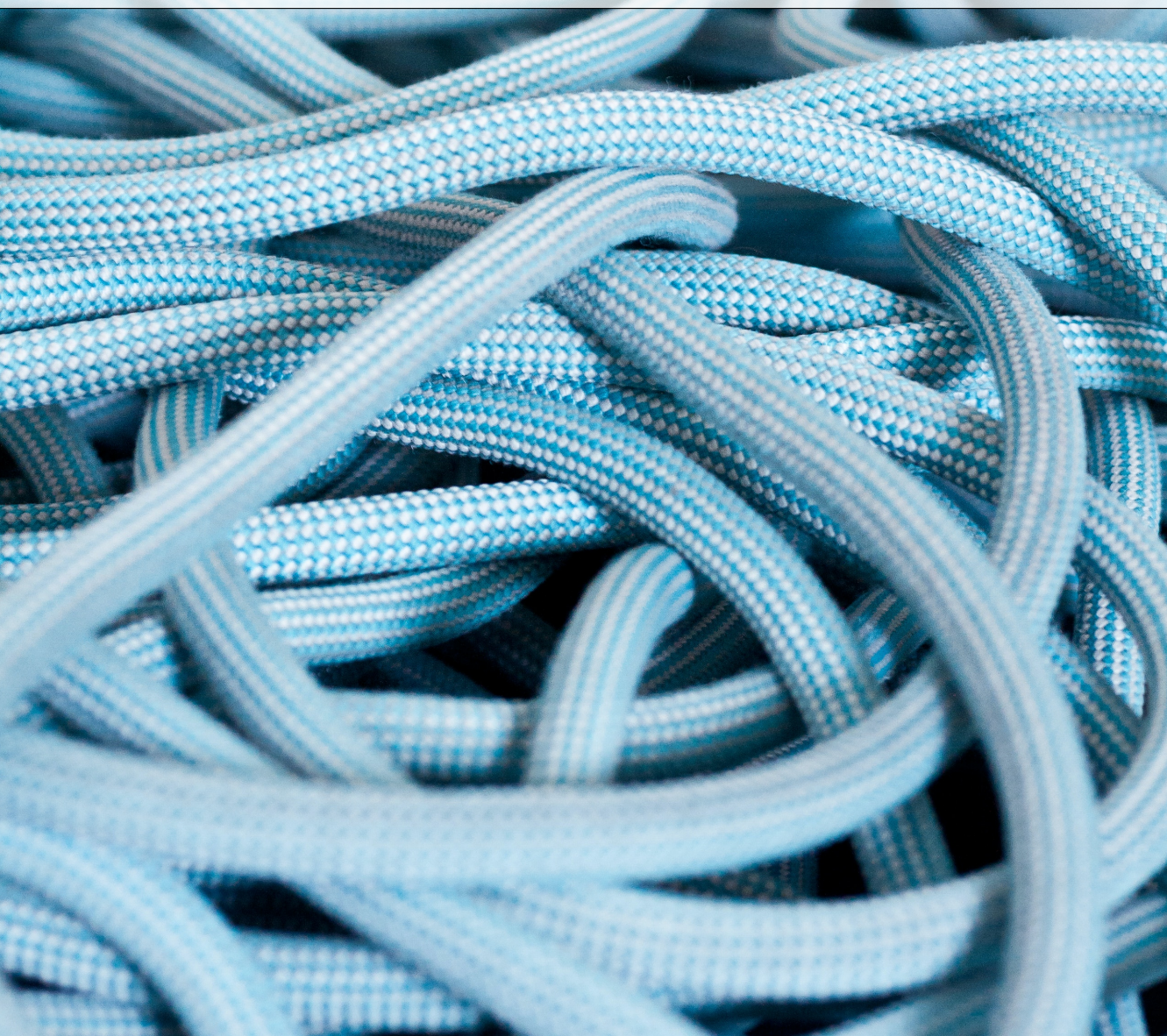
Internet Rzeczy

Laboratorium - ćwiczenia - symulacje

Łukasz Makowski



Internet Rzeczy
Laboratorium - ćwiczenia - symulacje

Łukasz Makowski

Politechnika Warszawska
Instytut Elektrotechniki Teoretycznej i Systemów Informacyjno-Pomiarowych

2022

Spis treści

Indeks skrótów i pojęć

IEEE Institute of Electrical and Electronics Engineers. 16, 18, 21, 68, 72, 99, 100, 105, 113

IoT Internet of Things. 12, 13, 15, 17–19, 34, 38, 45–47, 51, 52, 99, 100, 105, 106

IP Internet Protocol. 46–50, 54, 58, 59, 62, 64, 65, 68, 69, 72, 75, 76, 82, 90, 107, 108

IPv6 Internet Protocol version 6. 16, 46, 47, 62, 69

ISM Idustrial, Scientific, Medical. 99

JSON JavaScript Object Notation. 52

LAN Local Area Network. 71

LoRa . 38, 99

LTE Long Term Evolution. 18

LwM2M Lightweight M2M. 16, 17

MAC Medium Access Control. 58, 78, 79, 93

MANET Mobile Ad-hoc Network. 71–73, 98

MCS Modulation and Coding Set. 104, 105

MIMO Multiple Input, Multiple Output. 106, 107, 115

MQTT MQ Telemetry Transport. 16, 17, 19, 46, 52, 53, 69

MU-MIMO Multi-User Multiple Input, Multiple Output. 107

NAT Network Address Translation. 47

NB-IoT NarrowBand IoT. 38, 99

NED Network Description File. 24–26, 30, 36, 38, 42, 54, 56, 59–61, 73, 76, 78, 81, 83, 88–90, 98, 107, 108, 114, 115

NFS Network File System. 49

NTP Network Time Protocol. 49

OFDM Orthogonal Frequency-Division Multiplexing. 105, 106, 115

OFDMA Orthogonal Frequency-Division Multiple Access. 106

p2p peer-to-peer. 72

PER Packet Error Rate. 40

PPP Point-to-Point Protocol. 68

QAM Quadrature Amplitude Modulation. 103, 104

QoS Quality of Service. 18

QPSK Quadrature Phase Shift Keying. 102–104

REST Representational State Transfer. 17, 46, 51–53, 69

RFC Request for Comments. 18
RFID Radio Frequency Identification. 13
RPL Routing Protocol for Low-Power and Lossy Networks. 16
RU Resource Unit. 106

Sigfox . 38, 99
SNMP Simple Networking Management Protocol. 16, 49
SNMP MIB Simple Networking Management Protocol Management Information Base. 11
SNR Signal-to-Noise Ratio. 106
SSID Service Set IDentifier. 108, 109, 112

TAP . 68
TCP Transmission Control Protocol. 17, 18, 46, 50, 51, 64, 65, 69
TFTP Trivial File Transfer Protocol. 49
TLS Transport Layer Security. 52

UAV Unmanned Aerial Vehicle. 72
UDP User Datagram Protocol. 16–18, 46, 48–50, 53, 60–63, 69, 77, 78, 81, 85–87, 109
URI Uniform Resource Identifier. 51, 52

WAN Wide Area Network. 13
WiFi Wireless Fidelity. 38, 67, 68, 79, 99, 100, 103–111, 113, 115
WWW World Wide Web. 12, 51

XML Extensible Markup Language. 52, 82, 97, 109

ZigBee . 38, 68

Rozdział 1

Wstęp

Ludzie od lat dążą do tego, by opanować coraz więcej *rzeczy* i mieć nad nimi pełniejszą kontrolę. Jak powszechnie wiadomo, informacja o danej rzeczy, osobie, zjawisku daje choćby częściową możliwość kontroli. Zatem przesyłanie informacji od różnorodnych obiektów jest stale obecnym wątkiem w rozwoju ludzkiej cywilizacji.

W roku 1982 grupa pracowników naukowych Carnegie Mellon University podłączyła do sieci komputerowej automat wydający napoje. Zrobili to aby ograniczyć zbędne spacery do maszyny, gdy automat jest pusty, by wiedzieć od jak dawna puszki z napojami są do niego włożone i czy zdążyły się schłodzić, oraz z którego przycisku skorzystać aby otrzymać już zimny napój [1].

W roku 1990 Simon Hackett i John Romkey, w ramach rzuconego im wyzwania, podłączyli zwykły toster do sieci Internet.[1] Toster przesyłał dane do Internetu za pomocą podstawowego protokołu czyli TCP/IP. Zdalne nim zarządzanie było możliwe za pomocą protokołu SNMP MIB [2]. W roku 1991 dodano obsługiwane przez Internet ramię robota, które automatycznie wkładało chleb do tostera. Z powodu zastosowania standardowych protokołów Internetu, często pojawia się stwierdzenie, iż właśnie ten toster był pierwszą *rzeczą* w Internecie Rzeczy.

Te pierwsze eksperymenty pozwoliły uświadomić kolejnym pokoleniom, iż do Internetu można podłączyć w zasadzie wszystko.

W niniejszej publikacji będę posługiwał się terminem „Internet Rzeczy" pisanym wielkimi literami. Angielskie słowniki nie są zgodne co do formy zapisywania tego terminu i zdarza się kapitalizowanie tylko słowa „Internet" a pozostawienie „rzeczy" małą literą. Internet jest tylko jeden a więc to nazwa własna, co do której nie powinno być wątpliwości, iż należy pisać ją wielką literą. Dominujące w praktyce jest użycie „Internet of Things", które również występuje w renomowanych słownikach (np.: Merriam-Webster) i wynikającego z niego niezwykle popularnego skrótu „IoT". Dlatego w dalszym tekście będę używał nazwy „Internet Rzeczy" wzorem terminu angielskiego.

[1] Klasyczny amerykański toster firmy Sunbeam, opracowany w roku 1948 i produkowany w niemal niezmienionej formie przez około pół wieku.

Alternatywnie będę korzystał ze skrótu IoT, jako dostatecznie upowszechnionego w społecznej świadomości a na pewno wśród czytelników niniejszej publikacji.

1.1 Internet Rzeczy

Internet Rzeczy (ang. *Internet of Things*, IoT) mówiąc najprościej to wszelkiego rodzaju „rzeczy" połączone z siecią Internet. Czym się jednak różni od typowej sieci komputerowej, która przecież również jest zbudowana z *rzeczy* takich jak routery i serwery?

Względem tradycyjnego Internetu różnicę stanowi przede wszystkim **różnorodność urządzeń połączonych siecią oraz ich znacznie większa liczba** [3].

Internet Rzeczy stanowi kolejny etap rozwoju internetowych technologii sieciowych, które przybierały kolejne formy takie jak:

- sieć komputerów – to wczesny, wojskowy i akademicki etap rozwoju Internetu, gdy do sieci podłączona była niewielka liczba komputerów, dostępnych wówczas tylko dla specjalistów;
- sieć dokumentów – to etap, gdy istnienie Internetu przestało być celem samym w sobie a zaczął on być narzędziem pozyskiwania informacji z coraz łatwiej dostępnych zasobów sieciowych, czego osią stała się tak zwana „globalna pajęczyna sieciowa" (ang. *World Wide Web*, WWW);
- sieć międzyludzka – odkąd Internet przestał być jednokierunkowym strumieniem informacji przypominającym dawne, scentralizowane media a stał się demokratyczną przestrzenią wymiany poglądów i doświadczeń miedzy szerokimi masami ludzi którzy, zazwyczaj anonimowo, stali się głównymi twórcami treści w tak zwanej sieci Web 2.0;[2]
- sieć urządzeń – aktualny etap rozwoju, określany mianem Internet Rzeczy, w którym zaprogramowane przedmioty samoczynnie przekazują znacznie więcej informacji, niż robią to ludzie;

Dla porządku warto zauważyć upowszechnianie się koncepcji Web 3.0. Jednak ten temat, zbyt odległy niniejszej publikacji, zostawiam zainteresowanym czytelnikom do samodzielnej eksploracji.

Internet Rzeczy jest więc szeroką koncepcją w której przedmioty codziennego użytku, przedmioty często bardzo małe lub wręcz przeciwnie – maszyny i całe hale przemysłowe, uzyskują swe odbicie w świecie cyfrowym pod postacią danych, jakie generują, i z jakich korzystają. Owo odbicie może być tak dokładne, iż mówi się o „cyfrowym bliźniaku" (ang. *Digital Twin*) [4].

Tak jak dla fundamentalnych protokołów Internetu ważni są Vinton Cerf, Robert Kahn, David Reed i John Postel, tak jak dla WWW ważny jest Timothy Berners-

[2] Internet jest siecią demokratyczną w rozumieniu potencjalnie równej dostępności dla każdego obywatela a nie jako forma ustroju państwa.

-Lee, tak samo dla Internetu Rzeczy kluczową postacią jest Kevin Ashton. To właśnie on jako pierwszy użył określenia „Internet Rzeczy", podczas prezentacji dla firmy Procter & Gamble w roku 1999.

Ten globalny producent kosmetyków miał problem z dostarczeniem konsumentom jednego produktu. Szminki w charakterystycznym brązowym kolorze błyskawicznie znikały z półek sklepowych, chociaż były dostępne w magazynach. Ashton wpadł na pomysł, by wyposażyć je w znacznik RFID ale nie tylko to. W momencie, w którym półka wyposażona w czytnik zostałaby opróżniona, automatycznie wysyłałaby informację o konieczności kolejnej dostawy. Siecią, za pomocą której takie zamówienie dałoby się realizować był Internet [5].

Internet Rzeczy powstał więc w kontekście logistyki, oraz znanych już wcześniej radiowych identyfikatorów rzeczy (ang. *Radio Frequency Identification*, RFID). Jak zauważa sam Ashton, w roku 1999 „dostęp do Internetu był wdzwaniany, nie było WiFi, sieci komórkowe istniały dla rozmów i SMS, niemal nie było cyfrowej fotografii a GPS był tylko dla wojska" [6].[3]

Internet Rzeczy przeszedł imponującą ewolucję wraz z samym Internetem. Stał się pojęciem tak popularnym i modnym nawet wśród osób nietechnicznych, że na rynku mamy obecnie ogromną liczbę rozwiązań sprzętowych i oprogramowania oznaczonego etykietką „IoT". Nastąpił przy tym znaczący rozwój technik sieciowych, informatyki i elektroniki a zatem z inżynierskiego punktu widzenia ten złożony konglomerat wymaga szerokiej wiedzy z kilku dziedzin. Aby ułatwić zrozumienie tej tematyki można na przykład posłużyć się niedrogimi, sprzętowymi zestawami rozwojowymi, co pozwoli poznać IoT od strony urządzenia. Jednak by uzyskać szeroką perspektywę na tę tematykę, która obejmuje urządzenia instalowane masowo, praktycznie niezbędne jest wykorzystanie modeli symulacyjnych.

1.2 Narzędzia do symulacji sieci

W wyniku rozwoju elektroniki i technologii komunikacyjnych powstało wiele standardów i rozwiązań realizacji sieci urządzeń elektronicznych i komputerowych. Rozwijanie standardów i rozwiązań sieciowych byłoby bardzo trudne i kosztowne, gdyby nie istniały odpowiednie programy symulacyjne.

Testowanie IoT może odbywać się w czterech obszarach takich jak: funkcjonalność, wydajność, transmisja, bezpieczeństwo. Symulacjom mogą podlegać różne warstwy IoT, takie jak:

- węzły końcowe z sensorami i aktuatorami,
- bramki sieciowe,
- sieć rozległa (ang. *Wide Area Network*, WAN),

[3] Przed pojawieniem się tak zwanego „stałego dostępu" połączenie z Internetem realizowane było za pomocą naziemnych linii telefonicznych protokołami PPP lub wcześniej SLIP – tak zwane „wdzwanianie" (ang. *dial-up*).

- serwis przechowujący dane – tak zwana „chmura”,
- aplikacja biznesowa [7].

Programy do symulacji mogą oferować różny poziom realizmu i złożoności modelu sieci. Muszą umożliwiać symulowane działanie pojedynczych urządzeń oraz aspektów fizycznych transmisji, takich jak:

- opóźnienia w przekazywaniu i przetwarzaniu przesyłanych pakietów danych,
- propagacja sygnałów różnymi drogami, takimi jak radiowa, optyczna, akustyczna, przewodowa,
- zmienne warunki propagacji sygnałów,
- krótkookresowe odchylenia od ustalonych parametrów (ang. *jitter*)
- sposób zasilania (sieciowe, bateryjne, odnawialne),
- zasoby energii zmieniające się w czasie,
- mobilność węzłów sieci,
- zmienność topologii wynikająca z powyższych.

Ponadto dostęp do kodu źródłowego jest istotny, nawet jeśli nie zamierzamy samodzielnie wprowadzać poprawek do tego kodu. Gwarantuje on możliwość sprawdzenia zaimplementowanych modeli i ocenę ich poprawności przez szersze grono zainteresowanych ekspertów.

Zazwyczaj programy symulujące sieci są to symulatory zdarzeń dyskretnych, chociaż mogą też być emulatorami działającymi w czasie rzeczywistym. Korzystny w pierwszej z tych kategorii jest fakt, iż można przeanalizować działanie urządzeń i tworzonych przez nie sieci zarówno w mikroskali czasowej jak i w skali makro. Z kolei zaletą emulatorów jest możliwość połączenia ich z rzeczywistymi urządzeniami i aplikacjami sieciowymi.

Nie bez znaczenia jest też koszt tego oprogramowania, który dla różnych aplikacji rozkłada się w przedziale od oprogramowania darmowego (głównie wolne i otwarte oprogramowanie) do kosztu setek euro miesięcznie w modelu subskrypcji.

Jak można się domyśleć, dostępnych jest wiele tego rodzaju aplikacji, a wybrane z nich zostaną krótko omówione poniżej.

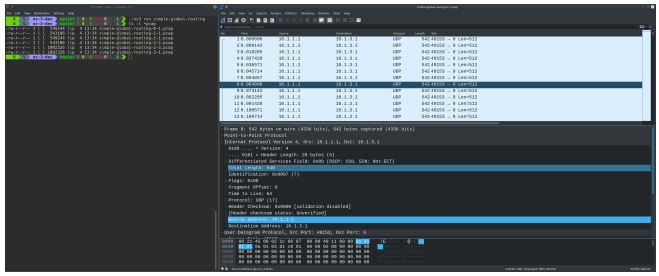
Network Simulator https://www.nsnam.org/

W środowisku naukowym najczęściej używanym symulatorem sieci jest Network Simulator. Jest to kontynuacja oprogramowania stworzonego w Lawrence Berkeley National Laboratory na terenie Uniwersytetu Kalifornijskiego w Berkeley około roku 1995, co czyni te oprogramowanie prawdopodobnie najstarszym w tej dziedzinie. Wersja druga (ns-2) powstała przy istotnym wsparciu amerykańskiej Agencji Zaawansowanych Projektów Badawczych w Obszarze Obronności (ang. *Defense Advanced Rese-*

arch Projects Agency, DARPA) i stała się głównym narzędziem badawczym w obszarze symulacji sieci. Wersja 3.36 została wydana w kwietniu 2022.

Aplikacja ta sama w sobie nie posiada środowiska graficznego i nie jest trywialna w użyciu. Jednak powstało kilka rozwiązań pozwalających na wizualizację sieci NS-3, takich jak NetAnim, PyViz lub NetSimulyzer [8].

Network Simulator nie ma w podstawowej wersji środowiska graficznego, gdyż nie jest ono aż tak potrzebne podczas symulacji. Zaprogramowany model sieci może być przetwarzany długi czas, więc nacisk jest tu położony bardziej na uzyskanie końcowego rezultatu w formie raportu. Raport ten może przyjąć formę pliku z zarejestrowanymi pakietami, które można następnie przeanalizować w specjalistycznym programie do ich analizy, takim jak Wireshark. Właśnie takie wykorzystanie jest pokazane na rysunku 1.



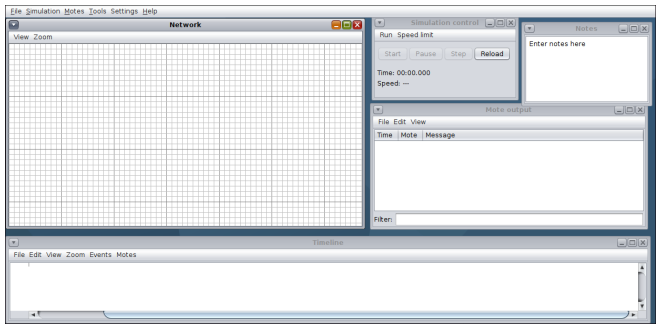
Rysunek 1. Użycie NS-3 w linii poleceń i analiza pakietów w programie Wireshark.

Network Simulator jest aktywnie rozwijany jako wolne i otwarte oprogramowanie dystrybuowane na licencji GNU GPL a jego kod źródłowy znajduje się pod adresem https://gitlab.com/nsnam/ns-3-dev. Network Simulator powstaje przede wszystkim dla systemów Linux, FreeBSD i MacOS. Jest dostępny w repozytoriach większości dystrybucji systemu GNU/Linux, gdzie nie trzeba kompilować go ze źródeł a jego pobranie i instalacja trwa kilkanaście sekund.

Cooja https://www.contiki-ng.org/

Cooja powstała na bazie i jako wsparcie dla systemu operacyjnego Contiki, przeznaczonego dla urządzeń o ograniczonych zasobach a więc takich jak urządzenia końcowe IoT [9]. System ten powstał w roku 2002 i uzyskał wsparcie firm takich jak Texas Instruments, Atmel (obecnie Microchip), ST Microelectronics, Cisco i wielu innych. Aktualnie rozwijana wersja systemu Contiki-NG reklamuje się jako „system dla urządzeń IoT następnej generacji" [10].

Cooja początkowo była tworzona dla systemu Linux oraz dla Windows wspartego przez Cygwin a następnie została też przeniesiona na system MacOS. Symulator posiada przyjazny interfejs graficzny napisany w języku Java, który został pokazany na rysunku 2.



Rysunek 2. Cooja gotowa do pracy.

Cooja jest dystrybuowana wraz z innymi narzędziami Contiki. Aplikacja emuluje rzeczywisty węzeł sieci na maszynie – gospodarzu, uruchamiając w tle proces Contiki. Aby mogło to działać, konieczne jest jednak poprawnie zainstalowane i skonfigurowane środowisko rozwojowe tego systemu. Możliwości Cooja są zatem takie, jak możliwości Contiki. Warto zwrócić uwagę na licznie dostępne protokoły, takie jak: IPv6 (ang. *Internet Protocol version 6*, IPv6), UDP, RPL (ang. *Routing Protocol for Low-Power and Lossy Networks*, RPL), CoAP, LwM2M, MQTT, BLE (ang. *Bluetooth Low Energy*, BLE), SNMP (ang. *Simple Networking Management Protocol*, SNMP), IEEE 802.15.4 i wiele innych a także możliwość implementowania kolejnych rozwiązań, i testowanie ich.

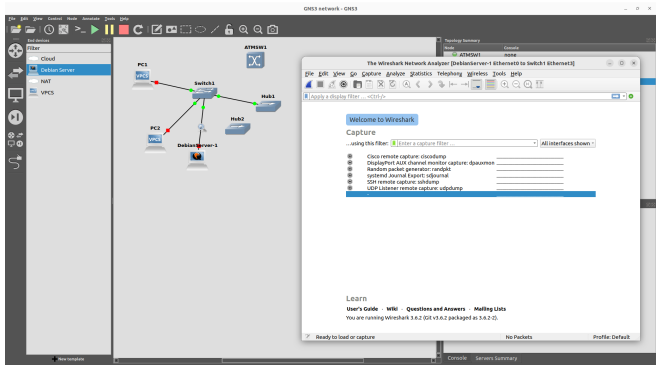
Contiki a wraz z nim Cooja są oprogramowaniem otwartoźródłowym, dystrybuowanym na 3-klauzulowej licencji BSD.

GNS3 https://gns3.com/

Kolejną otwartoźródłową aplikacją dystrybuowaną na licencji GNU GPL a służącą do symulacji sieci komputerowych jest Graphical Network Simulator-3, lub po prostu GNS3. Jest to środowisko graficzne oparte pierwotnie na aplikacji Dynamips będącej symulatorem routerów firmy Cisco. Stąd często GNS3 jest niepoprawnie uznawany za symulator routerów tej jednej firmy.

GNS3 powstał w roku 2008 i jest aktywnie rozwijany, zyskując coraz większą funkcjonalność. GNS3 w ogromnej mierze bazuje na pracy z wirtualnymi maszynami, które mogą emulować funkcjonowanie urządzeń sieciowych, systemów operacyjnych i innego oprogramowania w sieci. Oprócz symulacji routerów Cisco pojawiło się wsparcie dla symulacji rozwiązań firm Juniper, Alcatel, Dell, HP i innych. Należy jednak zauważyć, że do symulacji sprzętu komercyjnego zwykle potrzebne jest zakupienie wirtualnego obrazu urządzenia od jego producenta.

Symulowane transmisje mogą być analizowane za pomocą oprogramowania nasłuchowego, takiego jak tcpdump lub Wireshark. Przykładowe użycie tego tandemu aplikacji jest pokazane na rysunku 3.



Rysunek 3. Użycie GNS3 oraz Wireshark.

Dzięki tym możliwościom GNS3 jest używany przez duże firmy, takie jak między innymi: AT&T, NASA, Exxon, Walmart.

IoTIFY https://iotify.io/

IoTIFY jest dziełem szwajcarskiej firmy Ternary GmbH. Jest to oprogramowanie działające w stylu „Oprogramowanie jako usługa" (ang. *Software as a Service*, SaaS). IoTIFY od początku było tworzone z myślą o symulowaniu i analizowaniu działania Internetu Rzeczy w przeciwieństwie do większości innych aplikacji symulujących sieci, w których wsparcie dla IoT to dodatek załączony później.

Producent w opisie IoTIFY podkreśla symulacje aspektów skalowalności, istotne dla masowych sieci IoT. Symulowane są interakcje między urządzeniami a siecią wraz z możliwością połączenia środowiska symulowanego z rzeczywistymi technologiami chmurowymi w Internecie. Ponadto symulowane są interakcje między samymi urządzeniami. To obejmuje także aktuatory oddziałujące na środowisko, w którym umieszczone są inne sensory. IoTIFY wspiera podstawowe protokoły Internetu takie jak UDP, TCP, HTTP jak również te bardziej typowe dla sieci IoT takie jak MQTT, CoAP i LwM2M.

Ponieważ jest to oprogramowanie typu SaaS, to nie ma dostępu do jego kodu źródłowego. Istnieje możliwość skorzystania z udostępnionego interfejsu programistycznego REST API (ang. *Representational State Transfer*, REST) i na przykład przesłania danych do arkuszy Google Sheets. Dostęp do platformy IoTIFY jest płatny.

Common Open Research Emulator http://coreemu.github.io/core/

CORE jest narzędziem budowania wirtualnych sieci, tworzącym środowisko testowania aplikacji i protokołów. Nie jest symulatorem lecz emulatorem, działającym w czasie rzeczywistym, który może zostać połączony z fizycznymi sieciami i routerami. W jego skład wchodzi Extendable Mobile Ad-hoc Network Emulator (EMANE), z którym łączy

się za pomocą wirtualnego interfejsu sieciowego typu TAP a więc na poziomie łącza danych.

EMANE jest pakietem oprogramowania rozwijanym pod opieką U.S. Naval Research Laboratory a służącym do eksperymentowania w obszarze projektowania, rozwoju i testowania dynamicznych architektur sieciowych. Opiera się na zestawie wtyczek, odpowiedzialnych za warstwy fizyczną i łącza danych, co zapewnia możliwość testowania różnorodnych rozwiązań. Oprogramowanie to powstało w amerykańskiej firmie CenGen, wykupionej później przez mającą ponad pół wieku doświadczenia w branży obronnej firmę DRS (obecnie Leonardo DRS).

CORE jest rozwijany pod opieką firmy Boeing jako oprogramowanie otwartoźródłowe, dystrybuowane na 2-klauzulowej licencji BSD.

Cisco Packet Tracer https://www.netacad.com/courses/packet-tracer

Packet Tracer powszechnie znanej firmy Cisco został rozbudowany o symulację urządzeń IoT. Pozwala ona uzyskać szersze spojrzenie na sieci lokalne, funkcjonujące w nich urządzenia IoT oraz ich połączenia między sobą i z Internetem. Złożoność protokołów i komunikacji urządzeń IoT jest tu mocno uproszczona.

Cisco Packet Tracer jest zamkniętym oprogramowaniem działającym w systemach Windows, Linux, macOS, Android, iOS. Jest dostępny za darmo dla uczestników kursów Cisco jednak wymaga aktywnego konta.

Tetcos NetSim https://www.tetcos.com/

NetSim jest produktem indyjskiej firmy Tetcos, który oferuje możliwość symulowania zarówno typowych komputerowych sieci typowych dla Internetu jak też symulacje sieci z wielu innych kategorii. Są to między innymi: sieci Internetu Rzeczy, sieci czujnikowe, sieci kognitywne i pojazdowe zgodne ze standardami IEEE, komórkowe sieci telekomunikacyjne w standardach LTE (ang. *Long Term Evolution*, LTE) i 5G, transmisje sygnału cyfrowej telewizji satelitarnej drugiej generacji (ang. *Digital Video Broadcasting – Satellite – Second Generation*, DVB-S2).

W modelach uwzględniony jest szeroki zakres warstw transmisji danych począwszy od warstwy fizycznej z typowymi modulacjami cyfrowymi i modelami rozchodzenia fal radiowych w pasmach od HF do S a skończywszy na warstwie aplikacji. W obszarze IoT wspiera przede wszystkim standardy takie jak: IEEE 802.15.4, 6LoWPAN (ang. *IPv6 over Low-Power Wireless Personal Area Networks*, 6LoWPAN), standardy RFC 6550 i 3561, UDP, TCP, CoAP, kodeki głosu i obrazu, szyfrowanie oraz kontrolę jakości usługi (ang. *Quality of Service*, QoS). Możliwe jest interfejsowanie z pakietem Matlab oraz z analizatorem pakietów Wireshark.

Tetcos NetSim jest oprogramowaniem zamkniętym i płatnym.

MIMIC Simulator Suite https://www.gambitcomm.com/site/mimic-simulator.php

MIMIC to zestaw aplikacji do symulacji różnych aspektów komunikacji sieciowej. Potrafi zasymulować współistnienie setek serwerów i urządzeń IoT w heterogenicznych sieciach opartych o rozwiązania przewodowe oraz bezprzewodowe w tym sieci 3G i 4G. Oferuje także symulacje typowego sprzętu sieciowego firm takich jak Apple, Intel, Juniper, Cisco, Huawei i innych. Możliwa jest symulacja protokołu MQTT z dostępem do chmury firmy Amazon (ang. *Amazon Web Services*, AWS).

Jest to oprogramowanie dla systemu Windows, zamknięte i płatne.

IoT Simulator https://www.bevywise.com/iot-simulator/

BevyWise oferuje symulację sieci IoT z użyciem protokołu MQTT. Dostępna jest możliwość transmisji danych do rzeczywistych rozwiazań chmurowych firm Amazon, Microsoft i Google. IoT Simulator jest zamkniętym, płatnym oprogramowaniem dostępnym dla systemów Windows, Linux, Mac.

Boson NetSim https://www.boson.com/netsim-cisco-network-simulator

NetSim firmy Boson jest płatnym oprogramowaniem dedykowanym symulowaniu sprzętu i oprogramowania sieciowego firmy Cisco. Działa jako zwykła aplikacja oraz w przeglądarce, gdzie również oferuje symulację terminala routera. Boson NetSim jest oprogramowaniem zamkniętym i płatnym.

OMNeT++ https://omnetpp.org/

OMNeT++ jest symulatorem zdarzeń dyskretnych (ang. *discrete event simulator*) i na nim oparty jest niniejszy podręcznik. Dlatego tej aplikacji zostanie poświęcony osobny rozdział 2.

1.3 Pytania sprawdzające

1. Jaka jest geneza Internetu Rzeczy?
2. Na jakie cechy oprogramowania do symulacji sieci należy zwrócić uwagę?
3. Jaka jest korzyść z dostępu do kodu źródłowego oprogramowania do symulacji sieci?
4. Czym różni się emulator od symulatora sieci?
5. Jakie protokoły typowo występują w symulatorach sieci?
6. Jak symulatory sieci współpracują z rozwiązaniami chmurowymi?

Rozdział 2

Wprowadzenie do OMNeT++

Cel ćwiczenia

Celem niniejszego ćwiczenia jest wprowadzenie od podstaw do środowiska OMNeT++. Część teoretyczna rozdziału dotyczy jego instalacji i aspektów użytkowych. Część praktyczna obejmuje prostą symulację komunikacji między dwoma urządzeniami.

Środowisko programistyczne Eclipse, poprzez swój system wtyczek jest bardzo elastyczne, dzięki czemu umożliwia dostosowanie do różnych języków programowania i zestawów narzędzi programistycznych (ang. *toolchain*). Na Eclipse oparty jest między innymi OMNeT++, czyli szablon projektowy i zestaw bibliotek przeznaczony do symulacji sieci, protokołów i standardów komunikacyjnych.

