

Hajautetun järjestelmän toiminnan kannalta on tärkeätä synkronoida hajautettujen osien toiminta. Tämä on erityisen tärkeää tässä työssä kuvatun kaltaisessa järjestelmässä, jossa tapahtumien kesinäinen ajallinen suhde on ratkaiseva. Jos tietoa siirretään useiden järjestelmän komponenttien välillä on niiden ajallinen järjestys pystyttävä luotettavasti toteamaan.

Tässä työssä käytetään aikaleimaa eräänlaisena sekvenssinumerona. Tätä aikaleimaa vertailemalla voidaan selvittää kunkin tapahtuman ajallinen suhde toiseen tapahtumaan. Kuten kappaleesta 5.5.7 kerrottiin, sisältyy jokaiseen tulos- ja virhevektorin välittämiseen käytettyyn sanomaan aikaleima tai aikaleimoja.

Framework synkronoidaan näiden ”external”-verkojen kautta saapuvien aikaleimojen mukaan. Tällä tavoin synkronointiongelma siirtyy ”external”-verkkoja vastaavien frameworkin ulkopuolisten tiedon tuottamisesta vastaavien ohjelmien vastuulle. Esimerkiksi tässä työssä aikaleimojen muodostamisesta vastaa sensori 5.2.1 tai opettaja 5.5.3, ja ainoastaan yksi aikaleimojen lähde on kerrallaan käytössä. Jos jossakin järjestelmässä on useampia aikaleimojen lähteitä, jotka sijaitsevat eri lait-

[7] http://tools.ietf.org/html/rfc5246

teilla, täytyy näiden laitteiden keskinäisestä aikasynkronoinnista huolehtia. Tämän kaltainen synkronointi kuitenkin rajataan tämän työn ulkopuolelle.

Frameworkin toiminta jaksotetaan vastaanotettujen tulosvektori-pakettien aikaleimojen mukaan siten, että vain saman hetkiset paketit otetaan käsittelyyn. Koska pelkästään saman hetkisiä paketteja suoritetaan samanaikaisesti, on jokaisella frameworkin ”forward” ja ”backward” -syklillä yksi aikaleima. Tämä aikaleima välitetään järjestelmän läpi seuraavalle vastaanottajalle, joka voi olla esimerkiksi käyttöliittymä 5.2.4 tai opettaja 5.5.3. Jos kyseessä on opettaja ja verkkoja ollaan opettamassa, vastaa opettaja vastaanottamaansa tulosvektori-pakettiin (forward) virhevektoripaketilla (backward), jonka aikaleima on sama kuin vastaanotetun paketin.

Koska ethernet-verkko ei takaa UDP-pakettien saapumista lähetysjärjestyksessä, on paketit mahdollisesti järjestettävä vastaanottajan toimesta aikaleimojen mukaan. Tässä työssä kuitenkin oletettiin pakettien saapuvan järjestyksessä kaikkien laitteiden sijaitessa samassa aliverkossa.

5.5.9 Tietämyksen jakaminen

Tietämyksen jakamisella tarkoitetaan tässä tapauksessa sitä, että verkot voisivat opettaa toisiaan. Ajatus tällaisesta toimintamallista perustuu kappaleessa 4.7 esitettyihin tekniikoihin, joissa pyritään parantamaan järjestelmän yleistävyyttä lisäämällä siihen useita toisistaan poikkeavia verkkoja. Kyseisessä kappaleessa esitettyjen tekniikoiden yhteydessä käytetyt neuroverkot luodaan yleensä satunnaisesti tai muuten juuri kyseistä sovellusta varten. Tässä työssä kuitenkin hahmotellaan järjestelmää, jossa erityisen toimiviksi osoittautuneita verkkoja voitaisiin käyttää laajemmalti hyväksi. Näin verkkojen opettamiseksi tehty työ ei joutuisi hukkaan vaan voitaisiin uudelleenkäyttää. Tämä mahdollistaisi järjestelmän itseopetuksen vähentäen käyttäjän ja ylläpitäjän kuormaa.

Tämän tyyppisen ominaisuuden mahdollistamiseksi olisi järjestelmän kyettävä vaihtamaan neuroverkkoja toisen vastaavanlaisen järjestelmän kanssa. Osittain tästä syystä kappaleessa 5.5.7 esitellyssä kommunikaatioprotokollassa on sisällytetty sanomia verkkojen mainostamiseen, hakemiseen ja lisäämiseen.

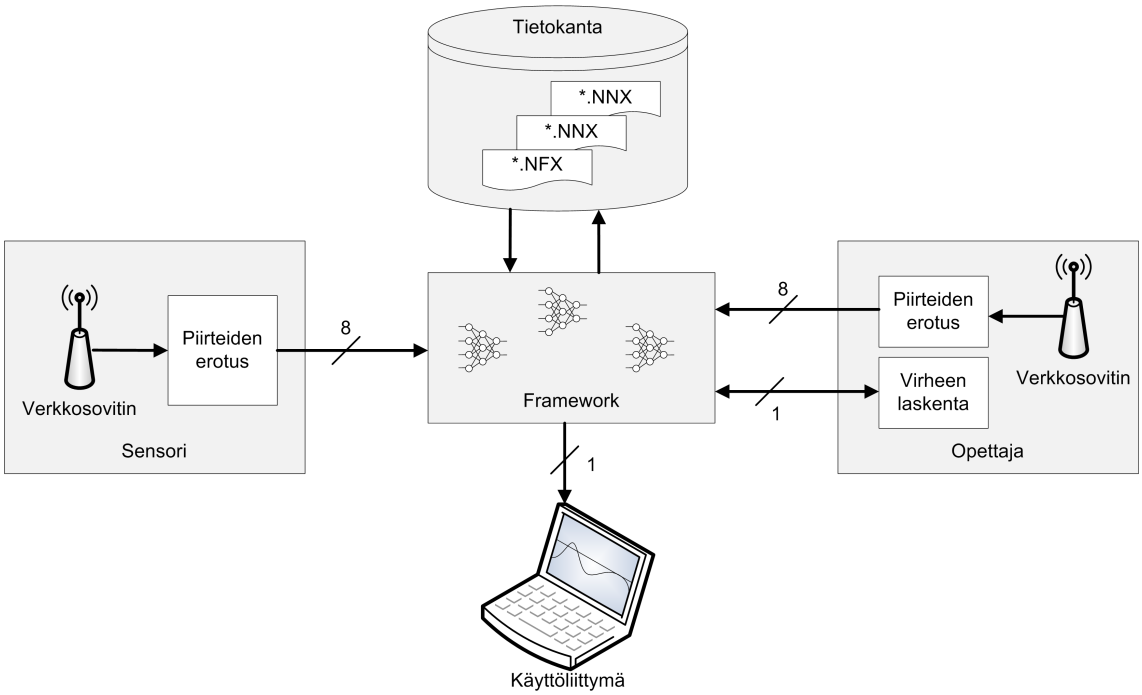
Tehokkuuden kannalta järkevin lähestymistapa voisi olla vaihtaa parhaiten menestyviä verkkoja. Tämä voisi toimia, jos oletetaan, että jokaisella verkon toimijalla on hyvät aikeet. Kuitenkin, jos näin olisi, tunkeutumisen tunnistusjärjestelmiä ei tarvittaisi. Mikään ei tietysti estä järjestelmän komponentteja tarjoamasta omasta mielestään parhaalla mahdollisella tavalla toimivaa verkkoaan toisen järjestelmän käyttöön, mutta vastaanottajan ei pidä luottaa tähän hyväntahtoisuuteen.

Jo kappaleessa 5.5.7 mainitun järjestelmän saastumisen estämiseksi täytyy uusiin verkkoihin suhtautua aluksi epäilevästi. Tämä tarkoittaa käytännössä pieniä painoarvoja uudesta verkosta johtaville yhteyksille. Opetussyklin aikana selviää miten uusi verkko käyttäytyy opetusdataan tai vanhoihin verkkoihin verrattuna. Tässä mielessä uutta verkkoa käsitellään samoin kuin sattumanvaraisesti luotua uutta verkkoa. Jos järjestelmään tuotu verkko osoittautuu toimivaksi, vahvistuu muiden verkkojen luottamus siihen ja siitä ulos johtavat yhteydet saavat opetussääntöjen mukaisesti suuremman painoarvon. Linjasta poikkeava verkko menettää ”uskottavuutensa” ja sen painoarvot lähestyvät nollaa kuten kappaleessa 5.5.3 kuvataan.

Verkkoja siirrettäessä tärkeä huomioitava asia on järjestelmän käyttäjien yksityisyyden suoja. Verkoissa ei saa liikkua mukana mitään tunnistetietoa sitä käyttäneestä järjestelmästä tai sen käsittelemästä liikenteestä. Toisin sanoen kappaleessa 5.5.5 käsitellyn nimeämisen täytyy olla sen tyyppistä, että se ei paljasta edellä mainittuja seikkoja.

6 Esimerkkisovellus

Tässä luvussa esitellään luvussa 5 kuvattuun arkkitehtuuriin perustuva esimerkkisovellus. Esimerkkisovelluksen tarkoituksena on toimia eräänlaisena todisteena arkkitehtuurin toimivuudesta (engl. *proof of concept*). Tämä esimerkkisovellus asennettiin koeverkkoon, jossa simuloitiin sekä normaalia että epänormaalia liikennettä.



Kuva 6.1: Esimerkkisovelluksen komponentit ja yhteydet.

Kuvassa 6.1 nähdään toteutuneen esimerkkisovelluksen komponentit: sensori, framework, käyttöliittymä ja opettaja. Kuvassa näkyvä tietokanta ei varsinaisesti ole oma komponenttinsa koska framework tallentaan ”.nfx” ja ”.nff” tiedostonsa suoraan levylle. Muuta tietoa ei tallenneta.

Sensori ja opettaja erottavat verkosta kaapatuista paketeista kahdeksan piirrettä. Piirteet ovat lukuja väliltä $[-1, 1]$. Piirrevektorit lähetetään frameworkille, jossa ne syötetään frameworkissa oleville neuroverkoille. Neuroverkot muodostavat piirrevektorisekvenssien perusteella yhden luvun mittaisen vektorin, jonka jatkuva arvo on myös väliltä $[-1, 1]$. Tämä arvo kertoo onko hyökkäys havaittu vai ei. Framework lähettää vektroin käyttöliittymälle ja opettajalle. Jos framework on opetustilassa se

odottaa opettajalta vastaavaa virhevektoria, jonka arvo on samalta väliltä. Opettaja ja sensori eivät voi olla aktiivisia samaan aikaan. Käyttöliittymässä täytyy asettaa hälytyksen kynnysarvo siten, että normaalissa tilanteessa vastaanotetun vektorin arvo pysyy sen alapuolella. Kynnysarvon ylitykset lasketaan hälytyksiksi.

