**ARKADIUSZ BOGACZ**

Wyższa Szkoła Bankowa w Poznaniu, Wydział Finansów i Bankowości
e-mail: bogacz.a.m@gmail.com

# Samokonfigurująca się sieć urządzeń IoT*

**Streszczenie.** Celem artykułu jest przeprowadzenie eksperymentu opracowania samokonfigurującej się sieci urządzeń internetu rzeczy. W pracy przeprowadzono analizę dostępnych rozwiązań chmury obliczeniowej, przedstawiono koncepcję bliźniaka cyfrowego oraz symulację sieci urządzeń IoT. Utworzono oraz opisano model bazujący na algorytmie uczenia maszynowego. Zademonstrowano wykorzystanie opracowanego modelu oraz potwierdzono tezę, że samokonfiguracja i budowanie topologii przez samą sieć jest możliwe dzięki wykorzystaniu rozwiązań chmurowych oraz algorytmów uczenia maszynowego.

**Słowa kluczowe:** uczenie maszynowe, internet rzeczy, chmura obliczeniowa, bliźniak cyfrowy

## 1. Wprowadzenie

Rynek internetu rzeczy rozrasta się w tempie wykładniczym. Co chwilę powstają nowe mikrokontrolery, mikrokomputery oraz płytki projektowe. Utworzono wiele interfejsów sieciowych i komunikacyjnych dla *Internet of Things* (IoT). Routing w sieciach informatycznych został w znacznej mierze zautomatyzowany, dzięki protokołowi DHCP bramka sieciowa sama nadaje adres IP podłączonym urządzeniom, natomiast konfiguracja urządzeń końcowych czy węzłów komunikacyjnych odbywa się w sposób manualny.

Urządzenia IoT powinny działać w trybie pracy ciągłej, a komunikacja odbywać się samoczynnie wewnątrz wyznaczonej sieci. Każde urządzenie, które istnieje w sieci lub jest do niej dodawane, należy skonfigurować. Odbywa się to przez wgranie na urządzenie gotowego rozwiązania lub plików konfiguracyjnych dla bardziej ogólnych systemów. Dlatego też niniejsza praca ma na celu udowodnienie tezy, że samokonfiguracja sieci urządzeń IoT jest możliwa. Do tego celu

* Artykuł został przygotowany na podstawie pracy magisterskiej autora pt. „Samokonfigurująca się sieć urządzeń IoT", napisanej pod kierunkiem dr. hab. Grzegorza Pawłowskiego, prof. WSB w Poznaniu.

należy przeanalizować dostępne rozwiązania, sprawdzić możliwość symulacji sieci oraz zadać pytanie, czy sieć sama w sobie byłaby w stanie ustalić swoją topologię i hierarchię urządzeń.

## 2. Chmura obliczeniowa jako platforma dla symulacji sieci

Obecnie na rynku IT bardzo popularna jest chmura obliczeniowa. Jest to rozwiązanie bazujące na przeniesieniu infrastruktury, platformy sprzętowej lub oprogramowania bezpośrednio do zewnętrznego dostawcy. Na rynku znajduje się wiele przedsiębiorstw dostarczających chmurę, a trzej najwięksi to Amazon z rozwiązaniem Amazon Web Services (AWS)[1], Microsoft z platformą Azure[2] oraz Google z Google Cloud Platform (GCP)[3]. Każda z wymienionych firm oferuje zbliżony zestaw usług – od prostego utrzymywania stron internetowych aż po skomplikowane rozwiązania z dziedziny uczenia maszynowego czy tworzenia symulacji i bliźniaka cyfrowego.

Aby określić, czy dana platforma nadaje się do symulacji samokonfigurującej się sieci urządzeń IoT, należy prześledzić zestawy dostępnych usług z dziedziny internetu rzeczy. Pierwszy dostawca rozwiązań, który został przeanalizowany, to AWS. Na tej platformie udostępniono aż dziewięć usług z dziedziny IoT oraz własny system czasu rzeczywistego (RTOS)[4] instalowany na urządzeniach. Jednym z narzędzi na platformie AWS jest IoT Greengrass[5] – rozwiązanie pozwalające na połączenie się urządzenia bezpośrednio z chmurą Amazona. W tej usłudze logika biznesowa została przeniesiona na urządzenia dysponujące odpowiednio dużą mocą obliczeniową. Kolejną usługą jest AWS IoT Core[6] – rozwiązanie, za pomocą którego urządzenia łączą się z chmurą, a dane przez IoT Core są przekazywane dalej do innych usług, takich jak AWS Lambda[7], S3[8], Dynamo DB[9] czy CloudWatch[10]. IoT Core oferuje bogaty wachlarz protokołów komunikacyjnych, a wśród nich znajdują się MQTT, HTTPS, MQTT po WSS (*websockets secure*), czyli te najczęściej używane w komunikacji w sieciach IoT. Do monitoringu oraz