OMNeT++ jest dystrybuowany na Publicznej Licencji Akademickiej (ang. *Academic Public License*). Zgodnie z licencją jest dostępny za darmo w pracach badawczych i dydaktyce oraz innej działalności nieprzeznaczonej dla zysku. Można go pobrać ze strony https://omnetpp.org/download/ lub z repozytorium w serwisie GitHub.

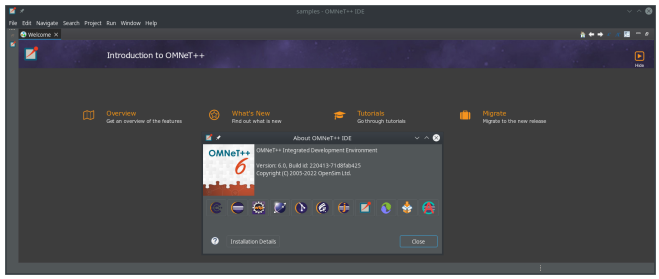
Do prac komercyjnych konieczne jest zakupienie wersji płatnej o nazwie OMNEST. Różnice między wersją na licencji APL i wersją komercyjną są kosmetyczne.

OMNeT++, w przeciwieństwie do OMNEST, między innymi nie posiada wygodnego instalatora i trzeba podjąć wysiłek samodzielnej kompilacji. Jednak jest ona mocno zautomatyzowana i zwykle przebiega bezproblemowo. Materiał wideo prezentujący proces instalacji tego środowiska w systemie Debian GNU/Linux towarzyszy niniejszemu podręcznikowi. Po kompilacji katalog z OMNeT++ zajmuje około 1,5 GB ale później potrzeba kilkukrotnie więcej miejsca na biblioteki, przykłady i własne projekty.

Ponadto OMNEST posiada możliwość eksportu modeli do grafiki wektorowej SVG, integrację z językiem SystemC oraz wsparcie dla standardu IEEE 1516 opisującego szablon projektowy i zasady Architektury Wysokiego Poziomu (ang. *High Level Architec-*

ture, HLA) [11]. W wersji płatnej dostępne jest też oficjalne wsparcie za pomocą poczty elektronicznej.

Okno powitalne OMNeT++ wraz z winietką programu pokazane jest na rysunku 4.

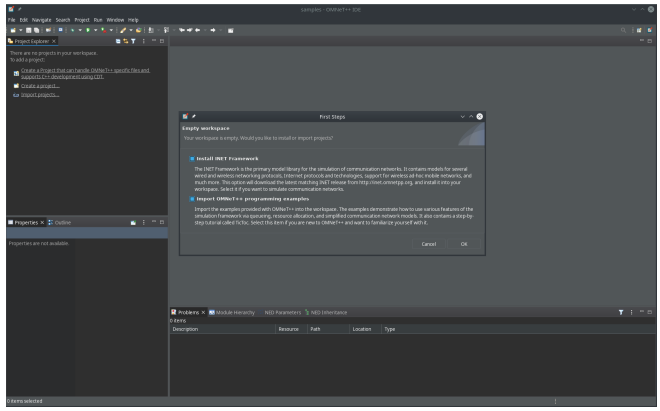


Rysunek 4. Okno powitalne i podstawowe informacje o OMNeT++.

2.1 Podstawy użytkowania OMNeT++

Pełen podręcznik użytkownika OMNeT++ liczy ponad 150 stron i szczegółowo omawia aspekty użytkowe środowiska [12]. Poniżej znajdzie się uproszczony opis, który powinien wystarczyć do szybkiego rozpoczęcia pracy z symulacjami.

Po zamknięciu okna powitalnego użytkownik jest zachęcany do instalacji podstawowego szablonu programistycznego, jakim jest biblioteka INET oraz zestawu przykładów. Okno pytające użytkownika o chęć instalacji tych elementów pokazane jest na rysunku 5.

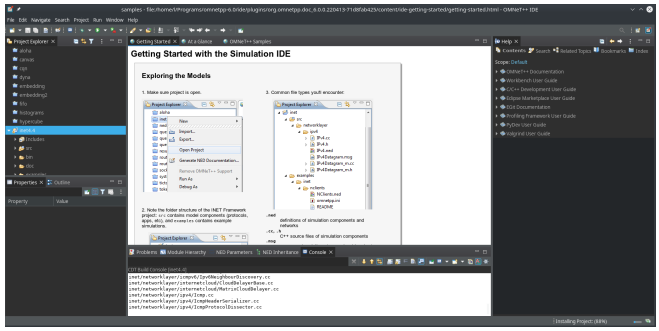


Rysunek 5. Pytanie o instalację podstawowej bilbioteki INET oraz przykładów.

Po zatwierdzeniu chęci instalacji wspomnianych kompoenentów, okno zmieni wygląd. W środkowej, głównej cześci okna pojawi kilka zakładek z przeglądowymi informacjami o samym środowisku i programach demonstracyjnych. Warto poświecić

chwilę i samemu przejrzeć zakładki „Getting Started”, „At a Glance” oraz „OMNeT++ Samples”.

Poniżej zobaczymy postęp kompilacji biblioteki INET, co chwilę trwa.Warto przyjrzeć się temu procesowi i zwrócić uwagę na to, jakie komponenty ta biblioteka zawiera. Później da się je odnaleźć w katalogu `omnetpp-6.0/samples/inet4.4/src/inet/` i podzielone na kategorie oraz w dokumentacji [13]. Okno środowiska na tym etapie pokazane jest na rysunku 6.



Rysunek 6. Przeglądowe informacje o OMNeT++ i proces kompilacji INET.

Biblioteka INET powinna skompilować się bez błędów („0 errors”) chociaż możliwe, że pojawią się ostrzeżenia. Po kompilacji zajmuje ona 2,1 GB przestrzeni dyskowej. Później można będzie zainstalować kolejne biblioteki i szablony przeznaczone do specjalistycznych zastosowań. Kod źródłowy przykładów, które są potrzebne do realizacji zakresu zajęć niniejszego podręcznika, zajmuje około 300 MB.

W oknie OMNeT++ możemy zobaczyć kilka podstawowych obszarów:

- na górze – menu tekstowe i graficzne,
- po lewej – przegląd projektów i ich właściwości,
- na środku – główne pole aplikacji w którym będziemy edytować kod,
- na dole – liczne pola informacyjne w tym konsola prezentująca postępy kompilacji,
- po prawej – informacje pomocnicze.

Jak każde środowisko programistyczne tak i OMNeT++ można później dostosować do własnych preferencji.

Pracę z OMNeT++ można podzielić na trzy podstawowe etapy:

- wczytywanie i edycja projektu,
- kompilacja i uruchamianie symulacji,
- analiza wyników w trakcie i po zakończeniu symulacji.

Wczytywanie istniejącego projektu jest bardzo proste. Wystarczy w panelu Project Explorer dwukrotnie kliknąć wybrany projekt. By projekt zamknąć należy we

wspomnianym panelu wybrać korzeń drzewa projektu (jego nazwę) po czym można posłużyć się górnym menu Project › Close Project. Alternatywnie można wybrać tę opcję z menu kontekstowego, czyli po kliknięciu prawym przyciskiem myszy na nazwie projektu w eksploratorze projektów.

Tworzenie nowego projektu od podstaw jest dostępne z górnego menu File › New › OMNeT++ Project. Pomocny skrót klawiszowy to Shift + Alt + N.

Na projekt składa się wiele plików a najważniejsze z nich można zaliczyć do kilku kategorii:

- kod źródłowy w C++ w plikach z rozszerzeniem `.cc` wraz z plikami nagłówkowymi `.h`,
- pliki .ned zawierające modele komponentów obecnych w symulowanych sieciach, jest to podstawowy plik opisujący sieć (ang. *Network Description File*, NED)
- pliki `.msg` opisujące wiadomości przesyłane między symulowanymi urządzeniami,
- pliki `.ini` z konfiguracją symulacji.

By lepiej zrozumieć środowisko OMNeT++ w dalszej części zrealizujemy prostą symulację.

2.1.1 Problemy z OMNeT++

Kompilacja W przypadku problemów z kompilacją środowiska OMNeT++ należy cierpliwie i uważnie przeczytać wyświetlone komunikaty. Typowym problemem jest brak potrzebnych bibliotek dodatkowych, które należy zainstalować wcześniej. W popularnych systemach rodziny GNU/Linux takich jak Debian, Ubuntu, Mint czy Fedora, instalacja tych dodatkowych bibliotek potrzebnych na etapie kompilacji jest trywialnie prosta, gdyż znajdują się one w repozytoriach systemu.

Później, w przypadku wystąpienia problemów z kompilacją symulacji, również należy analizować komunikaty błędów i ostrzeżeń. Typowym błędem osób z małym doświadczeniem programistycznym jest ignorowanie tych wiadomości.

Kolorystyka środowiska OMNeT++ Znaną bolączką środowisk opartych o Eclipse jest problem palety kolorów. Zdarza się to szczególnie w sytuacji, gdy w systemie operacyjnym korzystamy z ostatnio modnego, tak zwanego „ciemnego motywu". W takiej sytuacji niektóre z kolorów paneli OMNeT++ mogą być nieczytelne – na przykład czarny lub granatowy tekst na ciemno szarym tle.

Należy wówczas poszukać odpowiedniego ustawienia wybierając menu górnego Window › Preferences. Znalezienie właściwej opcji może wymagać pewnej dozy cierpliwości, ze względu na ogromną liczbę elementów o różnych, ustawialnych kolorach oraz nie zawsze oczywiste umiejscowienie tych ustawień w drzewie preferencji. Z pomocą przychodzi pole wyszukiwania.

Na przykład zmiana kolorów konsoli z informacjami o postępach kompilacji w drzewku ustawień znajduje się w C++ › Build › Console.

2.2 Wykonanie ćwiczenia

Wybieramy z górnego menu File › New › OMNeT++ Project... po czym otworzy się okno pozwalające na nadanie projektowi nazwy. Po kliknięciu Next >, możemy wybrać wzorzec projektu. Wybierzmy „Empty project". Można przejść przez kolejne okna dialogowe klikając Next >, by zakończyć tworzenie projektu przyciskiem Finish.

2.2.1 Deklaracja sieci i urządzeń – plik NED

Jak zostało wspomniane wcześniej, podstawowym plikiem opisującym symulowaną sieć jest plik z rozszerzeniem `.ned`. Tworzymy więc taki plik poprzez wybranie File › New › Network Description File (NED) i wybieramy w pojawiającym się oknie nasz projekt. Alternatywnie można prawym przyciskiem myszy kliknąć nazwę projektu w panelu „Project Explorer" i dalej podążyć przez New › Network Description File (NED). W drugim rozwiązaniu nie trzeba wybierać projektu z listy ale oba docelowo prowadzą do tego samego punktu, czyli nadania nazwy takiej jak na przykład `Network1.ned`. Musimy kliknąć Next po czym możemy wybrać szablon tego pliku. Wybierzmy „Empty NED file" i zatwierdźmy wybór przyciskiem Finish.

W głównym panelu otworzy się plik NED, który jest zwykłym plikiem tekstowym. Plik ten domyślnie otworzy się w trybie Design, czyli graficznego podglądu komponentów sieci. Aby plik edytować należy kliknąć zakładkę Source u dołu głównego panelu. Nasza pierwsza symulacja powinna zawierać dwa fragmenty, której niżej zostały przedstawione w listingach 1 oraz 2.

```
1 simple Node1
2 {
3     gates:
4         input in;
5         output out;
6 }
```

Listing 1. Opis modelu węzła w pliku NED.

Pierwszy fragment tego pliku opisuje model prostego węzła (urządzenia) sieci, nazwanego Node1. Urządzenie tego typu ma dwie bramki do przesyłania komunikatów: wejściową i wyjściową nazwane odpowiednio `in` oraz `out`.

```
8 network Network1
9 {
```

```
    @display("bgb=333,140");
    submodules:
        Alice: Node1 {
            @display("p=67,61");
        }
        Bob: Node1 {
            @display("p=244,61");
        }
    connections:
        Alice.out --> {  delay = 100ms; } --> Bob.in;
        Alice.in <-- {  delay = 100ms; } <-- Bob.out;
}
```

Listing 2. Opis modelu sieci w pliku NED.

Druga część pliku to opis sieci złożonej z dwóch węzłów (`submodules`) nazwanych `Alice` oraz `Bob`, bazujących na deklaracji urządzenia `Node1` z fragmentu pierwszego. W opisie sieci znajdują się również deklaracje połączeń (`connections`) między węzłami wraz z prostym, opóźnieniowym modelem przesyłania komunikatów. W pliku widzimy też linie zaczynające się od `@display`, które ustalają wygląd węzłów i sieci w ich graficznej reprezentacji dostępnej w zakładce Design.

Niestety, powyższy fragment nie wystarczy, by przeprowadzić symulację. Potrzebny jest jeszcze zapis funkcji węzłów, który jest realizowany w języku C++.

2.2.2 Funkcjonalność węzłów i sieci – kod źródłowy w C++

Plik źródłowy tworzymy podobnie jak plik NED, czyli albo z górnego menu File › New › File lub z menu kontekstowego projektu. W obu przypadkach trzeba plikowi nadać nazwę z rozszerzeniem `.cc`. Podobnie jak plik NED tak i plik z kodem źródłowym w C++ zostanie omówiony fragment po fragmencie w listingach od 3 do 6.

```
#include <string.h>
#include <omnetpp.h>

using namespace omnetpp;
```

Listing 3. Nagłówek kodu symulacji z podstawowymi deklaracjami.

W listingu 3 widzimy typowe zadeklarowanie użycia plików nagłówkowych oferujących potrzebne funkcje oraz deklarację użycia przestrzeni nazw `omnetpp`, co ułatwia pisanie kodu opartego o bibliotekę OMNeT++.

```
class Node1 : public omnetpp::cSimpleModule
{
```

```
8       protected:
9           virtual void initialize() override;
10          virtual void handleMessage(cMessage *msg) override;
11  };
12
13  Define_Module(Node1);
```

Listing 4. Zdefiniowanie modułu węzła wraz z jego funkcjami.

Fragment w listingu 4 jest deklaracją klasy o nazwie Node1 – pamiętamy, że C++ jest językiem obiektowym. Klasa ta bazuje na klasie `cSimpleModule` z biblioteki `omnetpp`. Jest to klasa, która dostarcza szereg funkcji potrzebnych do symulacji węzła w tym cztery podstawowe prototypy:

- void initialize()
- void handleMessage(cMessage *msg)
- void activity()
- void finish()

Te bazowe funkcje wirtualne należy nadpisać własnymi funkcjami, co będzie pokazane dalej. Należy zwrócić uwagę na łatwy do zapomnienia a wymagany przez składnię średnik w linii 6, czyli za deklaracją klasy `Node1`. Ponieważ stawianie średnika za nawiasem klamrowym nie jest typowe to łatwo zapomnieć, iż w tym miejscu jest on wymagany. Ostatnia linia z powyższego fragmentu kodu to wywołanie makra, które rejestruje klasę `Node1` jako widoczny dla OMNeT++ typ modułu, czyli urządzenia w symulowanej sieci.

```
15  void Node1::initialize()
16  {
17      if (strcmp("Alice", getName()) == 0) {
18          cMessage *msg = new cMessage("Message");
19          send(msg, "out");
20      }
21  }
```

Listing 5. Funkcja inicjalizacji węzła.

Fragment z listingu 5 to nadpisana metoda inicjalizacji urządzenia opisanego klasą `Node1`. Jeśli nazwą danego węzła sieci jest „Alice”, to stworzony zostanie nowy komunikat o treści „Message” i wysłany poprzez kanał „out”. To warunkowe przygotowanie komunikatu powoduje, iż tylko jedno z dwóch urządzeń istniejących w symulowanej tu sieci, czyli „Alice” rozpocznie swe funkcjonowanie od wysłania komunikatu przez bramkę o nazwie „out”.

Zadanie

- Zmień treść przekazywanego komunikatu.
- Zlikwiduj warunkowość tworzenia komunikatu podczas inicjalizacji urządzenia tak, aby wszystkie urządzenia sieci rozpoczynały pracę od wysyłania wiadomości.

```
23 void Node1::handleMessage(cMessage *msg)
24 {
25     send(msg, "out");
26 }
```

Listing 6. Funkcja obsługi otrzymanej wiadomości.

Końcowy fragment kodu, przedstawiony na listingu 6 to nadpisanie funkcji obsługującej odbieranie wiadomości przez węzeł sieci. Działanie funkcji pokazanej tutaj jest bardzo proste i polega na przekazaniu odebranej wiadomości do bramki „out". Zgodnie z powyższym kodem urządzenia przekazują więc sobie tę samą wiadomość w tę i z powrotem.

Zadanie

- Zaimplementuj prosty licznik odebranych wiadomości.
- Zaimplementuj dynamiczną zmianę treści wiadomości przekazywanej do bramki „out".

2.2.3 Konfiguracja symulacji – plik inicjalizacyjny

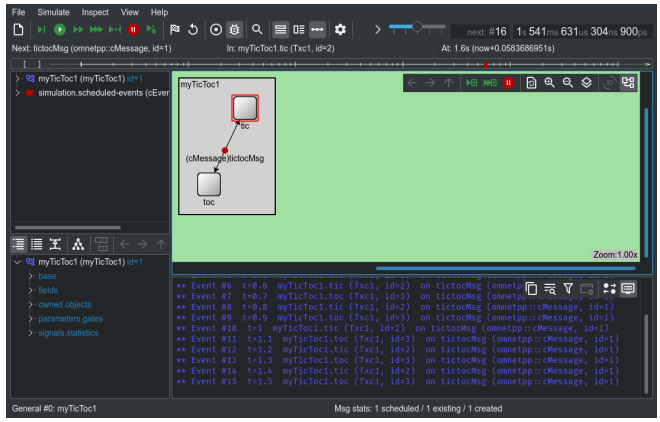
Ostatnią czynnością przed uruchomieniem symulacji jest stworzenie pliku inicjalizacyjnego. W tym celu wybieramy z menu górnego File › New › Initialization File (ini) lub analogiczną opcję z menu kontekstowego projektu. Plik ten otrzyma domyślną nazwę „omnetpp.ini", którą zostawiamy w tej postaci. Klikamy Finish po czym w głównym panelu ukaże się widok pliku konfiguracyjnego. Konfiguracja ta ma formę drzewa i domyślnie otwiera się gałąź „General". Przy polu tekstowym „Network to simulate" dostrzegamy niewielką, czerwoną ikonę informującą o błędzie polegającym na tym, że brakuje nazwy symulacji. Powinna to być wcześniej podana nazwa sieci a więc wprowadzamy myTictoc1 i już można uruchomić symulację.

2.2.4 Symulacja

Uruchomienie symulacji może być wykonane:

- z górnego menu: Run > Run,
- za pomocą skrótu klawiszowego Ctrl + F11,
- odpowiednią ikonką w górnym menu graficznym.

Środowisko może w tym momencie zapytać o zmianę konfiguracji, jeśli wcześniej projekt był debugowany, co potwierdzamy przyciskiem Yes. Jeśli wszystkie wcześniejsze kroki zostały wykonane prawidłowo to na ekranie powinno pojawić się nowe okno z wizualizacją symulacji. Pokazane jest ono na rysunku 7.



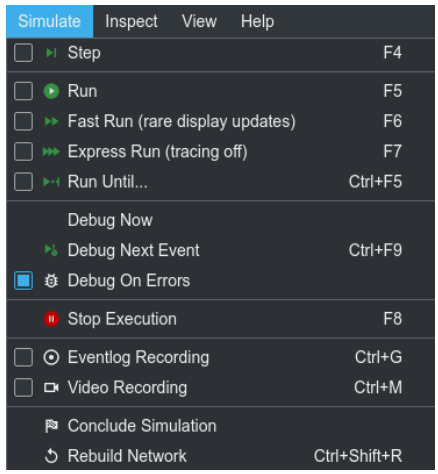
Rysunek 7. Okno symulacji sieci w OMNeT++.

W oknie symulacji zwraca uwagę duża przestrzeń z graficznym obrazem symulowanej sieci. Poniżej tego schematu pojawiają się komunikaty o zdarzeniach, jakie nastąpiły w czasie symulacji. Po lewej stronie mamy drzewka przedstawiające użyte struktury danych. Od góry mamy kolejno: typowe menu tekstowe i graficzne oraz licznik i oś czasu. Oś czasu jest logarytmiczna, co pozwala spojrzeć relatywnie daleko w przyszłość na zdarzenia zaplanowane do zasymulowania oraz z dużą rozdzielczością na zdarzenia dziejące się w przybliżeniu *teraz*. Zdarzenia na osi są zaprezentowane przez czerwone kropki, które przemieszczają się w lewo, czyli w stronę chwili $t = 0$, która odpowiada dokładnie *teraz*. Zdarzenia są numerowane i rejestrowane a to pozwala analizować je po zakończeniu symulacji.

By uruchomić symulację należy skorzystać z jednej z opcji w menu Simulate lub odpowiedniej ikony w menu graficznym. Rozwinięte menu Simulate pokazane jest na rysunku 8.

Symulację można uruchamiać w kilku trybach:

- tryb krokowy „Step” (F4), w którym symulacja pauzuje na każdym zdarzeniu,



Rysunek 8. Menu Simulate z różnymi sposobami uruchamiania symulacji.

- podstawowy tryb „Run” (F5), w którym możemy obserwować symulację w czasie rzeczywistym,
- tryb szybki „Fast Run” (F6) ze sporadycznymi aktualizacjami wizualizacji symulacji,
- tryb ultraszybki „Express Run” (F7) bez wizualnego śledzenia zdarzeń,
- tryb ograniczony czasem lub n-tym zdarzeniem.

By zakończyć symulację należy wybrać z menu Simulate > Stop Execution, albo adekwatną ikonę w górnym menu graficznym, albo po prostu skorzystać ze skrótu klawiszowego F8.

Zadanie

- Przećwicz wyżej opisane metody wykonywania symulacji.

2.2.5 Grafika symulacji

W pokazanym wyżej fragmencie pliku NED dało się zauważyć atrybuty `display`. Określały one wymiary płaszczyzny symulacji oraz położenie na niej ikon reprezentujących urządzenia. Atrybuty te mają większe możliwości i pozwalają w szerokim zakresie zmieniać wygląd graficznej reprezentacji urządzenia.

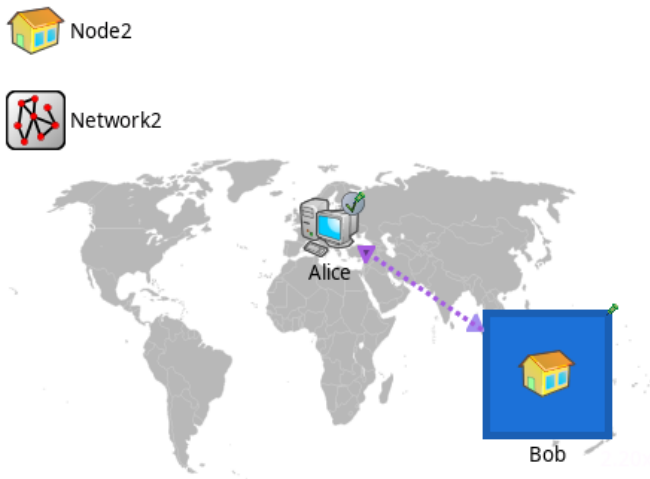
Są trzy podstawowe sposoby ustalania wyglądu elementów graficznych podczas tworzenia symulacji.

- za pomocą odpowiedniego kodu w pliku NED,
- za pomocą interfejsu graficznego dostępnego zakładce „design”:

– z menu kontekstowego elementu, który chcemy zmieniać, należy wybrać Properties,
– po uprzednim wybraniu elementu lewym przyciskiem myszy należy użyć skrótu klawiszowego Ctrl + Enter.

Kolory reprezentowane są w powszechnie znanym standardzie trójbajtowej wartości szesnastkowej RGB, poprzedzonej znakiem „#". Zbiór ikon dostarczanych z OMNeT++ znajduje się w katalogu aplikacji, w podkatalogu images.

Przykładowy, nieco bardziej złożony wygląd poprzedniej symulacji pokazany jest na rysunku 9.



Rysunek 9. Przykładowy widok sieci o zmodyfikowanych właściwościach wyglądu.

Zadanie

- Poeksperymentuj ze zmianami wyglądu symulacji.

2.2.6 Prosta diagnostyka symulacji

Od niniejszego miejsca omawiany kod wykorzystuje więcej niż jedną klasę. Wcześniej używana klasa `Node1()` została rozbudowana i zduplikowana ze zmianami nazw. W ten sposób powstały klasy `Alice()` oraz `Bob()` a następnie, na ich podstawie obiekty z analogicznymi nazwami. Dzięki temu w kodzie można uniknąć widzianego wcześniej porównywania łańcucha takiego jak „Alice" z nazwą obiektu pobieraną metodą `getName()`, w celu uzyskania różnego zachowania odmiennych obiektów. W dalszych przykładach obiekty urządzeń Alice i Bob zachowują się w różny sposób.

Komunikaty diagnostyczne, które wyświetlają się podczas symulacji można wygenerować przesyłając odpowiedni tekst do strumienia EV. Przykładowy kod realizujący to zadanie w ramach konstruktora obiektu reprezentującego urządzenie pokazany jest na listingu 7.

```
60 void Bob::initialize()
61 {
62     EV << "Starting..." << this->getName() << endl;
63 }
```

Listing 7. Wyświetlanie komunikatów diagnostycznych symulacji.

Słowo kluczowe języka C++ „this” odnosi się do obiektu a więc urządzenia symulowanego na podstawie jego klasy. W ten sposób można pobrać parametr konkretnego obiektu, w powyższym przykładzie – jego nazwę.

Komunikaty diagnostyczne mogą być dłuższe i wyświetlać wiele atrybutów różnych obiektów, takich jak identyfikatory otrzymywanych wiadomości, czas ich otrzymania, informacje o nadawcy i inne.

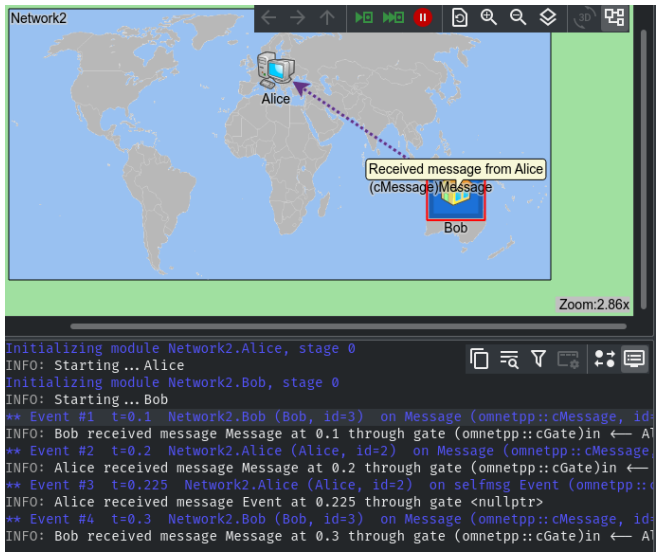
Przydatne w diagnostyce symulacji mogą być też chmurki z tekstem, wyświetlenie przy ikonce urządzenia. Tekst chmurki ustalany jest za pomocą prostej funkcji `bubble()`, która pobiera tylko jeden parametr, jakim jest prosty łańcuch znaków lub wskaźnik na taki łańcuch. Ten łańuch znaków w prosty sposób da się przygotować za pomocą funkcji takiej jak `snprintf()` a następnie podesłać funkcji `bubble()`. Chmurka po chwili wyświetlania samoczynnie znika z ekranu.

Przykład użycia powyższych możliwości pokazany jest na listingu 8.

```
65 void Bob::handleMessage(cMessage *msg)
66 {
67     char floatingBubble[100];
68
69     EV << this->getName() << " received message " << msg->getName()
       << " at " << msg->getArrivalTime() << " through gate " <<
       msg->getArrivalGate() << endl;
70     snprintf(floatingBubble, 100, "Received message from %s",
       msg->getSenderModule()->getName());
71     bubble(floatingBubble);
72     send(msg, "out");
73 }
```

Listing 8. Wyświetlanie komunikatów diagnostycznych symulacji.

Efekty działania powyższego kodu w postaci różnorodnych komunikatów diagnostycznych przedstawione są na rysunku 10.

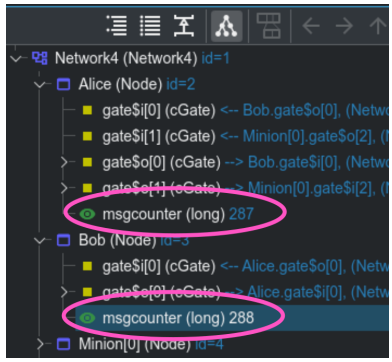


Rysunek 10. Różne komunikaty diagnostyczne w oknie symulacji.

Wizualizacje na graficznym kanwasie są bardzo dobrym rozwiązaniem na początek, dla małych sieci. Podejście to przestaje być najlepszym rozwiązaniem, gdy sieć ma jedną z cech takich jak:

- duża liczba urządzeń,
- duża liczba przekazywanych komunikatów,
- długi czas symulacji.

Bardziej zaawansowane symulacje w ogóle nie posługują się wizualizacją. Krokiem w tę stronę jest użycie makra `WATCH(zmienna)`, które dodaje zmienną do obserwowanych w ramach inspektora obiektów. Inspektor obiektów prezentujący zmienną licznika odebranych wiadomości pokazany jest na rysunku 11.



Rysunek 11. Obserwacja zmiennej za pomocą inspektora obiektów.

Zmienną do obserwacji należy dodać tylko raz a najprościej zrobić to podczas inicjalizacji obiektu, tak jak w przykładzie z listingu 9.

Zmienna ta jest prywatnym atrybutem każdego obiektu, musi wiec znaleźć się w definicji klasy, tak jak pokazane jest to w listingu 10.

```
6  class Node : public omnetpp::cSimpleModule
7  {
8      private:
9          long msgcounter = 0;
```

Listing 10. Zmienna licznika wiadomości jako prywatny atrybut.

Zmienna w inspektorze obiektów będzie zmieniała swą wartość oczywiście tylko wtedy, gdy kod aplikacji będzie tę wartość zwiększał.

Alternatywnie do powyższych metod zawsze można posłużyć się bardzo dobrym debuggerem dostępnym w OMNeT++/Eclipse, tak jak w przypadku tworzenia każdego innego oprogramowania, jednak może to być bardziej uciążliwe.

Zadanie

- Dodaj w metodzie `handleMessage()` zwiększanie licznika odebranych wiadomości i obserwuj zmianę tej wartości jednocześnie dla urządzeń Alice i Bob, za pomocą inspektora obiektów.

2.2.7 Zdarzenia w czasie

Rzeczywiste mikrokontrolery, stosowane w urządzeniach IoT, są wyposażone w liczniki zegarowe, tak zwane *timery*, dzięki którym można w nich zaplanować zadania do wykonania po upłynięciu pewnego czasu lub zadania cyklicznie. Podobny efekt da się uzyskać w symulacji.

Symulowane urządzenie może zaplanować zdarzenie do wykonania, które to zdarzenie trafi do niego samego ale po pewnym czasie. Można powiedzieć, że z punktu widzenia urządzenia jest to „notatka dla samego siebie z przyszłości". Do zaplanowania zdarzenia służy funkcja pokazana w listingu 11.

```
19 void Node::initialize()
20 {
21     WATCH(msgcounter);
```

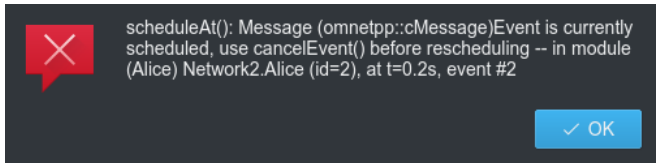
Listing 9. Dodawanie zmiennej do obserwowanych.

```
47          scheduleAt(simTime() + 0.025, event);
```

Listing 11. Zaplanowanie zdarzenia, do realizacji w przyszłości.

Aktualny czas symulacji można uzyskać funkcją `simTime()`. Funkcja ta zwraca zmienną typu `simtime_t`, która faktycznie jest tylko typem danych zdefiniowanym według klasy `SimTime`. W szczegółach może jest to odrobinę skomplikowane ale w użyciu wystarczy zapamiętać, że podstawową jednostką czasu w OMNeT++ jest sekunda, przy czym czas podajemy jako liczbę rzeczywistą a więc 0.025 oznacza 25 milisekund. W powyższym przykładzie zdarzenie nazwane `event` zostało zaplanowane na chwilę 25 milisekund od *teraz*. Mowa tu oczywiście o czasie symulacji a nie o czasie rzeczywistym.

Zaplanować zdarzenie można w dowolnym momencie pod warunkiem, że ono istnieje ale jeszcze nie zostało zaplanowane. Ponowna próba zaplanowania tego samego zdarzenia kończy się błędem wykonania takim jak na rysunku 12.



Rysunek 12. Komunikat błędu podczas wykonywania symulacji będący efektem próby ponownego zaplanowania tego samego zdarzenia.