Kuten kuvasta 6.1 voi päätellä, käyttää esimerkkisovellus kappaleessa 5.5.3 kuvattua opetusmallia A.

Esimerkkisovelluksen frameworkin luokkakaavio löytyy liitteestä C.

6.1 Toteutustekniikat

Kappaleessa 5.1 kuvattujen vaatimusten pojalta valittiin esimerkkisovelluksessa käytettävät toteutustekniikat.

6.1.1 Qt

Ensimmäiseksi valittiin ohjelmiston kehittämistä varten kehitysympäristö. Kehitysympäristöksi valikoitui alunperin norjalaisen Trolltechin, nykyisin Nokian, kehittämä Qt. Qt valittiin siksi, että se toimii nykyisin suurimmalla osassa kappaleessa 5.1 luetelluissa ympäristöissä [1], sekä siksi, että se tarjoaa lukuisan määrän hyödyllisiä, ympäristöriippumattomia palveluita (tietoliikenne, käyttöliittymä, jne.) helpottaen siirrettävän ohjelmiston kehittämistä erilaisille alustoille.

Työn toteuttamisessa käytettiin Qt:n versiota 4.7.

6.1.2 Libpcap

Toiseksi valitsin libpcap [2] ja WinPcap [3] kirjastot. Nämä ovat lähes yhteensopivat kirjastot ja ajurit ”raakojen” pakettien kaappaamiseen sekä UNIX- että Windows-ympäristöissä. Näitä kirjastoja käyttämällä vältytään omien verkkoajureiden kehittämiseltä sekä saadaan käyttöön helppo ja hyvin määritelty ohjelmointirajapinta verkkoliikenteen kuunteluun.

Libpcap tarjoaa myös mahdollisuuden paljon käytetyn tiedostoformaatin [52] hyödyntämiseen helposti. Formaattia voidaan käyttää kappaleessa 5.2.3 mainitun hyökkäysdatan ja opetusdatan tallentamiseen. Tästä on helpon integroitavuuden lisäksi se etu, että kyseistä formaattia tukevia muita sovelluksia löytyy todella paljon.

[1] http://qt.nokia.com/products/platform/platforms
[2] http://www.tcpdump.org/
[3] http://www.winpcap.org/

6.1.3 XML

Kolmanneksi valitsin XML-kuvauskielen [4]. XML-kuvauskieltä käytetään neuroverkkojen kuvaamiseen, siirtämiseen ja tallentamiseen. Lisäksi sitä käytetään kuvaamaan ja tallentamaan verkkojen välisiä yhteyksiä fameworkin sisällä. Ideana XML:n käyttö tällaiseen tarkoitukseen ei ole uusi, vaan esimerkiksi Rubtsov ja Butakov [53] ovat kehittäneet oman kielensä vastaavaan tarkoitukseen. Verkkojen tallentamiseen käytetyn tiedostotyypin tunnistaa tunnisteesta ".nnx" (Neural Network XML) ja frameworkin tallentamiseen käytetyn tiedostotyypin tunnisteesta ".nfx" (Neural Framework XML).

Verkon XML voi olla kuvaus jostakin tietystä verkosta ennen opettamista tai opettamisen jälkeen. Ennen opettamista kaikilla neuroneille ei välttämättä ole määritelty painoarvoja mutta opettamisen jälkeen on. Verkon XML voi olla myös verkon "aihio" (engl. *template*), jolloin kyseessä on verkko jolla ei ole nimeä ennen opettamista. Nimen puuttuessa framework muodostaa jokaiselle templatesta luodulle uudelle verkolle yksilöllisen GUID:n perustustuvan nimen. Verkon XML kuvaa verkon sisältämät ryhmät, neuronit, synapsit sekä niiden ominaisuudet. Tällä hetkellä ryhmät, neuronit ja synapsit ovat kaikki XML-elementtejä, ja niiden ominaisuudet on esitetty elementtien parametreina. Ilmeisesti parempi käytäntö olisi välttää parametrien käyttöä ja kuvata myös ominaisuudet omina, sisäkkäisinä elementteinään. Tämän työn kannalta valittu lähestymistapa on kuitenkin riittävä. Lisäksi XML-tiedostosta tulee valittua menetelmää käyttäen tiiviimpi ja mahdollisesti luettavampi. Seuraavassa on lueteltuna verkon XML:n mahdolliset elementit sekä niiden parametrit:

- **network** määrittelee verkon ja kapseloi muut elementit sisäänsä. Network voi sisältää seuraavat parametrit:
  - **name** on verkon tunniste.
- **neuron** määrittelee yhden neuronin ja voi sisältää seuraavat parametrit:
  - **name** on neuronin tunniste,
  - **type** on neuronin tyyppi (*act* = activation, *sum* = summation, *mul* = multiplication) ja
  - **actFun** on activation neuronin aktivaatiofunktion tyypi (*tanh* = hyperbolinen tangentti, *sig* = logistinen sigmoidi).

[4] http://www.w3.org/XML/

- **synapse** määrittelee neuronin yhden synapsin. Se sijaitsee aina **neuron**-lohkon sisällä ja voi sisältää seuraavat parametrit:
  - **source** on lähde neuronin tunniste,
  - **truncated** kertoo katkaistaanko back probacation -algoritmi tässä synapsissa vai ei,
  - **delayed** kertoo onko synapsilla oletuksena yhden syklin viive vai ei,
  - **permanent** kertoo onko synapsi pysyvä, jolloin sitä ei voi optimoida pois ja
  - **weight** on synapsin painoarvo.
- **group** määrittelee ryhmän ja voi sisältää seuraavat parametrit:
  - **name** on ryhmän tunniste ja
  - **visible** on lista ryhmistä, jotka voivat luoda yhteyksiä tähän ryhmään (poikkeuksellisesti elementtinä toteutettu).

Esimerkki verkon XML:stä löytyy liitteestä B.

Koska kappaleessa 5.5.1 mainittua verkon yhteyksien uudelleenjärjestämistä ei ole tällä hetkellä täysin toteutettu, ei ryhmän visible-ominaisuudella ole juuri käyttöä. Kannattaa kuitenkin huomata, että väärin asetettu näkyvyys voi estää käyttäjän määrittelemän yhteyden muodostumisen.

Frameworking XML kuvaa frameworkin sisältämät verkot ja niiden väliset yhteydet. Seuraavassa on lueteltuna frameworkin XML:n mahdolliset elementit sekä niiden parametrit:

- **framework** määrittelee frameworkin ja kapseloi muut elementit sisäänsä. Framework voi sisältää seuraavat parametrit:
  - **name** on frameworkin tunniste.
- **group** määrittelee ryhmän ja voi sisältää seuraavat parametrit:
  - **name** on ryhmän tunniste ja
  - **visible** on lista ryhmistä, jotka voivat luoda yhteyksiä tähän ryhmään (poikkeuksellisesti elementtinä toteutettu).
- **network** määrittelee verkon ja voi sisältää seuraavat parametrit:
  - **name** on verkon tunniste.

Esimerkki frameworkin XML:stä löytyy liitteestä A.

Framework lukee käynnistyessään ".nfx"-tunnisteisesta XML-tiedostosta ladattavien verkkojen nimet, ryhmät ja niiden näkyvyydet. Frameworkin tapauksessa ryhmän visible-ominaisuus on jo nykyisellään hyödyllinen. Tämä johtuu frameworkin tavasta yhdistää verkot toisiinsa (katso kappale 5.5.2). Visible-ominaisuudella voidaan kertoa mihin verkkoihin kunkin ryhmän verkot saavat liittyä niitä ladattaessa tai myöhemmin lisättäessä. Framework yrittää lukea ".nfx"-tiedoston sisältävästä hakemistosta ".nnx"-tunnisteisia tiedostoja joilla on sama nimi kuin ".nfx"-tiedoston network-elementeillä. Löydettyään tiedoston se luo tiedoston kuvaaman verkon ja lisää sen ".nfx"-tiedoston kuvaamaan ryhmään. Lisäämisen jälkeen verkko yhdistetään muihin verkkoihin näkyvyyssääntöjen sallimalla tavalla.

XML sopii neuroverkkojen kuvaamiseen melko hyvin, koska kieli rakentuu hierarkkisesti sisäkkäisistä elementeistä. Elementeillä voi olla erialisia elementin tyypistä riippuvia parametreja. Neuroverkot koostuvat myös sisäkkäisistä elementeistä, eli solmuista sekä niitä yhdistävistä yhteyksistä. Näillä molemmilla on parametreja, jotka täytyy tallentaa. Tällaisten hierarkkisten rakenteiden tallentamisessa XML on erittäin hyödyllinen. Lisäksi XML on ihmisen kannalta melko helposti luettavaa ja siksi sen käyttö helpottaa kehitystyötä.

Yksi syy XML:n käyttöön on Qt:stä löytyvä hyvä XML-tuki. Haittapuolena tekstipohjaisella XML:llä on sen sen vaatima tila. Tästä syystä binäärinen tiedon esittämien ja tallentaminen olisi tehokkaampaa varsinkin mobiilissa ympäristössä. Edut ovat kuitenkin tässä tapauksessa mielestäni haittoja merkittävämmät ja XML:n käyttö esimerkkijärjestelmässä perusteltua.

Kappaleessa 4.4 mainittuja geneettisiä menetelmiä silmällä pitäen XML:stä täytyy joko voida palauttaa jokin geneettisten algoritmien kannalta käytettävä kuvaus, tai sitten se on tallennettava verkon mukana osaksi XML:ää. Tällainen keinotekoinen DNA tarvitaan, koska XML ei sellaisenaan sovellu kovin hyvin geneettisten algoritmien käsiteltäväksi. Tämän keinotekoisen DNA:n tulisi sisältää verkon luomiseen alunperin käytetty informaatio (katso lisätietoja Salakosken työstä [37]).

6.2 Koejärjestely

Koska langattomille verkoille ei ole olevassa vastaavaa testidataa kuin langallisten verkkojen ID-järjestelmien tutkimuksessa ja testaamisessa usein käytetty DARPA99(KDD 99) (katso kuvaus Stolfo sivustolta [44]) jouduttiin rakentamaan testiverkko, jossa järjestelmää kokeiltiin.