[1] https://aws.amazon.com/ [dostęp: 28.01.2021].
[2] https://azure.microsoft.com/pl-pl/ [dostęp: 28.01.2021].
[3] https://cloud.google.com/ [dostęp: 28.01.2021].
[4] https://aws.amazon.com/freertos/?c=i&sec=srv [dostęp: 28.01.2021].
[5] https://aws.amazon.com/greengrass/ [dostęp: 28.01.2021].
[6] https://aws.amazon.com/iot-core/?c=i&sec=srv [dostęp: 28.01.2021].
[7] https://aws.amazon.com/lambda/ [dostęp: 28.01.2021].
[8] https://aws.amazon.com/s3/ [dostęp: 28.01.2021].
[9] https://aws.amazon.com/dynamodb/ [dostęp: 28.01.2021].
[10] https://aws.amazon.com/cloudwatch/ [dostęp: 28.01.2021].

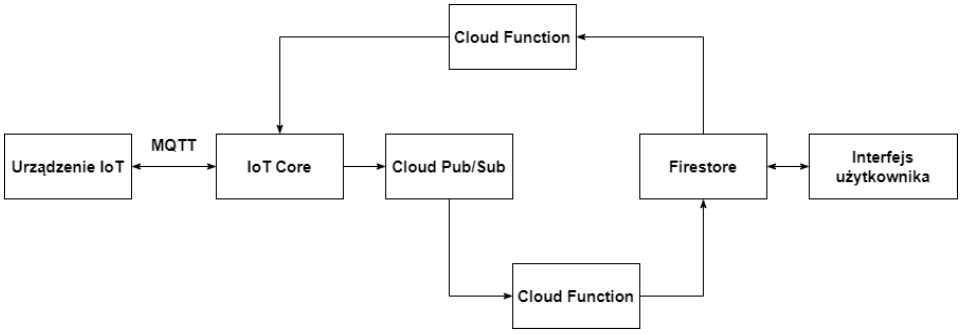
ochrony sieci IoT służą usługi AWS IoT Device Management[11] oraz IoT Device Defender[12]. AWS IoT Analytics[13] wykorzystuje się natomiast do analizy dużych wolumenów danych, upraszczając przy tym cały proces oraz automatyzując kolejne kroki wymagane przy analizie. Dla rozwiązań przemysłowych przewidziane jest narzędzie IoT SiteWise[14], które pozwala na optymalizację pod kątem metryk potrzebnych w przemyśle do zapobiegania problemom produkcyjnym. Do obsługi oraz zarządzania zdarzeniami służy usługa IoT Events[15], gdzie dzięki prostemu interfejsowi oraz instrukcjom sterującym typu „if-then-else" można sterować przepływem danych. Dodatkowo to rozwiązanie pozwala na połączenie z AWS IoT Core oraz AWS IoT Analytics. W celach wizualizacji AWS dostarcza IoT Things Graph[16], z kolei najważniejszą usługą, która pozwala na stworzenie symulacji samokonfigurującej się sieci IoT, jest AWS Sumerian[17] – narzędzie, w którym można zbudować środowisko symulacyjne oraz bliźniaka cyfrowego dla danego rozwiązania.

Drugi dostawca usług chmurowych, który był brany pod uwagę do przeprowadzenia symulacji sieci, to Google. Zestaw usług oferowanych na platformie GCP jest o wiele skromniejszy niż u konkurencji. Jednakże jako trzeci największy dostawca chmury na rynku Google oferuje zestaw narzędzi odpowiedni do symulacji sieci wraz z samokonfiguracją. W trakcie przeglądu usług zostało wybrane Google Cloud Platform IoT Core[18], które jest narzędziem stanowiącym trzon usług IoT. Rozwiązanie to dostarcza protokoły komunikacyjne HTTP i MQTT wraz z zabezpieczeniem dzięki TLS 1.2. W tę usługę wbudowano rejestr urządzeń oraz wyprowadzenie dla innych usług dostępnych w GCP.