Zdarzenie trafi do urządzenia, które je sobie zaplanowało w postaci wiadomości. Wiadomość ta musi jednak być uprzednio zadeklarowana i zainicjowana. Deklarowanie zdarzenia – wiadomości, jako atrybutu klasy, pokazane jest na listingu 12.

```
6  class Alice : public omnetpp::cSimpleModule
7  {
8      private:
9          cMessage *event = nullptr;
```

Listing 12. Deklaracja zdarzenia.

Natomiast inicjalizacja zdarzenia następuje w konstruktorze, tak jak zostało to pokazane na listingu 13.

```
26 void Alice::initialize()
27 {
28     EV << "Starting..." << this->getName() << endl;
29     event = new cMessage("Event");
```

Listing 13. Inicjalizacja wiadomości typu zdarzenie.

Poniżej, na listingu 14 pokazana jest cała metoda `handleMessage()` klasy `Alice()`.

```
34 void Alice::handleMessage(cMessage *msg)
35 {
36     EV << this->getName() << " received message " << msg->getName()
       << " at " << msg->getArrivalTime() << " through gate " <<
       msg->getArrivalGate() << endl;
37     if ( msg == event ) {
38         cDisplayString& displayString = getDisplayString();
39         displayString.setTagArg("i", 2, "0");
40         displayString.setTagArg("is", 0, "n");
41     } else {
42         bubble("Received message");
43         cDisplayString& displayString = getDisplayString();
44         displayString.setTagArg("i", 2, "50");
45         displayString.setTagArg("is", 0, "l");
46         send(msg, "out");
47         scheduleAt(simTime() + 0.025, event);
48     }
49 }
```

Listing 14. Reakcja na otrzymaną wiadomość i zdarzenie w klasie `Alice()`.

Jeśli urządzenie „Alice” otrzyma wiadomość typu `event`, to:

- zdejmuje zabarwienie ikony w oknie symulacji
- ustawia jej rozmiar na *normal*.

W przypadku otrzymania „zwykłej” wiadomości a więc od urządzenia „Bob”, z ikonką „Alice” dzieją się dwie rzeczy:

- nadawane jest 50% zabarwienie kolorem zadeklarowanym w pliku NED,
- zmieniana jest jej wielkość na *large*.

Zatem po każdej wiadomości od Boba, ikonka Alice staje się większa i nabiera innego odcienia by po 25 milisekundach czasu symulacji powrócić do normalnego wyglądu. Pozwala to zwizualizować na przykład przetwarzanie odebranego pakietu przez urządzenie. W bardziej zaawansowanej wersji może to być czas potrzebny na przygotowanie wiadomości odpowiedzi. Taki kod, realizowany przez `Alice()`, pokazany jest na listingu 15.

```
37 void Alice::handleMessage(cMessage *msg)
38 {
```

```
39      double processingTime;
40      if ( msg != event ) {
41          rcvdmsg = msg;
42          processingTime = uniform(0.1, 1.1);
43          EV << "Processing of message will take " << processingTime
        << " seconds.";
44          scheduleAt(simTime() + processingTime, event);
45      } else {
46          send(rcvdmsg, "out");
47      }
48  }
```

Listing 15. Symulacja zmiennego czasu przygotowywania odpowiedzi przez `Alice()`.

W powyższym przykładzie, po odebraniu zwykłej wiadomości od Boba, Alice zapamiętuje tę wiadomość w atrybucie o nazwie `rcvdmsg`. Następnie losuje czas symulowanego przetwarzania tej wiadomości za pomocą funkcji `uniform()`, która zwraca wartość typu `double` w zadanym przedziale. W przykładzie losowany jest czas między 100 ms a 1,1 s. Następnie szereguje do wykonania w przyszłości zdarzenie `event`. W przypadku odebrania zdarzenia `event`, kod wysyła wiadomość uprzednio zapamiętaną w `rcvdmsg` poprzez bramkę „out".

Bob realizuje inne zadanie – oczekuje, iż otrzyma odpowiedź w założonym czasie. By to sprawdzić, po wysłaniu wiadomości szereguje zdarzenie event do wykonania po 1 sekundzie od wysłania swej wiadomości do Alice. Jeśli wiadomość od Alice dotrze w czasie poniżej 1 sekundy, to poprzednie zdarzenie `event` jest anulowane i startowane jest nowe, z nowo ustawionym licznikiem czasu. Jest to realizowane we fragmencie metody handleMessage(), pokazanej na listingu 16.

```
79  void Bob::handleMessage(cMessage *msg)
80  {
81      if ( msg != event ) {
82          cancelEvent(event);
83          msgCounter++;
84          send(msg, "out");
85          scheduleAt(simTime() + 1.0, event);
86      } else {
```

Listing 16. Bob ustawia czas oczekiwania na wiadomość od Alice na czas 1 sekundy.

Jeśli komunikat od Alice nie dotrze na czas, czyli w ciągu 1 sekundy to wiadomość otrzymana przez metodę `handleMessage()` jest typu `event`. Wówczas zwiększany jest licznik przekroczeń czasu oczekiwania oraz prosty komunikat pojawia się w „dymku" nad ikoną Boba. Odpowiedni fragment kodu pokazany jest na listingu 17.

```
86      } else {
87          timeoutCounter++;
88          bubble("Timeout!");
89      }
90  }
```

Listing 17. Bob odnotowuje spóźnioną odpowiedź od Alice.

W powyższych przykładach przeanalizowaliśmy wykorzystanie licznika czasu symulacji oraz metody planowania zdarzeń do realizacji w przyszłości. Nawet, jeśli nie jest to najbardziej efektywny sposób ich programowania, to jest on prosty do użycia i może okazać się przydatny.

2.2.8 Dwukierunkowe łącza symetryczne

Rzeczywiste łącza sieciowe mogą być asymetryczne i w przypadku IoT takie podejście jest często spotykane. Inną przepustowość ma łącze „w górę" (ang. *uplink*) a inną łącze „w dół" (ang. *downlink*). Z popularnych standardów IoT o asymetrycznych łączach należy wymienić NB-IoT oraz Sigfox.

W OMNeT++ oczywiście można symulować zarówno łącza symetryczne jak i asymetryczne. Do tej pory posługiwaliśmy się łączami asymetrycznymi, jednak zapewne miały one ustawione te same parametry. Dla przypomnienia można spojrzeć na listing 2 pokazujacy fragment pliku NED, gdzie wprowadzane są odpowiednie deklaracje.

Wiele systemów transmisji zapewnia jednak łączność symetryczną. Spośród istotnych dla IoT należy wymienić: WiFi, BLE, ZigBee, LoRa.

Wiadomo, że fizyczne łącze między urządzeniami ma pewne parametry, których ciągłe wprowadzanie może być uciążliwe przy większej liczbie symulowanych urządzeń. Ponadto może być przydatna możliwość wprowadzenia opisu łącza tylko raz a następnie używanie go wielokrotnie – dla wielu połączeń pomiędzy różnymi urządzeniami.

Do tej pory widzieliśmy prosty model opóźnieniowy, wprowadzony w pliku NED, poniżej słowa kluczowego *connections*. Można taki kanał zadeklarować raz, tak jak pokazane jest to na listingu 18, gdzie został on nazwany `Channel`.

Przy okazji widzimy tu określenie koloru (RGB HTML), grubości i stylu linii reprezentującej ten kanał. Dostępne są trzy style linii:

- ciągła – „s" (domyślna),
- kreskowana – „da",
- punktowa – „d".

Dwukierunkowe, symetryczne łącze między urządzeniami Alice i Bob na bazie tego kanału będzie zadeklarowane w pliku NED tak, jak na listingu 19.

```
35      connections:
```

```
network Network4
{
    @display("bgb=1000,500,#C2DFFF,,0");
    @display("bgi=maps/world,s");
    types:
        channel Channel extends ned.DelayChannel
        {
            delay = 200ms;
            @display("ls=#CACA40,5,da");
        }
```

Listing 18. Deklaracja parametrów kanału łączności w prostym modelu opóźnieniowym.

```
        Alice.gate++ <--> Channel <--> Bob.gate++;
```

Listing 19. Dwukierunkowe, symetryczne łącze `Channel`.

Aby powyższy kod działał, trzeba inaczej niż dotychczas zadeklarować bramki danych wchodzących (dotychczas: „in") i wychodzących (dotychczas: „out"). Realizuje się to w postaci wektora, tak jak na listingu 20.

```
simple Node
{
    @display("i=device/device,,0");
    gates:
        inout gate[];
}
```

Listing 20. Wektoryzowane bramki wejścia-wyjścia.

Kanał w OMNeT++ jest klasą C++. Najprostszym kanałem jest `IdealChannel`, który nie wprowadza żadnych błędów ani opóźnień transmisji, nie ma też atrybutów.

Natomiast powyższy przykład z listingu 18, bazuje na prostym kanale opóźnieniowym `ned.DelayChannel`. Ta klasa ma tylko dwa parametry:

- `delay` – opóźnienie typu `double`, przy czym można podać jednostkę czasu (s, ms, us, ns, ps, fs, as),
- `disabled` – typu `boolean`, domyślnie jest ustawiony na `false` co umożliwia transmisję.

Można też skorzystać z dostępnego modelu `ned.DatarateChannel` o parametrach:

- `datarate` – przepustowość typu `double`, przy czym można dodać przyrostek typowych jednostek (bps, kbps, Mbps, Gbps, Tbps),

- per – stopa błędów pakietów (ang. *Packet Error Rate*, PER), wartość typu double w zakresie od 0 do 1 wyrażająca szansę na błąd transmisji pakietu,
- ber – stopa błędnych bitów (ang. *Bit Error Rate*, BER), wartość typu double w zakresie od 0 do 1 wyrażająca szansę na błąd transmisji bitu,
- delay – opóźnienie, podobnie jak dla klasy `ned.DelayChannel`.

Przykład kanału bazującego na przepustowości bitowej pokazany jest na listingu 21.

```
18        channel YellowChannel extends ned.DatarateChannel
19        {
20            datarate = 100bps;
21            delay = 100ms;
22            ber = 1e-2;
23            @display("ls=#123456,3,d");
24        }
```

Listing 21. Kanał w oparciu o klasę bazową w modelu przepustowości.

W jednej sieci może występować wiele grup urządzeń, posługujacych się różnymi kanałami.

Atrybuty kanału mogą być obliczone na podstawie innych wartości opisujących symulację. Wykorzystać to można do zamodelowania pogarszających się parametrów transmisji radiowej wraz ze wzrostem odległości między komunikującymi się urządzeniami.

W bardziej zaawansowanych symulacjach można samodzielnie, od podstaw zaimplementować klasę opisującą zachowanie kanału transmisyjnego. Jednak nie jest to konieczne, gdyż istniejące klasy można rozszerzać poprzez dodawanie własnych parametrów.

Po stronie kodu źródłowego realizującego logikę urządzeń również należy dokonać pewnych zmian, które odzwierciedlą wektorowy charakter bramek wejścia wyjścia. Dotychczas w celu wysłania wiadomości należało użyć nazwy bramki. Jednak w przypadku zastosowania bramki wektorowej a więc wielokrotnej, sama jej nazwa nie jest jednoznaczna. Trzeba posłużyć się specjalnym sufiksem nazwy oraz numerycznym indeksem. Przykład, jak to zrealizować, pokazany jest na listingu 22.

```
36    send(msg, "gate$o", 0);
```

Listing 22. Wysyłanie wiadomości przez bramkę zadeklarowaną wektorowo.

W powyższym przykładzie widzimy prefiks w postaci nazwy wektora (`gate`) oraz sufiks „$o”, który odpowiada bramkom wyjściowym. Dla bramek wejściowych używany jest sufiks „$i”. Trzecim parametrem funkcji `send()` jest indeks kolejnej bramki wejściowej bądź wyjściowej, liczony od wartości 0.

Zadanie

- Poeksperymentuj z przepustowościami i opóźnieniami kanałów.

2.2.9 Więcej urządzeń

Symulowanie komunikacji pomiędzy dwoma urządzeniami może być przydatne i adekwatnie modelować takie rozwiązania jak na przykład łączność pomiędzy urządzeniem BLE a telefonem komórkowym. Jednak rzeczywiste sieci zazwyczaj składają się z więcej niż dwóch urządzeń.

W opisie łączy dwukierunkowych widzieliśmy wektoryzowane bramki wejścia-wyjścia urządzeń. Podobnie można zwektoryzować urządzenia i „hurtem" wygenerować ich kilka. Pokazuje to przykład na listingu 23.

```
32        Minion[4]: Node {
33            @display("p=100,120,ri,100,150");
34        }
```

Listing 23. Deklaracja wielu urządzeń za pomocą wektora.

Różnica w deklarowaniu wielu urządzeń jest zatem niewielka w porównaniu z deklarowaniem ich pojedynczo. Należy jednak zwrócić uwagę na opis ich pozycjonowania na płaszczyźnie symulacji. W powyższym przykładzie zostaną one rozłożone na elipsie („ri") wpisanej w prostokąt, którego górny-lewy narożnik ma pozycję $(x = 100, y = 120)$ a jego wymiary szerokości (X) i wysokości (Y) to odpowiednio 100 i 150. Punkt $(0, 0)$ znajduje się w lewym-górnym narożniku obszaru symulacji.

Rozłożenie urządzeń na płaszczyźnie może być zrealizowane kilkoma sposobami, z parametrami analogicznymi do pokazanego wyżej przykładu:

- w rzędzie – „p=x,y,r,deltaX",
- w kolumnie – „p=x,y,c,deltaY",
- macierzowo – „p=x,y,m,liczbakolumn,deltaX,deltaY",
- pierścień – „p=x,y,ri,rx,ry",
- dokładne (**ex**act) – „p=x,y,x,deltaX,deltaY"

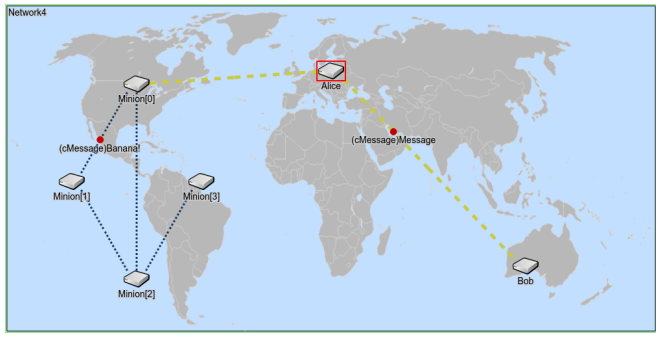
Podczas układania wizualnej strony symulacji, wszystkie urządzenia z jednego wektora prezentowane w zakładce „design" znajdują się w tym samym miejscu i nie należy się tym przejmować. Gdyby nie zastosować pokazanej wyżej linijki z dyrektywą `display`, to również podczas uruchomienia symulacji byłyby w tym samym miejscu, co utrudniłoby obserwacje.

Zadeklarowane urządzenia należy jeszcze połączyć w sieć. Kompletny fragment kodu z pliku NED realizujący połączenia między opisanymi wyżej urządzeniami i za pomocą dwóch omówionych wcześniej kanałów jest pokazany na listingu 24.

```
35      connections:
36          Alice.gate++ <--> Channel <--> Bob.gate++;
37          Minion[0].gate++ <--> YellowChannel <--> Minion[1].gate++;
38          Minion[0].gate++ <--> YellowChannel <--> Minion[2].gate++;
39          Minion[1].gate++ <--> YellowChannel <--> Minion[2].gate++;
40          Minion[2].gate++ <--> YellowChannel <--> Minion[3].gate++;
41          Alice.gate++ <--> Channel <--> Minion[0].gate++;
```

Listing 24. Deklaracja wielu urządzeń za pomocą wektora.

Okno wynikowej symulacji pokazane jest na rysunku 13.



Rysunek 13. Symulacja z wieloma urządzeniami o różnorodnych połączeniach.

Zadanie

- Dodaj kolejne urządzenia i rozbuduj sieć.
- Zaimplementuj prosty routing rozwijając metodę przetwarzania odebranej wiadomości.

2.3 Pytania sprawdzające

1. Do czego służy OMNeT++?
2. Jakie są różnice między OMNeT++ a OMNEST?
3. Jakie komponenty zawiera biblioteka INET?
4. Dlaczego o pakiecie OMNeT++ mówimy, iż jest symulatorem zdarzeń dyskretnych?

5. Co zawiera plik typu NED?
6. Jakie metody należy zaimplementować dla modelu prostego modułu komunikacyjnego?

Rozdział 3

Transmisja pakietowa

Cel ćwiczenia

Celem niniejszego ćwiczenia jest zapoznanie się z możliwościami symulowania standardowych transmisji Internetu za pomocą pakietu INET. Część teoretyczna rozdziału obejmuje przegląd wybranych standardów transmisji pakietowej, mających znaczenie dla IoT. W części praktycznej wykonane zostaną symulacje sieci i analiza przekazywanych pakietów.

Elementy transmisji mogą być traktowane niezależnie i modyfikowane bez wpływu na pozostałe ,dzięki zastosowaniu warstwowego modelu ISO/OSI. Typowo wyróżnia się siedem warstw, z którymi być może Czytelnik spotkał się już wcześniej. Dla przypomnienia krótko je tutaj wymienimy, bez charakteryzowania, zaczynając od warstwy najwyższej:

- aplikacji,
- prezentacji,
- sesji,
- transportowa,
- sieciowa,
- łącza danych,
- fizyczna.

Niniejsze ćwiczenie dotyczy głównie obszarów od warstwy sieciowej w górę.

Bywa w niektórych rozwiązaniach, że trzy najwyższe warstwy są mocno zintegrowane i trudno wskazać wyraźną granicę pomiędzy nimi. Dlatego często traktuje się łącznie, nazywając po prostu warstwą aplikacji. Takie podejście jest obecne w prostszym – bo czterowarstwowym modelu sieci, wywodzącym się jeszcze z sieci Arpanet, opracowanym przez Departament Obrony USA (ang. *Department of Defense*, DoD).

Realizacja IoT opiera się o typowe protokoły Internetu, takie jak:

- protokoły warstwy sieciowej (internetowej): IPv4 (ang. *Internet Protocol*, IP) oraz IPv6,
- protokoły warstwy transportowej: UDP, TCP,
- protokoły warstwy aplikacji: MQTT, CoAP, REST.

Protokołów i standardów istotnych dla IoT jest oczywiście znacznie więcej, jednak tutaj wymienione zostały wybrane, najważniejsze, mające znaczenie dla niniejszego ćwiczenia.

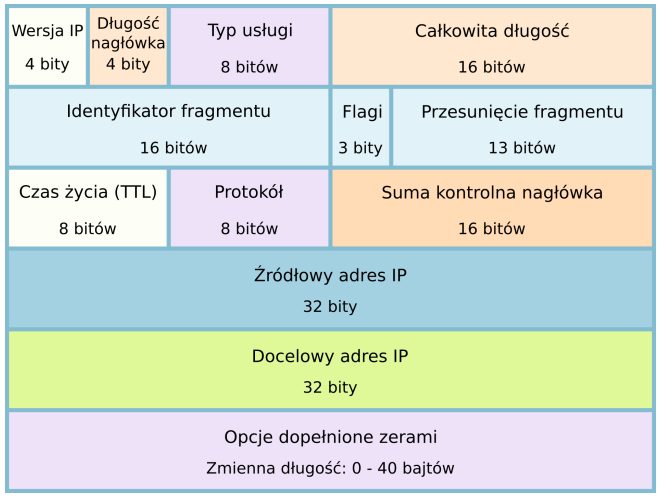
3.1 Wybrane protokoły Internetu

Poniższy fragment nie stanowi wyczerpującego przedstawienia protokołów ze względu na ograniczoną objętość niniejszego podręcznika. Zostaną zaprezentowane jedynie kluczowe aspekty standardów o istotnym znaczeniu dla IoT oraz wskazane publikacje, w których można znaleźć więcej informacji.

3.1.1 IPv4

Protokół IPv4 jest jednym z podstawowych protokołów Internetu, opracowanym już w roku 1981 [14]. Struktura nagłówka pakietu IPv4 pokazana jest na rysunku 14.

Zadaniem tego protokołu jest umożliwienie dostarczenia pakietów danych między odległymi urządzeniami określonymi ich adresami IP. W wersji IPv4 adres ten składa się z czterech bajtów. Teoretycznie daje to $2^{32} \approx 4,3 \times 10^9$ możliwych kombinacji [15].



Rysunek 14. Struktura nagłówka pakietu IPv4.

Jednak znaczne obszary przestrzeni adresowej mają specjalne przeznaczenie lub są w posiadaniu firm i organizacji, które do Internetu przystąpiły na wczesnym etapie jego rozwoju.

Strumień danych z aplikacji po stronie nadawcy jest dzielony na pakiety. Dla sieci Ethernet przyjęto, że górny limit wielkości pakietu wynosi 1500 bajtów. Wartość ta zależy od parametrów łącza i działania protokołów w innych warstwach. Pakiety przekraczające limit są dzielone na fragmenty. Każdy z fragmentów opisany jest adresem nadawcy, odbiorcy oraz pozycją wśród innych fragmentów zrealizowaną za pomocą identyfikatora oraz przesunięcia względem innych fragmentów. Identyfikatory i przesunięcie pozwalają na połączenie fragmentów pakietu u odbiorcy. Protokół IP nie gwarantuje jednak dostarczenia wiadomości i nie przekazuje informacji zwrotnej o dostarczeniu poszczególnych pakietów.

3.1.2 IPv6

Protokół IPv4 został ściśle powiazany z jego przestrzenią adresową, która w początkach Internetu wydawała się wystarczająca. Wkrótce jednak okazało się, że liczba urządzeń w sieci gwałtownie rośnie i przekracza możliwości tego protokołu.

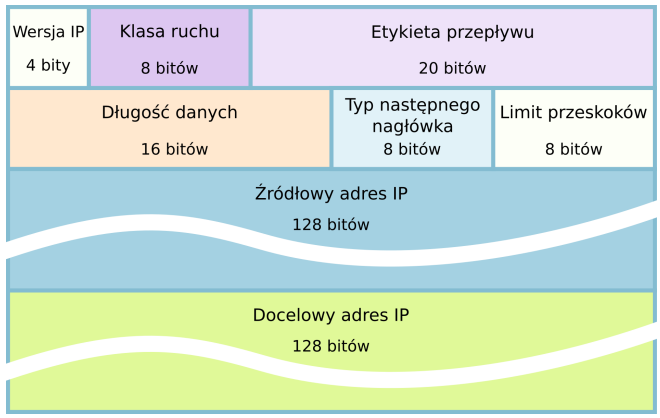
Tymczasowe rozwiązanie tego problemu oparte o sieci lokalne i translację adresów sieciowych (ang. *Network Address Translation*, NAT) mają wiele zalet ale wprowadzają dodatkowy narzut na przetwarzanie danych i stoją w sprzeczności z pierwotną wizją Internetu. Ponadto zauważono, że znaczne ramka IPv4 zawiera zbędne fragmenty a sam protokół wymusza działanie sieci w sposób nie przystający do postępu technicznego w warstwie fizycznej i łącza danych oferowanego przez sieci światłowodowe i bezprzewodowe.

Dlatego podjęto prace nad nową wersją protokołu, czyli IPv6. Jego pierwsza specyfikacja ukazała się już w roku 1995, ale była następnie kilkukrotnie poprawiana [16]. Format nagłówka IPv6 pokazany jest na rysunku 15.

Główne różnice względem IPv4 to:

- przestrzeń adresowa zwiększona do $2^{128} \approx 3,4 \times 10^{38}$,
- autokonfiguracja adresów,
- uproszczony nagłówek o stałej wielkości,
- brak fragmentacji w routerach – oszczędność czasu i energii,
- IPsec – autentykacja urządzeń i szyfrowanie transmisji w warstwie 3 ISO/OSI.

Zwiększona pula adresowa z pewnością jest korzystna dla IoT ze względu na potencjalnie bardzo dużą liczbę urządzeń tego rodzaju. Jednak dłuższe adresy oznaczają większe rozmiary nagłówka, który nie niesie użytecznych informacji. To jest problem dla urządzeń o ograniczonych zasobach energetycznych, które powinny realizować transmisję w sposób jak najwydajniejszy. Jest to szczególnie istotne w przypadku



Rysunek 15. Struktura nagłówka pakietu IPv6.

transmisji bezprzewodowych, na przykład zgodnych ze standardem IEEE 802.15.4 lub BLE. W tym celu opracowano specjalistyczne rozwiązanie, jakim jest 6LoWPAN [17].

Krótkie podsumowaniem protokołu IP w wersjach 4 i 6 stanowi poniższa tabelka porównawcza.

	IPv4	**IPv6**
Od kiedy istnieje	1981	1999
Standard	RFC791	RFC2460
Przestrzeń adresowa	2^{32}	2^{128}
Przykładowe adresy	194.29.151.9	2a00:1450:401b:804::200e
	10.1.1.10	::ffff:10.120.78.40
Adres zwrotny	127.0.0.1	::1
Maskowanie sieci	192.168.1.0/24	2a00:1450:4001::/48
Liczba adresów	około 4 mld	około 340 sekstylionów
Przyznawanie	CIDR	/48 teoretycznie dla każdego
Dopuszczalna fragmentacja	W każdym kroku	Tylko u nadawcy
Transmisja rozgłoszeniowa	Broadcast	Multicast, anycast
Bezpieczeństwo	Zależy od warstwy wyższej	Gwarantowane przez IPSEC
Długość nagłówka	Zmienna: 0-20 bajtów	Stała: 40 bajtów
Minimalne MTU	576	1280

3.1.3 UDP

Protokół UDP, czyli User Datagram Protocol, jest protokołem warstwy transportowej [18]. Służy do przekazywania pakietów IP pomiędzy odległymi urządzeniami sieciowymi, również poprzez routery pośredniczące. Nadawca nie otrzymuje potwierdzenia odebrania wiadomości przez adresata a tym samym nie ma gwarancji, że

komunikacja za pomocą UDP w ogóle się powiodła. Można więc zastanawiać się jakie są możliwe zastosowania takiego protokołu [15].

Protokół UDP, dzięki swojej prostocie i natychmiastowości, dobrze nadaje się do takich transmisji, przy których można sobie pozwolić na stratę niektórych pakietów. Nie wprowadza dodatkowego narzutu w postaci weryfikowania skuteczności komunikacji i jest protokołem bezstanowym. Pozwala to na relatywnie wiarygodną łączność w czasie rzeczywistym a więc na przykład transmisję strumieniową obrazu i dźwięku do wielu odbiorców jednocześnie, dzięki zastosowaniu technik takich jak *broadcast* i *multicast*. Ponadto UDP stosuje się w znanych usługach, takich jak DHCP, DNS oraz mniej znanych jak NFS, NTP, TFTP, SNMP i wielu innych.

Mylne jest jednak traktowanie UDP jako umożliwiającego transmisję jednokierunkową. Otwierane są dwa porty: docelowy u odbiorcy – na serwerze usługi sieciowej oraz źródłowy u nadawcy – kliencie tej usługi. Teoretycznie możliwe jest użycie jednego z 65536 portów ale w praktyce niektóre numery są zarezerwowane, część numerów jest ogólnie przyjęta dla konkretnych usług sieciowych a ponadto porty dzielą się na porty usług sieciowych (serwerowe) oraz tak zwane efemeryczne porty służące klientom do nawiązywania połączeń. Mechanizm portów w powiązaniu z routowaniem pakietów za pomocą protokołu IP pozwala na wzajemną komunikację z takim zastrzeżeniem, że kontrolę nad transmitowanymi pakietami po obu stronach ma aplikacja a więc wyższe warstwy stosu ISO/OSI.

Rysunek 16. Struktura nagłówka UDP.

Nagłówek ramki danych UDP ma dość prostą konstrukcję, co widać na rysunku 16, gdzie jest ona pokazana w kontekście użycia protokołu IP oraz transmitowanych danych. Oprócz wspomnianych portów znajduje się w niej również „całkowita długość" ramki, w co wliczane są również nagłówki UDP oraz IP. W przypadku zastosowania IPv4 ramka UDP może mieć zatem $65535 - 20 - 8 = 65507$ B, gdyż 20 B to najkrótszy nagłówek IPv4, natomiast nagłówek UDP ma zawsze długość 8 B. Zakładając użycie IPv6 możliwe są ramki dłuższe niż `0xffff` bajtów. Ostatnim elementem nagłówka UDP jest suma kontrolna liczona jako suma kolejnych wartości 16-bitowych ramki używającej pseudonagłówka (nagłówka tymczasowego z sumą kontrolną równą 0). Jeśli suma ta przekracza zakres wartości 16-bitowej, to należy dodawać jej górną i dolną tak długo, aż w rezultacie uzyskany wynik będzie mieścił się w 16 bitach.

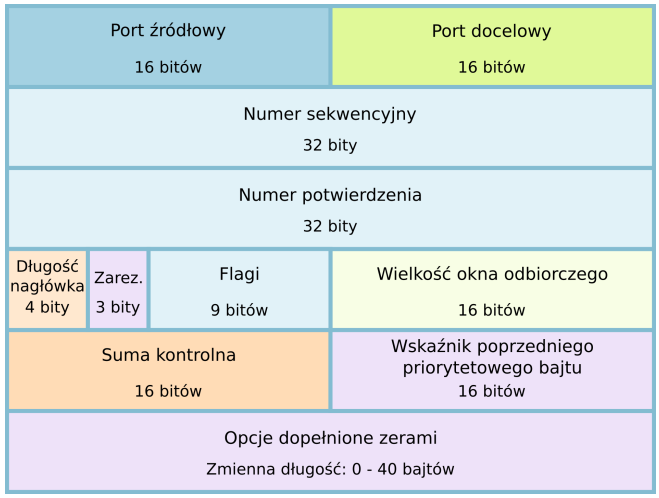
3.1.4 TCP

TCP czyli *Transmission Control Protocol* jest jednym z najpowszechniej kojarzonych protokołów Internetu ze względu na popularne powiązanie TCP/IP. W tym momencie warto zauważyć, że rola protokołu TCP jest bliska tej, jaką spełnia UDP. Obydwa są protokołami warstwy transportowej, która wspiera protokół IP w dostarczaniu pakietów pomiędzy odległymi urządzeniami w sieci, oddzielonymi przez routery [15, 19].

TCP ma jednak takie cechy, które pod pewnymi względami stanowią zaprzeczenie protokołu UDP. TCP jest protokołem, którego główne cechy to:

- połączeniowym – komunikacja jest inicjowana za pomocą trójetapowej sekwencji (SYN, SYN+ACK, ACK) a po zakończeniu transmisji zamykana,
- z kontrolą stanu – działanie gniazda TCP/IP można opisać za pomocą maszyny stanów, takich jak na przykład: `LISTEN`, `ESTABLISHED`, `FIN-WAIT2`, `CLOSED`,
- z synchronizacją – obie strony komunikacji kontrolują poprawność przekazania pojedynczych pakietów a w razie problemów żądają retransmisji.

Nagłówek TCP jest dość złożoną strukturą, mogącą mieć różną wielkość, ze względu na zmienną liczbę użytych „opcji”. Jego diagram pokazany jest na rysunku 17.



Rysunek 17. Struktura nagłówka TCP.

Podobnie jak w przypadku UDP tak i nagłówek TCP rozpoczyna się od portu nadawcy i portu odbiorcy. Mechanizm w obu przypadkach jest niemalże identyczny stąd i taka sama, 16-bitowa wielkość tych pól.

Następnie w nagłówku znajdziemy dwie dość duże, bo 32-bitowe, liczby pozwalające na określanie kolejności pakietów w transmisji i potwierdzanie ich odbioru. Podczas transmisji pakiety mogą dotrzeć do odbiorcy różnymi drogami i w innej kolejności,

niż zostały wysłane. Odbiorca musi zatem śledzić ich kolejność za pomocą numerów sekwencyjnych oraz potwierdzać otrzymanie poszczególnych fragmentów nadawcy.

Następne 4 bity nagłówka zawierają liczbę za pomocą której można określić wielkość nagłówka występującą w danym pakiecie. Ponieważ jest to liczba w zakresie zaledwie od 0 do 15 to posłużono się mnożnikiem $\times 4$. Wynik stanowi liczbę oktetów (bajtów) nagłówka TCP. Nagłówek bez opcji będzie miał wielkość wyrażoną jako „5", czyli 20 bajtów. Cztery bity dopełniające oktet zawierający informację o długości nagłówka są zarezerwowane.

Następnie w nagłówku znajdziemy 9 bitów flag, z których cztery najważniejsze to:

- `SYN` – flaga synchronizacji transmisji, służąca do nawiązywania połączenia,
- `ACK` – flaga potwierdzająca, na przykład odbiór innego pakietu,
- `FIN` – flaga oznaczająca chęć zakończenia transmisji,
- `RST` – flaga resetująca połączenie, wymuszająca ponowne uzgodnienie sekwencji.

Następnie na dwóch bajtach zapisana jest wielkość okna pojedynczej transmisji, za pomocą której odbiorca może przedstawić nadawcy swoje możliwości techniczne pod względem możliwości odbioru pakietów o pewnej wielkości. Pozwala odbiorcy na ograniczenie prędkości napływu danych.