Koska uusien verkkojen automaattista luomista ei ole esimerkkisovelluksessa

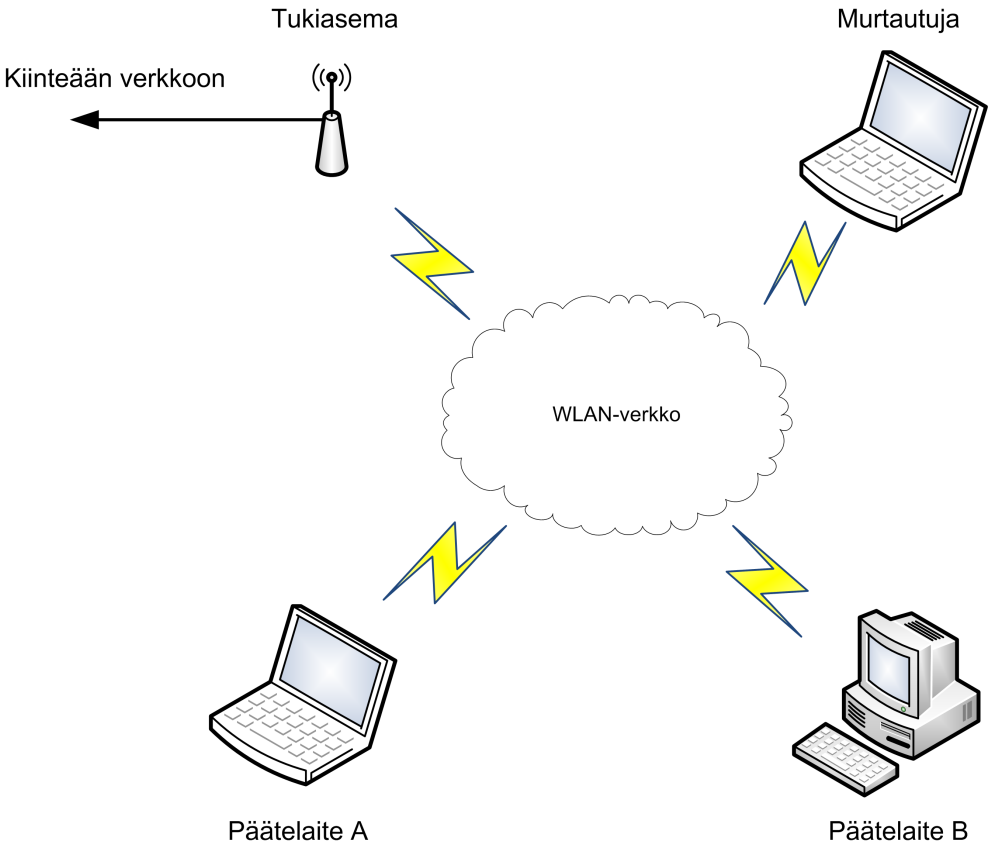
vielä toteutettu, jouduttiin parasta neuroverkkomallia hakemaan kokeilemalla.

Parhaan verkkomallin löydyttyä kokeiltiin sen toimivuutta suorittamalla joitakin testihyökkäyksiä.

Hyökkäysten liskäksi kokeilin järjestelmän hajauttamista useammalle laitteelle.

6.2.1 Testiverkko

Järjestelmän toimivuuden kokeilemiseksi rakennettiin pieni testiverkko. Testiverkon rakenne on esitetty kuvassa 6.2.



Kuva 6.2: Testiverkkon rakenne.

Testiverkko koostuu kahdesta päätelaitteesta (Päätelaite A ja Päätelaite B), tukiasemasta (Tukiasema) ja murtautumiseen käytetystä päätelaitteesta (Murtautuja).

Tukiasemana toimi langalliseen verkkoon liitetty Buffalo Airstation Wireles G (WHR-HP-G54) tukiasema.

Päätelaite A oli langattomalla verkkosovittimella (Intel WiFi Link 4965AGN) varustettu kannettava tietokone. Käyttöjärjestelmänä koneella oli Windows Vista.

Päätelaite B oli langattomalla verkkosovittimella (A-link WL54USB) varustettu

tietokone. Käyttöjärjestelmänä koneella oli Linux (Ubuntu 11.04 muokatulla 2.6.38.8 ytimellä [5]).

Murtautuja oli langattomalla verkkosovittimella (Buffalo WLI-UC-GN) varustettu kannettava tietokone. Käyttöjärjestelmänä koneella oli Linux (Ubuntu 11.04 2.6.38.11 ytimellä).

Kappaleessa 5.2 kuvatut IDS-järjestelmän osat sijoitettiin hyökkäyksentunnistuskokeissa verkon laitteille siten, että Päätelaite B:llä toimivat sensori, opettaja, tietokanta, framework ja käyttöliittymä. Päätelaite A toimi liikennegeneraattorina ja hyökkäysten kohteena. Murtautuja-koneella suoritettiin erilaisia hyökkäyksiä käyttäen kappaleessa 6.2.3 lueteltuja ohjelmistoja.

Hajauttamiskokeessa kokeiltiin järjestelmän osien yhteistoimintaa komponenttien sijaitessa ei koneilla. Tässä kokeessa käyttöliittymä sijoitettiin Murtautuja-koneelle ja Päätelaite A:lla sijainneelle virtuaaliselle koneelle. Muuten komponentit sijaitsivat samoin kun hyökkäyksentunnistuskokeissa. Kommunikaatio Murtautuja-koneelle sijoitetun käyttöliittymän ja Päätelaite B:lle sijoitetun frameworkin välillä kulkei langallista verkkoa pitkin kappaleessa 5.5.7 kuvattujen syiden takia. Kappaleessa kuvattujen parannusten jälkeen liikenne voitaneen kuitenkin tulevaisuudessa kuljettaa langattoman verkon kautta.

Verkossa käytettiin IEEE 802.11g -standardin mukaisia laitteita. Salaustekniikkana toimi 128-bittinen WEP.

6.2.2 Neuroverkkotyypit

Järjestelmää kokeiltaessa huomattiin, että neuroverkon tyypillä on suuri merkitys verkon kyvylle oppia ja tunnistaa hyökkäyksiä. Kokeilin manuaalisesti muokata verkosta joitakin erilaisia versioita, mutta nopeasti kävi ilmi, että manuaalinen muokkaaminen on liian työlästä. Verkkojen luomiseen ja kokeiluun tarvitaan ehdottomasti kappaleessa 4.4 esitettyjä tekniikoita.

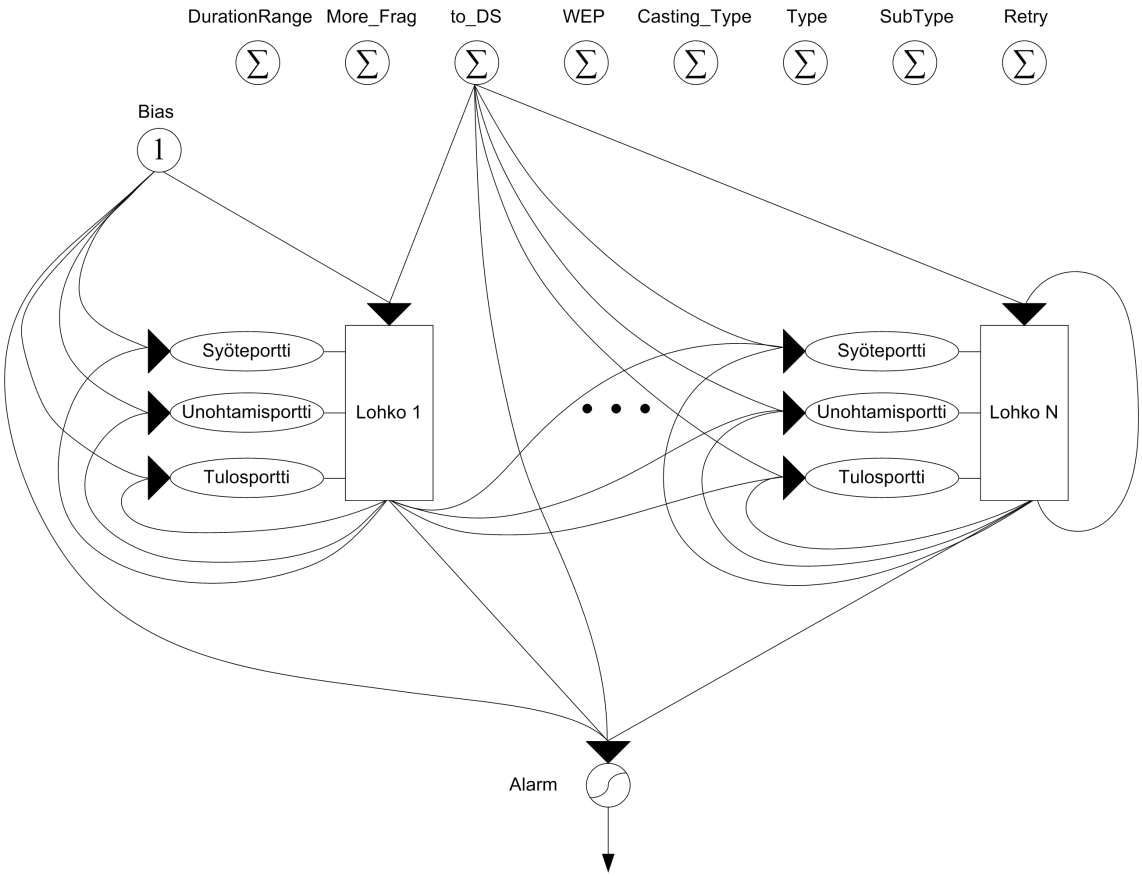
Ennen lopullista testiä kokeiltiin LSTM-lohkoja sekä ”kurkistusyhtyksillä” että ilman. Edellisen lisäksi kokeiltiin verkkoja, joissa syöteneuronit eivät olleet tai olivat liitettynä LSTM-lohkojen lisäksi myös suoraan tulosneuroniin. Lopuksi kokeiltiin vielä muuttaa LSTM-lohkojen määrää kolmesta kahteenkymmeneen.

Verkon tyypin lisäksi kokeiltiin muuttaa oppimiskerrointa ja momenttia.

Lopulliseksi verkoksi valittiin kahdellakymmenellä LSTM-lohkolla varustettu verkko, jonka LSTM-lohkoissa käytettiin ”kurkistusyhteyksiä” ja syöteneuronit oli liitetty LSTM-lohkojen syötteiden ja kaikkien porttien lisäksi ainoaan tulosneuro-

[5] http://www.aircrack-ng.org/doku.php?id=zd1211rw

niin. Myös kaikkien LSTM-lohkojen tulokset oli kytketty niin omiin kuin toisenkin lohkojen portteihin ja toisten lohkojen syötteeseen. Esimerkkiverkko on esitetty kaaviona kuvassa 6.3 ja XML-muodossa liitteessä B. Oppimiskertoimeksi valittiin lopulta 0,1 ja momentiksi 0,35.