W ramach testów platformy GCP została wykonana prosta symulacja sieci, w początkowej fazie bez użycia algorytmów uczenia maszynowego. Aby można było utworzyć symulację sieci, trzeba było uwzględnić komponenty chmury Google. Jako rdzeń zastosowano usługę IoT Core, która następnie została połączona z narzędziem Cloud Pub/Sub[19]. Bazą danych wykorzystaną do przechowywania stanu sieci została Google Firestore[20]. Jako elementy scalające poszczególne komponenty zostały napisane funkcje sieciowe w usłudze Cloud Functions[21]. Całość przedstawiono na rysunku 1, gdzie zasada działania symulacji

[11] https://aws.amazon.com/iot-device-management/?c=i&sec=srv [dostęp: 28.01.2021].
[12] https://aws.amazon.com/iot-device-defender/?c=i&sec=srv [dostęp: 28.01.2021].
[13] https://aws.amazon.com/iot-analytics/?c=i&sec=srv [dostęp: 28.01.2021].
[14] https://aws.amazon.com/iot-sitewise/?c=i&sec=srv [dostęp: 28.01.2021].
[15] https://aws.amazon.com/iot-events/?c=i&sec=srv [dostęp: 28.01.2021].
[16] https://aws.amazon.com/iot-things-graph/?c=i&sec=srv [dostęp: 28.01.2021].
[17] https://aws.amazon.com/sumerian/ [dostęp: 28.01.2021].
[18] https://cloud.google.com/iot-core [dostęp: 28.01.2021].
[19] https://cloud.google.com/pubsub/docs [dostęp: 28.01.2021].
[20] https://firebase.google.com/docs [dostęp: 28.01.2021].
[21] https://cloud.google.com/functions [dostęp: 28.01.2021].

sieci jest oparta na sprzężeniu zwrotnym. Symulowane urządzenie wysyła odpowiedni komunikat do IoT Core za pomocą protokołu MQTT, a tam urządzenie jest zapisane w rejestrach. Następnie IoT Core rozgłasza za pomocą Pub/Sub otrzymaną informację, którą przechwytuje utworzony komponent w usłudze Cloud Functions. Funkcja ta jest spoiwem pozwalającym na aktualizację stanu urządzenia zarówno po stronie bazy danych Firebase, jak i IoT Core. Dodatkowo została ona napisana w języku programowania JavaScript. Po zmianie aktualnego stanu w bazie danych, IoT Core jest o tym automatycznie informowane.




**Rysunek 1.** Schemat działania sieci IoT

Źródło: opracowanie własne.

Jak wynika z rysunku 1, GCP pozwala na utworzenie prostej symulacji sieci, jednak ubogi zestaw usług nie pozwala na odwzorowanie rzeczywistego działania sieci jako bliźniaka cyfrowego, jak również brakuje tutaj narzędzia do bardziej złożonej analizy danych.

Ostatnim dostawcą usług chmurowych branym pod uwagę w tworzeniu symulacji sieci jest Microsoft z usługą Azure. To drugi co do wielkości dostawca rozwiązań chmurowych, rozmiarem ustępuje tylko Amazonowi. Podobnie jak AWS, zestaw narzędzi dla IoT jest pokaźny. Microsoft jako jeden z nielicznych dostawców zapewnia usługi obliczeń brzegowych dla sieci IoT, a sama chmura pozwala na monitoring, komunikację oraz zarządzanie milionami urządzeń w czasie rzeczywistym. Podczas przeglądu usług potrzebnych do symulacji samokonfigurującej się sieci IoT zapoznano się z usługami dostępnymi na platformie Azure. Pierwsze narzędzie to Azure IoT Hub[22], w którym następuje wymiana wiadomości między urządzeniami IoT a poszczególnymi elementami chmury. Azure IoT Edge[23]

[22] https://docs.microsoft.com/pl-pl/azure/iot-hub/about-iot-hub [dostęp: 28.01.2021].

[23] https://docs.microsoft.com/pl-pl/azure/iot-edge/about-iot-edge?view=iotedge-2020-11 [dostęp: 28.01.2021].

to rozwiązanie, w którym logika biznesowa w formie gotowego kontenera jest umieszczana na urządzeniu, wymagającym przez to większej mocy obliczeniowej. Między urządzeniami a IoT Hub można uruchomić usługę IoT Device Provisioning Service[24], służącą jako router kierujący wiadomości do poszczególnych instancji IoT Hub. Automatyzację konfiguracji zapewnia usługa Azure IoT Plug and Play[25], w której tworzy się model napisany w języku DTDL[26]. Taki model może potem zostać wykorzystany do tworzenia bliźniaka cyfrowego w chmurze Azure. Jako że rozwiązania z dziedziny IoT w założeniu mają operować na milionach urządzeń w czasie rzeczywistym, a przesyłane dane są liczone w terabajtach, potrzebna jest ich odpowiednia wizualizacja oraz eksploracja. Do tego zadania najlepiej nadaje się narzędzie Azure Time Series Insights[27]. Wizualizacja danych na mapach jest możliwa dzięki Azure Maps[28]. Najbardziej kompleksowe rozwiązanie to Azure IoT, gdzie zostały zaimplementowane gotowe scenariusze użycia, symulacja urządzeń generujących dane telemetryczne lub bliźniak cyfrowy zakładu produkcyjnego wraz z wykorzystaniem kompleksowej analizy danych. Ostatnim przydatnym narzędziem z wachlarza chmury Azure jest Digital Twins[29] – usługa pozwalająca na utworzenie bliźniaka cyfrowego.