Nagłówek TCP, o ile nie występują „opcje", kończy 16-bitowa suma kontrolna oraz wskaźnik priorytetu danych używany w powiązaniu z flagą `URG`.

3.1.5 REST

Representational State Transfer czyli REST to wzorzec projektowy architektury zorientowanej na usługi (*SOA*) oparty o istniejące rozwiązania WWW i protokół HTTP [20]. Jest odzwierciedleniem tendencji projektowania i programowania obiektowego w architekturze sieciowej, opartej o lekką, bezstanową komunikację klient-serwer. Prosta, bezstanowa architektura umożliwia współbieżnie komunikowanie się z serwerem wielu aplikacjom klienckim, na przykład urządzeniom IoT) Twórcą REST jest Roy Fielding, który opracował tą metodykę projektowania aplikacji sieciowych w ramach swojej pracy doktorskiej w roku 2000 [21]. Model REST skłania tworzenia aplikacji zorientowanych na dane (ang. *data-driven programming*) co dobrze wpisuje się w Internet Rzeczy będący źródłem ogromnych ilości danych.

Informacja jest zapisywana i odczytywana z zasobów (obiektów) sieciowych serwera. Zasoby te są uwidocznione w jednorodny, hierarchiczny sposób i reprezentują dane. Identyfikatory *Uniform Resource Identifier*, URI stanowią uchwyty zasobów oraz pozwalają na wywoływanie ich metod. Dzięki temu możliwe jest skuteczne zaimplementowanie podejścia model-widok-kontroler (MVC) z wyraźnym oddzieleniem *frontendu* i *backendu*. Ta cecha REST ułatwia wdrożenie tej metodyki w środowisku

rozproszonym, z różnorodnymi klientami i urządzeniami będącymi źródłem danych, co jest typowe dla IoT.

REST nie stanowi nowego standardu a jest metodyką tworzenia aplikacji w oparciu o już istniejące techniki. Komunikacja REST jest z założenia zgodna ze standardem HTTP a więc lekka w implementacji, co współgra z ograniczonymi możliwościami obliczeniowymi i komunikacyjnymi typowego urządzenia IoT. Używa tych samych metod zapytań co HTTP:

- GET – pobranie danych z zasobu,
- POST – umieszczenie danych w zasobie,
- DELETE – skasowanie danych z zasobu.

Aby jeszcze bardziej ułatwić pracę z danymi, dobra implementacja REST wspiera wiele opisowych, przejrzystych formatów takich jak *Extensible Markup Language*, XML, *JavaScript Object Notation*, JSON. Główną ich cechą jest elastyczność pozwalająca na łatwe rozszerzanie funkcjonalności systemu bez konieczności modyfikacji lub rezygnowania z rozwiązań zaimplementowanych wcześniej.

3.1.6 MQTT

Protokół *MQ Telemetry Transport* powstał w roku 1999 jako efekt współpracy między przedstawicielami firm IBM – Andy Stanford-Clark oraz Arcom – Arlen Nipper. W roku 2014 został przyjęty jako standard organizacji OASIS a w wersji 3.1.1 jest to standard ISO/IEC [22]. Od wersji 3.1 jest to standard otwarty, ogólnodostępny co przyczyniło się do upowszechnienia jego implementacji i realnych wdrożeń. Nadal jest rozwijany a jego aktualna wersja ma numer 5 i nadal jest standaryzowana przez OASIS [23].

MQTT to lekki protokół typu publikuj/subskrybuj użyteczny dla urządzeń o ograniczonych zasobach, pracujących w problematycznych sieciach. Z drugiej strony jest to rozwiązanie wysoce skalowalne, umożliwiające komunikację milionom urządzeń. Jest to więc standard szczególnie użyteczny i popularny w IoT. Dopasowanie jest tak dobre, że na stronie internetowej organizacji rozwijającej ten protokół jest on przedstawiony jako „*The Standard for IoT Messaging*". Jest skalowalny i skutecznie daje się go używać w sieciach o niskiej przepustowości i małej dostępności.

MQTT ma przydzielone dwa standardowe porty: 1883 oraz 8883, odpowiednio dla transmisji bez szyfrowania oraz wykorzystującej *Transport Layer Security*, TLS. Klienci łączą się z serwisem, który nazywany jest tu *brokerem* a więc jednostką która „wie" jak dystrybuować gromadzone dane. Klienci przekazują dane do „tematów" określonych wielopoziomowymi ścieżkami W przypadku pojawienia się nowej informacji broker automatycznie dystrybuuje ją subskrybentom tematu.

Tematem może być na przykład: `dom/salon/temperatura` czyli ciąg znaków z jednej strony łatwy do interpretacji przez człowieka a z drugiej bliski idei jednolitych lokatorów (URI). Stosowane są znaki specjalne takie jak "+" zastępujący jeden dowolny

fragment tematu (pomiędzy znakami "/") oraz znak "#" zastępujący wiele poziomów tematu. Jest też znak specjalny "$" oznaczający tematy specjalne.

Protokół MQTT jest powszechnie wspierany przez usługi chmurowe wielu dużych dostawców usług sieciowych (Amazon Web Services, Microsoft Azure, Google Cloud) jak też ogromną liczbę mniejszych, niezależnych firm, które specjalizują się w tym standardzie (HiveMQ, EMQX, Eclipse, Mosquitto, CloudMQTT). Wynika to z otwartości standardu – w istocie rzeczy każdy może samodzielnie uruchomić usługę brokera MQTT i nie jest to trudne.

3.1.7 CoAP

Jeśli omówiony wyżej protokół MQTT określimy jako „lekki" to CoAP należałoby opisać jako „ultralekki". Powstał on z inicjatywy popularnego producenta rdzeni mikrokontrolerów – firmy ARM. Firmie tej zależy na tym by jej rozwiązania upowszechniały się a więc z biznesowego punktu widzenia istotne było umożliwienie mikrokontrolerom porozumiewanie się z serwisami RESTowymi [20]. Podobnie jak podstawowe protokoły Internetu tak i ten jest ustandaryzowany przez IETF a autorem jest Zach Shelby [24].

Chociaż CoAP bazuje i jest ukierunkowany na REST, to jest to format binarny. Z tego względu bardziej przypomina protokoły warstwy sieciowej niż warstwy aplikacji, w której faktycznie się znajduje. Dzięki swej prostocie jest możliwy do zaimplementowania nawet na mikrokontrolerach o niewielkich zasobach pamięci RAM i *flash* i mających łącza o bardzo małej przepustowości. Zgodnie z powyższymi założeniami i podobnie jak REST tak i CoAP jest protokołem bezstanowym. Mimo dużych ograniczeń nie zignorowano kwestii bezpieczeństwa transmisji, którą zapewnia w tym przypadku lekki protokół szyfrowania w oparciu o transmisję UDP (*Datagram Transport Layer Security*, DTLS).

3.2 Wykonanie ćwiczenia

Do wykonania tego ćwiczenia konieczne jest posiadanie zainstalowanego pakietu INET [25]. Jego instalacja jest sugerowana podczas instalacji OMNeT++ ale nieopatrznie można ten element pominąć. W takim przypadku należy skorzystać z opcji w menu środowiska Help » Install Simulation Models... i doinstalować INET.

Poprawnie zainstalowany pakiet INET jest widoczny tak jak inne, „zwykłe" projekty, w panelu „Project Explorer". Jeśli dostarczana przez niego funkcjonalność ma być używana w innych projektach, to powinien pozostawać otwarty. Ponadto, w projektach zależnych od INET należy prawidłowo ustawić własności projektu tak, żeby wykorzystywały bibliotekę. By to zrobić należy wybrać z menu Project » Properties by otworzyć okno własności projektu. Następnie należy z listy wybrać Project References i tam zaznaczyć inet4.4. Trzeba też w drzewku kategorii ustawień rozwinąć OMNeT++ i wybrać Makemake. W panelu „Makemake" trzeba wybrać linię z tekstem „src: makemake"

i Options. Otworzy się kolejne okno, w którym w panelu Compile trzeba zaznaczyć obie opcje „Add include(...)” a w panelu Link zaznaczyć „Link with libraries(...)” oraz „Add libraries(...)”.

Pominięcie któregoś z tych elementów skutkuje tym, że potrzebne funkcje nie są dostępne. Wówczas projekt w ogóle się nie skompiluje lub, jeśli się skompiluje i uruchomi, to zgłosi przy tym błąd.

Proces instalacji i wykonanie pierwszego projektu z użyciem pakietu INET są zaprezentowane na filmie towarzyszącym niniejszemu podręcznikowi. Omówienie tego przykładu znajduje się niżej.

3.2.1 Pierwsza sieć w INET

Pierwszy projekt wykorzystujący INET będzie symulował jeden z najpowszechniej znanych protokołów Internetu, czyli *Internet Control Message Protocol*, ICMP, znany potocznie jako „ping”.

Opis sieci w pliku NED wykorzystującej INET rozpoczyna się tak samo, jak inne projekty OMNeT++, czyli z użyciem dyrektywy `package`, tak jak pokazane jest to na listingu 25.

```
1 package pingu.simulations.pingu1;
```

Listing 25. Nagłówek projektu wykorzystującego INET jest typowy.

Następna linijka kodu NED wykorzystuje istotną część pakietu INET. Jest to przywołanie automatycznego konfiguratora sieci IPv4, który zadba o właściwą adresację urządzeń i zapewni routing między urządzeniami. Ten kluczowy fragment kodu pokazany jest na listingu 26.

```
3 import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;
```

Listing 26. Przywołanie automatycznego konfiguratora sieci IPv4.

Dalsze kilkanaście linijek, pokazane na listingu 27 również ściśle opiera się o INET i przygotowuje elementy potrzebne do zbudowania sieci.

```
5 import inet.node.ethernet.Eth1G;
6 import inet.node.ethernet.Eth100M;
7 import inet.node.ethernet.Eth10M;
8 import inet.node.ethernet.EthernetSwitch;
```

Listing 27. Deklaracja chęci użycia typowych kanałów Ethernetu.

W przykładzie wykorzystane są symulacje łączy o przepustowości 1 Gbps, 100 Mbps i 10 Mbps. Oprócz wymienionych w przykładzie, dostępne są także przepustowości 10 Gbps, 40 Gbps, 100 Gbps, 200 Gbps i 400 Gbps. Zauważmy też, że *switch* Ethernetowy

jest traktowany tak, jak inne łącza. W warstwie Ethernetu pakiet OMNET pozwala też symulować urządzenia takie jak `EthernetHost` i `EthernetHub`.

Następne dwie linie, z listingu 28, pozwolą na deklarowanie typowych węzłów sieci.

```
10 import inet.node.inet.StandardHost;
11 import inet.node.inet.Router;
```

Listing 28. Podstawowe węzły sieci przewodowej w bibliotece INET.

W powyższym fragmencie zadeklarowana jest chęć użycia submodułów typu `StandardHost` oraz `Router`, co w połączeniu z wcześniej zadeklarowanym `EthernetSwitch` oraz kilkoma typami łącza ethernetowego już pozwala na zbudowanie dość złożonej sieci. Ponadto możliwe jest zasymulowanie urządzeń takich jak:

- `MulticastRouter` rozszerzający klasę `Router`,
- `WirelessHost` rozszerzający klasę `StandardHost`,
- `AdhocHost` rozszerzający klasę `WirelessHost`,
- `ManetRouter` rozszerzający klasę na `AdhocHost`,
- `SensorNode`.

Bazową klasą jest `ApplicationLayerNodeBase`, na której bazują `StandardHost`, `Router` i `SensorNode`.

Do wizualizacji symulacji sieci z użyciem INET potrzebna jest jeszcze kanwa graficzna. Deklarujemy ją jak na listingu 29.

```
13 import inet.visualizer.canvas.integrated.IntegratedCanvasVisualizer;
```

Listing 29. Deklaracja użycia graficznej kanwy graficznej w INET.

Ten wizualizator potrafi graficznie reprezentować bardzo wiele aspektów działania sieci, takich jak:

- urządzenia w środowisku ich umieszczenia,
- fizyczne przeszkody między urządzeniami komunikującymi się radiowo,
- połączenia a w tym połączenia typu *ad hoc*,
- trasy sieciowe,
- trwającą komunikację i propagację sygnałów,
- przekazywanie pakietów przez różne warstwy sieci,
- ruch urządzeń,
- tabele i statystyki.

Tak przygotowany zestaw metod pozwala na zadeklarowanie sieci w stylu przedstawionym w poprzednim rozdziale. Ponieważ sieć z niniejszego przykładu jest już

nieco większa, więc zostanie omówiona w dwóch fragmentach. Pierwszy z nich został pokazany na listingu 30.

```
network pingu
{
    @display("bgb=1047,371");
    submodules:
        configurator: Ipv4NetworkConfigurator {
            @display("p=91,284");
        }
        visualizer: IntegratedCanvasVisualizer {
            @display("p=30,284");
        }
```

Listing 30. Deklaracja użycia konfiguratora i kanwy dla sieci symulowanej z wykorzystaniem INET.

Jak widzimy w powyższym kodzie, zarówno `configurator` jak i `visualizer` są podmodułami sieci, tak jak funkcjonujące w niej urządzenia. Zostały tu przywołane `Ipv4NetworkConfigurator` oraz `IntegratedCanvasVisualizer`, wcześniej zaimportowane w nagłówku pliku NED.

Dalszy ciąg kodu, pokazanay na listingu 31, zawiera deklarację właściwych urządzeń z symulowanej sieci.

```
        Alice: StandardHost {
            @display("p=130,203");
        }
        Bob: StandardHost {
            @display("p=899,203");
        }
        Carol: Router {
            @display("p=267,62");
        }
        David: Router {
            @display("p=459,203");
        }
        Frank: EthernetSwitch {
            @display("p=636,61");
        }
        Host[5]: StandardHost;
```

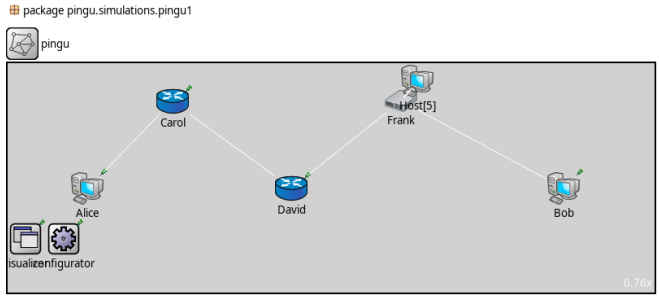
Listing 31. Deklaracja urządzeń dla symulowanej sieci z wykorzystaniem INET.

W scenariuszu tej symulacji urządzenia `Alice` i `Bob` próbują się skomunikować. Pośredniczą w tym dwa routery: `Carol` i `David` oraz jeden switch o nazwie `Frank`. Do switcha, oprócz `Boba`, podłączonych jest także 5 standardowych hostów. Żeby to było możliwe, konieczne jest zapisanie połączeń, tak jak na listingu 32.

```
44      connections allowunconnected:
45          {
46              Alice.ethg++ <--> Eth100M <--> Carol.ethg++;
47              Carol.ethg++ <--> Eth1G <--> David.ethg++;
48              David.ethg++ <--> Eth100M <--> Frank.ethg++;
49              Frank.ethg++ <--> Eth100M <--> Bob.ethg++;
50              Frank.ethg++ <--> Eth100M <--> Host[0].ethg++;
51              Frank.ethg++ <--> Eth100M <--> Host[1].ethg++;
52              Frank.ethg++ <--> Eth10M <--> Host[2].ethg++;
53              Frank.ethg++ <--> Eth10M <--> Host[3].ethg++;
54              Frank.ethg++ <--> Eth10M <--> Host[4].ethg++;
55          }
```

Listing 32. Deklaracja połączeń między węzłami sieci.

Powyższy kod jest bardzo prosty i powinien być zrozumiały bez dodatkowych wyjaśnień. Po przełączeniu do trybu graficznego powinniśmy zobaczyć widok taki, jak na rysunku 18.



Rysunek 18. Diagram pierwszej symulacji z użyciem INET.

Zauważmy, że zadeklarowane wektorem urządzenia `Host` na rysunku są zbite w jednym miejscu. Nie zostały przesunięte w trybie „Design” oraz nie zostały im nadane atrybuty `@display`. Nie jest to problemem dla symulacji, gdyż jej okno będzie wyglądało troszkę inaczej, o czym przekonamy się za chwilę.

Żeby uruchomić symulację trzeba jeszcze przygotować jej konfigurację w pliku inicjalizacyjnym. Tworzymy więc zwyczajowy plik „omnetpp.ini”, który w tym przykładzie będzie się składał z mniej niż dwudziestu linii. Początkowy fragment poleca

konfiguratorowi sieci aby zapisał w logach projektu adresację, topologię, połączenia i trasy sieci a pokazany jest na listingu 33.

```
[General]
pingu.configurator.dumpAddresses = true
pingu.configurator.dumpTopology = true
pingu.configurator.dumpLinks = true
pingu.configurator.dumpRoutes = true
```

Listing 33. Ustawienie wyświetlania komunikatów o konfiguracji sieci.

Informacje te pojawią się w panelu konsoli poniżej widoku symulacji ale ich odnalezienie może być tam trudne ze względu na obecny tam gąszcz różnych wpisów. Dlatego warto pamiętać, że po uruchomieniu symulacji można wejść w ikonę „configurator" za pomocą podwójnego kliknięcia. Tekst opisu struktury sieci pojawi się wtedy w konsoli ponownie.

Gwiazdki na początku pokazanych wyżej linii zastępują nazwę sieci, która dzięki temu może być dowolna a dany element konfiguracji będzie jej dotyczył.

Następne dwie linie są pokazane na listingu 34 a dotyczą konfiguracji protokołu IPv4.

```
*.*.ipv4.arp.typename = "GlobalArp"
*.*.ipv4.routingTable.netmaskRoutes = ""
```

Listing 34. Prosta konfiguracja IPv4 w INET.

Zgodnie z powyższą konfiguracją mapowanie adresów logicznych IPv4 na adresy kontroli dostępu do medium, zwane adresami fizycznymi lub po prostu adresami MAC (*Medium Access Control*, MAC), jest realizowane z użyciem globalnej tablicy ARP (*Address Resolution Protocol*, ARP). Upraszcza to późniejszą komunikację poprzez ograniczenie wysyłanych pakietów ARP i ułatwia skupienie się na warstwie sieciowej. Widzimy też, że nie zadeklarowano żadnych masek sieciowych, co upraszcza adresację urządzeń.

Podobnie jak wyżej, początkowe gwiazdki w liniach konfiguracji odnoszą się do sieci o dowolnych nazwach. Druga z gwiazdek odnosi się do wszystkich urządzeń w tej sieci. Nie trzeba dzieki temu martwić się o potencjalne zmiany nazwy sieci ani liczbę urządzeń w tej sieci i ich nazwy. Jednocześnie daje to możliwość przygotowania w pewnym zakresie indywidualnej konfiguracji dla wybranego urządzenia.

Kolejne dwie linie konfiguracji decydują o wyświetlaniu interfejsów sieciowych urządzeń w oknie symulacji. Pokazane są na listingu 35.

```
*.visualizer.interfaceTableVisualizer.displayInterfaceTables = true
*.visualizer.interfaceTableVisualizer.nodeFilter = "not (*switch*
    or *Switch* or *AP*)"
```

Listing 35. Włączenie wyświetlania interfejsów wybranych urządzeń.

Domyślnie interfejsy nie są wyświetlane i należy je włączyć samodzielnie. Za pomocą pola `nodeFilter` można wybrać, dla których typów urządzeń ta informacja będzie dostępna w kanwie graficznej.

Następny fragment, z listingu 36 odnosi się do funkcjonowania jednego tylko węzła – `Alice`.

```
13 *.Alice.numApps = 1
14 *.Alice.app[0].typename = "PingApp"
15 *.Alice.app[0].startTime = uniform(3s,7s)
16 *.Alice.app[0].printPing = true
17 *.Alice.app[0].destAddr = "Bob(ipv4)"
```

Listing 36. Uruchomienie aplikacji w węźle `Alice`.

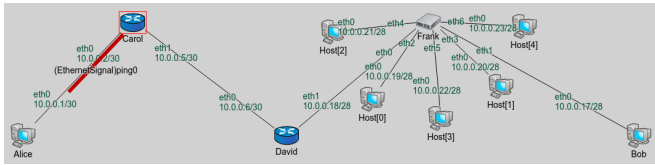
Zgodnie z tą konfiguracją, `Alice` uruchamia jedną aplikację. Będzie to „PingApp”, jedna z wielu aplikacji dostępnych w `inet.applications`. Pierwszy ping zostanie wysłany po losowym czasie między 3 a 7 sekund od momentu rozpoczęcia symulacji. Kolejne „pingi” `Alice` będzie wysyłać po uzyskaniu odpowiedzi na wcześniejsze. Adresatem pingów jest węzeł o nazwie „Bob”, który musi w sieci istnieć aby symulacja mogła się uruchomić. Pingi te są realizowane w oparciu o protokół IPv4.

Ostatnia linijka konfiguracji to deklaracja nazwy sieci, która musi być zgodna z nazwą w pliku NED. Pokazane jest to na listingu 37.

```
19 network = pingu
```

Listing 37. Określenie nazwy sieci, której dotyczy konfiguracja.

Wyżej opisane dwa pliki są kompletną symulacją. Jest ona bardzo prosta, dlatego nie trzeba tworzyć jakiegokolwiek pliku w C++. Po uruchomieniu symulacja prezentuje się jak na rysunku 19.



Rysunek 19. Pierwsza symulacja z użyciem INET po uruchomieniu.

Warto zauważyć, że ikony zapisanych wektorowo węzłów `Host` zostały automatycznie rozłożone wokół switcha, do którego są podłączone.

W panelu symulacji można teraz odkrywać niektóre komponenty sieci za pomocą podwójnego kliknięcia w ich ikonę.

Zadanie

- Rozbuduj sieć symulowaną w tym ćwiczeniu.
- Zmień konfigurację z „GlobalArp” na „Arp” i zaobserwuj wymianę pakietów ARP.

3.2.2 Symulacja protokołu UDP

Protokół UDP jest prostszym z dwóch najważniejszych rozwiązań stosowanych w warstwie transportowej Internetu. Poniże omówiony będzie przykład symulacji, z pomocą której można analizować jego funkcjonowanie.

Jeśli chodzi o plik NED to ma on niemal taką samą strukturę, jak w poprzednim przykładzie, dlatego jego opis tutaj zostanie całkowicie pominięty. Jedyną zmianą jest zmiana nazwy sieci (`network`) z `pingu` na `pingu2` oraz umieszczenie tej symulacji w osobnym katalogu i pliku o innej nazwie `simulations` ▸ `pingu2` ▸ `pingu2.ned`, co powoduje zmianę jak w listingu 38.

```
1 package pingu.simulations.pingu2;
```

Listing 38. Zmieniona nazwa pliku NED musi w nim być odzwierciedlona.

Jak jednak zaraz się przekonamy, przy niezmienionej strukturze sieci można zrealizować kilka różnych symulacji za pomocą jednego pliku konfiguracyjnego.

Linie z ogólną konfiguracją działania symulacji pozostały niezmienione względem poprzedniego przykładu. Dla przypomnienia zostaną tu jeszcze raz podane w całości, chociaż bez omawiania, w listingu 39.

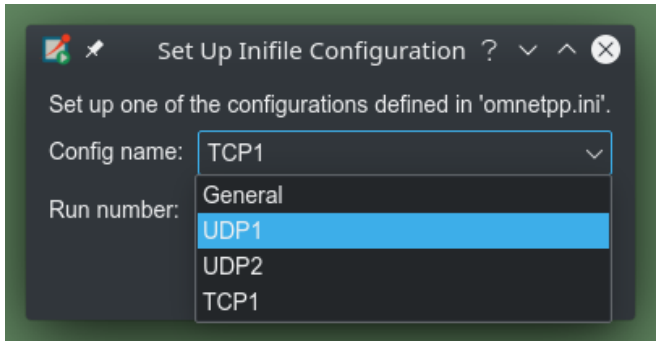
```
1  [General]
2  pingu2.configurator.dumpAddresses = true
3  pingu2.configurator.dumpTopology = true
4  pingu2.configurator.dumpLinks = true
5  pingu2.configurator.dumpRoutes = true
6
7  *.*.ipv4.arp.typename = "GlobalArp"
8  *.*.ipv4.routingTable.netmaskRoutes = ""
9
10 *.visualizer.interfaceTableVisualizer.displayInterfaceTables = true
11 *.visualizer.interfaceTableVisualizer.nodeFilter = "not (*switch*
       or *Switch* or *AP*)"
```

```
12
13 network = pingu2
```

Listing 39. Ogólna konfiguracja symulacji.

Jak wspomniano wyżej, jedyną zmianą w tym fragmencie jest zmiana nazwy sieci, która musi być zgodna z nazwą użytą w pliku NED.

Widzieliśmy już wielokrotnie linie zawierające słowo kluczowe `General` ujęte w nawiasy kwadratowe. Nawiasy te służą do wyszczególnienia sekcji konfiguracji. Sekcja taka w ogólności, oprócz „General" jest *de facto* odrębną symulacją. Dzięki temu w jednym pliku można zamieścić kilka, nieco różniących się scenariuszy symulacji i łatwo je przełączać. Wybór, która wersja ma zostać uruchomiona, jest podejmowany po uruchomieniu projektu. Okno wyboru wygląda jak na rysunku 20.



Rysunek 20. Wybór scenariusza symulacji.

Zatem w pierwszej omawianej tu symulacji, nazwanej `UDP1`, deklarowana jest specyficzna charakterystyka działania węzła `Alice`, co pokazane jest na listingu 40.

```
15 [UDP1]
16 *.Alice.numApps = 1
17 *.Alice.app[0].typename = "UdpSourceApp"
18 *.Alice.app[0].source.packetLength =  int(truncnormal(1024B, 512B))
19 *.Alice.app[0].source.productionInterval = exponential(100ms)
20 *.Alice.app[0].io.destAddress = "Bob(ipv4)"
21 *.Alice.app[0].io.destPort = 1000
```

Listing 40. Konfiguracja urządzenia jako źródła pakietów UDP w symulacji o nazwie `UDP1`.

Na urządzeniu `Alice` uruchamiana jest jedna symulowana aplikacja, którą jest generator pakietów UDP. Wielkość pakietu ustalana jest losowo, zgodnie z rozkładem normalnym o wartości średniej 1024 bajtów, z odchyleniem standardowym 512 bajtów. OMNeT++ oferuje kilka rozkładów, takich jak:

- normalny,

- wykładniczy,
- trójkątny,
- jednorodny,
- Studenta,
- beta,
- gamma,
- Erlanga,
- χ^2,
- Cauchy'ego,
- Weibulla,
- Pareta.

Interwał generowania pakietów w powyższym przykładzie określony jest losowo, zgodnie z rozkładem wykładniczym, przy czym wartość średnia to 100 ms. `Alice` będzie wysyłać pakiety do urządzenia `Bob`, za pomocą protokołu IPv4, na port o numerze 1000.

Można zgadnąć, że w powyższym kodzie zmiana `ipv4` na `ipv6` wystarczy, aby skorzystać z symulacji protokołu IPv6. Mogłoby tak być, jednak domyślnie nowsza wersja protokołu IP jest wyłączona. Można ją włączyć dla wszystkich urządzeń zmieniając `false` na `true` we fragmencie pokazanym na listingu 41.

```
23 *.*.hasIpv6 = false
```

Listing 41. Globalne włączenie obsługi IPv6.

Pozostaje jeszcze skonfigurować odbiór pakietów UDP na urządeniu `Bob`. Niezbędne minimum to trzy linie pokazane na listingu 42.

```
25 *.Bob.numApps = 1
26 *.Bob.app[0].typename = "UdpSinkApp"
27 *.Bob.app[0].io.localPort = 1000
```

Listing 42. Konfiguracja odbioru pakietów UDP w urządzeniu `Bob`.

Podobnie jak `Alice`, tak i `Bob` ma uruchomioną jedną aplikację, którą jest `UdpSinkApp`, nasłuchująca na porcie 1000. Jedyną funkcją tej aplikacji jest odbiór pakietów a ponieważ UDP jest protokołem bezpołączeniowym, to pakiety te będą przez węzeł odbiorczy tylko konsumowane.

Zadanie

- Sprawdź, co się stanie, gdy w węźle `Alice` albo `Bob` zostanie użyty inny port, niż w drugim z urządzeń.

3.2.3 Dwukierunkowa komunikacja UDP

Protokół UDP nie potwierdza odebrania pakietów, przez co często nazywa się go „bezpołączeniowym". Jednak komunikacja dwukierunkowa za jego pomocą jest możliwa, co zobaczymy w poniższym, prostym przykładzie.

Do pliku konfiguracyjnego dodajemy sekcję kolejnej symulacji a w niej ustalamy parametry urządzenia Alice, tak jak pokazane jest to na listingu 43.

```
29 [UDP2]
30 *.Alice.numApps = 1
31 *.Alice.app[0].typename = "UdpBasicBurst"
32 *.Alice.app[0].startTime = exponential(1s)
33 *.Alice.app[0].burstDuration = exponential(15ms)
34 *.Alice.app[0].sendInterval = exponential(5ms)
35 *.Alice.app[0].sleepDuration = exponential(10s)
36 *.Alice.app[0].messageLength = int(truncnormal(1024B, 512B))
37 *.Alice.app[0].destAddresses = "Bob(ipv4)"
38 *.Alice.app[0].destPort = 1000
39 *.Alice.app[0].localPort = 1000
40 *.Alice.app[0].chooseDestAddrMode = "once"
```

Listing 43. Bardziej rozbudowana konfiguracja węzła `Alice`.

Tym razem aplikacją uruchamianą na `Alice` jest `UdpBasicBurst`, co symuluje wysłanie serii pakietów UDP w krótkim czasie. Parametry czasowe tej serii opisane są rozkładami wykładniczymi. Kolejno są to:

- początek pierwszej transmisji względem początku symulacji,
- czas emitowania jednej serii pakietów,
- interwał wysyłania pakietów jednej serii,
- czas uśpienia pomiędzy kolejnymi seriami.

Dalej skonfigurowana jest losowa wielkość pakietu. Linie 37 i 38 są już znane z poprzedniego przykładu i nie wymagają omówienia. Pojawia się tu jednak konfiguracja portu lokalnego, a więc służącego do nasłuchiwania. Ostatnia linia określa jak mają być ustalane adresy, na które kierowane są pakietów. Może to być:

- „once" – tylko raz, na całą symulację,

- „perBurst” – osobno, na każdą serię pakietów,
- „perSend” – osobno, na każdy pakiet.

W przypadkach `perBurst` i `perSend` należy w parametrze `destAddress` podać listę możliwych urządzeń docelowych, oddzielonych spacjami.

Konfiguracja urządzenia Bob jest bardzo prosta a jedyną zmianą względem poprzedniej jest użyta aplikacja, co widać na listingu 44.

```
42 *.Bob.numApps = 1
43 *.Bob.app[0].typename = "UdpEchoApp"
44 *.Bob.app[0].localPort = 1000
```

Listing 44. Konfiguracja nasłuchującego węzła `Bob`.

Jak można łatwo zgadnąć, `UdpEchoApp` odbiera pakiety i odpowiada nadawcy ich zawartością. Mamy zatem prosty model komunikacji dwukierunkowej.

Zadanie

- Sprawdź działanie symulacji z innymi parametrami czasowymi.
- Sprawdź w działaniu opcje `perBurst` i `perSend`, z pomocą obecnych w symulowanej sieci urządzeń `Host[]`.

3.2.4 Symulacja echo TCP

Niniejsza symulacja przypomnia w ogólności poprzednią, gdyż `Alice` nadaje wiadomości do `Boba` a ten odpowiada zawartością pakietu. Jednak tutaj użyjemy protokołu TCP. Węzłem inicjującym komunikację również jest tu `Alice` a konfiguracja tego urządzenia pokazana jest na listingu 45

```
46 [TCP1]
47 *.Alice.numApps = 1
48 *.Alice.app[0].typename = "TcpSessionApp"
49 *.Alice.app[0].tOpen = 0.2s
50 *.Alice.app[0].tSend = 0.4s
51 *.Alice.app[0].tClose = 5s
52 *.Alice.app[0].sendBytes = intuniform(100B, 200B)
53 *.Alice.app[0].connectAddress = "Bob(ipv4)"
54 *.Alice.app[0].connectPort = 5000
55 *.Alice.app[0].localPort = 4000
```

Listing 45. Konfiguracja nawiazywania sesji TCP/IP z urządzenia `Alice`.

Użytą aplikacją jest TcpSessionApp, która nawiązuje połączenie, realizuje transmisję i kończy swe działanie. Zdarzenia związane z tym połączeniem mają charakterystyczne punkty czasowe, takie jak:

- tOpen – chwila rozpoczęcia otwierania połączenia,
- tSend – chwila rozpoczęcia wysyłania danych,
- tClose – chwila zamknięcia połączenia.

Liczba wysyłanych danych określona jest tu rozkładem jednorodnym między 100 B a 200 B. Dane przesyłane są za pomocą IPv4 do urządzenia Bob, na port 5000. Aplikacja TcpSessionApp może też działać jako serwer nasłuchując na jakimś porcie, co też tu widzimy z przykładowym portem 4000.