Kuva 6.3: Testatun neuroverkon rakenne. Kuva ei sisällä kaikkia yhteyksiä.

6.2.3 Suoritetut hyökkäykset

Esimerkkisovellusta oli alun perin tarkoitus kokeilla suorittamalla kaikkia kappaleessa 2.3 kuvattuja hyökkäystyyppejä. Kävi kuitenkin ilmi, ettei kaikkia hyökkäyksiä voida suorittaa kovin helposti, joten päätin jättää Duration ja ChopChop -hyökkäykset pois.

Syy ChopChop-hyökkäyksen poisjättämiseen on se, että sen havaitseminen vaatisi välttämättä ”IsWepValid”-piirrettä, jota ei kuitenkaan pystytty toteuttamaan. Piirteen toteuttaminen tarkoittaisi käytännössä ICV:n oikeellisuuden tarkastamista, ja se taas vaatisi paketin salauksen purkamista. Valitsemani lähestymistapa kuitenkin nojasi raakojen 802.11 -pakettien kaappaamiseen monitor-tilassa. Päätin siis

työmäärää vähentääkseni luopua kyseiseen hyökkäyksen kokeilemisesta. Korvasin ”IsWepValid”-piirteen ”Retry”-piirteellä.

Syy Duration-hyökkäyksen pois jättämiseen oli yksinkertaisesti se, ettei käyttämäni Aircrack-ng ohjelmisto tukenut suoraan kyseisen hyökkäyksen toteuttamista. Olisi tietysti ollut mahdollista käyttää jotakin muuta ohjelmistoa kappaleessa 2.3 kuvattujen räätälöityjen pakettien lähettämiseen, mutta jälleen kerran päätettiin ajan säästämiseksi jättää kyseinen hyökkäys kokeilematta.

Kuten jo edellä mainittiin, hyökkäykset toteutettiin Aircrack-ng ohjelmistopaketin sisältämillä apuohjelmilla [54].

Lisätietoa deauthentication [6] ja fragmentation [7] hyökkäysten suorittamisesta löytyy ohjelmiston kotisivuilta [54].

6.2.4 Suoritetut kokeet

Tarkoituksena oli alun perin koota nelikenttä vääristä positiivisista, vääristä negatiivisista, oikeista negatiivisista ja oikeista positiivisista tunnistuksista useammalle verkkotyypille. Suunnitelmasta kuitenkin luovuttiin suuren työmäärän vuoksi. Niinpä päädyttiin lyhyesti kokeilemaan kappaleessa 6.2.3 mainittuja hyökkäyksiä kappaleen 6.2.2 neuroverkkotyypeillä. Parhaaksi osoittautuneen verkon toimintaa normaalissa sekä hyökkäystilanteessa kokeiltiin lyhyellä testijaksolla. Tämä on mielestäni täysin riittävä koesarja, koska työn tarkoituksena ei ollut optimoida valmiin järjestelmän suorituskykyä vaan toteuttaa toimiva prototyyppijärjestelmä.

Testatut neuroverkot alustettiin satunnaisilla painoarvoilla. Opetuksessa käytettiin valvottua opetusta. Järjestelmälle opetettiin ensin koeverkon normaali toiminta altistamalla se normaalille liikenteelle opetustilassa. Opetusjakson kesto vaihteli noin 100 kaapatusta paketista yli 6000 kaapattuun pakettiin ja verkkoa pyrittiin sinä aikana käyttämään normaalisti, kuten WWW-selailuun. Lopullisessa testissä päädyin opettamaan verkkoa pienissä jaksoissa ja kokeilemaan miten se tunnisti kappaleessa 6.2.3 kuvattuja hyökkäyksiä.

Muiden opetusmetodien käyttö, sekä varsinaisten hyökkäysten opettaminen rajattiin ajan puutteen vuoksi kokeilun ulkopuolelle.

Lisäksi kokeilin parantaa normaalin toiminnan ja hyökkäyksen välistä eroa ylimääräisellä varsinaista opetusta edeltävällä opetusjaksolla. Siinä ajatuksena oli alustaa verkko tunnistamaan kaikki syötteet hyökkäyksiksi asettamalla tavoitetaso hyökkäystä vastaavaksi ja syöttämällä verkolle satunnaisia syötteitä. Tämä ei kuitenkaan

[6] http://www.aircrack-ng.org/doku.php?id=deauthentication
[7] http://aircrack-ng.org/doku.php?id=fragmentation

jostakin syystä toiminut aivan odotetulla tavalla.

Käytettyä liikennettä ei tallennettu millään tavoin lähinnä tiedon valtavan määrän takia. Hyökkäysstatistiikkaa ei voitu tallentaa kyseisen ominaisuuden puuttumisen vuoksi. Hyökkäysten havaitsemista arvioitiin käyttöliittymän laskurista sekä silmämääräisesti käyttöliittymän palkkia tarkkailemalla. Laskuri laski kuinka monta kertaa verkon syöte nousi yli asetetun kynnysarvon. Jos asetettu kynnysarvo ylitettiin hyökkäyksen aikana tai heti sen jälkeen tulkittiin se oikeaksi positiiviseksi tunnistukseksi. Jos taas arvo ei ylittynyt hyökkäyksen aikana, tulkittiin se vääräksi negatiiviseksi tunnistukseksi. Jos arvo ylittyi selvästi hyökkäyksen ulkopuolella, tulkittiin se vääräksi positiiviseksi tunnistukseksi.

Testit suoritettiin useita eri kertoja koska verkot alustetaan satunnaisesti. Satunnaisesta alkutilanteesta johtuen verkko ei aina kykene oppimaan vaikka sillä siihen edellytykset olisikin. Löydettyäni mielestäni tyydyttävästi toimineen verkon ajoin sille kappaleen 6.2.3 hyökkäyksiä kymmenen kertaa. Lisäksi kokeilin seurata toimintaa normaalitilanteessa noin 30000 kaapatun paketin, eli noin 17 minuutin ajan.

Hyökkäysten lisäksi kokeiltiin järjestelmän hajauttamista. Tämä kokeilu suoritettiin käynnistämällä järjestelmän eri komponentteja eri tietokoneilla. Tässä kokeilussa tosin käytin langallista verkkoa kappaleessa 5.5.7 kuvatun protokollan puutteen vuoksi. Tämä ei tosin vaikuta lopputulokseen siinä mielessä, että langaton ja langallinen verkko eivät ohjelmiston näkökulmasta eroa toisistaan. Komponentteina käytin yhtä sensoria, yhtä opettajaa, yhtä frameworkia ja kahta käyttöliittymää.

6.2.5 Tulokset ja analyysi

Kappaleessa 6.2.4 kuvattujen mittausten perustella koottiin nelikentän 6.1, jossa esitetään oikeat positiiviset, väärät positiiviset ja väärät negatiiviset. Oikeiden negatiivisten laskeminen jätettiin tekemättä. Tämä johtuu siitä, että ennen laskemista pitäisi määritellä tulkitaanko yksi kaapattu paketti, tutkittu minuutti, tai jokin muu vastaava tunnistukseksi. Nelikentän oikeat positiiviset ja väärät negatiiviset -kentät kuvaavat yksittäisiä hyökkäyksiä siten, että D:llä merkitty suhdeluku kertoo deauthentication-hyökkäysten kentän mukaisen määrän verrattuna kokonaismäärään ja F:llä merkitty suhdeluku Fragmentation-hyökkäyksen kentän mukaisen määrän verrattuna kokonaismäärään. Väärät positiiviset -kenttä kuvaa hälytyksiä normaalin toiminnan aikana.

Kuten huomataan, pystyi esimerkkiverkko tunnistamaan hyökkäykset täysin aukottomasti. Ainoaksi ongelmaksi jäi 16 väärää positiivista. Nämä, noin kerran minuutissa ilmenneet väärät positiiviset saattavat johtua liian lyhyestä opetusjaksosta

	Positiiviset	**Negatiiviset**
Oikeat	D:10/10, F:10/10	-
Väärät	16	D:0/10, F:0/10

Taulukko 6.1: Testin tulokset.

ja tästä johtuen opetuksessa huomaamatta jääneestä normaalista sanomasekvenssistä, jota verkko nyt piti epänormaalina. Kaikesta huolimatta tulos ylittää odotukseni ja sen perusteella LSTM-verkko näyttää kykenevän myös verkkoliikenteen analysointiin.

Myös järjestelmän hajauttaminen toimi loistavasti. Järjestelmän komponentit tunnistivat toisensa ja muodostivat yhteyden keskenään häiritsemättä toisiaan riippumatta missä aliverkon koneessa ne oli käynnistetty.

Testien perusteella ilmeni myös seuraavia huomioita:

- Fragmentation-hyökkäysten havaitseminen näyttää olevan paljon helpompaa kuin de-authentication-hyökkäysten. Melkein jokainen satunnaisesti alustettu verkko oppi tunnistamaan Fragmentation-hyökkäykset täydellisesti, mutta de-authentication-hyökkäyksiä tunnistavia verkkoja löytyi yllättävän vähän. Ilmeisesti de-authentication-hyökkäys eroaa vähemmän normaalista liikenteestä kuin Fragmentation-hyökkäys.

- LSTM-lohkojen lisäämien vähentää verkon tuloksen huojuntaa, mutta saattaa jostakin syystä johtaa helposti täysin havaitsemiseen kykenemättömään verkkoon. Tämä saattaa johtua niin sanotusta ylioppimisesta kuten kappaleessa 4.6.2 mainittiin.

- LSTM-lohkojen määrä osoittautui merkittävimmäksi eroksi eri verkkojen välillä.

- Opetusjakson pidentäminen saattaa huonontaa hyökkäyksien havaitsemista, mutta toisaalta vääriä tunnistuksia tulee vähemmän. Pitkässä opetuksessa verkko approksimoi liikaa ja oppii pitämään lähes kaikkea normaalina. Tämä saattaa liittyä opetettujen luokkien vähyyteen. Verkolle opetettiin vain yksi luokka (normaali), jota kohti koko verkko hiljalleen siirtyy muiden opetettavien luokkien puuttuessa.

- Edellisen kohdan ongelmaa korjaamaan tarkoitettu, varsinaista opetusta edeltävä, verkon opettaminen satunnaisella syötteelle ja hyökkäystä vastaavalla mallilla saattaa huonontaa tulosta. Vaatisi syvällisempää analyysiä selvittää mitä verkossa tapahtuu. Ongelman ratkaisemiseksi kannattaisi ehkä kokeilla näiden kahden opetusjakson (normaali liikenne ja satunnainen syöte) jaksottaista vaihtelua.