Po dokonaniu przeglądu oraz podstawowym przetestowaniu rozwiązań chmurowych dla IoT, to Azure najbardziej nadaje się do przeprowadzenia symulacji samokonfigurującej się sieci urządzeń IoT. Czynnikiem przeważającym jest usługa Azure Digital Twins, dająca najbardziej zaawansowane możliwości dokonania symulacji. Język DTDL pozwala na tworzenie modeli, które są potem możliwe do użycia w więcej niż jednej usłudze. Dodatkowym czynnikiem są najniższe koszty eksploatacji narzędzi IoT na chmurze Azure.

## 3. Cyfrowy bliźniak jako narzędzie w symulacji sieci

Bliźniak cyfrowy to cyfrowa replika obiektu świata rzeczywistego, istoty żywej lub istniejącego procesu. Dodatkowo jest on połączony ze swoim odpowiednikiem i zachowuje się dokładnie w taki sam sposób. Firmy udostępniające usługi chmurowe mają w gamie swoich produktów rozwiązania z dziedziny bliźniaka

[24] https://docs.microsoft.com/pl-pl/azure/iot-dps/about-iot-dps [dostęp: 28.01.2021].

[25] https://docs.microsoft.com/pl-pl/azure/iot-pnp/overview-iot-plug-and-play [dostęp: 28.01.2021].

[26] https://docs.microsoft.com/pl-pl/azure/digital-twins/concepts-models [dostęp: 28.01.2021].

[27] https://docs.microsoft.com/pl-pl/azure/time-series-insights/overview-what-is-tsi [dostęp: 28.01.2021].

[28] https://docs.microsoft.com/pl-pl/azure/azure-maps/about-azure-maps [dostęp: 28.01.2021].

[29] https://docs.microsoft.com/pl-pl/azure/digital-twins/overview [dostęp: 28.01.2021].

cyfrowego, jednak obecnie to raczej prosta symulacja, a nie pełne zbliźniaczenie. Problem ten najlepiej oddają kryteria, jakie musi spełnić bliźniak cyfrowy, aby optymalnie odwzorować istniejący świat:

– model wirtualny – bliźniak cyfrowy powinien mieć model istniejącego obiektu, na obecnym poziomie wiedzy jest to osiągalne i praktykowane,

– odwzorowanie ożywionej i nieożywionej materii – obecnie możliwe, natomiast są pewne ograniczenia,

– odwzorowanie fizyki – dostępne są narzędzia, silniki fizyczne typu Phisx[30], ale nadal istnieją pewne ograniczenia w odwzorowaniu rzeczywistej fizyki,

– połączenie – obecnie w dużej mierze niewykonalne, model obiektu otrzymuje dane w formie liczbowej lub zdarzeń,

– odwzorowanie rzeczywistego zachowania – olbrzymia liczba zmiennych nie pozwala na osiągnięcie tego kryterium odwzorowania, dla prostych obiektów maszyny stanu mogą zawierać olbrzymią liczbę stanów,

– symbioza świata rzeczywistego z wirtualnym – obecna technologia nie pozwala na tego typu symbiozę, dodatkowo prowadzone są badania nad takim rozwiązaniem[31].