Na urządzeniu Bob jest uruchomiona aplikacja nasłuchująca TcpEchoApp, która odsyła nadawcy pewne dane, na bazie pakietów otrzymanych. Jej prostą konfigurację widzimy na listingu 46.

```
57 *.Bob.numApps = 1
58 *.Bob.app[0].typename = "TcpEchoApp"
59 *.Bob.app[0].localPort = 5000
60 *.Bob.app[0].echoFactor = uniform(10.0, 20.0)
```

Listing 46. Konfiguracja węzła Bob, nasłuchującego połączeń TCP.

Bob oczekuje na połączenia TCP nadchodzące na port 5000. Po odebraniu pakietu odpowiada nadawcy wysyłając do niego między 10 a 20 razy większą ilością danych, niż otrzymał. Ponieważ pakiety TCP/IP podlegają fragmentacji, powinno dać się zobaczyć odpowiedź w postaci serii pakietów wysyłanych przez Boba do Alice. Ponadto Alice będzie potwierdzać każdy z pakietów za pomocą ACK. Symuluje to prostą komunikację typu pytanie-odpowiedź, gdzie odpowiadający serwuje dane potrzebne stronie inicjującej komunikację. Po zakończeniu sesji TCP/IP wyżej omówiona symulacja kończy się.

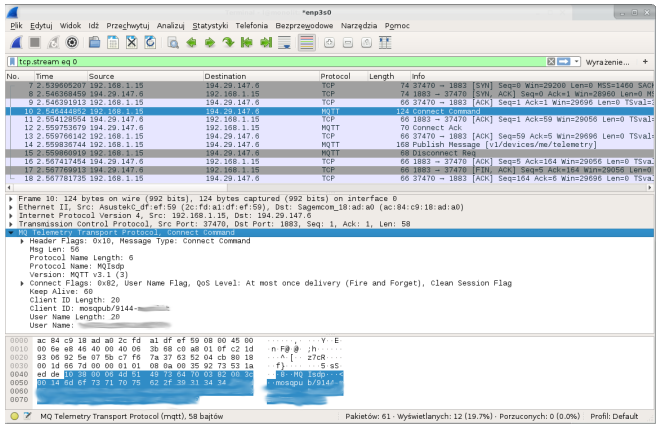
Zadanie

- Spraw, by urządzenia Host[*] również nawiązały połączenia z urządzeniem Bob ale niech ich parametry czasowe połączeń będą dane wybranymi przez Ciebie rozkładami losowymi.
- Sprawdź, co się stanie, gdy Alice spróbuje nawiązać połączenie na innym porcie, niż otwarty na urządzeniu Bob lub z innym urządzeniem w sieci, które na docelowym porcie nie nasłuchuje.

3.2.5 Analiza pakietów

Do tej pory oglądaliśmy wizualizacje podstawowych transmisji internetowych. Wiemy jednak, że złożoność protokołów sieciowych jest znacznie wyższa, niż pokazują to proste metafory graficzne w OMNeT++.

W celu bliższego poznania standardów sieciowych skorzystamy z możliwości zapisania transmisji w pliku typu PCAP. Użytkownikom programów takich jak tcpdump, nmap i Wireshark nie trzeba tego rozwiązania przedstawiać. Pozostałe osoby będą mogły zapoznać się z nim za pośrednictwem ostatniego z wymienionych tu programów, gdyż właśnie z Wireshark skorzystamy w celu analizy zapisanych pakietów [26, 27].



Rysunek 21. Przykładowa analiza pakietów w programie Wireshark.

Wireshark pojawił się już w niniejszym podręczniku, podczas omawiania aplikacji do symulacji sieci, w rozdziale 1.2. Jest to wieloplatformowa aplikacja graficzna pozwalającą na przechwytywanie pakietów za pomocą biblioteki libpcap oraz analizę ich zawartości. Może działać zarówno w czasie rzeczywistym jak też otwierać pliki zawierające zapis wcześniej przechwyconej wymiany pakietów. Zrzut ekranu prezentujący ten program pokazany jest na rysunku 21.

W OMNeT++ do nagrywania symulowanych pakietów służy moduł `PcapRecorder`. Należy dodać jego konfigurację w pliku konfiguracyjnym „omnetpp.ini”. W omawianym tu przykładzie przedstawiona wyżej konfiguracja `[TCP1]` została skopiowana jako konfiguracja `[TCP2]` i to na drugiej z nich będziemy dalej pracować. Włączenie nagrywania dla hosta `Alice` realizowane jest jak na listingu 47.

```
72 *.Alice.numPcapRecorders = 1
73 *.Alice.pcapRecorder[*].pcapFile = "/tmp/Alice.pcap"
```

Listing 47. Włączenie jednego strumienia nagrywania pakietów.

Tutaj występuje tylko jeden rejestrator, nazywany w kodzie `pcapRecorder`, który będzie łapać wszystkie pakiety. Jednak, jak można się łatwo domyśleć, istnieje możliwość skonfigurowania wielu. Wrócimy do tego zagadnienia za chwilę.

Pakiety będą zapisywane w pliku `tmp/Alice.pcap`. Gwiazdka w nawiasach kwadratowych oznacza, że docelowy plik jest przeznaczony dla wszystkich rejestratorów.

Analogicznie skonfigurowany jest rejestrator hosta `Bob`, co przedstawia listing 48.

```
79 *.Bob.numPcapRecorders = 1
80 *.Bob.pcapRecorder[*].pcapFile = "/tmp/Bob.pcap"
```

Listing 48. Włączenie jednego strumienia nagrywania pakietów w hoście Bob.

W zasadzie to wszystko co jest potrzebne, jednak gdyby uruchomić symulację w tej postaci, to po chwili pojawi się komunikat błędu. Nie można zapisać pakietów, dla których nie jest wyznaczona suma kontrolna CRC. OMNeT++ standardowo nie oblicza tej sumy podczas symulacji. W omawianym przypadku należy to jednak włączyć, tak jak pokazano na listingu 49.

```
82 **.crcMode = "computed"
83 **.fcsMode = "computed"
```

Listing 49. Włączenie obliczania sum kontrolnych dla symulowanych pakietów.

Dane o pakietach zapisywane są do plików na zakończenie symulacji. Dopóki symulacja działa dobrze i nie kończy się w sposób nieprawidłowy jest to dobre, bo wydajne podejście. Jednak przy takim ustawieniu, gdy pojawi się błąd krytyczny uniemożliwiający dalsze wykonywanie się kodu symulacji, to nic do plików się nie zapisze. Aby to zmienić należy w pliku inicjalizacyjnym dodać linię konfigurującą rejestrator taką, jak w listingu 50.

```
85 **.pcapRecorder[*].alwaysFlush = true
```

Listing 50. Wymuszenie ciągłego zapisu pakietów do pliku.

Po tej zmianie każdy pakiet jest zapisywany indywidualnie, co w przypadku niewielkich symulacji uruchamiancyh na typowych komputerach nie powinno mieć zauważalnego wpływu na wydajność ich wykonywania.

Wyżej zostało omówione proste przechwytywanie pakietów dla urządzeń o nieskomplikowanym połączeniu sieciowym. Jednak urządzenia brzegowe, pośredniczące w komunikacji mogą mieć tych interfejsów wiele. Mogą mieć też interfejsy różnych typów, na przykład z jednej strony BLE, gdy z drugiej jest przewodowe połączenie *ethernetowe* albo inne połączenie radiowe, jak na przykład WiFi.

Ekstremalnym przypadkiem są routery i przełączniki sieciowe. Zwykle mają od kilku do kilkudziesięciu „portów” o różnej charakterystyce łącza fizycznego, jak na przykład

gigabitowy ethernet za pomocą skrętki miedzianej oraz połączenia światłowodowe. Współcześnie często oferują też komunikację WiFi lub są jej dedykowane.

Podstawową klasą łącza sieciowego w OMNeT++ jest `NetworkInterface`, którą znajdziemy w pakiecie `inet.networklayer.common`. Ta klasa ma liczne rozszerzenia uściślające działanie komunikacji różnego typu:

- EthernetInterface – podstawowy interfejs ethernetowy,
- PppInterface – interfejs komunikacji typu punkt-punkt (ang. *Point-to-Point Protocol*, PPP) znany na przykład z modemowych połączeń wdzwanianych,
- Ieee80211Interface – interfejs WiFi,
- Ieee802154NarrowbandInterface – wąskopasmowa komunikacja w standardzie IEEE 802.15.4 jak na przykład ZigBee,
- Ieee802154UwbIrInterface – komunikacja szerokopasmowych impulsów radiowych zgodna ze standardem IEEE 802.15.4,
- TunInterface – interfejs sieciowy tunelowany (glstun), operujący w warstwie 3 ISO/OSI, czyli na poziomie pakietów IP,
- LoopbackInterface – sieciowy interfejs zwrotny,
- ExtInterface – moduł za pomocą którego można połączyć symulacje z rzeczywistym, fizycznym łączem ethernetowym na jeden z trzech sposobów oferowanych przez moduły dziedziczące:
  - ExtLowerEthernetInterface – fizyczne łącze ethernetowe pozwalające na symulacje sprzętu-w-pętli (ang. hardware-in-the-loop),
  - ExtUpperEthernetInterface – łącze typu TAP, czyli pracujące w warstwie 2 ISO/OSI, warstwie łącza danych, skonfigurowane na komputerze realizującym symulacje,
  - ExtUpperIeee80211Interface – rzeczywiste zewnętrzne łącze TAP zrealizowane dla interfejsu typu WiFi.

- AckingWirelessInterface – abstrakcyjna klasa o bardzo uproszczonym modelu warstw 1 i 2 ISO/OSI, dla przypadków gdy fizyczne własności łącza mogą być zupełnie zignorowane,
- EthernetCutthroughInterface – symuluje interfejsy typu *cut-through* czyli rozpoczynające przekazywanie ramki ethernetowej, zanim zostanie ona odebrana w całości,
- InterfaceService – klasa wspierająca, odpowiedzialna za przekazywanie pakietów przez kolejne etapy przetwarzania, w typowych symulacjach używana automatycznie,
- LayeredEthernetInterface – klasa wspierająca, odpowiedzialna za przetwarzanie pakietów w warstwie ethernetowej, używana automatycznie,
- VirtualInterface – łącze wirtualne,

- WirelessInterface – łącze bezprzewodowe ogólnego typu.

Większość powyższych klas znajdziemy w pakiecie `inet.linklayer`, oprócz:

- `ExtInterface`, która znajduje się w `inet.emulation`,
- `InterfaceService`, która znajduje się w `inet.protocolelement`.

Dla złożonych urządzeń, które jednocześnie transmitują i otrzymują dane z wielu innych albo za pomocą różnorodnych interfejsów, zapisywanie całego ruchu sieciowego a w szczególności do jednego pliku może utrudnić późniejszą analizę komunikacji. Na szczęście w OMNeT++ można włączyć wiele rejestratorów, osobno dla każdego łącza danej maszyny. Przykład takiego rozwiązania zaprezentowany jest na listingu 51.

```
87 *.Carol.numPcapRecorders = 2
88 *.Carol.pcapRecorder[0].moduleNamePatterns = "eth[0]"
89 *.Carol.pcapRecorder[0].pcapFile = "/tmp/Carol-eth0.pcap"
90 *.Carol.pcapRecorder[1].moduleNamePatterns = "eth[1]"
91 *.Carol.pcapRecorder[1].pcapFile = "/tmp/Carol-eth1.pcap"
```

Listing 51. Wiele rejestratorów dla jednego urządzenia sieciowego.

Z powyższego przykładu widzimy, że kolejne rejestratory tworzą tablicę, indeksowaną od 0. Dla porządku indeksy rejestratorów są zgodne z indeksami interfejsów ethernetowych.

Zadanie

- Rozbuduj symulowaną sieć o własne urządzenia.
- Wprowadź więcej komunikacji między urządzeniami, zarówno prawidłowej jak i nieprawidłowej, z użyciem przetestowanych wcześniej protokołów (ICMP, TCP, UDP, ARP).
- Włącz przechwytywanie pakietów na urządzeniu o wielu interfejsach, z podziałem rejestracji na odrębne pliki.
- Przeanalizuj zebrane zapisy za pomocą programu Wireshark.

3.3 Pytania sprawdzające

1. Jakie są podobieństwa i różnice między IPv4 i IPv6?
2. Jakie są główne cechy i zastosowania protokołu UDP?
3. Jakie są główne cechy protokołu MQTT?
4. Jakie są obszary zastosowań protokołów CoAP oraz REST?

5. Jakie moduły i klasy INET są potrzebne do symulacji prostej sieci przewodowej?
6. Jak wykorzystać rejestrację pakietów PCAP do ich analizy?

Rozdział 4

Sieci MANET

Cel ćwiczenia

Celem niniejszego ćwiczenia jest zapoznanie się z działaniem mobilnych sieci bezprzewodowych ad-hoc (*Mobile Ad-hoc Network*, MANET). Część teoretyczna krótko przedstawia jeden z dynamicznych i rozproszonych algorytmów odnajdywania trasy routingu w sieci bezprzewodowej. W części praktycznej wykonywane są symulacje sieci MANET z uwzględnieniem mobilności węzłów i przeszkód terenowych.

Sieć ad-hoc jest siecią lokalną (*Local Area Network*, LAN), której urządzenia (węzły) tworzą strukturę sieciową spontanicznie. Nie mają one z góry założonych relacji ani predefiniowanych tras transmisji pakietów. W trakcie działania optymalizują połączenia między sobą tak, by sieć działała jak najefektywniej pod względem wybranego kryterium, którym może być na przykład:

- jak najwyższa średnia przepustowość sieci,
- jak najmniejsze opóźnienia w przekazywaniu danych,
- jak najmniejsza częstość błędnie zrealizowanych transmisji,
- jak najniższe zużycie energii,
- jak najdłuższe funkcjonowanie sieci węzłów zasilanych bateryjnie.

Optymalizacja takiej dynamicznej struktury jest dość złożona ze względu na bezprzewodowy charakter sieci i dlatego nie skalują się one dobrze, i są ograniczone do sieci LAN.

Sieci ad-hoc mogą być tworzone przez urządzenia o identycznych parametrach, gdzie nie ma wyszczególnionych punktów dostępowych, routerów i innego sprzętu sieciowego. Każde z urządzeń mające interfejs sieciowy i odpowiednie oprogramowanie może, w razie potrzeby, stać się routerem dla innych urządzeń w takiej sieci. Dzięki

komunikacji „każdy z każdym” (*peer-to-peer*, p2p) można ich działanie zorganizować w taki sposób, że nie potrzebują scentralizowanych usług sieciowych a przez to mogą być znacznie tańsze w uruchomieniu. Dzięki zastosowaniu standardów internetowych mogą współpracować z typowym sprzętem Sieci ad-hoc są też istotną alternatywą w sytuacji problemów dostępności do sieci stacjonarnych. sieciowym i urządzeniami stacjonarnymi.

Mimo swych zalet nie są stosowane zamiast sieci stacjonarnych co do zasady, gdyż ze względu na swój tymczasowy charakter i dynamiczne nawiązywanie połączeń są znacznie bardziej narażone na atak.

Sieci ad-hoc, których węzły mogą poruszać się w trakcie działania tejże sieci, są nazywane MANET [28]. Szczególnym ich przypadkiem są sieci pojazdów samochodowych oraz sieci tak zwanych dronów czyli bezzałogowych statków powietrznych (*Unmanned Aerial Vehicle*, UAV).

4.1 Rozproszone ustalanie tras

W celu ustalenia struktury połączeń sieciowych węzły sieci ad-hoc muszą posłużyć się rozproszonym algorytmem. Istniało wiele takich algorytmów a dziś praktyczne znaczenie ma *Ad Hoc On-Demand Distance Vector*, AODV, który został opracowany w roku 2003, w wyniku współpracy firmy Nokia z uniwersytetami w Cincinnati oraz Kalifornijskiego [29]. AODV znaczy „wektor odległości [tworzony] na żądanie *ad hoc*”. Jest to algorytm używany między innymi przez ZigBee (IEEE 802.15.4).

AODV nie wymaga apriorycznej informacji do rozpoczęcia działania. Wpisy w bazie tras tworzone są na życzenie węzłów źródłowych za pomocą cyklu zapytań i odpowiedzi. Stąd właśnie określenie „*on demand*” w nazwie algorytmu. AODV jest więc algorytmem typu reaktywnego.

Węzeł źródłowy próbując odkryć trasę do węzła docelowego rozsyła w tym celu broadcastowe zapytanie `RREQ`. Transmisja tego typu jest prosta w przypadku sieci radiowych z dookólną transmisją. Węzeł odbierający taką wiadomość może nie dysponować żądaną trasą i wówczas samemu rozsyła zapytanie `RREQ` oraz zapamiętuje, od jakiego węzła pochodziło zapytanie, które samemu otrzymał. W końcu trafi ono do węzła, który zna aktualną trasę do węzła docelowego. Możliwe, że sam odbiorca jest węzłem docelowym. Odpowiada on wówczas pakietem `RREP`, który jest unikastowo propagowany przez węzły wcześniej wysyłające zapytania aż trafi do węzła, z którego pochodziło oryginalne zapytanie.

W opisany wyżej sposób powstaje rozproszona informacja o trasie łączącej punkt źródłowy z końcowym. Jednak w tablicy routingu węzła nie jest zapisywana cała trasa a jedynie jej krótki fragment opisany czterema polami:

- adres IPv4 węzła docelowego,
- adres IPv4 następnego węzła na trasie do węzła docelowego,

- liczba skoków (ang. *hop count*) do węzła docelowego,
- numer sekwencyjny danego połączenia.

Liczba skoków pozwala określić długość danej ścieżki i wybierać krótsze trasy. Numer sekwencyjny jest zwiększany przy każdym użyciu danego połączenia a dzięki temu aktualna, używana trasa jest łatwa do odróżnienia od trasy nieaktywnej. AODV reaguje na dynamiczne zmiany zachodzące w topologii sieci i pozwala sprostać wyzwaniom stawianym przed MANET.

Jednak im większa sieć, tym bardziej długotrwały jest proces odnajdywania trasy i budowania tablic routingu. Ponadto przechowywanie tras jest ograniczone wielkością przeznaczonej na tę strukturę ilości pamięci, która jest cennym bo istotnie ograniczonym zasobem w urządzeniach wbudowanych i radiowych modułach komunikacyjnych. Im większa sieć tym większa też dynamika zmian a zatem większa liczba pakietów służących odnajdywaniu tras. Z tych powodów AODV źle się skaluje.

4.2 Wykonanie ćwiczenia

Ćwiczenie zaczynamy od stworzenia nowego projektu w OMNeT++. Pamiętamy aby zmienić właściwości projektu i dodać odniesienie do pakietu INET. W tym celu zaznaczamy nowo utworzony projekt w drzewku projektów Project Explorer a następnie wybieramy z górnego menu: Project > Properties > Project References. Zamiast klikania w górnym menu można posłużyć się menu kontekstowym projektu, dostępnym pod prawym przyciskiem myszy lub użyć skrótu klawiszowego Alt+Enter. Upewniamy się, że pakiet INET jest zaznaczony.

4.2.1 Prosta sieć bezprzewodowa

Pierwsza symulacja transmisji bezprzewodowej niewiele różni się od transmisji przewodowych. Zaczniemy jej analizę od pliku z rozszerzeniem NED, czyli opisującego urządzenia w sieci. Zgodnie z pierwszą jego linią nosi on nazwę `WLAN1.ned`:

```
1 package wireless.simulations.WLAN1;
```

Listing 52. Deklaracja pakietu dostarczanego plikiem NED.

Następnie importowane są elementy pakietu INET, które będą używane w opisanych niżej symulacjach:

```
3 import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;
4 import inet.node.contract.INetworkNode;
5 import
    inet.physicallayer.wireless.common.contract.packetlevel.IRadioMedium;
6 import inet.visualizer.contract.IIntegratedVisualizer;
```

Listing 53. Importowanie niezbędnych pakietów biblioteki INET.

Konfigurator `Ipv4NetworkConfigurator` jest nam już znany z poprzednich ćwiczeń więc jego omówienie tutaj jest zbędne.

Moduł `INetworkNode` zawiera wspólny interfejs dla wszystkich typów urządzeń sieciowych. Ich późniejsza deklaracja wygląda nieco inaczej niż dotychczas to widzieliśmy.

Moduł `IRadioMedium` organizuje dostęp do współdzielonego medium fizycznego, w ramach którego odbywa się komunikacja. Śledzi działanie urządzeń radiowych i realizowanych przez nie transmisji. Obsługuje źródła zakłóceń, szumu oraz szumu tła. Moduł oblicza gdzie, kiedy i jak transmisja sygnałów, zakłócenia, i szumy docierają do poszczególnych odbiorników.

Moduł `IIntegratedVisualizer`, jak łatwo zgadnąć, jest odpowiedzialny za wizualizację symulacji. Sam w sobie pakiet ten nie zajmuje się wizualizacjami jednak łączy dostępne moduły wizualizacji, które następnie trzeba skonfigurować w pliku `ini`. W kolejnych przykładach zobaczymy użycie wizualizatorów między innymi takich jak:

- physicalLinkVisualizer – warsty fizycznej,
- dataLinkVisualizer – warstwy łącza danych,
- networkRouteVisualizer – warstwy sieciowej,
- mediumVisualizer – warstwy „medium”,
- mobilityVisualizer – mobilności urządzeń.

Podstawowym urządzeniem w symulowanej sieci będzie węzeł o nazwie Alice, która zajmie się produkcją pakietów. Przygotowana zostanie dla niej odrębna konfiguracja w postaci sieci o tej samej nazwie – Alice, stanowiąca bazę dla kolejnych sieci. Początek tej konfiguracji pokazany jest na listingu 54

```
9  network Alice
10 {
11     parameters:
12         @display("bgb=400,300");
13
14         @figure[rcvdBobUDP](type=indicatorText; pos=35,15;
       anchor=w; font=,10; textFormat="Pakietów UDP odebranych przez
       Bob: %g."; initialValue=0);
15
       @statistic[packetReceived](source=Bob.app[0].packetReceived;
       record=figure(count); targetFigure=rcvdBobUDP);
```

```
16          @figure[txAliceUDP](type=indicatorText; pos=35,35;
    anchor=w; font=,10; textFormat="Pakietów UDP nadanych przez
    Alice: %g."; initialValue=0);
17          @statistic[packetSent](source=Alice.app[0].packetSent;
    record=figure(count); targetFigure=txAliceUDP);
```

Listing 54. Początek konfiguracji podstawowej sieci – Alice.

Powyższy fragment dotyczy głównie aspektów wizualnych symulacji. Obszar tła stanowiącego umowne granice symulacji to prostokąt o rozmiarze 400 na 300 metrów. Następnie skonfigurowane jest wyświetlanie w oknie wizualizacji podstawowej statystyki pakietów wysłanych przez Alice oraz odebranych przez węzeł Bob. Statystyka dla węzła odbierającego jest wyświetlana nad statystyką pakietów wysłanych aby łatwiej, zgodnie z intuicją odczytywać wartość tego ułamka. Węzeł Bob zostanie jednak dodany i skonfigurowany później.

Kolejny fragment to konfiguracja wizualizacji, łączności radiowej, warstwy sieciowej oraz jednego urządzenia – Alice. Pokazane jest to na listingu 55.

```
18      submodules:
19          visualizer: <default("IntegratedCanvasVisualizer")> like
    IIntegratedVisualizer {
20              @display("p=50,250");
21          }
22          radioMedium: <default("UnitDiskRadioMedium")> like
    IRadioMedium {
23              @display("p=100,125");
24          }
25          configurator: Ipv4NetworkConfigurator {
26              @display("p=100,250");
27          }
28          Alice: <default("WirelessHost")> like INetworkNode {
29              @display("p=33,155;i=device/laptop");
30          }
```

Listing 55. Konfiguracja modułów bazowej sieci.

Konfiguracja wizualizacji opiera się o zintegrowaną kanwę graficzną, na której różne elementy graficzne w postaci na przykład strzałek, linii, okręgów, ikon mogą być rysowane przez inne, wyspecjalizowane metody. W przypadku medium radiowego zakładamy, że urządzenia korzystają z anten dookólnych o charakterystyce idealnego koła. W kolejnych liniach widzimy znaną nam już, automatyczną konfigurację sieci IPv4.

W ostatnim fragmencie powyższego listingu pojawia się konfiguracja węzła Alice. Po raz pierwszy widzimy konfigurację hosta w nowym stylu, na bazie `INetworkNode`. Jednak Alice to dość proste urządzenie bezprzewodowe. `WirelessHost` jest to standardowy host IPv4 z dodanym wsparciem dla standardu IEEE 802.11 dostarczonego pod postacią modułu `Ieee80211Interface`.

Sieć złożona z jednego urządzenia oczywiście nie jest naszym celem. Dlatego rozszerzamy bazową konfigurację opartą o Alice, tak jak zostało to pokazane na poniższym listingu.

```
33 network WLAN1 extends Alice
34 {
35     submodules:
36         Bob: <default("WirelessHost")> like INetworkNode {
37             @display("p=133,211;i=device/laptop");
38         }
39 }
```

Listing 56. Rozszerzenie konfiguracji Alice o konfigurację węzła Bob.

Warto w tym momencie zwrócić uwagę na współrzędne obu węzłów sieci i obliczyć dystans między nimi.

Sieć nazwana `WLAN1` rozszerza konfigurację sieci Alice. Wszystkie ustawienia przygotowane w ramach sieci Alice są nadal aktualne i aktywne. Można też niektóre z wcześniejszych ustawień zmienić poprzez przypisanie nowych wartości wybranym atrybutom. Jest to alternatywna metoda względem używanego dotychczas budowania opisu sieci od początku. Ma ona zarówno zalety jak i wady.

Jej główną zaletą jest to, że na bazie wcześniejszych eksperymentów można budować kolejne, tyle że w uproszczony sposób. Taka konfiguracja rozszerzająca może być bardzo krótka i pozwala skupić się na nowo wprowadzanych aspektach symulacji.

Jej główną wadą jest swoiste „zamrożenie” sieci bazowych, gdyż jakakolwiek w nich zmiana wpływa na sieci zbudowane na ich podstawie. Po drugie konfiguracja staje się rozczłonkowana na wiele segmentów, często o rozbieżnych ustawieniach, co zmusza do przeanalizowania, jakie ustawienia są obowiązujące dla każdej z symulacji.

Będziemy korzystać z tego podejścia lecz liczba segmentów konfiguracji bazowych pozostanie ograniczona.

W pliku `WLAN1.ned` są jeszcze konfiguracje innych sieci, które zostaną omówione dalej. Teraz przejdziemy do konfiguracji funkcjonowania tej sieci, złożonej z urządzeń Alice i Bob. Jak poprzednio, konfiguracja ta jest zapisana w pliku `omnetpp.ini`.

Analogicznie do pliku NED, gdzie pewna bazowa konfiguracja sieci jest rozszerzana tak i plik inicjalizujący symulację ma podobną strukturę. Najpierw przeanalizujemy fragment po fragmencie ustawienia ogólne, których początek jest pokazany na listingu 57.

```
[General]
sim-time-limit = 60s

*.configurator.dumpAddresses = true
*.configurator.dumpLinks = true
*.configurator.dumpTopology = true
*.configurator.dumpRoutes = true

*.*.ipv4.arp.typename = "Arp"
```

Listing 57. Początek ogólnej konfiguracji symulacji.

Sekcja ogólna zwyczajowo nazwana tu `General` nie jest symulacją samą w sobie i nie należy jej uruchamiać. Powinna zawierać te podstawowe parametry konfiguracji, które raczej nie ulegną zmianom pomiędzy kolejnymi eksperymentami.

Druga linijka pliku ogranicza czas trwania symulacji do 60 sekund czasu symulowanego. Oczywiście w czasie rzeczywistym symulacja ta może wykonać się znacznie dłużej, na przykład jeśli będziemy korzystać z wielu wizualizacji i powoli analizować zdarzenia, lub też znacznie szybciej, gdy uruchomimy ją w trybie ekspresowym.

Następne cztery linijki włączają wyświetlenie adresów urządzeń, łącz sieciowych, topologii sieci i tras między urządzeniami na początku symulacji. Odpowiednie komunikaty informacyjne pojawiają się w oknie symulacji poniżej kanwy graficznej. Warto wcześniej pamiętać, że liczba wyświetlanych tam komunikatów jest bardzo duża i znalezienie poszukiwanej informacji może chwilę potrwać. Jednak wspomniane wyżej parametry konfiguracyjne są prezentowane na początku.

Ostatnia linia to znany z wcześniejszego ćwiczenia przełącznik używania lokalnej lub globalnej tablicy ARP. Różnica jest taka, że w przypadku lokalnej tablicy ARP urządzenia będą najpierw wymieniać się komunikatami protokołu ARP a dopiero później będą mogły zrealizować właściwą transmisję danych.

Konfiguracja aplikacji wysyłającej pakiety UDP przez węzeł Alice jest pokazana na listingu 58.

```
*.Alice.numApps = 1
*.Alice.app[0].typename = "UdpBasicBurst"
*.Alice.app[0].startTime = 0s
*.Alice.app[0].sendInterval = uniform(23ms, 27ms)
*.Alice.app[0].burstDuration = uniform(240ms, 260ms)
*.Alice.app[0].sleepDuration = uniform(740ms, 760ms)
*.Alice.app[0].messageLength = int(uniform(576B, 1520B))
*.Alice.app[0].destAddresses = "Bob"
*.Alice.app[0].destPort = 1000
```

```
20 *.Alice.app[0].localPort = 21000
21 *.Alice.app[0].chooseDestAddrMode = "once"
```

Listing 58. Konfiguracja aplikacji wysyłającej pakiety UDP na węźle Alice.

Powyższy fragment nie wymaga szczegółowego komentarza, gdyż używaliśmy już tego rozwiązania wcześniej. Węzłem docelowym dla Alice jest Bob, którego konfigurację widzimy poniżej:

```
23 *.Bob.numApps = 1
24 *.Bob.app[0].typename = "UdpSink"
25 *.Bob.app[0].localPort = 1000
```

Listing 59. Konfiguracja aplikacji odbierającej pakiety UDP w węźle Bob.

Również i ten węzeł ma bardzo prostą strukturę, znaną z wcześniejszych przykładów. Jedyną jego rolą jest odbieranie pakietów UDP na porcie 1000.

Dalszy fragment konfiguracji opisuje parametry sieci bezprzewodowej. Widzimy go na listingu 60.

```
27 *.*.wlan[*].typename = "AckingWirelessInterface"
28 *.*.wlan[*].radio.transmitter.communicationRange = 125m
29 *.*.wlan[*].radio.receiver.ignoreInterference = true
```

Listing 60. Konfiguracja interfejsu radiowego.

W pierwszej linii wybrany jest typ interfejsu bezprzewodowego, ustalony tu na `AckingWirelessInterface`. Jest to bardzo prosty, abstrakcyjny interfejs sieciowy stosowny dla symulacji, w których poziomy 1 i 2 modelu ISO/OSI nie są istotne. Zakłada on dookólną transmisję i wymaga użycia `UnitDiskRadioMedium` w konfiguracji sieci, co widzieliśmy podczas analizy pliku NED.

Głównym parametrem tego interfejsu jest zasięg transmisji określony w omawianym przykładzie na 125 metrów. Kolejna linijka konfiguracji również dotyczy warstwy radiowej a więc fizycznej i decyduje, czy symulacja ma ignorować interferencje między urządzeniami. Przez interferencje rozumiemy tu transmisje docierające na odległość większą niż zasięg transmisji a więc nieskuteczne ale jednak wprowadzające zauważalne zakłócenia. Ponieważ pierwszy przykład ma być prosty, to w tej chwili tego aspektu nie bierzemy pod uwagę.

Kolejne linie ustalają konfigurację warstwy MAC.

```
30 *.*.wlan[*].mac.useAck = false
31 *.*.wlan[*].mac.fullDuplex = false
32 *.*.wlan[*].mac.headerLength = 32B
33
```

```
34 *.*.**.bitrate = 54Mbps
```

Listing 61. Konfiguracja warstwy MAC.

Ponieważ mamy tu tylko dwa urządzenia to korzystanie z potwierdzeń (`useAck`) jest wyłączone. Komunikacja, jak w większości interfejsów radiowych jest półdupleksowa, czyli jednokierunkowa w danym momencie, z możliwością zmiany kierunku transmisji. Następnie widzimy określenie długości nagłówka warstwy MAC. W ostatniej linii ustalana jest przepustowość komunikacji, dla której wybraną tutaj wartością jest 54 Mbps, charakterystyczne dla sieci WiFi zgodnych z IEEE 802.11g.

Pozostały fragment konfiguracji podstawowej dotyczy wizualizacji symulacji. Pokazany jest na listingu 62

```
36 *.visualizer.physicalLinkVisualizer.displayLinks = true
37 *.visualizer.dataLinkVisualizer.displayLinks = true
38 *.visualizer.networkRouteVisualizer.displayRoutes = true
39
40 *.visualizer.mediumVisualizer.displaySignals = false
41
42 *.Alice.wlan[*].radio.displayCommunicationRange = true
43 *.Bob.wlan[*].radio.displayCommunicationRange = true
```

Listing 62. Konfiguracja wizualizacji.