- Opetusvaiheessa verkko minimoi virheen todella nopeasti (alle 100 opetuskierrosta) lähes nollaan. Tästä johtuen jäljellä olevan virheen määrä ei tässä tapauksessa ole sopiva kriteeri opettamisen lopettamiselle, vaan opetusta täytyy jatkaa kunnes verkko on oppinut kaikki normaalin tilanteen mahdolliset tapahtumat.

Testiaineisto jäi valitettavan pieneksi. Tämä on mielestäni kuitenkin hyväksyttävää, koska pidin itse arkkitehtuurin ja siihen liittyvien ideoiden kehittämistä tärkeämpänä kuin järjestelmän lopullista suorituskykyä. Pienuudestaan huolimatta aineistosta voitiin kutenkin saada yllä esitettyjä suuntaa antavia tuloksia. Saadut tulokset kertovat, että jatkokehittämällä järjestelmästä voitaisii mahdollisesti saada käyttökelpoisen työkalun paitsi hyökkäysten havaitsemiseen, myös muihin oppivista ja itseorganisoituvista järjestelmistä hyötyviin sovelluksiin.

7 Yhteenveto

Työssä tutustuttiin kattavasti langattomaan lähiverkkotekniikkaan, tunkeutumisen tunnistustekniikkaan ja neuroverkkoihin. Lisäksi suunniteltiin yleiskäyttöinen alusta hajautettujen neuroverkkojen suunnitteluun, opettamiseen ja käyttämiseen. Tätä alustaa hyväksi käyttäen toteutettiin langattoman lähiverkon tunkeutumisen tunnistusjärjestelmän prototyyppi, jota testattiin erilaisilla hyökkäystyypeillä.

7.1 Tulokset

Mielestäni saavutin työlleni asettamani tavoitteet. Esittelin hajautetun, itseorganisoituvan ja neuroverkkoihin perustuvan langattomien verkkojen tunkeutumisen havaitsemisjärjestelmän arkkitehtuurin. Arkkitehtuuri sisältää monia ehdotuksia joiden tavoitteena on tehdä järjestelmän käytöstä helppoa ja siirtää vastuuta järjestelmän ylläpidosta maallikkokäyttäjältä järjestelmälle itselleen. Voidaan ajatella, että järjestelmä voi itse hoitaa asiota, joista järjestelmän käyttäjällä ei ole mitään käsitystä. Tällä tavoin voidaan välttää erilaisiin tietoturvaohjelmistoihin liittyvät ohjelmiston esittämät vaikeat kysymykset, joihin järjestelmän käyttäjällä ei usein ole mitään valmiuksia vastata. Tällöin voi olla parempi, että keinoäly, jolla on edes pieni käsitys siitä mikä on normaalia ja mikä ei, suorittaa tulkinnan.

Testiaineiston vähyydestä huolimatta voidaan mielestäni tehdä esitettyjä ratkaisuja puoltava johtopäätös. Kokeilujen perusteella ei paljastunut mitää, mikä pakottaisi esitetyn järjestelmän hylkäämiseen. Tulokset olivat päin vastoin rohkaisevia, ja suorastaan kehottavat esitettyjen ratkaisujen jatkokehittämiseen.

LSTM-verkon valitseminen oli harkittu riski koska kirjallisuudesta ei löytynyt yhtään mainintaa kyseisen verkkomallin soveltamisesta tällaiseen tehtävään. Riski kannatti mielestäni kuitenkin ottaa, koska pääsin tutustumaan uuteen ja mielestäni tällä hetkellä mielenkiintoisimpaan neuroverkkotekniikkaan. Saamani arvokkaan kokemuksen lisäksi valitsemani yleiskäyttöinen toteutustapa jättää auki monia jatkokehityssuuntia.

7.2 Jatkokehitysideoita

Koska Hochreiterin ja Schmidhuberin kehittämää tehokkaampaa LSTM-opetusalgoritmia [30] ei ehditty toteuttaa tämän työn puitteissa olisi mielenkiintoista jatkaa alustan kehittämistä kyseistä algoritmia silmällä pitäen. Tämä voisi tapahtua siten, että alusta tukisi neuroverkon aliverkkoja tai ryhmiä, joiden sisäisenä opetusalgoritmina käytettäisiin kyseistä menetelmää vaikka muuten käytettäisiin nyt toteutettua BPTT-algoritmia.

Haluaisin toteuttaa myös muut työssäni esittämät ideat, kuten verkkojen geneettinen muodostaminen (kappaleessa 4.4), verkkojen komitea ja asiantuntijaverkko (kappaleessa 4.7), erilaiset opetusmetodit (kappaleessa 5.5.3), protokollan tehostaminen (kappaleessa 5.5.7) ja itse frameworkiin liittyvät toteuttamatta jääneet ominaisuudet.

Itse näen toteutetun tyyppisessä järjestelmässä suurta potentiaalia. Saman tyyppisen järjestelmän varaan voisi kehittää jopa uusia bisnesmalleja: voitaisiin perustaa neuroverkko-farmeja, joissa voisi jalostaa erilaisiin tarkoituksiin soveltuvia uusia neuroverkkotyyppejä ja myydä niitä. Toinen vaihtoehto olisi tarjota erialisia neuroverkkopalveluita palvelimella tai pilvessä: jokin yritys voisi esimerkiksi tarjota asiakkailleen mahdollisuutta ajaa omia neuroverkkojaan tai liittyä valmiisiin neuroverkkoihin.

Tulevaisuuden sovellukset voisivat olla jotakin täysin muuta kuin nyt esitelty tunkeutumisen havaitsemisjärjestelmä. Vain mielikuvitus rajoittaa uusien sovellutusten kehittämistä.

Lähteet

[1] Yongguang Zhang, Wenke Lee ja Yi-An Huang, *Intrusion Detection Techniques for Mobile Wireless Networks*, Wireless Networks, 9 (2003), s. 545–556.

[2] Hongyu Yang, Lixia Xie ja Jizhou Sun, *Intrusion Detection Solution to WLANs*, kirjassa ”Proceedings of the IEEE 6th Circuits and Systems Symposium on Emerging Technologies: Frontiers of Mobile and Wireless Communication”, IEEE, s. 553–556, 2004.

[3] Zhanfei Ma, Xuefeng Zheng, Dongkui Li, Xuebao Li ja Liping Yang, *Distributed Architecture for Intrusion Detection System Based on Multi-SoftMan*, kirjassa ”The 5th International Conference on Wireless Communications, Networking and Mobile Computing(WiCOM 2009)”, IEEE, s.1–4, 2009.

[4] Jia-Jun Xiong ja Jing Zhang, *A kind of multilayer intrusion detection system using mobile agent*, kirjassa ”Second International Conference on Machine Learning and Cybernetics”, IEEE, s. 1951–1955, 2003.

[5] Jin-Gang Cao ja Gu-Ping Zheng, *Research on distributed intrusion detection system based on mobile agent*, kirjassa ”Proceedings of the Seventh International Conference on Machine Learning and Cybernetics” IEEE, s. 1394–1399, 2008.

[6] Mouhcine Guennoun, Khalil El-Khatib, *A Scalable Wireless Intrusion Detection System*, (IJCSIS) International Journal of Computer Science and Information Security, 1 (2009), s. 53–62.

[7] Helena Telkänranta, *Eläimetkin oppivat oivaltaen*, Tiede, 1 (2005).

[8] IEEE 802.11 Working Group, ”Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications”, IEEE, New York, 2007.

[9] Wi-Fi Alliance, ”Discover and Learn”, saatavilla WWW-muodossa <URL: `http://www.wi-fi.org/discover_and_learn.php`>, viitattu 19.4.2011.

[10] IEEE, ”IEEE 802.11™: WIRELESS LOCAL AREA NETWORKS (LANs)”, saatavilla WWW-muodossa <URL: `http://standards.ieee.org/about/get/802/802.11.html`>, viitattu 18.4.2011.

[11] Christopher Low, ”Understanding Wireless attacks & detection”, SANS Institute, Bethesda USA, 2005.

[12] IEEE 802.11 Working Group, ”Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications” IEEE, New York, 1999, saatavilla PDF-muodossa <URL: http://pdos.csail.mit.edu/decouto/papers/802.11.pdf>.

[13] Gianluigi Ciambriello ja Savino Ordine, ”Why you should not trust Wired Equivalent Privacy (WEP)”, Mälardalen University, Västerås Sweden, 2008.

[14] Karen Scarfone ja Peter Mell, *Guide to Intrusion Detection and Prevention Systems (IDS)*, National Institute of Standards and Technology, Gaithersburg USA, 2007.

[15] Communications Security Establishment Canada (CSEC), *802.11 Wireless LAN Vulnerability Assessment (ITSPSR-21A)*, Communications Security Establishment Canada, Ottawa Canada, 2009.

[16] Brad Antoniewicz, *802.11 Attacks*, McAfee Inc., Santa Clara USA, 2008.

[17] Mohteshim Hussain, ”Passive & active attacks against wireless LAN’s”, University of Hertfordshire, England U.K, 2005.

[18] Jungwoo Ryoo, Young B. Choi, Tae Hwan Oh ja Gregory Corbin, *A multi-dimensional classification framework for developing context-specific Wireless Local Area Network attack taxonomies*, International Journal of Mobile Communications, 7 (2009).

[19] Mouhcine Guennoun, Aboubakr Lbekkouri ja Khalil El-Khatib, *Selecting the Best Set of Features for Efficient Intrusion Detection in 802.11 Networks*, kirjassa ”ICTTA 2008. 3rd International Conference on Information and Communication Technologies: From Theory to Applications”, IEEE, s. 1–4, 2008.

[20] Sara Bury, Paul Smith, Dwight Makaroff, Nicholas J.P. Race ja David Hutchison, ”Challenges for Intrusion Detection in Community Wireless Mesh Networks”, saatavilla PDF-muodossa <URL: http://www.dcs.gla.ac.uk/Conferences/EuroSys2008/posters/14_Dwight_Makaroff.pdf>, 4.4.2008.

[21] Scott Fluhrer, Itsik Mantin ja Adi Shamir, *Weaknesses in the Key Scheduling Algorithm of RC-4*, kirjassa ”Selected Areas in Cryptography, Lecture Notes in Computer Science”, Springer, Berlin Germany, s. 1–24, 2001

[22] M.A.C.Sandarenu, ”Using Artificial Neural Networks for Solving Complex Problems”, University of Moratuwa, Moratuwa Sri Lanka, 2008.

[23] Christopher M. Bishop, ”Neural Networks for Pattern Recognition”, Oxford University Press, Oxford U.K, 1995.