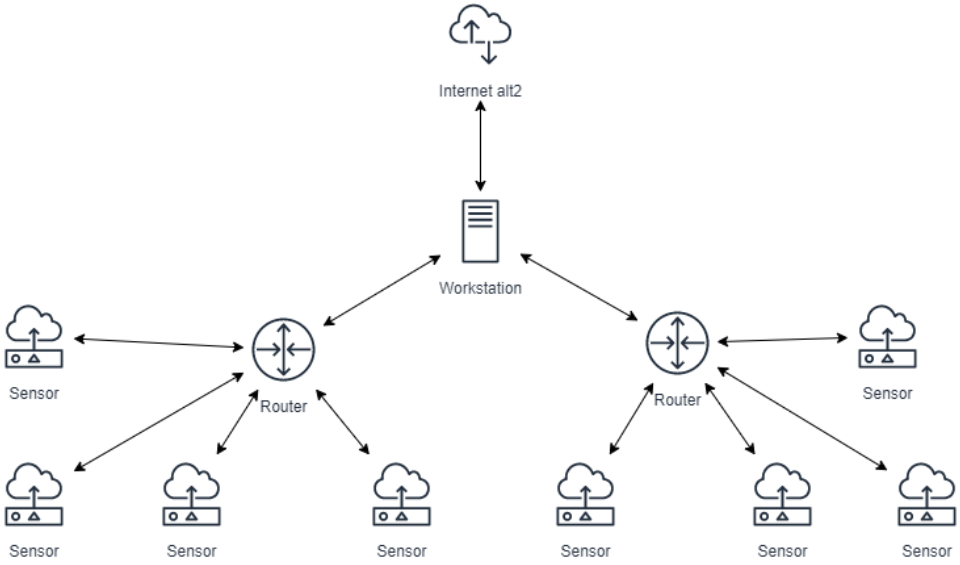
Przedstawione kryteria są wymagane, aby była mowa o pełnym bliźniaku cyfrowym. Technologia, przez ciągły rozwój i odkrycia, pozwala na coraz większe połączenie świata rzeczywistego i wirtualnego. Zagadnienie bliźniaka cyfrowego zostało poruszone w niniejszej pracy ze względu na symulację sieci urządzeń IoT. Mimo że AWS oraz Azure udostępniają podstawową możliwość utworzenia modelu sieci, to nadal jest to rozwiązanie dalekie od bliźniaka cyfrowego.

## 4. Uczenie maszynowe w samokonfiguracji

Aby sieć nazwać samokonfigurującą, należy utworzyć system, który pozwoli na nadawanie pewnej określonej charakterystyki, np. klasyfikacji w ramach klas, urządzeniom bez udziału człowieka. Pozwalałoby to na automatyczne przydzielanie roli danemu urządzeniu, które zostaje podłączone do istniejącej sieci. Do celów symulacji sieci została wybrana topologia gwiazdy. W sieci zostały umieszczone trzy typy urządzeń, stacje o dużej mocy obliczeniowej, urządzenia do routingu oraz sensory o najmniejszej mocy obliczeniowej. Przykładowa topologia została umieszczona na rysunku 2.

[30] https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Introduction.html [dostęp: 28.01.2021].

[31] http://2045.com/ [dostęp: 28.01.2021].




**Rysunek 2.** Przykładowa topologia sieci

Źródło: opracowanie własne.

Aktualnie używa się produkcyjnie wiele algorytmów uczenia maszynowego. Do najprostszych zadań używane są algorytmy regresji, naiwny klasyfikator Bayesa lub algorytm „k najbliższych sąsiadów". Z kolei bardziej skomplikowane problemy, jak rozpoznawanie twarzy czy obiektów na obrazie, wymagają skomplikowanych algorytmów używających sieci neuronowych. Uczenie maszynowe dzieli się na trzy podstawowe rodzaje: nadzorowane, nienadzorowane oraz uczenie poprzez wzmacnianie. W pierwszym dane są wcześniej opracowane przez człowieka. Do algorytmów wprowadza się obiekty, na podstawie których algorytmy się uczą, oraz wymaganą odpowiedź, do której się dąży. Drugie nie wymaga udziału człowieka. Do metod stosowanych w tym przypadku należą analiza składowych głównych oraz analiza skupień. Trzecim rodzajem uczenia maszynowego jest uczenie poprzez wzmacnianie. W tym przypadku nie są obecne dane uczące, ale środowisko, którego model będzie się uczył poprzez system kar i nagród.

Do przeprowadzenia eksperymentu samokonfiguracji sieci został wybrany algorytm z dziedziny uczenia nadzorowanego. A dokładniej drzewo decyzyjne, które jest wszechstronnym algorytmem używanym zarówno do zadań z regresji, jak i klasyfikacji.

W eksperymencie przeprowadzonym w celu udowodnienia tezy, że samokonfiguracja sieci jest możliwa, zostały wykorzystane narzędzia programistyczne

Anaconda[32] i Jupyter Notebook[33]. Są to narzędzia służące do kontroli środowiska wirtualnego dla języka programowania Python[34], instalacji i zarządzania pakietami oraz do wizualizacji danych. Biblioteki potrzebne do obróbki danych, a następnie do nauczenia algorytmu to numpy[35], pandas[36], scikit-learn[37] oraz kilka bibliotek pomocniczych.