Widzimy w powyższym fragmencie, że połączenia na poziomie warstwy fizycznej, łącza danych i sieciowej mają być wizualizowane. Propagacja sygnałów w ramach medium radiowego została wyłączona, gdyż z powodu błędu w OMNeT++ wersja 6.0 może ona powodować blokowanie się symulacji. Jednak jej brak w niczym nie przeszkadza, gdyż jest to tylko dodatkowy element graficzny na ekranie, i tak prezentowany za pomocą czytelnych strzałek. Dla testu można tę dodatkową wizualizację transmisji włączyć, zobaczyć co prezentuje i jeśli okaże się funkcjonować prawidłowo i być przydatną – pozostawić włączoną. W wersji 6.0.1 błąd ten już nie powinien wystąpić.

Wizualizacja zasięgu skutecznej komunikacji z węzła Alice i z węzła Bob jest włączona. Na kanwie graficzne będzie reprezentowana przez granatowe okręgi o promieniach równych opisanemu wyżej zasięgowi transmisji radiowej i środkach w punkcie będącym położeniem urządzeń.

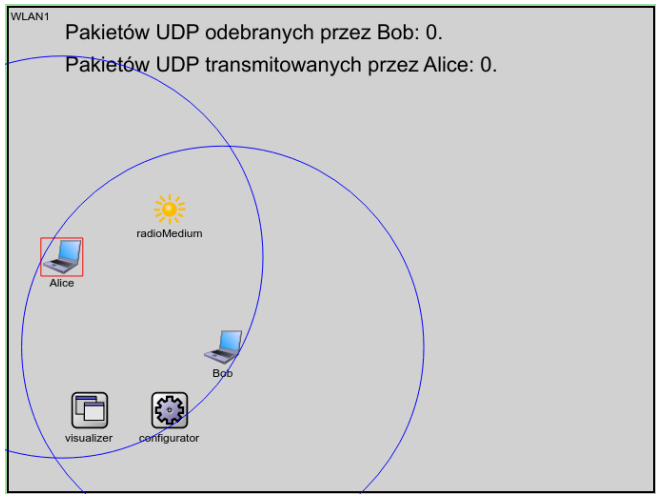
Powyższy fragment to niemal kompletna konfiguracja pierwszej sieci. Jednak dla porządku i spójności z dalszymi symulacjami jest jedynie bazą, rozszerzoną przez właściwą konfigurację łączności między dwoma urządzeniami bezprzewodowymi, pokazaną na listingu 63

```
46 [Config TwoWayRadio]
47 description = Prosta komunikacja bezprzewodowa
```

```
48 extends = General
49 network = WLAN1
```

Listing 63. Konfiguracja prostej łączności bezprzewodowej między dwoma urządzeniami.

Po uruchomieniu tej symulacji i wybraniu scenariusza „TwoWayRadio”, powinniśmy zobaczyć widok, jak na rysunku 22.



Rysunek 22. Symulacja dwóch urządzeń gotowa do uruchomienia.

Z rysunku widzimy, że urządzenia są w swoim zasięgu. Możemy uruchomić symulację w trybie normalnym, z pełnymi animacjami (F5) i przeanalizować szczegółowo wymianę pakietów. Uruchomienie bez animacji, z okresowym aktualizowaniem kanwy graficznej (F6) daje inny ale również wartościowy wgląd w komunikację. Jednak symulacja ta jest bardzo prosta i jej realizacja w trybie przyspieszonym kończy się niemal natychmiast. Po jej zakończeniu warto spojrzeć na komunikaty a w szczególności podsumowanie prezentowane w kafelku znajdującym się poniżej kanwy graficznej.

Warto zauważyć i zapamiętać fakt, że pokazana w kanwie symulacji liczba pakietów wysłanych i odebranych aktualizuje się jeszcze raz po naciśnięciu przycisku „OK”, na koniec symulacji. Dopiero ta wartość obejmuje całą symulację i jest ostateczna.

Zadanie

- Wyłącz limit czasowy symulacji i poobserwuj ją przez pewien czas w trybie „Fast Run” – bez animacji pojedynczych pakietów.
- Zmodyfikuj konfigurację tak, żeby węzeł Bob odpowiadał nadawcy za pomocą odebranych pakietów UDP.
- Zmodyfikuj konfigurację tak, żeby urządzenia znalazły się tuż poza swoimi zasięgami.

4.2.2 Routing bezprzewodowy

Z powyższego fragmentu ćwiczenia poznaliśmy podstawowe aspekty komunikacji bezpośredniej między dwoma urządzeniami. W praktyce często występuje koordynator, router pośredniczący i to podejście zaobserwujemy w niniejszej części ćwiczenia.

Początek konfiguracji tego eksperymentu pokazany jest na listingu 64.

```
51 [Config Coordinator]
52 description = Router bezprzewodowy
53 extends = General
54 network = WLAN2
55
56 *.Carol.forwarding = true
```

Listing 64. Początek konfiguracji z routerem pośredniczącym.

Widzimy z powyższego fragmentu, że ta symulacja, podobnie jak poprzednia, również jest zbudowana na bazie konfiguracji „General”. Jednak odwołuje się ona do sieci `WLAN2`, której za chwilę się przyjrzymy. Ostatnia linia jest kluczowa dla niniejszej symulacji, gdyż deklaruje ona, iż węzeł o nazwie Carol będzie routować pakiety. Węzeł nosi nazwę zaczynającą się na literę „C” od „Connectivity”. Zobaczmy zatem co znajduje się w pliku NED opisującym sieć `WLAN2` – listing 65.

```
41 network WLAN2 extends Alice
42 {
43     submodules:
44         Bob: <default("WirelessHost")> like INetworkNode {
45             @display("p=233,177;i=device/laptop");
46         }
47         Carol: <default("WirelessHost")> like INetworkNode {
48             @display("p=133,211;i=device/accesspoint");
49         }
```

```
}
```

Listing 65. Topologia sieci z routerem.

Sieć WLAN2 rozszerza podstawową sieć Alice, złożoną z jednego urządzenia – Alice, o dwa kolejne węzły: Bob i Carol. Bob jest takim samym urządzeniem jak w poprzedniej symulacji ale znajduje się w innym miejscu. Jest na tyle oddalony od Alice, że nie mogą prowadzić bezpośredniej komunikacji. Na szczęście między nimi, w obszarze wspólnym znajduje się Carol, z możliwością przekazywania pakietów. Wróćmy do analizy pliku inicjalizującego sieć a konkretnie fragmentu z listingu 66.

Pierwsza linia dostarcza konfiguratorowi sieci wytyczne odnośnie używanej w niej adresacji i odwołuje się do zewnętrznego pliku „netconf.xml". Plik ten zaraz przeanalizujemy ale dla uzupełnienia należy stwierdzić, że dwa pozostałe parametry włączają prezentowanie adresów urządzeń i tras sieciowych w wizualizacji symulacji.

Plik „netconf.xml" jest oczywiście w standardzie XML. Jego zawartość jest krótka więc poniżej jest pokazany w całości:

```
<config>
 <interface hosts='Carol' address='192.168.1.1'
    netmask='255.255.255.0'/>
 <interface hosts='**' address='192.168.1.x'
    netmask='255.255.255.0'/>
 <autoroute metric='errorRate'/>
</config>
```

Listing 67. Topologia sieci z routerem zapisana w pliku XML.

Router, czyli węzeł Carol ma ręcznie przypisany typowy adres 192.168.1.1. Pozostałe hosty zostaną ponumerowane automatycznie w ramach sieci 192.168.1.1/24. Wraz z opisaną wyżej konfiguracją w pliku inicjalizacyjnym da to prostą strukturę łańcucha: Alice $\longleftrightarrow$ Carol $\longleftrightarrow$ Bob. Atrybut „metric" pola „autoroute" określa sposób podejmowania decyzji podczas wyznaczania trasy. Dostępne są cztery opcje:

- `hopCount` – preferowanie trasy o najmniejszej liczbie urządzeń pośredniczących, jest to domyślna miara autoroutera,
- `dataRate` – preferowanie połączeń o wyższej przepustowości,

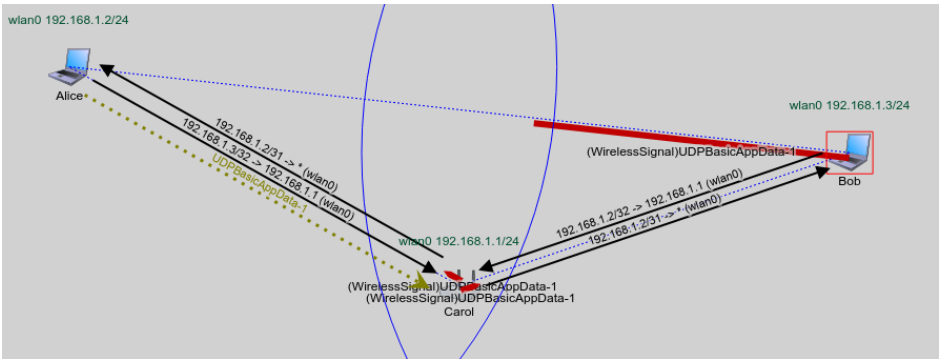
```
*.configurator.config = xmldoc("netconf.xml")

*.visualizer.interfaceTableVisualizer.displayInterfaceTables = true
*.visualizer.routingTableVisualizer.displayRoutingTables = true
```

Listing 66. Konfiguracja automatycznego konfiguratora sieci IPv4.

- `delay` – wybieranie trasy dającej najmniejsze opóźnienie,
- `errorRate` – wybieranie trasy o najmniejszej liczbie błędów podczas transmisji.

W tak prostej sieci wybór miary jakości trasy jest praktycznie bez znaczenia. Miara `errorRate` została tu wskazana, gdyż w dynamicznych sieciach bezprzewodowych liczba nieprawidłowych łączności może być duża.

To wszystko co potrzebne, do realizacji sieci z routerem. Omówiona symulacja, uruchomiona bez zmian da widok taki, jak na rysunku 23



Rysunek 23. Alice i Bob komunikujący się za pośrednictwem routera Carol.

Zadanie

- Zmieniaj adresy urządzeń i obserwuj czy, i jak wpływa to na przekazywanie pakietów, i transmisje.
- Odsuń węzeł Bob od pozostałych węzłów sieci i dodaj kolejny router tak, żeby nadal realizowała się transmisja z Alice do Bob.

4.2.3 Interferencje i zakłócenia

W sieciach bezprzewodowych, ze względu na ich obszarowy charakter znacznie łatwiej o wzajemne zakłócenia między urządzeniami i sieciami. Może dochodzić również do zakłócania transmisji z urządzenia jego własnym sygnałem w wyniku odbić i interferencji ale wykracza to poza wąskie ramy niniejszego opracowania.

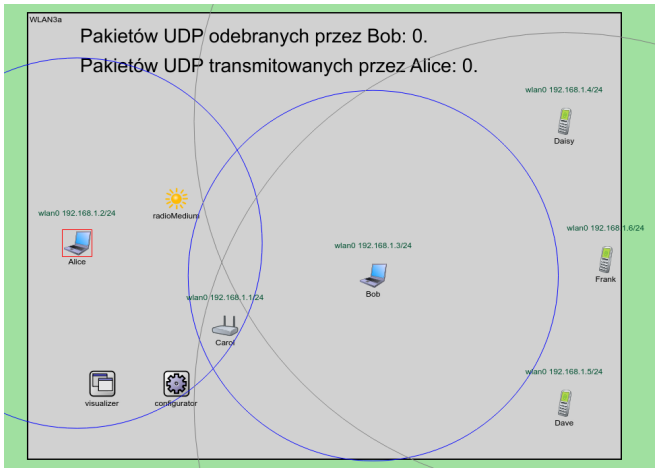
Przykład w tej części ćwiczenia bazuje na omówionej wyżej sieci z routerem a więc występują w nim Alice, Bob oraz Carol w swoich dotychczasowych rolach. Sieć jest rozszerzona o jeszcze trzy urządzenia zgodnie z poniższym fragmentem pliku NED.

```
52 network WLAN3a extends WLAN2
```

```
{
    submodules:
        Daisy: <default("WirelessHost")> like INetworkNode {
            @display("p=362,73;i=device/cellphone");
        }
        Dave: <default("WirelessHost")> like INetworkNode {
            @display("p=362,263;i=device/cellphone");
        }
        Frank: <default("WirelessHost")> like INetworkNode {
            @display("p=390,166;i=device/cellphone");
        }
}
```

Listing 68. Dodatkowe węzły sieci będące źródłem zakłóceń.

Te trzy węzły stanowią odrębną sieć taką, że Daisy i Dave wysyłają swoje pakiety do Franka. Dla odróżnienia mają inne ikony, niż wcześniej występujące urządzenia, jednak nie ma to znaczenia dla ich funkcjonowania. Sieć ta, po uruchomieniu symulacji wygląda ja na rysunku 24.



Rysunek 24. Pozycje urządzeń w eksperymencie z zakłóceniami.

Ponieważ zasięgi wszystkich urządzeń są takie same, to z rysunku widzimy, że trzy nowe węzły znajdują się poza zasięgiem któregokolwiek z urządzeń wcześniejszych. Jednak oprócz zasięgu skutecznej komunikacji istnieje zasięg interferencji, w którym siła sygnału z urządzeń jest jeszcze na tyle duża, że wpływa negatywnie na inne transmisje. Dla urządzeń Daisy i Dave jest on zaznaczony szarymi łukami – wycinkami okręgu zasięgu interferencji o dość dużym promieniu. Skonfigurowanie tej sieci znajduje się w pliku inicjalizacyjnym, który teraz przeanalizujemy.

```
[Config Interferers]
description = Kolizje i interferencje
extends = Coordinator
network = WLAN3a  # [WLAN3a|WLAN3b]
sim-time-limit = 300s

*.*.**.bitrate = 11Mbps
*.*.wlan[*].mac.useAck = false # [false|true]
```

Listing 69. Początek konfiguracji sieci z interferencjami.

Symulacja tej sieci będzie prowadzona przez 300 sekund a więc pięć razy dłużej, niż poprzednie. Jednocześnie przepustowość sieci została zmniejszona mniej więcej pięciokrotnie, do 11 Mbps. Zostało to zrobione celowo, aby dla tej niewielkiej sieci zwiększyć prawdopodobieństwo pojawienia się zakłóceń transmisji, które to zjawisko chcemy badać. Używanie potwierdzeń ACK dla transmisji zostało wyłączone.

Poprzednio omówione przykłady ignorowały interferencje. Ich włączenie i określenie zasięgu jest pokazane na listingu 70, który nie wymaga szczegółowego komentarza.

```
*.*.wlan[*].radio.receiver.ignoreInterference = false
*.*.wlan[*].radio.transmitter.interferenceRange = 250m
*.D*.wlan[*].radio.displayInterferenceRange = true
```

Listing 70. Włączenie interferencji.

Kolejny fragment pliku inicjalizacyjnego, widoczny na listingu 71 to znana nam już konfiguracja wysyłania serii pakietów UDP.

```
*.D*.numApps = 1
*.D*.app[0].typename = "UdpBasicBurst"
*.D*.app[0].startTime = 0s
*.D*.app[0].sendInterval = uniform(20ms, 40ms)
*.D*.app[0].burstDuration = uniform(40ms, 100ms)
*.D*.app[0].sleepDuration = uniform(10ms, 20ms)
*.D*.app[0].messageLength = int(uniform(576B, 1520B))
*.D*.app[0].destAddresses = "Frank"
*.D*.app[0].destPort = 2000
*.D*.app[0].localPort = 22000
*.D*.app[0].chooseDestAddrMode = "once"
```

Listing 71. Konfiguracja wysyłania pakietów UDP na potrzeby testu interferencji.

W niniejszym eksperymencie zwracamy uwagę na czas startu aplikacji „UdpBasicBurst" w urządzeniach Daisy i Dave. Możemy ten czas zmienić aby sprawdzić statystycznie wpływ tej transmisji na skuteczność komunikacji między urządzeniami Alice i Bob. Dla wartości opóźnienia startu równej 0 wpływ ten powinien być największy a dla wartości 300 sekund – żaden. Odbiorcą pakietów jest Frank, którego konfigurację widzimy na następnym listingu.

```
88 *.Frank.numApps = 1
89 *.Frank.app[0].typename = "UdpSink"
90 *.Frank.app[0].localPort = 2000
```

Listing 72. Konfiguracja odbierania pakietów UDP przez węzeł Frank.

Ponieważ liczba połączeń jest tu znacznie większa niż we wcześniejszych eksperymentach, to warto wyłączyć nadmiarowe elementy graficzne, takie jak prezentowanie tras. Realizue to poniższa linia konfiguracji.

```
92 *.visualizer.routingTableVisualizer.displayRoutingTables = false
```

Listing 73. Konfiguracja odbierania pakietów UDP przez węzeł Frank.

Zadanie

- Zmieniaj czas startu transmisji z urządzeń Daisy i Dave oraz narysuj wykres pokazujący jak to opóźnienie wpływa na stosunek liczby pakietów wysłanych z Alice do odebranych przez Bob.
- Powtórz powyższe z ustawieniem flagi `useAck` na wartość `true`.
- Zmień konfigurację sieci z WLAN3a na WLAN3b i powtórz oba powyższe punkty. Zbierz wyniki w postaci czterech krzywych na jednym wykresie.
- Spróbuj wyjaśnić, dlaczego uzyskujemy różne wartości i dlaczego właśnie takie w powyższych sytuacjach.

Podczas realizacji powyższych zadań, symulacje mogą zakończyć się z błędem lub niespodziewanym wynikiem.

Pakiety są tworzone w wyższej warstwie symulowanego urządzenia i kierowane do wysłania. Ponieważ używany model urządzenia jest bardzo prosty, to może zdarzyć się, że pakiety te będą generowane zbyt często i urządzenie otrzyma kolejne dane, gdy nie zdążyło jeszcze wysłać wcześniejszych. W takiej sytuacji symulacja zatrzyma się i zgłosi błąd. Jest to błąd wewnętrzny pakietu INET zauważony w roku 2022. Być może dostępna w czasie eksperymentów wersja będzie już tego błędu pozbawiona. Prostym remedium na taki zbieg okoliczności jest modyfikacja czasu `startTime` o czas $t \in (-0,5; +0,5)$

sekund i ponowne przeprowadzenie symulacji. Tak małe przesunięcie to mniej niż 2‰ czasu trwania symulacji, można więc uznać je za statystycznie dopuszczalne.

Inną dziwną sytuacją, również ze względu na niefortunny zbieg okoliczności dla pewnego ustawienia, może być taka, że uzyskana liczba pakietów jest znacznie niższa, niż dla wartości sąsiednich. Teoretycznie możemy spodziewać się „eleganckiej" i „gładkiej" krzywej zależności liczby skutecznych transmisji od czasu pracy urządzeń zakłócających, działających podczas eksperymentu. Jednak uzyskiwany wynik jest tylko jedną próbką krzywej, która *de facto* jest zaszumiona. Ze względu na ograniczenia czasowe zajęć może nie być możliwości przeprowadzenia długotrwałych testów z liczeniem statystyk. Zakładamy, że lokalne minimum w średniej kroczącej miałoby znikomy wpływ na końcowy wynik w postaci idealnego wykresu. Dlatego i tu prostym remedium jest przesunięcie czasu `startTime` o $t \in (-0,5; +0,5)$ sekund i ponowne przeprowadzenie symulacji.

4.2.4 Detekcja nośnej i unikanie kolizji

W kolejnym eksperymencie zastosujemy *Carrier-Sense Multiple Access with Collision Avoidance*, CSMA/CA, czyli algorytm polegający na krótkotrwałym nasłuchiwaniu pasma w celu wykrycia ewentualnych innych transmisji i czasowego wycofywania się z transmisji, gdy to takiego zakłócenia dochodzi. Konfiguracja tej symulacji polega w zasadzie tylko na zmianie funkcjonowania interfejsów radiowych urządzeń, więc w listingu 74 jest ona pokazana w całości.

```
94  [Config CSMA-CA]
95  description = Carrier Sense Multiple Access, Collision Avoidance
96  extends = Interferers
97  network = WLAN3a  # [WLAN3a|WLAN3b]
98  *.*.wlan[*].mac.useAck = false # [false|true]
99
100 *.*.wlan[*].typename = "WirelessInterface"
101 *.*.wlan[*].radio.typename = "UnitDiskRadio"
102 *.*.wlan[*].mac.typename = "CsmaCaMac"
103 *.*.wlan[*].mac.ackTimeout = 100us
104 *.*.wlan[*].queue.typename = "DropTailQueue"
105 *.*.wlan[*].queue.packetCapacity = -1
```

Listing 74. Konfiguracja odbierania pakietów UDP przez węzeł Frank.

Na początku widzimy parametr decydujący o przekazywaniu potwierdzeń ACK. Następnie znany z wcześniejszych eksperymentów typ urządzeń „AckingWirelessInterface" zostaje tu zamieniony na ogólny interfejs radiowy „WirelessInterface". Zaletą tego interfejsu jest to, że jego moduł jest rozszerzany przez moduł `CsmaCaInterface` a więc rozwiazanie będące przedmiotem aktualnych rozważań. Moduł `CsmaCaInterface`

z odpowiednio dobranymi parametrami może być dobrym przybliżeniem rozwiązania używanego w IEEE 802.11b.

Typ transmisji radiowej jest tu ustawiony na najpowszechniejsze rozwiazanie stosowane w sieciach lokalnych, czyli transmisję dookólną – `UnitDiskRadio`. Obecny tu model jest bardzo prosty, gdyż operuje tylko na dwóch zasięgach: skutecznej transmisji i interferencji. W istocie korzystaliśmy z niego już wcześniej ale w przypadku symulowania CSMA/CA musimy jawnie zadeklarować jego użycie.

Widzimy też ograniczenie czasu oczekiwania na pakiet potwierdzający odebranie wiadomości. To ustawienie oczywiście działa tylko wtedy, gdy potwierdzenia ACK są włączone. Przy włączonych potwierdzeniach pakiet jest zaliczany jako „wysłany" tylko wtedy, gdy urządzenie otrzyma potwierdzenie.

Ostatnie dwie linie konfigurują kolejkę pakietów. Jeśli nie ma miejsca w kolejce to pakiet jest odrzucany. Długość kolejki określona wartością -1 oznacza brak limitu jej długości.

Zadanie

- Powtórz poprzednią serię eksperymentów: przy włączonym i wyłączonym ACK, przy dwóch rozmieszczeniach sieci `WLAN3a` oraz `WLAN3b` i narysuj wykres zawierający cztery krzywe wyników.
- Porównaj wyniki uzyskane przy zastosowanym CSMA/CA z wynikami uzyskanymi wcześniej dla zwykłego interfejsu bezprzewodowego.
- Włącz użycie potwierdzeń ACK ale zmieniaj czas `ackTimeout` w zakresie $(1\mu s; 100\mu s)$. Zachowaj wyniki i przygotuj wykres.

4.2.5 Sieć typu ad hoc z wykorzystaniem protokołu AODV

Sieci radiowe, bezprzewodowe ze swej natury są tworami dynamicznymi. W tej części ćwiczenia przyjrzymy się symulacji sieci, w której urządzenia pozostają w ruchu a topologia sieci musi się cały czas zmieniać aby pakiety między Alice i Bob mogły być transmitowane.

Konfiguracja w pliku NED

W pierwszym kroku za pomocą wykresu zdarzeń rozpoznamy prostą, jeszcze stacjonarną symulację bazową. Chociaż urządzenia pozostają w bezruchu to wykorzystują protokół AODV do ustalenia tras wysyłania pakietów. Przyjrzyjmy się plikowi NED, którego początek widnieje na listingu 75.

```
1 package wireless.simulations.AdHocWLAN;
2
```

```
import inet.networklayer.configurator.ipv4.Ipv4NetworkConfigurator;
import inet.node.contract.INetworkNode;
import
    inet.physicallayer.wireless.common.contract.packetlevel.IRadioMedium;
import inet.visualizer.contract.IIntegratedVisualizer;
import inet.environment.common.PhysicalEnvironment;
```

Listing 75. Nagłówek pliku NED symulacji sieci Ad-hoc.

Jego konstrukcja jest niemal identyczna z przedstawioną na listingu 53. Różnicą jest oczywiście nazwa symulacji w pierwszej linii oraz to, że w dalszych krokach będziemy używać parametrów środowiska fizycznego.

Następny fragment, przedstawiony na listingu 76 jest niemal tym samym, co omówiony wyżej listing 54. Zatem on również nie wymaga ponownego komentarza a jedynie przypomnienia, że służy wizualizacji statystyk w obszarze kanwy graficznej, podczas wykonywania się symulacji.

```
network AdHocWLAN1
{
    parameters:
        @display("bgb=400,300");

        @figure[rcvdBobUDP](type=indicatorText; pos=35,15;
    anchor=w; font=,10; textFormat="Pakietów UDP odebranych przez
    Bob: %g."; initialValue=0);

    @statistic[packetReceived](source=Bob.app[0].packetReceived;
    record=figure(count); targetFigure=rcvdBobUDP);
        @figure[txAliceUDP](type=indicatorText; pos=35,35;
    anchor=w; font=,10; textFormat="Pakietów UDP nadanych przez
    Alice: %g."; initialValue=0);
        @statistic[packetSent](source=Alice.app[0].packetSent;
    record=figure(count); targetFigure=txAliceUDP);
```

Listing 76. Konfiguracja wyświetlania komunikatów pomocniczych na kanwie graficznej.

Dalsze linie pliku NED, pokazane na listingu 77 zawierają konfigurację submodułów.

```
    submodules:
        visualizer:
    <default(firstAvailableOrEmpty("IntegratedCanvasVisualizer"))>
    like IIntegratedVisualizer if typename != "" {
            @display("p=50,250");
```

```
22          }
23          configurator: Ipv4NetworkConfigurator {
24              @display("p=100,250");
25          }
26          radioMedium: <default("UnitDiskRadioMedium")> like
    IRadioMedium {
27              @display("p=100,125");
28          }
29          physicalEnvironment: PhysicalEnvironment {
30              @display("p=580,425");
31          }
```

Listing 77. Konfiguracja podstawowych parametrów środowiska symulacji.

Te powyższe kilka linii kolejno ustawia i umożliwia użycie:

- wizualizację,
- konfigurację sieci IPv4,
- medium radiowe w postaci transmisji dookólnej,
- środowisko fizyczne w postaci przeszkód terenowych utrudniających lub uniemożliwiających transmisję.

Pozostałe linie pliku NED ustawiają sześć węzłów symulowanej sieci. Głównymi węzłami są Alice oraz Bob, których konfiguracja pokazana jest na listingu 78.

```
32          Alice: <default("WirelessHost")> like INetworkNode {
33              @display("p=100,50;i=device/pocketpc");
34          }
35          Bob: <default("WirelessHost")> like INetworkNode {
36              @display("p=300,250;i=device/pocketpc");
37          }
```

Listing 78. Przykładowa konfiguracja węzła mobilnej sieci radiowej.

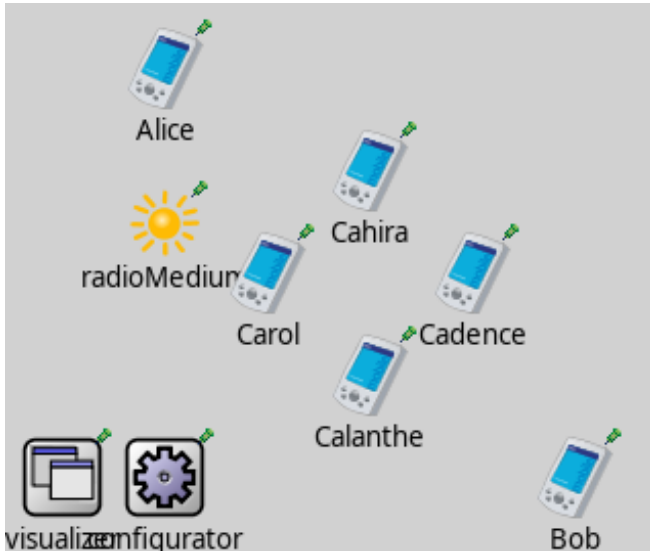
Pozostałe cztery węzły to punkty dostępowe sieci, które będą mogły przekazywać pakiety pomiędzy Alice i Bob. Ich konfiguracja, stanowiąca zarazem koniec pliku NED jest widoczna na poniższym listingu i nie wymaga komentarza.

```
38          Carol: <default("WirelessHost")> like INetworkNode {
39              @display("p=150,150;i=device/pocketpc");
40          }
41          Cadence: <default("WirelessHost")> like INetworkNode {
42              @display("p=250,150;i=device/pocketpc");
43          }
```

```
44          Cahira: <default("WirelessHost")> like INetworkNode {
45              @display("p=200,100;i=device/pocketpc");
46          }
47          Calanthe: <default("WirelessHost")> like INetworkNode {
48              @display("p=200,200;i=device/pocketpc");
49          }
50  }
```

Listing 79. Przykładowa konfiguracja węzła mobilnej sieci radiowej.

Rozmieszczenie urządzeń względem siebie, będące efektem tej konfiguracji, jest pokazane na rysunku 25. Można się domyślać, że węzły Alice i Bob nie będą mogły komunikować się bezpośrednio a będą musiały skorzystać z pośrednictwa pozostałych.



Rysunek 25. Rozmieszczenie urządzeń sieci ad-hoc dla testów AODV.

Konfiguracja w pliku inicjalizacyjnym

Możemy teraz przejść do omówienia szczegółów działania symulowanej sieci zawartych w pliku inicjalizacyjnym, którego początek widzimy niżej.

```
1  [General]
2  record-eventlog = true
3  *.*.ipv4.arp.typename = "Arp"
```

Listing 80. Początek pliku inicjalizującego sieć.

Pierwsze z powyższych ustawień w tym podręczniku jeszcze się nie pojawiło. Jak łatwo się domyśleć, włącza ono zapisywanie w trakcie symulacji dziennika zdarzeń. Znajdziemy go w katalogu `results`. Plik logów może szybko urosnąć do setek megabajtów, dlatego z tą opcją warto rozważnie dobierać czas trwania symulacji. Tablica ARP będzie przechowywana lokalnie co pozwoli nam zaobserwować proces jej tworzenia.

Kolejne linie zawierają konfigurację transmisji pomiędzy Alice i Bob. Alice uruchamia jedną aplikację – „UdpSourceApp" wysyłając co czas do około sekundy pakiety o wielkości od 576 do 1520 bajtów. Odbiorcą jest Bob, który również uruchamia tylko jedną aplikację – „UdpSink". Tuż pod konfiguracją aplikacji realizowanych na tych dwóch węzłach jest deklaracja przepustowości ustalona na 54 Mbps.

Dalej znajdziemy konfigurację sieci, z automatycznym przydzieleniem adresów z puli 10.0.0.0/24, bez udziału tras statycznych i z wyłączoną ich optymalizacją. Optymalizacja w przypadku sieci ad-hoc, pozbawionej łącz stałych i tak jest bez znaczenia. Podobne fragmenty pojawiały się już w tym podręczniku, dlatego nie będą tu pokazane ponownie.

Nowością jest za to typ urządzeń funkcjonujących w tej sieci, co widać poniżej:

```
23 *.Alice.typename = "AodvRouter"
24 *.Bob.typename = "AodvRouter"
25 *.Ca*.typename = "AodvRouter"
```

Listing 81. Konfiguracja węzłów jako routerów AODV.

W OMNeT++ „AodvRouter" jest to „WirelessHost", który już widzieliśmy w akcji ale rozszerzony o funkcję AODV.

Dalszy fragment konfiguracji to ustalenie parametrów medium radiowego, pokazane na listingu 82.

```
27 *.*.wlan[*].typename = "WirelessInterface"
28 *.*.wlan[*].radio.typename = "UnitDiskRadio"
29 *.*.wlan[*].radio.transmitter.communicationRange = 120m
30 *.A*.wlan[*].radio.displayCommunicationRange = true
31 *.B*.wlan[*].radio.displayCommunicationRange = true
32 *.*.wlan[*].radio.receiver.ignoreInterference = false
```

Listing 82. Konfiguracja warstwy fizycznej – radiowej.

Tak jak wcześniej, korzystamy z radia dookólnego, ustalając jego zasięg na 120 metrów. Zasięg ten będzie pokazany na kanwie graficznej tylko dla węzłów Alice i Bob. Zakłócenia interferencyjne będą brane pod uwagę przy analizie skuteczności odbioru pakietów.

Kolejne kilka linii to ustawienie warstwy MAC tak, aby używała już nam znanego algorytmu CSMA/CA. Limit czasowy potwierdzeń został tu ustalony na 100 μs a długość nagłówka MAC na 32 bajty. Kolejka pakietów ma długość 10 a pakiety nadmiarowe będą odrzucane.