[24] Jake Ryan, Meng-Jang Lin ja Risto Miikkulainen, *Intrusion Detection with Neural Networks*, kirjassa ”Advances in Neural Information Processing Systems”, (Michael I. Jordan, Michael J. Kearns ja Sara A. Solla, toim.), MIT Press, Cambridge USA, s. 943–949, 1998.

[25] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke ja Jürgen Schmidhuber, *A Novel Connectionist System for Unconstrained Handwriting Recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 31 (2009), s. 855–868.

[26] Laurene Fausett, ”Fundamentals of Neural Networks, Architectures, Algorithms, and Applications”, Prentice-Hall, New Jersey USA, 1994.

[27] Daniel Rios, ”ARTIFICIAL NEURAL NETWORKS - A neural network tutorial”, saatavilla WWW-muodossa, <URL: `http://www.learnartificialneuralnetworks.com/`>, viitattu 23.1.2011.

[28] R.J.Frank, N.Davey ja S.P.Hunt, *Time Series Prediction and Neural Networks*, Journal of Intelligent and Robotic Systems, 31 (2001), s. 91–103.

[29] Sepp Hochreiter, Yoshua Bengio, Paolo Frascioni ja Jürgen Schmidhuber, *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies*, kirjassa ”Field Guide to Dynamical Recurrent Networks”, (J. Kolen ja S. Kremer, toim.), Wiley-IEEE Press, New York USA, 2001.

[30] Sepp Hochreiter ja Jürgen Schmidhuber, *Long Short-Term Memory*, Neural computation, 9 (1997), s. 1735–1780.

[31] Felix A. Gers, Nicol N. Schraudolph ja Jürgen Schmidhuber, *Learning precise timing with lstm recurrent networks*, The Journal of Machine Learning Research, 3 (2003), s. 115–143.

[32] Felix A. Gers, Jürgen Schmidhuber ja Fred Cummins, *Learning to forget: continual prediction with LSTM*, kirjassa ”ICANN 99. Ninth International Conference on Artificial Neural Networks” IEEE, New York USA, s. 850–855, 1999.

[33] Wen-Biao Gan, Elaine Kwon, Guoping Feng, Joshua R Sanes ja Jeff W Lichtman, *Synaptic dynamism measured over minutes to months: age-dependent decline in an autonomic ganglion*, Nature Neuroscience, 6 (2003), s. 956–960.

[34] Mikael Bodén *A guide to recurrent neural networks and backpropagation*, kirjassa "The DALLAS project. Report from the NUTEK-supported project AIS-8: Application of Data Analysis with Learning Systems, 1999-2001", (Anders Holst, toim.), SICS, Kista Sweden, 2002, s. 17–26.

[35] Justin Bayer, Daan Wierstra, Julian Togelius ja Jürgen Schmidhuber, *Evolving memory cell structures for sequence learning*, kirjassa "ICANN '09 Proceedings of the 19th International Conference on Artificial Neural Networks: Part II" Springer-Verlag, Berlin Germany, s. 755–764, 2009.

[36] Conrad Tiflin ja Christian W. Omlin, *LSTM Recurrent Neural Networks for Signature Verification*, kirjassa "The Southern African Telecommunication Networks & Applications Conference (SATNAC 2003)", University of the Western Cape, Cape Town South Africa, 2003.

[37] Harri Salakoski, "Neuroverkkojen tuottaminen geneettisen algoritmin avulla", Tampereen yliopisto, Tampere, 2002.

[38] Xuanwu Zhou, *Bionic evolution based intrusion detection system*, kirjassa "CCDC '09. Chinese Control and Decision Conference", IEEE, New York USA, s. 1355–1360, 2009.

[39] J F Dale Addison, Stefan Wermter ja Garen Z Arevian, *A Comparison of Feature Extraction and Selection Techniques*, kirjassa "Proceedings of the International Conference on Artificial Neural Networks", Springer, Berlin Germany, s. 212–215, 2003.

[40] Paul Bourke, "BMP image format", saatavilla WWW-muodossa <URL: `http://paulbourke.net/dataformats/bmp/`>, viitattu 19.2.2011.

[41] Athira. M. Nambiar, Asha Vijayan ja Aishwarya Nandakumar, *Wireless intrusion detection based on different clustering approaches*, kirjassa "Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India", ACM, New York USA, s. 42–48, 2010.

[42] Shi Zhong, Taghi M. Khoshgoftaar ja Shyarn V. Nath, *A clustering approach to wireless network intrusion detection, Tools with Artificial Intelligence*, kirjassa "ICTAI

2005. 17th IEEE International Conference on Tools with Artificial Intelligence" IEEE, New York USA, s. 190–196, 2005.

[43] Magdalena Balazinska ja Paul Castro, *Characterizing Mobility and Network Usage in a Corporate Wireless Local-Area Network*, kirjassa "MobiSys '03 Proceedings of the 1st international conference on Mobile systems, applications and services", ACM, New York USA, s. 303–316, 2003.

[44] Salvatore J. Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis ja Philip K. Chan, "KDD-CUP-99 Task Description", saatavilla WWW-muodossa <URL: `http://kdd.ics.uci.edu/databases/kddcup99/task.html`>, viitattu 19.2.2011.

[45] Irina Klevecka ja Janis Lelis, *Pre-Processing of Input Data of Neural Networks: The Case of Forecasting Telecommunication Network Traffic*, Telektronikk, 3 (2008), s. 168–178.

[46] Nikko Ström, *Sparse Connection and Pruning in Large Dynamic Artificial Neural Networks*, KTH (Royal Institute of Technology), Tukholma Ruotsi, 1997.

[47] EDinformatics, *THE ADOLESCENT BRAIN – WHY TEENAGERS THINK AND ACT DIFFERENTLY–*, saatavilla WWW-muodossa <URL: `http://www.edinformatics.com/news/teenage _brains.htm`>, viitattu 19.2.2011.

[48] Yinyin Liu, Janusz A. Starzyk ja Zhen Zhu, *Optimizing number of hidden neurons in neural networks*, kirjassa "Proceedings of the 25th conference on Proceedings of the 25th IASTED International Multi-Conference: artificial intelligence and applications", ACTA Press, Anaheim USA, 2007, s. 121–126.

[49] George Bebis ja Michael Georgiopoulos, *Optimal feed-forward neural network architectures*, IEEE Potentials, (1994), s. 27–31.

[50] Wenjian Wang, Weizhen Lu, Andrew Y T Leung, Siu-Ming Lo, Zongben Xu ja Xiekang Wang, *Optimal Feed-Forward Neural Networks Based on the Combination of Constructing and Pruning by Genetic Algorithms*, kirjassa "IJCNN '02. Proceedings of the 2002 International Joint Conference on Neural Networks" IEEE, New York USA, 2002, s. 636–641.

[51] Klaus Wilhelm, *Nuku hyvin, aivot siivoavat*, Tieteen Kuvalehti, 9 (2008), s. 68–69.

[52] ”Libpcap File Format”, saatavilla WWW-muodossa <URL: http://wiki.wireshark.org/Development/LibpcapFileFormat>, viitattu 15.9.2011.

[53] D.V. Rubtsov ja S.V. Butakov, *XML-Based Format for Trained Neural Network Definition*, Academic Open Internet Journal, 4 (2001), s. 1–7.

[54] ”Aircrack-ng”, saatavilla WWW-muodossa <URL: http://www.aircrack-ng.org>, viitattu 19.2.2011.

A NFX-esimerkki

Esimerkki frameworkin kuvaavasta NFX-tiedostosta. Framewokrin nimi on ”framework1” ja se sisältää kaksi ryhmää. Ryhmässä ”Active” on yksi verkko ”Network1”. Ryhmä ”external” on erikoisryhmä, johon framework dynaamisesti lisää ulkoisia ryhmiä. ”Active” -ryhmän ”visible”-elementti kertoo, että siinä oleviin verkkoihin voi liittyä ainoastaan ”external” -rymässä olevia verkkoja. ”external” -ryhmän ”visible” -elemetti keroo, että siihen voi liittyä minkä tahansa muun ryhmän verkkoja.

```
<?xml version="1.0" encoding="UTF–8"?>
<framework name="framework1">
  <group name="Active">
    <visible>external</visible>
    <network name="Network1"/>
  </group>
  <group name="external">
    <visible>*</visible>
  </group>
</framework>
```

B NNX-esimerkki

Esimerkki verkon kuvaavasta NNX-tiedostosta. Verkko sisältää syöte-, piilo- sekä tuloskerroksen. Syötekerros koostuu kahdeksasta syöteneuronista. Piilokerros koostuu kolmesta LSTM-lohkosta. Tuloskerros koostuu yhdestä tulosneuronista.

```
<?xml version="1.0" encoding="UTF-8"?>
<network name="Network1">
  <group name="input">
    <neuron name="DurationRange" type="sum" />
    <neuron name="More_Frag" type="sum" />
    <neuron name="to_DS" type="sum" />
    <neuron name="WEP" type="sum" />
    <neuron name="Casting_Type" type="sum" />
    <neuron name="Type" type="sum" />
    <neuron name="SubType" type="sum" />
    <neuron name="Retry" type="sum" />
  </group>
  <group name="LSTM">
    <visible>.,input</visible>
    <group name="Block1">
      <visible>.,Block2,Block3,.input</visible>
      <neuron name="In" type="act" actFun="tanh">
        <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
        <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
        <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
        <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
        <synapse source="bias" truncated="1" delayed="0" permanent="0" />
      </neuron>
      <neuron name="InGate" type="act" actFun="sig">
        <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
        <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
        <synapse source="State" truncated="1" delayed="1" permanent="0" />
        <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
        <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
        <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
        <synapse source="bias" truncated="1" delayed="0" permanent="0" />
```

```
  </neuron>
  <neuron name="ForgetGate" type="act" actFun="sig">
    <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
    <synapse source="State" truncated="1" delayed="1" permanent="0" />
    <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source="bias" truncated="1" delayed="0" permanent="0" />
  </neuron>
  <neuron name="OutGate" type="act" actFun="sig">
    <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
    <synapse source="State" truncated="1" delayed="1" permanent="0" />
    <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source="bias" truncated="1" delayed="0" permanent="0" />
  </neuron>
  <neuron name="State" type="sum">
    <synapse source="InMul" truncated="0" delayed="0" permanent="0" weight="1" constant
        ="1"/>
    <synapse source="ForgetMul" truncated="0" delayed="0" permanent="0" weight="1"
        constant="1"/>
  </neuron>
  <neuron name="InMul" type="mul">
    <synapse source="In" truncated="0" delayed="0" permanent="0" weight="1" constant="1
        "/>
    <synapse source="InGate" truncated="0" delayed="0" permanent="0" weight="1"
        constant="1"/>
  </neuron>
  <neuron name="ForgetMul" type="mul">
    <synapse source="State" truncated="0" delayed="0" permanent="0" weight="1" constant
        ="1"/>
    <synapse source="ForgetGate" truncated="0" delayed="0" permanent="0" weight="1"
        constant="1"/>
  </neuron>
  <neuron name="OutMul" type="mul">
    <synapse source="State" truncated="0" delayed="0" permanent="0" weight="1" constant
        ="1"/>
    <synapse source="OutGate" truncated="0" delayed="0" permanent="0" weight="1"
        constant="1"/>
  </neuron>
</group>
```

```
<group name="Block2">
  <visible>.,Block1,Block3,.input</visible>
  <neuron name="In" type="act" actFun="tanh">
    <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
    <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source="bias" truncated="1" delayed="0" permanent="0" />
  </neuron>
  <neuron name="InGate" type="act" actFun="sig">
    <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
    <synapse source="State" truncated="1" delayed="1" permanent="0" />
    <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source="bias" truncated="1" delayed="0" permanent="0" />
  </neuron>
  <neuron name="ForgetGate" type="act" actFun="sig">
    <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
    <synapse source="State" truncated="1" delayed="1" permanent="0" />
    <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source="bias" truncated="1" delayed="0" permanent="0" />
  </neuron>
  <neuron name="OutGate" type="act" actFun="sig">
    <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
    <synapse source="State" truncated="1" delayed="1" permanent="0" />
```