Aby nauczyć algorytm, potrzebne są dane uczące. W tym celu zostały wykorzystane informacje o płytkach deweloperskich, na których rozwijane są prototypy urządzeń. Dane te zostały zebrane ze stron producentów oraz sprzedawców rozwiązań, zapisane do pliku *comma-separated values* (CSV), a do projektu zaimportowane z wykorzystaniem biblioteki pandas. Rezultat zamieszczono na rysunku 3.

| | name | architecture | cores | cpu_speed | cpu_type | ram | memory | type | wifi | bluetooth | device_type |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | ondroid n2+ | 64bit | 6 | 2.4GHz | RISC | 2048MB | 8MB | microcomputer | False | False | router |
| 10 | arduino micro | 8bit | 1 | 16MHz | RISC | 2.5kB | 32kB | microcontroller | False | False | sensor |
| 19 | beaglebone black | 64bit | 2 | 1.0GHz | RISC | 512MB | none | microcomputer | False | False | router |
| 20 | beaglebone white | 32bit | 1 | 1.5GHz | RISC | 1024MB | none | microcomputer | True | True | sensor |
| 15 | ondroid mc1 | 64bit | 8 | 2.0GHz | RISC | 2048MB | none | microcomputer | False | False | router |
| 2 | raspberry pi 4 8gb | 64bit | 4 | 1.5GHz | RISC | 8192MB | none | microcomputer | True | True | workstation |
| 3 | raspberry pi 3B+ | 64bit | 4 | 1.4GHz | RISC | 1024MB | none | microcomputer | True | True | router |
| 6 | lattepanda delta | 64bit | 4 | 2.4GHz | CISC | 4096MB | none | microcomputer | True | False | workstation |

**Rysunek 3.** Dane zaimportowane z pliku CSV

Źródło: opracowanie własne na podstawie not katalogowych producentów.

Aby przeprowadzić proces uczenia maszynowego, zaimportowane dane poddano obróbce. W tym celu każdej kolumnie obiektu **DataFrame** został nadany typ danych. Kolumny **architecture**, **cpu_speed**, **cpu_type**, **ram**, **memory** oraz **type** otrzymały typ kategoryczny, natomiast kolumna **name** otrzymały typ *string* reprezentujący łańcuch znaków, a **cores** typ stałopozycyjny *int*. Po nadaniu odpowiednich typów poszczególnym kolumnom dane zostały rozdzielone na zbiory kategoryczne i liczbowe, a te przypisane do odpowiednich preprocesorów z biblioteki scikit-learn. Dalsza część uczenia algorytmu drzewa decyzyjnego wymagała utworzenia macierzy parametrycznej dla klasyfikatora, z których w trakcie pracy klasyfikator dobiera odpowiednie parametry względem drzewa. Tutaj należy

[32] https://www.anaconda.com/products/individual [dostęp: 28.01.2021].
[33] https://jupyter.org/ [dostęp: 28.01.2021].
[34] https://www.python.org/ [dostęp: 28.01.2021].
[35] https://numpy.org/ [dostęp: 28.01.2021].
[36] https://pandas.pydata.org/ [dostęp: 28.01.2021].
[37] https://scikit-learn.org/stable/ [dostęp: 28.01.2021].

wspomnieć, że dla drzew decyzyjnych jednym z parametrów jest współczynnik zanieczyszczenia zbioru. Miara zanieczyszczenia określa jednorodność zbioru, a mierzy się to, używając albo wskaźnika Giniego, albo współczynnika entropii. Współczynnik Giniego jest określony następującym równaniem:

$$G_i = 1 - \sum_{k=1}^{n} P_{i,k}^{\ 2}$$

gdzie:
$P_{i,k}$ – współczynnik występowania klas $k$ wśród próbek uczących w $i$-tym węźle.

Natomiast współczynnik entropii określamy równaniem:

$$H_i = -\sum_{\substack{k=1 \\ P_{i,k} \neq 0}}^{n} P_{i,k} \log\left(P_{i,k}\right)$$

gdzie:
$P_{i,k}$ – współczynnik występowania klas $k$ wśród próbek uczących w $i$-tym węźle.

W tworzeniu klasyfikatora wykorzystano rekursywną eliminację cech wraz z walidacją krzyżową (RFECV), a następnie rozpoczęto proces uczenia modelu. Nauczony model osiągnął wynik 0,791(6), gdzie wartość 1 oznaczałaby idealne dopasowanie do danych uczących. Po tym wyniku można wstępnie założyć, że model nie został przeuczony. W otrzymanym modelu zostały wybrane 34 cechy otrzymane z danych uczących, które wraz z wagami umieszczono w tabeli 1.