Dalszy fragment, pokazany na listingu 83 zawiera nowe elementy związane z mobilnością urządzeń.

```
41 *.*.mobility.initialZ = 1.5m
42 *.*.mobility.constraintAreaMinZ = 0m
43 *.*.mobility.constraintAreaMaxZ = 2m
```

Listing 83. Parametry mobilności w osi Z.

Wszystkie urządzenia będą początkowo na wysokości Z=1,5 m. Podczas realizacji symulacji ich położenie będzie mogło się zmienić ale wysokość musi pozostać w przedziale $z \in \langle 0, 2 \rangle$ metrów.

Ostatnie trzy linie ogólnej konfiguracji sieci pokazane są na listingu 84.

```
45 *.visualizer.physicalLinkVisualizer.displayLinks = false
46 *.visualizer.dataLinkVisualizer.displayLinks = true
47 *.visualizer.networkRouteVisualizer.displayRoutes = true
```

Listing 84. Konfiguracja wizualizacji.

Wizualizacja połączeń w warstwie fizycznej będzie wyłączona gdyż w przypadku sieci radiowych z dookólną anteną takich połączeń może być bardzo dużo. Zaciemniłoby to obraz jednak zainteresowani czytelnicy mogą włączyć tę opcje by samemu się przekonać, jak tu wygląda. Włączone będzie wizualizowanie transmisji w warstwie danych oraz w warstwie sieciowej.

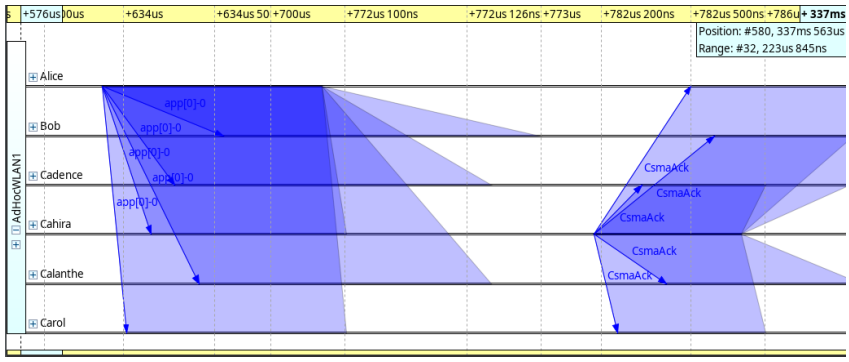
Testowanie AODV

Uruchomienie symulacji „General” nie jest możliwe i trzeba stworzyć właściwą konfigurację. Ponieważ omówiona wyżej konfiguracja zawiera niemal kompletny opis działania potrzebnych warstw sieci oraz węzłów tej sieci wraz z ich aplikacjami testowymi to wystarczy prosta konfiguracja rozszerzająca taka jak na listingu 85.

```
49 [Config Stationary]
50 sim-time-limit = 1s
51 description = Ad-hoc stacjonarnie
52 network = AdHocWLAN1
```

Listing 85. Sieć stacjonarana z wykorzystaniem AODV – właściwa konfiguracja symulacji.

Jak widać jedynym parametrem jest ograniczenie symulacji do wykonywania się przez zaledwie sekundę. To wystarczy by zebrać przykłady wszystkich istotnych pakietów. Po wykonaniu się tej symulacji w katalogu `results` powstanie plik z rozszerzeniem „elog", który można otworzyć w OMNeT++. Przykładowa prezentacja pliku dziennika zdarzeń pokazana jest na rysunku 26.



Rysunek 26. Graficzna analiza dziennika zdarzeń.

„Łapiąc" wykres lewym przyciskiem myszy można go przesuwać. Przytrzymując Ctrl można za pomocą kółka myszy zmieniać powiększenie.

Podczas analizy dziennika mogą wystąpić problemy z zarządzeniem tym okienkiem. Klikając prawym przyciskiem myszy w obrębie wykresu można otworzyć menu kontekstowe, w którym da się szybko skonfigurować widok za pomocą Preset Configuration. Również w menu kontekstowym jest Show/Hide, gdzie można wyłączyć Show Component Method Calls. Jest też podmenu dotyczące powiększenia, które można szybko przełączyć tak, by zobaczyć cały przebieg, co będzie jednak kosztowało sporo czasu przy wielu zarejestrowanych zdarzeniach. Należy pamiętać, że dłuższy plik elog na słabym komputerze może odświeżać się bardzo długo.

Zadanie

- Wykonaj wyżej omówioną symulację komunikacji urządzeń stacjonarnych i przeanalizuj dziennik zdarzeń.

Mobilność urządzeń sieci ad-hoc

Druga symulacja z tego zestawu powoduje włączenie ruchu urządzeń. Początek tego fragmentu konfiguracji pokazany jest niżej.

```
54 [Config Mobility]
55 record-eventlog = false
56 description = Ad-hoc mobilnie
```

```
57 network = AdHocWLAN1
```

Listing 86. Początek konfiguracji sieci moblinej.

Jak widać zapisywanie dziennika zdarzeń zostało wyłączone przy czym ta symulacja nie jest ograniczona czasowo. Wykorzystuje ona tę samą sieć złożoną z sześciu wspomnianych wyżej urządzeń: Alice, Bob oraz cztery routery z nazwami na literę „C".

Mobilność węzła Alice jest skonfigurowana tak, jak na listingu 87.

```
59 *.A*.mobility.typename = "LinearMobility"
60 *.A*.mobility.speed = 1.4mps
61 *.A*.mobility.constraintAreaMinX = 50m
62 *.A*.mobility.constraintAreaMinY = 50m
63 *.A*.mobility.constraintAreaMaxX = 140m
64 *.A*.mobility.constraintAreaMaxY = 150m
```

Listing 87. Początek konfiguracji sieci moblinej.

Alice będzie przemieszczać się z prędkością $1,4m/s$ ruchem liniowym w obszarze prostokąta ograniczonego współrzędnymi $x \in \langle 50; 140 \rangle$ oraz $y \in \langle 50; 150 \rangle$. Po dotarciu do krawędzi węzeł „odbije się" od niej i będzie kontynuował ruch liniowy. Początek układu współrzędnych znajduje się w lewym-górnym rogu kanwy graficznej.

Węzeł Bob porusza się ruchem pseudolosowym ale do pewnego stopnia deterministycznym. Jego konfiguracja jest pokazana na listingu 88.

```
66 *.B*.mobility.typename = "GaussMarkovMobility"
67 *.B*.mobility.alpha = 0.9
68 *.B*.mobility.speed = 1.4mps
69 *.B*.mobility.angleStdDev = 120.0deg
70 *.B*.mobility.speedStdDev = 1.4mps
71 *.B*.mobility.margin = 10m
72 *.B*.mobility.constraintAreaMinX = 220m
73 *.B*.mobility.constraintAreaMinY = 50m
74 *.B*.mobility.constraintAreaMaxX = 360m
75 *.B*.mobility.constraintAreaMaxY = 250m
```

Listing 88. Ruch pseudolosowy węzła Bob.

Współczynnik „alpha" określa stopień determinizmu w ruchu węzła przy czym wartość zero oznacza ruch całkowicie losowy. Maksymalna wartość tego parametru to 1, oznaczająca całkowicie deterministyczny ruch. Parametr „margin" daje bufor bezpieczeństwa przed „ścianami" obszaru ograniczającego, którego węzeł nie powinien przekroczyć.

Pozostałe węzły, czyli routery pośredniczące poruszają się po okręgu, zgodnie z konfiguracją pokazaną na listingu 89.

```
77 *.Ca*.mobility.typename = "CircleMobility"
78 *.Ca*.mobility.cx = 200m
79 *.Ca*.mobility.cy = 150m
80 *.Ca*.mobility.r = 70m
81 *.Ca*.mobility.speed = 1.4mps
82 *.Cadence.mobility.startAngle = 0deg
83 *.Cahira.mobility.startAngle = 90deg
84 *.Calanthe.mobility.startAngle = 180deg
85 *.Carol.mobility.startAngle = 270deg
```

Listing 89. Konfiguracja krążących routerów.

Pozostaje jeszcze kwestia ustawienia wizualizacji ruchu, pokazana na listingu 90.

```
87 *.visualizer.mobilityVisualizer.displayMobility = true
88 *.visualizer.mobilityVisualizer.displayPositions = true
89 *.visualizer.mobilityVisualizer.displayOrientations = true
90 *.visualizer.mobilityVisualizer.displayVelocities = true
91 *.visualizer.mobilityVisualizer.displayMovementTrails = true
92 *.visualizer.mobilityVisualizer.trailLength = 100
```

Listing 90. Wizualizacja mobilności węzłów.

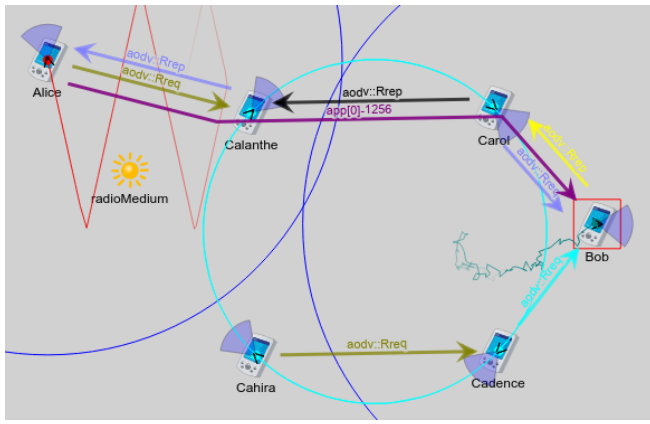
Pierwsza z opcji na powyższym fragmencie to „przełącznik główny” wizualizacji mobilności. Pozycje urządzeń są prezentowane kolorowymi kropkami, gdyż ikona zajmująca większy obszar nie wskazuje dość jednoznacznie właściwego miejsca. Skierowanie ruchu węzła w określoną stronę jest pokazywane wycinkiem koła. Prędkość jest wizualizowana długością wektora i pozwala wizualnie porównać względne prędkości urządzeń. Dla ułatwienia oceny trasy po której przemieszczają się węzły, są za nimi rysowane ścieżki, które stopniowo znikają. Efekt tej symulacji pokazany jest na rysunku 27.

Tak działającym urządzeniom można komunikację nieco „utrudnić” poprzez dodanie do symulacji przesłony (muru) uniemożliwiającej komunikację.

```
97 *.radioMedium.obstacleLoss.typename = "IdealObstacleLoss"
98 *.physicalEnvironment.config = xmldoc("mury.xml")
```

Listing 91. Konfiguracja środowiska fizycznego.

Zastosowana przesłona działa idealnie co oznacza, że w pełni tłumi sygnał radiowy. Realistyczna symulacja uwzględniłaby rzeczywiste parametry dielektryka, z którego

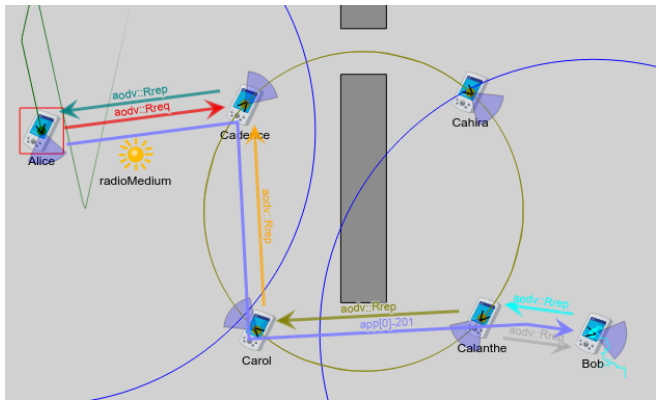


Rysunek 27. Wizualizacja komunikacji w mobilnej sieci ad-hoc.

zbudowana jest przesłona oraz wzięła pod uwagę efekty interferencyjne, które zachodzą na krawędziach takiego obiektu. Kształt tego rodzaju obiektów tworzących środowisko jest zapisywany w pliku XML. Zawartość pliku przygotowanego dla tej symulacji jest pokazany na listingu 92 a wizualizacja jego zastosowania na rysunku 28.

```
<environment>
 <object position="min 190 0 -5" orientation="0 0 0" shape="cuboid
    20 70 25" material="concrete" fill-color="140 140 140"
    opacity="1.0"/>
 <object position="min 190 90 -5" orientation="0 0 0" shape="cuboid
    20 100 25" material="concrete" fill-color="140 140 140"
    opacity="1.0"/>
</environment>
```

Listing 92. Konfiguracja prostopadłościanów tworzących przesłonę.



Rysunek 28. Mobilna sieć ad-hoc oraz przesłona utrudniająca komunikację.

Zadanie

- Zmień parametry konfigurujące mobilność urządzeń bez zmiany algorytmu ruchu (np.: kierunek ruchu, prędkość) i sprawdź efekt wprowadzonych zmian.
- Zmień mobilność wybranych urządzeń korzystając z innych algorytmów ruchu, zgodnie dokumentacją pakietu INET.
- Dla przypadku z przesłoną sprawdź jak mniejszy i większy zasięg łączności między węzłami wpływa na transmisję między urządzeniami Alice i Bob.
- Zmieniaj wymiary i położenie przesłon obserwując wpływ na nawiązywanie i zrywanie połączeń.

4.3 Pytania sprawdzające

1. Jakie są cechy charakterystyczne i własności sieci MANET.
2. Jak działa algorytm AODV?
3. Czym jest i jak działa CSMA/CA?
4. Jakie są podstawowe parametry warstwy fizycznej i dostępu do medium w prostej symulacji transmisji bezprzewodowej za pomocą pakietu INET?
5. Jak używać i jakie są własności rozszerzania konfiguracji sieci w plikach NED?
6. Jaki wpływ na transmisje w sieciach bezprzewodowych mają zakłócenia, odbicia, tłumienie i interferencje?

Rozdział 5

Lokalne sieci bezprzewodowe

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się ze specyficznymi cechami standardu WiFi. Część teretyczna dotyczy modulacji, kodowania i innych algorytmów wykorzystywanych współcześnie w tym standardzie. Część praktyczna obejmuje symulację urządzeń i punktów dostępowych z uwzględnieniem realnego wpływu środowiska fizycznego na transmisje.

Do realizacji bezprzewodowych sieci IoT stosuje się powszechnie znane standardy, takie jak:

- WiFi, czyli powszechnie znany standard prywatnych sieci lokalnych IEEE 802.11,
- Bluetooth, obecnie BLE, dla sieci osobistych, w lekkich urządzeniach przenośnych, który swe początki ma w standardzie IEEE 802.15.1,
- ZigBee, dedykowany komunikacji bezpiecznej i energooszczędnej, w szczególności sieciom budynkowym, opracowany na fundamencie IEEE 802.15.4,
- Thread, rozbudowany standard dedykowany IoT, również oparty o IEEE 802.15.4,
- LoRa, otwarty standard dla sieci wielkoobszarowych, korzystający z nielicencjonowanych pasm *Idustrial, Scientific, Medical*, ISM z wszystkimi korzyściami i problemami z tym zwiazanymi,
- Sigfox, zamknięty, komercyjny standard o niewielkiej przepustowości ale bardzo dalekiego zasięgu,
- NB-IoT, będący rozszerzeniem sieci *Global System for Mobile Communications*, GSM, wykorzystujący wsparcie bogatej infrastruktury telekomunikacyjnej operatorów sieci komórkowych.

Wymienione wyżej standardy nie wyczerpują listy dostępnych rozwiązań a ich pojawienie się tutaj ma wskazać pewne warunki brzegowe niniejszego ćwiczenia. Jeśli

przyjrzeć się tym protokołom, dostrzeżemy wiele wspólnych cech, takich jak modulacje sygnałów i kodowanie informacji na ich niższych poziomach oraz tymczasowość połączeń na wyższych.

5.1 WiFi

Standard IEEE 802.11 wraz z jego licznymi dodatkami, rozszerzeniami i aktualizacjami jest powszechnie znany pod nazwą implementacyjną jako WiFi. Najnowsze wersje WiFi ze znacznym opóźnieniem upowszechniają się w urządzeniach IoT i wbudowanych, gdzie wciąż powszechne są proste i tanie moduły 802.11b/g/n [30]. Nowsze rozwiązania pojawiają się w routerach, gdzie coraz częściej standard 802.11ac (znany też jako WiFi 5) jest wypierany przez 802.11ax (czyli WiFi 6) [31–33].

Mniej więcej co cztery lata IEEE *Computer Society* publikuje uaktualnioną i ujednoliconą wersję standardu 802.11. W momencie pisania niniejszego podręcznika najnowsza nosi przyrostek „-2020" a została opublikowana 26 lutego 2021 [34]. Dokument ten liczy 4226 stron zatem oczywiste jest, że ze względu na ograniczoną objętość niniejszego podręcznika, nie ma tutaj możliwości omówienia wszystkich aspektów tego skomplikowanego standardu.

Dlatego poniższy opis ograniczy się do wybranych, kluczowych rozwiązań stosowanych w obecnie używanych wersjach WiFi.

5.1.1 Modulacje i kodowanie cyfrowe

WiFi, jak każda inna transmisja cyfrowa, korzysta z modulacji, które możemy określić jako „cyfrowe". Jednak otaczająca nas rzeczywistość, taka jak fale elektromagnetyczne a w tym fale radiowe, jest analogowa. Transmitowane sygnały radiowe są ciągłe w dziedzinie czasu i amplitudy, zupełnie inaczej niż intuicyjnie możemy spodziewać się po transmisjach cyfrowych [35].

Sygnał analogowy możemy rozłożyć na zbiór funkcji o tym samym charakterze, takich jak na przykład funkcje sinus i cosinus, ale różniących się parametrami takimi jak częstotliwość, amplituda i przesunięcie fazowe. To przekształcenie realizowane jest przez transformację Fouriera, daną wzorem 5.1.

$$G(f) = \int_{-\infty}^{\infty} g(t) e^{-i2\pi f t} dt \tag{5.1}$$

Odwrotna operacja, czyli złożenie sygnału ciągłego ze zbioru funkcji można zrealizować odwrotną transformacją Fouriera opisaną wzorem 5.2.

$$g(t) = \int_{-\infty}^{\infty} G(f) e^{i2\pi f t} df \tag{5.2}$$

Występująca wyżej wartość $e^{i2\pi f}$ powinna być czytelnikom znana z analizy zespolonej, ze wzoru Eulera:

$$e^{i2\pi f} = \cos(2\pi f) + i\sin(2\pi f) \tag{5.3}$$

W przypadku transmisji sygnałów składową rzeczywistą ($\Re$) nazywamy synfazową (ang. *In-phase*, I) a składową urojoną ($\Im$) nazywamy kwadraturową (ang. *Quadrature*, Q). Są one względem siebie ortogonalne.

Powyższe wzory odnoszą się do ciągłej dziedziny częstotliwości i czasu ale istnieją też ich wersje dyskretne, które dobrze poddają się implementacji za pomocą podzespołów opartych na bramkach logicznych. Modulacja cyfrowa to ustalanie wartości I-Q na pewnych, skwantowanych poziomach, na podstawie wartości cyfrowej przeznaczonej do wysłania. Zgodnie ze wzorem Eulera i na podstawie sumy wartości I oraz Q generowany jest sygnał ciągły. Po odpowiednim jego wzmocnieniu można go wyemitować za pomocą anteny w postaci fali radiowej.

Kluczowanie fazy

Najprostszą modulacją cyfrową jest binarne kluczowanie fazy (*Binary Phase Shift Keying*, BPSK). BPSK jest „*binarne*" a więc ma tylko dwa stany dyskretne opisane tabelą 1.

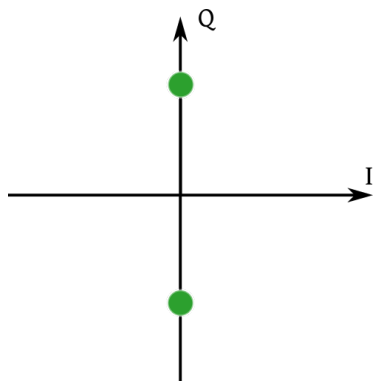
Tabela 1. Pozycje punktów diagramu konstelacji BPSK.

I	Q
0	+1
0	-1

Możliwe sytuacje da się przedstawić za pomocą punktów na wykresie synfazowo-kwadraturowym – analogicznie do przedstawiania wartości zespolonej na wykresie z osiami stanowiącymi składową rzeczywistą i urojoną. Wykres I-Q zawierający te punkty nazywa się „diagramem konstelacji". Dla BPSK diagram ten jest pokazany na rysunku 29.

Punkty diagramu konstelacji są końcami wektora zaczepionego w początku układu współrzędnych a reprezentującego dyskretne stany sygnału. Wektora jako takiego zwykle się nie rysuje aby nie zaciemniać rysunku. Długość wektora czyli dystans między początkiem układu współrzędnych a punktem n odpowiada amplitudzie sygnału.

Wiemy jednak, że ton prosty (sinus, cosinus) „płynie" w czasie zmieniając swoją wartość chwilową a można też powiedzieć, że zmienia swoją fazę. Kąt między dodatnią częścią osi odciętych (synfazowa, rzeczywistą) a wektorem to przesunięcie fazowe. „Kluczowanie" oznacza bezzwłoczne przełączanie sygnału pomiędzy możliwymi jego stanami. W przypadku BPSK są to dwa, symetryczne przesunięcia fazowe: $\pm\pi/2$. Teoretycznie nie występuje sygnał bez przesunięcia ale obserwując rzeczywisty sygnał BPSK, na przykład za pomocą oscyloskopu, zobaczymy sygnał o charakterze sinusoidalnym



Rysunek 29. Diagram konstelacji BPSK.

($\varphi = 0$) oraz tenże sygnał odwrócony ($\varphi\prime = \pm\pi$). Różnica w podejściu teroetycznym i przy praktycznej obserwacji jest taka sama i wynosi π czyli 180°.

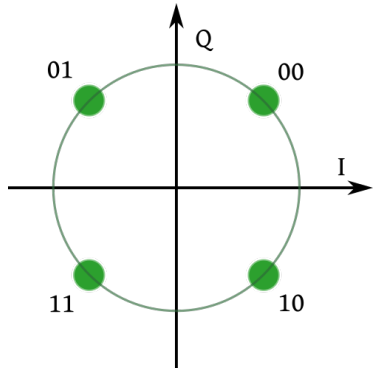
Fragment sygnału odpowiadający punktowi diagramu konstelacji nosi nazwę *symbolu*. W przypadku BPSK mamy tylko dwa symbole. Jeden z nich może reprezentować cyfrową wartość „0" a drugi wartość „1". Zatem BPSK jest najprostszą modulacją cyfrową wykorzystującą kluczowanie fazy.

Diagram konstelacji może być bardziej złożony i zawierać na przykład cztery punkty. Wówczas każdemu z nich można przyporządkować parę bitów. Ten typ kluczowania fazy nazywamy „kwadraturowym" (*Quadrature Phase Shift Keying*, QPSK) gdyż składowa synfazowa i kwadraturowa są modulowane niezależnie (czyli w kwadraturze, ortogonalnie).

Tabela 2. Pozycje i możliwe wartości punktów diagramu konstelacji QPSK.

I	Q	wartość binarna
+0,7	+0,7	00
-0,7	+0,7	01
-0,7	-0,7	11
+0,7	-0,7	10

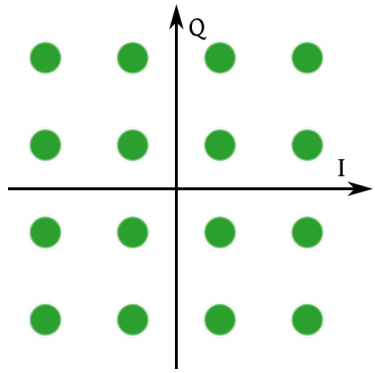
W powyższej tabeli widzimy wartość pozycji $(I; Q)$ punktów diagramu konstelacji równą $(\pm 0,7; \pm 0,7)$ po to, aby punkty diagramu konstelacji znalazły się na okręgu jednostkowym. Na rysunku 30 łatwo można zauważyć, że wartości bitowe zostały dobrane zgodnie z kodem Graya. Dzięki temu dla punktów sąsiadujących na okręgu różnią się tylko jednym bitem. To pozwala zmniejszyć częstość pojawiania się błędów bitowych w rzeczywistych transmisjach poddanych tłumieniu i szumom.



Rysunek 30. Diagram konstelacji QPSK.

Kwadraturowa modulacja amplitudy

Drugim typem modulacji stosowanym w WiFi jest kwadraturowa modulacja amplitudy (*Quadrature Amplitude Modulation*, QAM). Ponieważ jest to modulacja amplitudy, wielu autorów uważa ją za modulację analogową. Jednak poziomy sygnału składowych $(I; Q)$ powinny przyjmować wartości skwantowane więc w praktyce jest to modulacja cyfrowa. Liczba tych poziomów decyduje o liczbie punktów diagramu konstelacji. Przyjmując 4 poziomy na każdej ze składowych otrzymujemy diagram złożony z 16 punktów. Jest to najprostsza modulacja typu QAM stosowana w praktyce a zarazem w WiFi. Wygląd przykładowego diagramu konstelacji jest przedstawiony na rysunku 31. Zauważmy, że punkty diagramu są rozlokowane równomiernie, na siatce kwadratów.



Rysunek 31. Diagram konstelacji 16-punktowej QAM.

W modulacji QAM długość wektora łączącego środek układu współrzędnych oraz punkty diagramu konstelacji nie jest identyczna dla wszystkich punktów. Oznacza to, że radiowy sygnał analogowy ma zmienną amplitudę, podobnie jak sygnał analogowej modulacji amplitudowej. Zmiana amplitudy QAM ma jednak charakter kwantowy i odbywa się w dyskretnych chwilach czasowych.

Modulację QAM z 16 punktami diagramu konstelacji określamy precyzyjniej jako 16–QAM. Możemy się zatem domyślać, że istnieją jeszcze inne jej wersje. W WiFi stosowane są też 64–QAM, 256–QAM oraz 1024–QAM o diagramach konstelacji w postaci siatek o wymiarach odpowiednio: 8×8, 16×16 i 32×32. Im więcej punktów diagramu tym więcej bitów na *symbol* jest przesyłanych a więc rośnie przepustowość transmisji. Na przykład 256–QAM ma 256 symboli, czyli 256 możliwych sekwencji bitowych a zatem każdy symbol przesyła od razu cały bajt.

Im więcej punktów diagramu tym większe ich zagęszczenie w przestrzeni ograniczonej maksymalnymi napięciami sygnałów stosowanych w układzie modulującym. Oznacza to też mniejsze różnice między nimi. Zatem ze wzrostem złożoności diagramu, w przypadku analizy rzeczywistego sygnału radiowego obarczonego szumem rośnie ryzyko błędu transmisji.

Kodowanie

Drugim z czynników, które istotnie wpływają na przepustowość jest częstość kodowania. Częstość kodowania jest prezentowana jako ułamek zwykły o wartości nie większej od jeden. Licznik reprezentuje liczbę bitów danych otrzymanych od strony wyższych warstw stosu sieciowego. Mianownik to liczba bitów przeznaczonych do użycia w trakcie modulacji, które przy użyciu pewnego algorytmu kodują bity otrzymane z warstwy wyższej.

Skoro mianownik jest większy niż licznik to znaczy, że pewne dodatkowe bity nie będące pierwotną informacją są wytwarzane w wyniku kodowania. Bity te niosą dodatkową informację – redundantnie opisują oryginalne bity danych a dzięki temu zwiększają ochronę transmisji przed zakłóceniami. Tego rodzaju podejście jest stosowane w wielu typach transmisji cyfrowych i również w WiFi.

Tabela 3. Fragment tabeli *Modulation and Coding Set*, MCS standardu 802.11ax.

MCS	Modulacja	Kodowanie	Przepustowość 1 kanału 20 MHz [Mbps]
MCS0	BPSK	1/2	8
MCS1	QPSK	1/2	16
MCS2	QPSK	3/4	24
MCS3	16–QAM	1/2	33
MCS4	16–QAM	3/4	49
MCS5	64–QAM	2/3	65
MCS6	64–QAM	3/4	73
MCS7	64–QAM	5/6	81
MCS8	256–QAM	3/4	98
MCS9	256–QAM	5/6	108
MCS10	1024–QAM	3/4	122
MCS11	1024–QAM	5/6	135

Modulacje i kodowania tworzą zestawy (MCS), którymi w prosty sposób można indeksować różne sposoby transmisji. Wartości MCS wraz z modulacjami, kodowaniami określone standardem IEEE 802.11ax oraz przepustowość na jednym kanale o szerokości 20 MHz są pokazane w tabeli 3.

Współczesne transmisje WiFi mogą korzystać też z szerszych kanałów oraz wielodostępu przestrzennego, który będzie omówiony dalej. Te cechy transmisji pozwalają na uzyskanie znacznie wyższych przepustowości obecnie sięgających w praktyce kilku Gbps. Te dodatkowe opcje nie zostały ujęte w tabeli by zachować jej czytelność i ograniczyć zajmowane miejsce. Dla urządzeń IoT transmisja jednym kanałem o szerokości 20 MHz to często jedyne dostępne rozwiązanie.

5.1.2 Ortogonalne dzielenie częstotliwości

Ortogonalne dzielenie częstotliwości (*Orthogonal Frequency-Division Multiplexing*, OFDM) jest dość złożonym, wieloetapowym procesem przetwarzania sygnału transmisji cyfrowych [36, 37]. Najprościej da się go opisać jako wiele współbieżnych transmisji realizowanych za pomocą modulacji na sąsiadujących częstotliwościach, w zakresie pewnego pasma.

Pierwszy raz w WiFi to rozwiązanie zostało użyte już w 802.11a z roku 1999. Jednak ta wersja standardu się nie przyjęła dlatego na działającą implementację trzeba było poczekać. Ponownie OFDM pojawiło się w WiFi wraz z opublikowaniem w roku 2003 standardu 802.11g konsolidującego najlepsze rozwiązania z 802.11a oraz 802.11b.

W nadajniku strumień zakodowanych bitów jest rozdzielany (demultipleksowany) na wiele równoległych strumieni. Każdy z tych strumieni trafia na dedykowany modulator, z przypisaną mu indywidualną częstotliwością podnośnej. Częstotliwości podnośnych tych modulatorów równomiernie wypełniają zakres częstotliwości odpowiadających docelowej szerokości pasma transmisji, czyli na przykład 20 MHz. Mamy tu więc do czynienia z podziałem w dziedzinie częstotliwości (*Frequency Division Multiplexing*, FDM) przy czym kanały częstotliwościowe pozostają od siebie niezależne, czyli „ortogonalne", mimo tego iż w pewnym stopniu nakładają się na siebie.

W przypadku 802.11ac podnośne są oddalone co 312,5 kHz. Od 802.11ax zostały one czterokrotnie zagęszczone, czyli mają odstęp równy 78,125 kHz.

Z każdego z modulatorów uzyskiwana jest informacja o aktualnym symbolu a więc chwilowa wartość $(I; Q)$, z których można wyprowadzić informację o fazie i amplitudzie tego sygnału. W połączeniu z informacją o częstotliwości powiązanej z konkretnym modulatorem, pozwala to zrealizować odwrotną, cyfrową transformację Fouriera w celu wypracowania jednego sygnału zawierającego wszystkie strumienie cyfrowe. Do tego momentu przekształcenie to jest w pełni cyfrowe a więc daje się zaimplementować w postaci struktury szybkich bramek logicznych.

Ten wspólny, nadal cyfrowy sygnał zostaje przekształcony do sygnału analogowego i poddany mieszaniu z docelową częstotliwością, na której ma odbywać się transmisja. Po wzmocnieniu do właściwego poziomu trafia do anteny, skąd zostaje wyemitowany.

W odbiorniku należy wykonać operacje w odwrotnej kolejności:

- pozyskać sygnał z anteny,
- zmiksować go do niższych częstotliwości,
- przekształcić do sygnału cyfrowego,
- dokonać analizy Fourierowskiej w celu wyodrębnienia poszczególnych kanałów,
- dokonać demodulacji, czyli uzyskać wartości $(I; Q)$ i odpowiadający im symbol,
- odkodować sekwencję bitową.

Pozostaje pytanie – po co dokonuje się tak skomplikowanego przetwarzania? Otóż w przypadku modulacji na jednej częstotliwości widmowa gęstość mocy transmisji miałaby nierównomierny rozkład, bliższy rozkładowi normalnemu (gaussowskiemu). Im bliżej górnej i dolnej granicy pasma, tym słabszy byłby tam sygnał. Można powiedzieć, że te obszary zostałyby „zmarnowane", biorąc pod uwagę ograniczenia na moc transmisji nałożone obowiązującymi regulacjami. Wykorzystanie OFDM pozwala na wydajniejsze rozproszenie mocy w dostępnym pasmie a tym samym poprawę wartości stosunku sygnału do szumu (*Signal-to-Noise Ratio*, SNR).