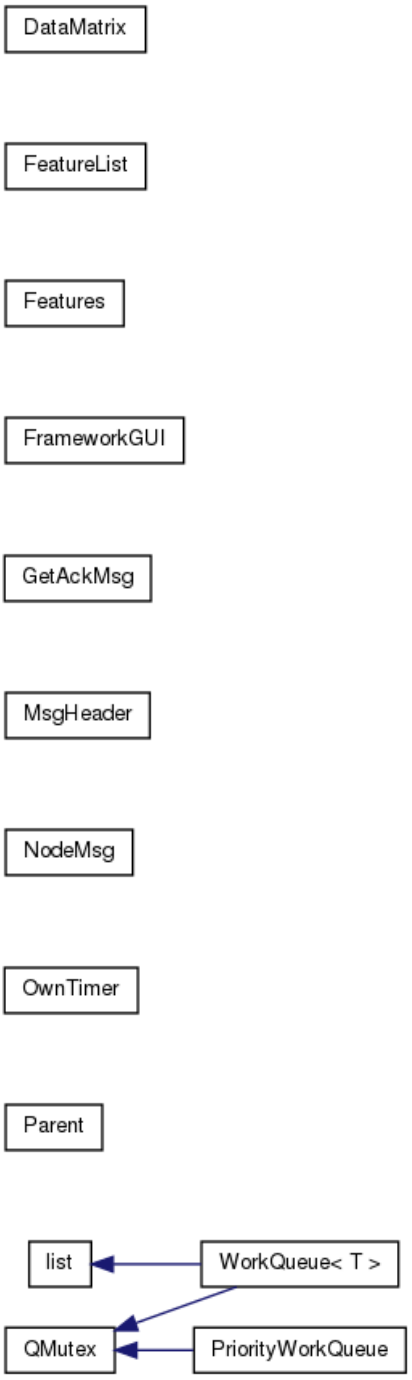
```
    <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source="bias" truncated="1" delayed="0" permanent="0" />
  </neuron>
  <neuron name="State" type="sum">
    <synapse source="InMul" truncated="0" delayed="0" permanent="0" weight="1" constant
        ="1"/>
    <synapse source="ForgetMul" truncated="0" delayed="0" permanent="0" weight="1"
        constant="1"/>
  </neuron>
  <neuron name="InMul" type="mul">
    <synapse source="In" truncated="0" delayed="0" permanent="0" weight="1" constant="1
        "/>
    <synapse source="InGate" truncated="0" delayed="0" permanent="0" weight="1"
        constant="1"/>
  </neuron>
  <neuron name="ForgetMul" type="mul">
    <synapse source="State" truncated="0" delayed="0" permanent="0" weight="1" constant
        ="1"/>
    <synapse source="ForgetGate" truncated="0" delayed="0" permanent="0" weight="1"
        constant="1"/>
  </neuron>
  <neuron name="OutMul" type="mul">
    <synapse source="State" truncated="0" delayed="0" permanent="0" weight="1" constant
        ="1"/>
    <synapse source="OutGate" truncated="0" delayed="0" permanent="0" weight="1"
        constant="1"/>
  </neuron>
</group>
<group name="Block3">
  <visible>.,Block1,Block2,.input</visible>
  <neuron name="In" type="act" actFun="tanh">
    <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
    <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
    <synapse source="bias" truncated="1" delayed="0" permanent="0" />
  </neuron>
  <neuron name="InGate" type="act" actFun="sig">
    <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
    <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
    <synapse source="State" truncated="1" delayed="1" permanent="0" />
```

```
  <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
  <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
  <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
  <synapse source="bias" truncated="1" delayed="0" permanent="0" />
</neuron>
<neuron name="ForgetGate" type="act" actFun="sig">
  <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
  <synapse source="State" truncated="1" delayed="1" permanent="0" />
  <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
  <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
  <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
  <synapse source="bias" truncated="1" delayed="0" permanent="0" />
</neuron>
<neuron name="OutGate" type="act" actFun="sig">
  <synapse source="..input.Retry" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.to_DS" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.DurationRange" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.More_Frag" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.WEP" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.Casting_Type" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.Type" truncated="0" delayed="0" permanent="0" />
  <synapse source="..input.SubType" truncated="0" delayed="0" permanent="0" />
  <synapse source="State" truncated="1" delayed="1" permanent="0" />
  <synapse source=".Block1.OutMul" truncated="1" delayed="0" permanent="0" />
  <synapse source=".Block2.OutMul" truncated="1" delayed="0" permanent="0" />
  <synapse source=".Block3.OutMul" truncated="1" delayed="0" permanent="0" />
  <synapse source="bias" truncated="1" delayed="0" permanent="0" />
</neuron>
<neuron name="State" type="sum">
  <synapse source="InMul" truncated="0" delayed="0" permanent="0" weight="1" constant
      ="1"/>
  <synapse source="ForgetMul" truncated="0" delayed="0" permanent="0" weight="1"
      constant="1"/>
</neuron>
<neuron name="InMul" type="mul">
  <synapse source="In" truncated="0" delayed="0" permanent="0" weight="1" constant="1
      "/>
  <synapse source="InGate" truncated="0" delayed="0" permanent="0" weight="1"
      constant="1"/>
</neuron>
<neuron name="ForgetMul" type="mul">
  <synapse source="State" truncated="0" delayed="0" permanent="0" weight="1" constant
      ="1"/>
  <synapse source="ForgetGate" truncated="0" delayed="0" permanent="0" weight="1"
      constant="1"/>
</neuron>
<neuron name="OutMul" type="mul">
  <synapse source="State" truncated="0" delayed="0" permanent="0" weight="1" constant
      ="1"/>
```

```
        <synapse source="OutGate" truncated="0" delayed="0" permanent="0" weight="1"
            constant="1"/>
      </neuron>
    </group>
  </group>
  <group name="output">
    <visible>LSTM.Block1,LSTM.Block2,LSTM.Block3,input</visible>
    <neuron name="alarm" type="act" actFun="tanh">
      <synapse source=".input.Retry" truncated="0" delayed="0" permanent="0" />
      <synapse source=".input.to_DS" truncated="0" delayed="0" permanent="0" />
      <synapse source=".input.DurationRange" truncated="0" delayed="0" permanent="0" />
      <synapse source=".input.More_Frag" truncated="0" delayed="0" permanent="0" />
      <synapse source=".input.WEP" truncated="0" delayed="0" permanent="0" />
      <synapse source=".input.Casting_Type" truncated="0" delayed="0" permanent="0" />
      <synapse source=".input.Type" truncated="0" delayed="0" permanent="0" />
      <synapse source=".input.SubType" truncated="0" delayed="0" permanent="0" />
      <synapse source=".LSTM.Block1.OutMul" truncated="0" delayed="0" permanent="0" />
      <synapse source=".LSTM.Block2.OutMul" truncated="0" delayed="0" permanent="0" />
      <synapse source=".LSTM.Block3.OutMul" truncated="0" delayed="0" permanent="0" />
    </neuron>
  </group>
</network>
```

C Luokkahierargia

Framework-sovelluksen luokkahierarkia.



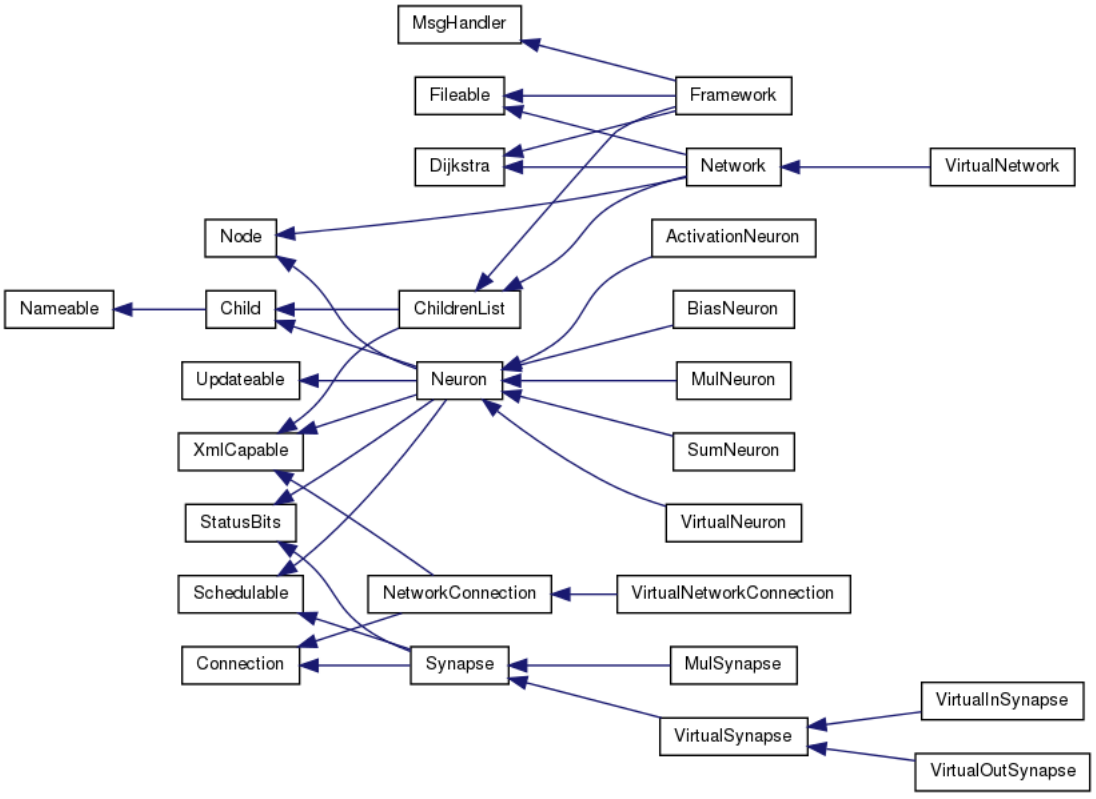
Scheduler

SchedulerList

SocetAPI

SynapseMsg

Timestamp



D Algoritmit

D.1 Eteenpäin Algoritmi

Seuraavissa algoritmeissa esitetään tässä työssä käytetty verkossa eteenpäin (engl. *forward*) suuntautuva algoritmi.