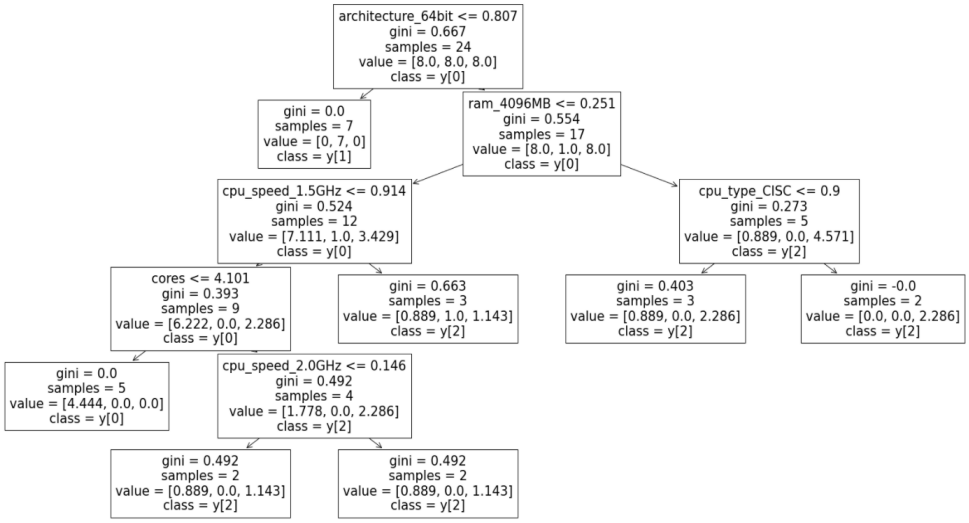
**Tabela 1.** Wybrane cechy nauczonego modelu wraz z wagami

| Cecha | Waga |
|---|---|
| architecture_64bit | 6,151751 |
| ram_4096MB | 1,747157 |
| cores | 1,254288 |
| cpu_speed_1.5GHz | 6,522367 |
| cpu_type_CISC | 1,945670 |

Źródło: opracowanie własne.

Pozostałe z 34 cech otrzymały wagę 0, a ostatnia, najmniej ważna cecha, **cpu_speed_2.0GHz** –1,036668e-16. Jako wynik uczenia modelu otrzymano drzewo decyzyjne, które jako wskaźnik zanieczyszczenia używa współczynnika

Giniego, maksymalną głębokość ma ustaloną na 7 węzłów przy minimalnej liczbie dwóch liści. Nauczony model przedstawiony został na rysunku 4.

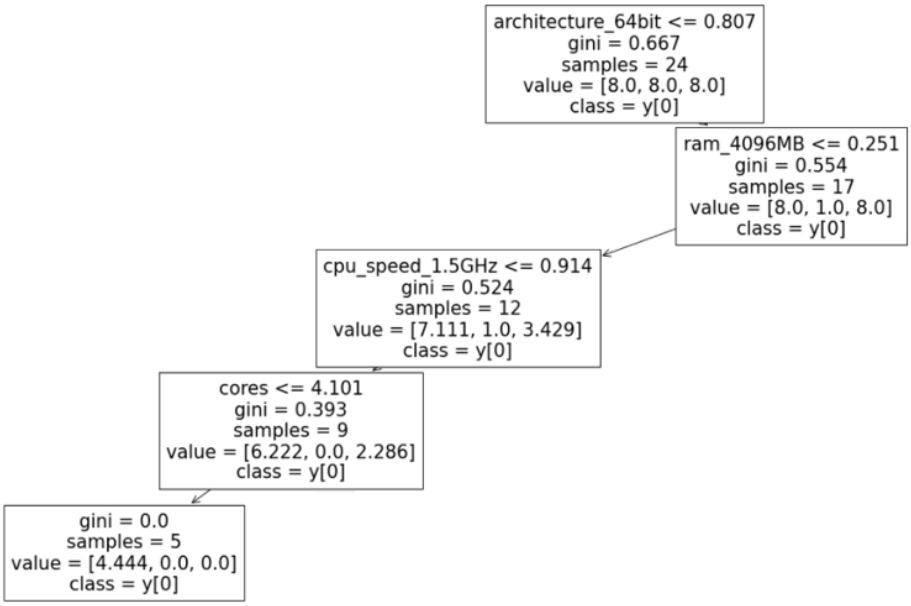



**Rysunek 4.** Wizualizacja nauczonego drzewa decyzyjnego

Źródło: opracowanie własne.