Od 802.11ax OFDM zostało rozszerzone do *Orthogonal Frequency-Division Multiple Access*, OFDMA, co pozwala na współdzielenie pasma przez wielu użytkowników w tym samym czasie. Podnośne stają się „jednostkami zasobów" (*Resource Unit*, RU), które mogą być przedzielane klientom w miarę potrzeb. Urządzenia IoT, które zazwyczaj nie potrzebują dużych przepustowości, mogą korzystać z niewielkiej liczby RU niejako „doklejając się" do istniejących transmisji. Nie zakłócają transferów realizowanych z dużymi przepustowościami i nie blokują całego kanału WiFi a współdzielą go.

5.1.3 Wielodostęp przestrzenny

Przepustowość można zwiększać poprzez zmianę modulacji, zmianę kodowania, poszerzenie pasma, multipleksowanie i kilka innych, drobnych zabiegów. Jednak można ją tez zwiększyć poprzez zwielokrotnienie samych nadajników i odbiorników. To podstawowa koncepcja stojąca za *Multiple Input, Multiple Output*, MIMO [38]. W WiFi to rozwiązanie zostało użyte po raz pierwszy w wersji 802.11n z roku 2009. Obecnie jest powszechnie stosowane w routerach, w których często możemy zobaczyć liczne anteny będące znakiem rozpoznawczym tego rozwiązania. Specyfikacja routera często zawiera kilka liczb w stylu $4 \times 4 : 4$, które oznaczają kolejno liczbę anten, modułów radiowych i możliwych do realizacji strumieni transmisji.

MIMO pozwala na realizowanie kierunkowej transmisji przy użyciu anten dookólnych. Sygnał przeznaczony do transmisji przesyłany jest do anten nadawczych z przesunięciem fazowym. Ze względu na przesunięcie fazowe sygnał emitowany tworzy

czoło fali odchylonej pod pewnym kątem względem płaszczyzny anten a więc może być skierowany w stronę urządzenia klienckiego Jednocześnie sygnał ten jest słabszy w innych kierunkach a dzięki temu zmniejszają się zakłócenia wzajemne urządzeń Wi-Fi. Wynika to z podstawowej zasady fizyki dotyczącej rozchodzenia się fal, czyli zasady Huygensa–Fresnela.

MIMO zazwyczaj jest rozwiązaniem asymetrycznym – urządzenie klienckie nadal transmituje sygnał dookólnie, przy użyciu jednej anteny i jednego strumienia. Trudno jest opracować anteny niewielkich rozmiarów (na przykład mikropaskowe) na tyle precyzyjnie by dało się to rozwiązanie stosować w urządzeniach klienckich wciąż powszechnie korzystających z tanich modułów 802.11b/g/n.

Mając wiele anten można zastosować jeszcze jedno rozwiązanie jakim jest *Multi-User Multiple Input, Multiple Output*, MU-MIMO, czyli podział przeznaczenia anten dla poszczególnych klientów. Pozwala to na realizację niemal niezakłócających się transmisji jednocześnie z wieloma urządzeniami.

5.2 Wykonanie ćwiczenia

Korzystając z wcześniejszych doświadczeń spróbujemy teraz zasymulować działanie sieci WiFi obejmujące routing, mobilność urządzeń oraz wpływ zagadnień takich jak poziom szumów i moc sygnału na skuteczność realizacji komunikacji.

Ogólne ustawienia tej symulacji w dużej mierze są takie same, jak we wcześniej omawianych. Dlatego nie będzie tu prezentacji i omówienia całego kodu linia po linii a jedynie najważniejszych i nowych jego elementów oraz różnic istotnych względem poprzednich przykładów.

Sieć opisana plikiem NED składa się z czterech urządzeń należących do dwóch kategorii:

- Alice – `WirelessHost`,
- Bob – `WirelessHost`,
- Carol – `AccessPoint`,
- Calanthe – `AccessPoint`.

Zgodnie z dobrze nam już znanym stylem Alice i Bob będą komunikować się za pośrednictwem punktów dostępowych Carol i Calanthe. Oprócz konfiguracji urządzeń plik zawiera też znane nam już deklaracje użycia:

- automatycznej konfiguracji adresów IPv4 (`Ipv4NetworkConfigurator`),
- środowiska fizycznego (`PhysicalEnvironment`),
- graficznego wizualizatora funkcjonowania sieci (`IntegratedVisualizer`).

Fragment, którego zawartość jest tu nowością, został pokazany na poniższym listingu.

```
44          radioMedium: Ieee80211ScalarRadioMedium {
45              @display("p=80,260");
46          }
```

Listing 93. Konfiguracja głównych parametrów warstwy fizycznej.

Medium radiowe typu `Ieee80211ScalarRadioMedium` jest nieco bardziej realistyczne, niż proste medium o kołowym kształcie zasięgu transmisji. Uwzględnia ono zanikanie sygnału wraz z odległością od źródła tego sygnału.

Plik NED tej symulacji ma jeszcze wariantowe użycie innego modelu oraz dodatkowy element próbnika spektrum radiowego, które jednak zostaną omówione później, przy opisie wykorzystujących je symulacji.

W tym przykładzie nie będzie globalnej tablicy ARP a więc urządzenia będą musiały zdobyć swoje powiązania ARP–IP. Proces nawiązywania połączenia WiFi ma jednak o wiele więcej etapów a rozpoznawanie adresu fizycznego i sieciowego jest tylko drobnym elementem tego łańcucha zdarzeń. W omawianym przykładzie nawiązanie połączeń trwa dość długo, bo około 1 sekundy czasu symulowanego.

Przepustowość łącza została ustalona na typowe 54 Mbps. Jest ona realizowana mocą równą zaledwie 1 mW, z użyciem anteny izotropowej (dookólnej). Typ zastosowanego tu radia w nomenklaturze OMNeT++ to `Ieee80211ScalarRadio`. Typ ten powinien być i jest zgodny z typem medium radiowego zadeklarowanym w pliku NED.

```
4  *.*.**.bitrate = 54Mbps
5
6  *.*.wlan[*].radio.typename = "Ieee80211ScalarRadio"
7  *.*.wlan[*].radio.antenna.typename = "IsotropicAntenna"
8  *.*.wlan[*].radio.transmitter.power = 1mW
```

Listing 94. Konfiguracja głównych parametrów warstwy fizycznej.

Konfiguracja włącza też prezentację zasięgu komunikacji urządzeń Alice i Bob za pomocą omawianych już granatowych okręgów.

Kolejne kilka linii konfiguracji dotyczą zarządzania dostępem do sieci WiFi a konkretnie: użycia identyfikatorów sieci (ang. *Service Set IDentifier*, SSID).

```
13 *.Carol.wlan[0].mgmt.ssid = "CarolNet"
14 *.Calanthe.wlan[0].mgmt.ssid = "CalantheNet"
15 *.Alice.wlan[*].agent.defaultSsid = ""
16 *.Bob.wlan[*].agent.defaultSsid = ""
```

Listing 95. Użycie SSID w punktach dostępowych i urządzeniach ruchomych.

W tym przykładzie Carol i Calanthe mają po jednym interfejsie sieciowym, którym nadają SSID odpowiednio „CarolNet” i „CalantheNet”. Urządzenia ruchome Alice i Bob nie określają preferencji łączenia się z punktem dostępowym i połączą się z dowolnym, który akurat będzie dla nich dostępny.

W kolejnych liniach pliku inicjalizującego symulację znajduje się konfiguracja pary producent–konsument pakietów UDP, realizowana przez Alice i Bob za pomocą „UdpSourceApp” oraz „UdpSink”. Alice wysyła co około sekundę (±100 ms) pakiety o wielkości od 576 B do 1400 B.

Ta symulacja uwzględnia jedną, prostopadłościenną przesłonę, która ma wpływ na nawiązywanie połączeń z punktami dostępowymi przez urządzenia Alice i Bob. Podobnie jak w przykładzie omówionym wcześniej a dotyczącym sieci ad-hoc, również i ta przesłona będzie idealnym dielektrykiem. Zawartość pliku XML została pokazana poniżej.

```
1 <environment>
2  <object position="min 315 185 -5" orientation="0 0 0"
      shape="cuboid 10 40 25" material="concrete" fill-color="140 140
      140" opacity="1.0"/>
3 </environment>
```

Listing 96. Opis przesłony użytej w symulacjach WiFi.

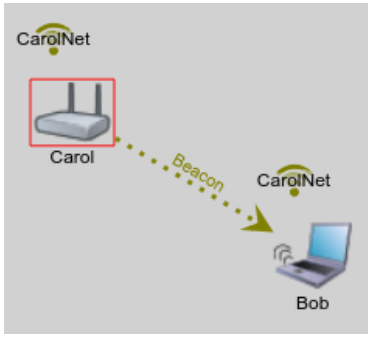
Konfiguracja włącza prezentowanie połączeń warstwy fizycznej, łącza danych i sieciowej tak, jak widzieliśmy to we wcześniej omawianych symulacjach. Jednak oprócz tego pojawia się tu kilka nowych elementów w znaczącym stopniu dotyczących WiFi. Pierwsza ich część została pokazana na listingu 97.

```
35 *.visualizer.*.ieee80211Visualizer.displayAssociations = true
36 *.visualizer.*.ieee80211Visualizer.minPower = -110dBm
37 *.visualizer.*.ieee80211Visualizer.maxPower = -80dBm
```

Listing 97. Konfiguracja wizualizacji aspektów specyficznych dla łączenia z punktem dostępowym WiFi.

Dołączenie urządzenia ruchomego do punktu dostępowego, czyli „asocjacja”, będzie prezentowane za pomocą ikony, której kształt powszechnie kojarzy się z dostępnością łączności radiowej. Przy większej liczbie punktów dostępowych mają one różne kolory ikon asocjacji. Liczba łuków ikony zależy od siły odbieranego sygnału ustalonej atrybutami `minPower` i `maxPower`, czyli w tym przykładzie na zakres (-80 dBm, -120 dBm). Przykład tej wizualizacji pokazany jest na rysunku 32.

Dotarcie sygnału transmisji WiFi do urządzenia również może być wizualizowane. Jest to możliwe w symulacji uruchomionej w trybie „Step/Run” (F5). Odpowiednia linia konfiguracji jest pokazana na listingu 98 a efekt jej działania na rysunku 33.



Rysunek 32. Wizualizacja asocjacji urządzenia Bob z punktem dostępowym Carol.

Rysunek 33. Urządzenie Alice odbierające słaby sygnał WiFi.

```
38 *.visualizer.*.mediumVisualizer.displaySignalArrivals = true
```

Listing 98. Konfiguracja wizualizacji odbioru sygnału WiFi.

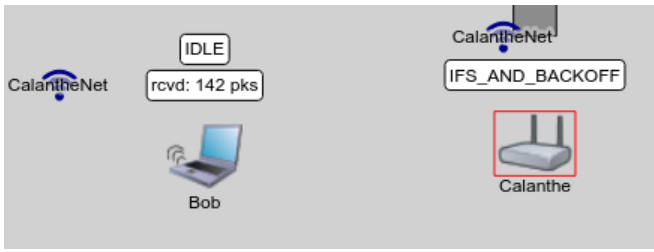
Ostatni fragment ogólnego kodu symulacji konfiguruje wyświetlanie „dymków" z informacjami obok ikonek urządzeń.

```
39 *.visualizer.*.infoVisualizer.displayInfos = true
40 *.visualizer.*.infoVisualizer.modules = "*.*.app[0] OR
       *.*.wlan[0].mac.dcf.channelAccess.contention"
41 *.visualizer.*.infoVisualizer.format = "%t"
```

Listing 99. Konfiguracja wizualizacji odbioru sygnału WiFi.

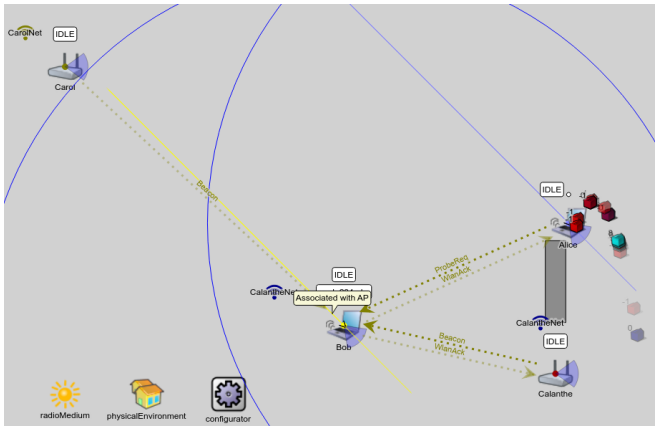
Widoczny wyżej kod włącza prezentowanie informacji w „dymkach", włącza monitorowanie modułu aplikacji czyli transmisji w warstwie sieciowej oraz aktywności warstwy łącza danych. Ostatnia linia oznacza, że teksty te będą prezentowane w prosty sposób tekstowy. Efekt tego fragmentu konfiguracji jest pokazany na rysunku 34

Na bazie powyższej konfiguracji zbudowane są trzy symulacje. Pierwsza z nich prezentuje mobilne urządzenia WiFi (Alice i Bob), które poruszają się ruchem liniowym między dwoma routerami (Calanthe i Carol). Obszary ich mobilności mają te same



Rysunek 34. Dynamiczna wizualizacja transmisji.

wymiary kwadratu o boku 150 metrów ale są od siebie odsunięte. Oba urządzenia rozpoczynają ruch z azymutem 225° i poruszają się z prędkością 1,4 m/s a więc poruszają się równolegle do siebie. W konfiguracji włączone są jeszcze wizualizacja kierunku i prędkości ich ruchu oraz pozostawiany ślad, tak jak widzieliśmy to we wcześniejszych symulacjach. Działanie tej symulacji w pełnej okazałości jest pokazane na rysunku 35.



Rysunek 35. Symulacja przełączenia między punktami dostępowymi (ang. *handover*).

Element, który wcześniej nie był w niniejszym podręczniku wykorzystywany to wizualizacja dostarczania i odbierania pakietów oraz ich „gubienia”, gdy z jakichś przyczyn nie mogły zostać dostarczone odbiorcy. Odzwierciedlają to kolorowe „pudełka” wypadające z ikony urządzenia. Konfiguracja tego elementu jest pokazana na listingu 100.

```
70 *.visualizer.*.packetDropVisualizer.displayPacketDrops = true
71 *.visualizer.*.packetDropVisualizer.nodeFilter = "Alice or Bob"
72 *.visualizer.*.packetDropVisualizer.labelFormat = "%r"
```

Listing 100. Konfiguracja wizualizacji odbioru sygnału WiFi.

Zadanie

- Zmień parametry konfigurujące mobilność urządzeń i sprawdź efekt wprowadzonych zmian.
- Zmień mobilność wybranych urządzeń korzystając z innych algorytmów ruchu, zgodnie dokumentacją pakietu INET.
- Zmieniaj moc transmisji i obserwuj wpływ tego parametru na nawiązywanie i zrywanie połączeń urządzeń ruchomych z punktami dostępowymi.
- Przygotuj eksperyment w którym dwa nieruchome urządzenia łączą się z punktem dostępowym umieszczonym za przesłoną i przeanalizuj zmianę liczby poprawnie dostarczanych pakietów w zależności od używanej mocy tranmitowanego sygnału.

Kolejna symulacja ma dość prostą konfigurację, dlatego najłatwiej ją omówić i zrozumieć w całości. Widzimy ją na listingu 101

```
79 *.Carol.numWlanInterfaces = 2
80 *.Carol.wlan[0].mgmt.ssid = "CarolNet2.4"
81 *.Carol.wlan[1].mgmt.ssid = "CarolNet5"
82 *.Carol.wlan[1].radio.bandName = "5 GHz"
83 *.Calanthe.numWlanInterfaces = 2
84 *.Calanthe.wlan[0].mgmt.ssid = "CalantheNet2.4"
85 *.Calanthe.wlan[1].mgmt.ssid = "CalantheNet5"
86 *.Calanthe.wlan[1].radio.bandName = "5 GHz"
87 *.Bob.wlan[0].radio.bandName = "5 GHz"
```

Listing 101. Konfiguracja sieci dwupasmowej.

W tej symulacji punkty dostępowe dostają do dyspozycji dwa interfejsy sieciowe: jeden w pasmie 2,4 GHz a drugi w pasmie 5 GHz. Spośród urządzeń ruchomych Alice nadal korzysta z pasma 2,4 GHz ale Bob działa w zakresie 5 GHz. Każdy z interfejsów sieciowych *de facto* stanowi odrębną sieć z własnym identyfikatorem SSID.

Ponieważ Alice i Bob nie współdzielą pasma to ich jedyną możliwością komunikacji jest użycie punktów dostępowych retransmitujących pakiety. W poprzedniej symulacji transmisja również odbywała się poprzez routery jednak tutaj będzie to jeszcze lepiej widoczne.

Trzeci wariant symulacji wprowadza bardziej realistyczną transmisję sygnałów radiowych wraz z możliwością analizy ich propagacji w czasie i przestrzeni. Realizm uwzględnia radiowy szum tła oraz nieidealny charakter dielektryka przeszkody. Ten fragment konfiguracji pokazany jest na listingu 102.

```
95 *.radioMedium.backgroundNoise.power = nan dBmW
96 *.radioMedium.backgroundNoise.powerSpectralDensity = -120dBmWpMHz
97 *.radioMedium.obstacleLoss.typename = "DielectricObstacleLoss"
```

Listing 102. Konfiguracja realistycznych parametrów środowiska.

Adekwatnie do realizmu otoczenia tak i charakterystyka warstwy fizycznej interfejsu radiowego jest nieco bogatsza. Widzimy ją na listingu 103.

```
99  *.*.wlan[*].radio.typename = "Ieee80211DimensionalRadio"
100 *.*.wlan[*].radio.antenna.typename = "IsotropicAntenna"
101 *.*.wlan[*].radio.transmitter.power = 1mW
102 *.*.wlan[*].radio.transmitter.frequencyGains = "\
103   left c-b*1.5 -40dB linear c-b -28dB linear \
104   c-b*0.5-1MHz -20dB linear c-b*0.5+1MHz \
105   0dB \
106   linear c+b*0.5-1MHz 0dB linear c+b*0.5+1MHz -20dB \
107   linear c+b -28dB linear c+b*1.5 -40dB right"
108 *.*.wlan[*].radio.channelNumber = 0
```

Listing 103. Konfiguracja warstwy fizycznej interfejsu radiowego.

Zastosowany tu model radia `Ieee80211DimensionalRadio` analizuje i pozwala na symulacje częstotliwości sygnałów radiowych rozchodzących się w czasie i przestrzeni. Nadal korzystamy z anteny izotropowej emitującej sygnał z mocą 1 mW. Atrybut `frequencyGains` zawiera szczegółowy opis charakterystyki częstotliwościowej transmisji WiFi, zgodny ze standardem IEEE. Widmo to jest opisane od lewej do prawej strony przy czym oznaczenia literowe to:

- c – „*center*”, częstotliwość środkowa kanału transmisyjnego w warstwie fizycznej,
- b – „*band*”, szerokość pasma podstawowego.

Zgodnie z tą konfiguracją radio jest skonfigurowane do pracy w kanale 1 WiFi – w INET 4.4.1 kanały są indeksowane od 0.

Pozostały fragment konfiguracji dotyczy ustawień wizualizacji i wiele z tych elementów było już wcześniej w użyciu i zostało omówione na kartach niniejszego podręcznika.

Wyjątkiem jest konfiguracja wizualizacji spektrum radiowego. Sygnały można przedstawić na płaszczyźnie częstotliwość–moc (analiza widmowa) lub na płaszczyźnie częstotliwość–czas (spektrogram). Można też włączyć obydwa typy wykresu jednak wówczas zabierają one znaczny obszar okna symulacji. Odpowiednie linie konfiguracji są pokazane niżej.

```
115 *.visualizer.*.mediumVisualizer.displaySpectrums = true
116 *.visualizer.*.mediumVisualizer.displaySpectrograms = false
```

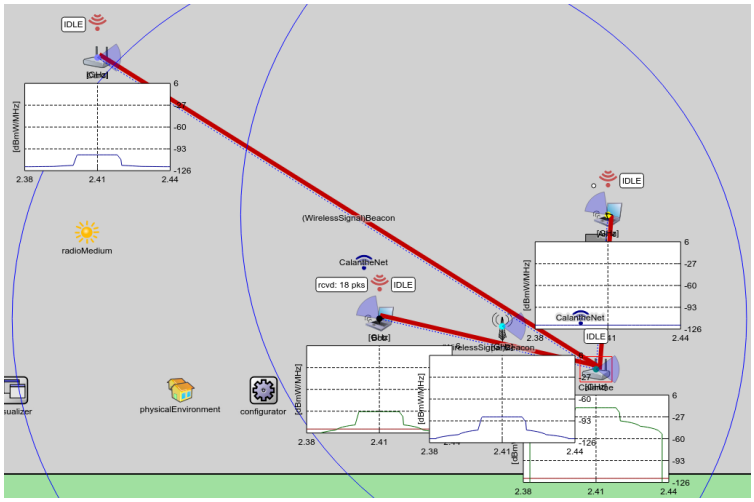
Listing 104. Włączenie charakterystyk częstotliwościowych, wyłączone spektrogramy.

Zakres osi mocy charakterystyki widmowej może być ustalany automatycznie ale często prowadzi to do ustawień nieadekwatnych do rzeczywiście występujących sygnałów. Można ten zakres ustanowić ręcznie, tak jak pokazane jest to na poniższym listingu.

```
118 *.visualizer.*.mediumVisualizer.autoPowerAxis = false
119 *.visualizer.*.mediumVisualizer.signalMinPowerDensity = -120dBmWpMHz
120 *.visualizer.*.mediumVisualizer.signalMaxPowerDensity = 0dBmWpMHz
```

Listing 105. Ustalenie zakresu mocy sygnałów analizowanych spektralnie.

Obraz tej symulacji pokazany jest na rysunku 36 a żeby ją uzyskać trzeba skorzystać z odpowiedniej konfiguracji w pliku NED. Konfiguracja ta jest widoczna na listingu 106.



Rysunek 36. Widok symulacji z użyciem realistycznego medium radiowego.

```
51     submodules:
52         visualizer: IntegratedVisualizer {
53             @display("p=40,260");
54         }
55         radioMedium: Ieee80211DimensionalRadioMedium {
56             @display("p=75,184");
```

```
57          }
58          probe: Probe {
59              @display("p=50,50");
60          }
```

Listing 106. Realistyczne radio i próbnik skonfigurowane w pliku NED.

Próbnik pozwala podejrzeć charakterystykę częstotliwości w danym miejscu i czasie, również podczas symulowanej propagacji sygnału. Można go przesuwać, przytrzymując lewy przycisk myszy przy naciśniętym przycisku Shift. To rozwiązanie jest wygodne przy wolno realizowanej lub całkowicie zatrzymanej symulacji.

Zadanie

- Używając próbnika przeanalizuj charakterystykę widmową sygnału w różnych punktach obszaru symulacji.
- Zmieniając konfigurację gęstości widmowej szumu tła powtórz analizę.
- Szczególnie uważnie przeanalizuj propagację sygnału przez przesłonę.
- Powtórz powyższe dla innych typów przesłony (np.: „wood").
- Opracuj charakterystykę zależności mocy maksymalnej sygnału w zależności od odległości od punktu nadającego ten sygnał.
- Opracuj charakterystykę zależności mocy maksymalnej sygnału w zależności od grubości przesłony oddzielającej próbnik od punktu nadającego sygnał.

5.3 Pytania sprawdzające

1. Jakie standardy sieci bezprzewodowych są obecnie w użyciu i jakie są kluczowe różnice między nimi?
2. Jakie są główne cechy kluczowania fazy i jaki jest związek tej modulacji z WiFi?
3. Jakie są główne cechy kwadraturowej modulacji amplitudy i jaki jest związek tej modulacji z WiFi?
4. Jakie są główne cechy, zasada działania i cel OFDM?
5. Jakie są główne cechy i zasady działania MIMO?
6. Jak środowisko fizyczne wpływa na transmisje sygnałów?

Klucz odpowiedzi

Wstęp

1. Patrz strony: 11 – 13, publikacja: [3]
2. Patrz strona: 13
3. Patrz strona: 13
4. Patrz strony: 13, 15, 17
5. Patrz strony: 15 – 18
6. Patrz strony: 13 – 19, publikacja: [7]

Wprowadzenie do OMNeT++

1. Patrz strona: 21, publikacja: [12]
2. Patrz strona: 21
3. Patrz strona: 45, publikacja: [13]
4. Patrz strona: 29
5. Patrz strony: 24, 25 – 26
6. Patrz strony: 26 – 28

Transmisja pakietowa

1. Patrz strony: 46 – 48, publikacja: [15]
2. Patrz strony: 48, publikacja: [18]
3. Patrz strony: 52, publikacja: [23]
4. Patrz strony: 51 – 52, 53, publikacje: [21, 24]
5. Patrz strona: 54 – 60, publikacja: [25]
6. Patrz strony: 66 – 69, publikacja: [25]

Sieci MANET

1. Patrz strony: 71 – 73, publikacja: [28]
2. Patrz strony: 72 – 73, publikacja: [28, 29]
3. Patrz strona: 87, publikacja: [25]
4. Patrz strony: 83 – 88, publikacja: [25]
5. Patrz strony: 81 – 83, publikacja: [25]
6. Patrz strony: 83 – 97, publikacja: [35]

Lokalne sieci bezprzewodowe

1. Patrz strona: 99, publikacja: [33]
2. Patrz strony: 101 – 103, publikacja: [35]
3. Patrz strony: 103 – 105, publikacja: [35]
4. Patrz strony: 105 – 106, publikacja: [36, 37]
5. Patrz strony: 106 – 107, publikacja: [38]
6. Patrz strony: 107 – 115, publikacja: [35]

Materiały multimedialne

Filmy prezentujące omówione w podręczniku symulacje, znajdują się pod adresem:

https://www.youtube.com/playlist?list=PLQGMGoY5x6p7vqB_H0DFq0w7ZoPKRkTuJ

alternatywnie:

https://tinyurl.com/yck32fja

Bibliografia

[1] *CMU SCS Coke Machine Home Page*, https://www.cs.cmu.edu/~coke/, [Online; accessed 13-July-2022].

[2] J. Romkey, „Toast of the IoT: The 1990 Interop Internet Toaster", *IEEE Consumer Electronics Magazine*, t. 6, nr. 1, s. 116–119, 2017. doi: 10.1109/MCE.2016.2614740.

[3] D. Evans, „The internet of things. how the next evolution of the internet is changing everything, whitepaper", *Cisco Internet Business Solutions Group (IBSG)*, 2011.

[4] A. Fuller, Z. Fan, C. Day i C. Barlow, „Digital Twin: Enabling Technologies, Challenges and Open Research", *IEEE Access*, t. 8, s. 108 952–108 971, 2020. doi: 10.1109/ACCESS.2020.2998358.

[5] K. Maney, „Meet Kevin Ashton, Father of the Internet of Things", *Newsweek Magazine*, 2015, [Online; accessed 03-July-2022].

[6] RAIN RFID, *Questions and Answers between the RAIN RFID Alliance and Kevin Ashton*, https://rainrfid.org/wp-content/uploads/2015/12/Kevin-Ashton.pdf, [Online; accessed 03-July-2022], paź. 2015.

[7] IOTIFY, *Testing the Internet of Things*, https://iotify.io/ebook/, [Online; accessed 05-July-2022], wrz. 2019.

[8] E. Black, S. Gamboa i R. Rouil, „NetSimulyzer: A 3D Network Simulation Analyzer for Ns-3", w *Proceedings of the Workshop on Ns-3*, ser. WNS3 '21, Virtual Event, USA: Association for Computing Machinery, 2021, s. 65–72, isbn: 9781450390347. doi: 10.1145/3460797.3460806. adr.: https://doi.org/10.1145/3460797.3460806.

[9] A. Velinov i A. Mileva, „Running and Testing Applications for Contiki OS Using Cooja Simulator", czer. 2016, s. 279.

[10] G. Oikonomou, S. Duquennoy, A. Elsts, J. Eriksson, Y. Tanaka i N. Tsiftes, „The Contiki-NG open source operating system for next generation IoT devices", *SoftwareX*, t. 18, s. 101 089, 2022, issn: 2352-7110. doi: https://doi.org/10.1016/j.softx.2022.101089.

[11] „IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Framework and Rules”, *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, s. 1–38, 2010. doi: 10.1109/IEEESTD.2010.5553440.

[12] A. Varga i O. Ltd., *OMNeT++ User Guide Version 6.0*, [Online; accessed 08-July-2022], 2021.

[13] *What Is INET Framework?*, [Online; accessed 13-July-2022].

[14] J. Postel, „Internet Protocol”, RFC Editor, STD 5, wrz. 1981, http://www.rfc-editor.org/rfc/rfc791.txt. adr.: http://www.rfc-editor.org/rfc/rfc791.txt.

[15] W. R. Stevens i K. Fall, *TCPIP Illustrated: The Protocols v. 1*, 2nd. USA: Addison-Wesley Publishing Company, 2009, isbn: 0321336313.

[16] S. Deering i R. Hinden, „Internet Protocol, Version 6 (IPv6) Specification”, RFC Editor, STD 86, lip. 2017.

[17] N. Kushalnagar, G. Montenegro i C. Schumacher, „IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals”, RFC Editor, RFC 4919, sierp. 2007, http://www.rfc-editor.org/rfc/rfc4919.txt. adr.: http://www.rfc-editor.org/rfc/rfc4919.txt.

[18] J. Postel, *User Datagram Protocol*, RFC 768, sierp. 1980. doi: 10.17487/RFC0768. adr.: https://www.rfc-editor.org/info/rfc768.

[19] *Transmission Control Protocol*, RFC 793, wrz. 1981. doi: 10.17487/RFC0793. adr.: https://www.rfc-editor.org/info/rfc793.

[20] M. Amundsen, S. Ruby i L. Richardson, *RESTful Web APIs*. O'Reilly Media, Inc., 2013, isbn: 9781449358068.

[21] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures”, prac. dokt., University of California, Irvine, 2000.

[22] „Information technology — Message Queuing Telemetry Transport (MQTT) v3.1.1”, International Organization for Standardization, Geneva, CH, Standard, czer. 2016.

[23] A. Banks, E. Briggs, K. Borgendale i R. Gupta, red., *MQTT Version 5.0*, Standard, mar. 2019.

[24] Z. Shelby, K. Hartke i C. Bormann, *The Constrained Application Protocol (CoAP)*, RFC 7252, czer. 2014. doi: 10.17487/RFC7252. adr.: https://www.rfc-editor.org/info/rfc7252.

[25] *INET Framework Documentation*, [Online; accessed 2-August-2022].

[26] N. Kumar, Y. Ramdoss i Y. Orzach, *Network Analysis Using Wireshark 2 Cookbook: Practical Recipes to Analyze and Secure Your Network Using Wireshark 2*, 2nd. Birmingham: Packt Publishing, Limited, 2018, isbn: 978-1786461674.

[27] C. Sanders, *Practical Packet Analysis, 3E: Using Wireshark to Solve Real-World Network Problems*. No Starch Press, mar. 2017, isbn: 978-1593278021.

[28] J. Loo, J. L. Mauri i J. H. Ortiz, red., *Mobile Ad Hoc Networks: Current Status and Future Trends*, 1st. CRC Press, kw. 2016, isbn: 978-1439856505.

[29] C. Perkins, E. Belding-Royer i S. Das, *Ad hoc On-Demand Distance Vector (AODV) Routing*, Internet Requests for Comments, RFC, lip. 2003.

[30] M. S. Gast, *802.11 Wireless Networks: The Definitive Guide*, 2nd. O'Reilly Media, maj 2005, isbn: 978-0596100520.

[31] M. S. Gast, *802.11ac: A Survival Guide: Wi-Fi at Gigabit and Beyond*, 1st. O'Reilly Media, sierp. 2013, isbn: 978-1449343149.

[32] L. Ward, *802.11ac Technology Introduction*, `https://www.rohde-schwarz.com/de/file/1MA192_7e_80211ac_technology.pdf`, [Online; accessed 2022-11-20].

[33] L. Ward, *IEEE 802.11ax Technology Introduction*, `https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/premiumdownloads/premium_dl_brochures_and_datasheets/premium_dl_whitepaper/IEEE-802-11ax-Technology-Introduction_wp_3609-9470-52_v0100.pdf`, [Online; accessed 2022-11-25].

[34] „IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks–Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications", *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)*, s. 1–4379, 2021. doi: 10.1109/IEEESTD.2021.9363693.

[35] S. Haykin i M. Moher, *An Introduction to Analog and Digital Communications*, 2nd. John Wiley & Sons, Inc., list. 2008, isbn: 978-0-470-46087-0.

[36] H. Rohling, *OFDM Concepts for Future Communication Systems*, 1st, H. Rohling, red. Springer Berlin, Heidelberg, 2011, isbn: 978-3-642-17496-4.

[37] A. R. S. Bahai, B. R. Saltzberg i M. Ergen, *Multi-carrier digital communications: theory and applications of OFDM*, 2nd. New York : Sringer ScienceBusiness Media, 2004, isbn: 0-387-22575-7.

[38] G. Tsoulos, *MIMO system technology for wireless communications*, G. Tsoulos, red. CRC/Taylor & Francis Group, 2006, isbn: 978-0-8493-4190-8.