Algotritmissa esiintyvät merkinnät ovat:

x on piirrevektori,
y on tulosvektori,
i on solmun syöte,
o on solmun tulos,
h on aktivaatiofunktio,
n on piirteen tai tuloksen indeksi,
l on taso jota ollaan suorittamassa,
j on syöttävän solmun indeksi,
k on vastaanottavan solmun indeksi ja
J on suoritettavien solmujen joukko.

Algorithm 1 Alustetaan syöteneuronit piirrevektorista

Require: J = tyhjä ja algoritmi 11 suoritettu.

1: $n \leftarrow 0$
2: **for all** solmuille j **do**
3: **if** $j =$ syöteneuroni **then**
4: $i_j \leftarrow x_n$
5: lisää j joukkoon J
6: $n \leftarrow n + 1$
7: **else**
8: $i_j \leftarrow 0, 0$
9: **end if**
10: **end for**

Algorithm 2 Lasketaan verkon aktivaatiot

Require: Algoritmi 1 suoritettu.

$l \leftarrow$ suurimman l joukossa J
while $J \neq$ tyhjä **do**
  while suurin l joukossa $J = l$ **do**
    $o_j \leftarrow h_j(i_j)$
    for all solmuille k joihin solmusta j on yhteys **do**
      $i_k \leftarrow i_k + w_{kj}o_j$
      if k ei kuulu joukkoon J **then**
        lisää k joukkoon J
      end if
    end for
  end while
  $l \leftarrow l - 1$
end while

Algorithm 3 Luetaan tulosneuronit tulosvektoriin

Require: Algoritmi 2 suoritettu.

$n \leftarrow 0$
for all solmuille j **do**
  if $j =$ syöteneuroni **then**
    $y_n \leftarrow o_j$
    $n \leftarrow n + 1$
  end if
end for

D.2 Taaksepäin algoritmi

Seuraavissa algoritmeissa esitetään tässä työssä verkon opettamiseen käytetty taaksepäin (engl. *backward*) suuntautuva algoritmi.

Algotritmissa esiintyvät merkinnät ovat:

e on tuloksen virhevektori,
d on syötteen virhevektori,
i on solmun syöte,
o on solmun tulos,
h' on aktivaatiofunktion derivaatta,
δ on solmun virhe,
n on virheen indeksi,
l on taso jota ollaan suorittamassa,
j on syöttävän solmun indeksi,
α on oppimiskerroin,
η on momentti,
E on neliövirhe,
k on vastaanottavan solmun indeksi ja
K on suoritettavien solmujen joukko.

Algorithm 4 Alustetaan tulosneuronit virhevektorista

Require: K = tyhjä ja algoritmi 3 suoritettu.

1: $n \leftarrow 0$
2: **for all** solmuille k **do**
3: **if** $k =$ tulosneuroni **then**
4: $\delta_k \leftarrow e_n$
5: lisää k joukkoon K
6: $n \leftarrow n + 1$
7: **else**
8: $\delta_k \leftarrow 0,0$
9: **end if**
10: **end for**

Algorithm 5 Lasketaan verkon virheet

Require: Algoritmi 4 suoritettu.

$l \leftarrow 0$
while $K \neq$ tyhjä **do**
  while pienin l joukossa $K = l$ **do**
    $k \leftarrow$ solmu jonka l pienin joukossa K
    poista solmu k joukosta K
    for all solmuille j joista on yhteys solmuun k **do**
      if solmun j truncated lippu ei ole asetettu **then**
        $\delta_j \leftarrow \delta_j + \delta_k w_{kj}$
        if j ei kuulu joukkoon K **then**
          lisää j joukkoon K
        end if
      end if
    end for
    $\delta_j \leftarrow \delta_j h'_j(i_j)$
  end while
  $l \leftarrow l + 1$
end while

Algorithm 6 Luetaan syöteneuronit virhevektoriin

Require: Algoritmi 5 suoritettu.

$n \leftarrow 0$
for all solmuille k **do**
  if $k =$ syöteneuroni **then**
    $d_n \leftarrow \delta_k$
    $n \leftarrow n + 1$
  end if
end for

Algorithm 7 Päivitetään painokertoimet

Require: Algoritmi 6 suoritettu.

for all painoille w_{kj} **do**
  $\frac{dE}{dw_{kj}} \leftarrow \delta_k o_j$
  $\Delta w_{kj}^t \leftarrow \alpha \frac{dE}{dw_{kj}} + \eta \Delta w_{kj}^{t-1}$
  $w_{kj} \leftarrow w_{kj} + \Delta w_{kj}^t$
end for

D.3 Suoritusjärjestys-algoritmi

Kappaleen 5.5.4 algoritmi suoritusjärjestyksen määrittelyyn.

Algotritmissa esiintyvät merkinnät ovat:

l on suoritustaso,
l_{hi} on solmun ylin suoritustaso,
l_{low} on solmun alin suoritustaso,
j on syöttävän solmun indeksi,
k on vastaanottavan solmun indeksi ja
J on käsittelemättömien solmujen joukko.

Algorithm 8 Tasojen päivitys

Require: J = tyhjä ja l_{low} asetettu ylemmällä tasolle.

1: Suorita algoritmi 9
2: Suorita algoritmi 10
3: Suorita algoritmi 11

Algorithm 9 Tasojen alustus

1: **for all** solmuille j **do**
2: **if** $j =$ tulossolmu **then**
3: $l_j \leftarrow l_{low}$
4: **else**
5: $l_j \leftarrow \infty$
6: **end if**
7: lisää j joukkoon J
8: **end for**

Algorithm 10 Dijkstran algoritmi

Require: Algoritmi 9 suoritettu.

1: $l_{hi} \leftarrow l_{low}$
2: $l \leftarrow l_{low}$
3: **while** $J \neq$ tyhjä **do**
4: $k \leftarrow$ solmu jonka l_{low} pienin joukossa J
5: poista solmu k joukosta J
6: **if** $k =$ verkko **then**
7: Suorita algoritmi 8 solmulle k
8: **else**
9: $l_{hi_k} \leftarrow l_{low_k}$
10: **end if**
11: $l \leftarrow l_{hi_k} + 1$
12: **for all** joukossa J oleville solmuille j joihin solmusta k on yhteys **do**
13: **if** $l < l_{low_j}$ **then**
14: $l_{low_j} \leftarrow l$
15: **end if**
16: **end for**
17: **if** $l_{hi} < l_{hi_k}$ **then**
18: $l_{hi} \leftarrow l_{hi_k}$
19: **end if**
20: **end while**

Algorithm 11 Etäisyyksien maksimoiminen

Require: J = tyhjä ja algoritmi 10 suoritettu.

1: $l_{low} \leftarrow l_{hi}$
2: $l \leftarrow l_{hi}$
3: **for all** solmuille j **do**
4: **if** $j =$ syötesolmu **then**
5: $l_{hi_j} \leftarrow l$
6: **end if**
7: lisää j joukkoon J
8: **end for**
9: **while** $J \neq$ tyhjä **do**
10: $j \leftarrow$ solmu jonka l_{hi} suurin joukossa J
11: poista solmu j joukosta J
12: **if** $j =$ verkko **then**
13: Suorita algoritmi 11 solmulle j
14: **else**
15: $l_{low_k} \leftarrow l_{hi_k}$
16: **end if**
17: $l \leftarrow l_{low_j} - 1$
18: **for all** joukossa J oleville solmuille k joihin solmusta j on yhteys **do**
19: **if** $l_{hi_k} >= l$ **then**
20: aseta viive yhteydelle w_{kj}
21: **else**
22: **if** solmuun k ei johda yhteyttä solmusta joka vielä joukossa J **then**
23: $l_{hi_k} \leftarrow l$
24: **end if**
25: **end if**
26: **end for**
27: **if** $l_{low} > l_{low_j}$ **then**
28: $l_{low} \leftarrow l_{low_j}$
29: **end if**
30: **end while**

E Sanomarakenne

Järjestelmän kommunikaatiossa käytettyjen sanomien sisältämät tietotyypit ja niiden rakenne. Kaikki tietotyypit koostuvat Qt:n omista tietotyypeista. Näin tehtiin siksi, että voitiin käyttää Qt:n sarjallistamisominaisuutta[1] hyväksi lähetettäessä paketteja verkon yli. Qt:n tietotyypin ovat 'q' tai 'Q' -alkuisia.

Tyypin nimi:	Timestamp	
Kentän tyyppi:	*quint32*	*quint32*
Kentän nimi:	tv_sec	tv_usec

Tyypin nimi:	DataMatrix	
Kentän tyyppi:	*quint32*	*QVector<double>*
Kentän nimi:	width	data

Tyypin nimi:	Features	
Kentän tyyppi:	Timestamp	DataMatrix
Kentän nimi:	ts	matrix

Tyypin nimi:	MsgHeader			
Kentän tyyppi:	*quint32*	*quint8*	*QString*	*QString*
Kentän nimi:	magic	type	source	destination

Tyypin nimi:	NodeMsg	
Kentän tyyppi:	MsgHeader	*QString*
Kentän nimi:	header	node

Tyypin nimi:	SynapseMsg	
Kentän tyyppi:	MsgHeader	Features
Kentän nimi:	header	data

[1] http://doc.qt.nokia.com/stable/datastreamformat.html

Tyypin nimi:	AckMsg	
Kentän tyyppi:	MsgHeader	*quint8*
Kentän nimi:	header	status

Tyypin nimi:	GetAckMsg		
Kentän tyyppi:	MsgHeader	*quint8*	*QString*
Kentän nimi:	header	status	node