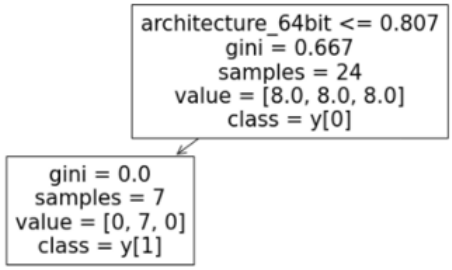
Po nauczeniu sprawdzono poprawność działania drzewa. Do tego celu wylosowano jedno urządzenie ze zbioru uczącego, a następnie użyto metody decision_path do określenia drogi klasyfikacji. Urządzenie wylosowane to płytka LattePanda mająca cztery rdzenie procesora w architekturze 64 bit typu CISC o szybkości 1,8 GHz, 2048 MB pamięci RAM i 32 GB pamięci flash. Po sklasyfikowaniu urządzenia przez drzewo otrzymano następujące kroki klasyfikacji wraz z wagami: dla cechy **architecture_64bit** urządzenie otrzymało wynik 0,80661322, dla **ram_4096MB** wartość cechy wyniosła 0,25064558. Cecha **cpu_speed_1.5GHz** otrzymała wagę 0,91434287, a cecha **cores** – 4,10107009. Ostatnia sprawdzana cecha **type_microcontroller** z wynikiem –2,0 zakończyła proces klasyfikacji. LattePanda została sklasyfikowana jako y[0], co oznacza, że w sieci pełni funkcję routera. Przebieg dopasowania klasy do płytki został przedstawiony na rysunku 5.

Sprawdzono również działanie drzewa dla innego typu urządzenia. Wylosowane zostało Arduino mega2560 z jednym rdzeniem o szybkości 16 MHz w architekturze 8-bitowej, 8 kB pamięci RAM oraz z 256 kB pamięci flash. Wyniki dla tej płytki to **architecture_64bit** mniejszy niż 0,80661322 oraz **type_microcontroller** z wynikiem –2,0. Idąc zgodnie ze ścieżką decyzyjną, Arduino zostało sklasyfikowane jako czujnik lub urządzenie brzegowe (rys. 6).




**Rysunek 5.** Wybór klasy dla LattePanda

Źródło: opracowanie własne.




**Rysunek 6.** Wybór klasy dla Arduino

Źródło: opracowanie własne.

## 5. Podsumowanie

Eksperyment przeprowadzony w celu uzyskania samokonfigurującej się sieci urządzeń IoT wykazał, że tworzenie tego typu rozwiązań z wykorzystaniem aktualnie dostępnych technologii jest możliwe. W tym celu przeprowadzono

dokładną analizę dostępnych rozwiązań z dziedziny chmury obliczeniowej dla urządzeń internetu rzeczy. Dostępne rozwiązania zapewniają kompleksową obsługę urządzeń oraz obliczeń zarówno brzegowych, jak i bezpośrednio w chmurze. Komunikacja odbywa się w sposób bezpieczny, dane mogą być analizowane przez różne dostępne narzędzia, a całość działa w czasie rzeczywistym.

Chociaż sam bliźniak cyfrowy nie jest jeszcze w pełni osiągalny z wykorzystaniem dostępnej technologii, istnieje możliwość uzyskania pewnego poziomu cyfryzacji rozwiązań. Usługa Azure Digital Twins pozwala na utworzenie kompleksowego modelu z wykorzystaniem języka DTDL, a samo środowisko pozwala już w pewnym stopniu uzyskać bliźniaka cyfrowego na podstawowym poziomie.

Uczenie maszynowe to potężne narzędzie do tworzenia holistycznych rozwiązań. Dostępne obecnie algorytmy, począwszy od regresji czy klasyfikacji, a kończąc na skomplikowanych sieciach neuronowych, pozwalają na budowanie rozwiązań, które same dostosowują się do zmiennych warunków. Wybrany algorytm do eksperymentu spełnił swoje zadanie i potwierdził tezę, że samokonfiguracja przez ustalenie topologii sieci jest możliwa.

Wykorzystanie w warunkach rzeczywistych nauczonego modelu powinno zostać poddane dalszym badaniom. Należy zbudować w pełni działające środowisko symulacyjne z wykorzystaniem dostępnych narzędzi, aby odwzorować warunki rzeczywiste. Przy wykorzystaniu bliźniaka cyfrowego i modeli istniejących urządzeń IoT, warto byłoby utworzyć sieć oraz za pomocą klasyfikatora ustalić jej topologię.

## Literatura

Athreya A., DeBruhl B., Tague P., 2013, *Designing for Self-Configuration and Self-Adaptation in the Internet of Things*, Carnegie Mellon University.
Boschetti A., Massaron L., 2017, *Python. Podstawy nauki o danych. Wydanie II*, Gliwice: Wyd. Helion.
Cormen T., Leiserson Ch., Rivest R., Clifford S., 2020, *Wprowadzenie do algorytmów*, Warszawa: WN PWN.
Edelman J., Lowe S., Oswalt M., 2019, *Programowalność i automatyzacja sieci*, Gliwice: Wyd. Helion.
Fall K., Stevens W., 2013, *TCP/IP od środka. Protokoły*, Gliwice: Wyd. Helion.
Géron A., 2018, *Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow*, Gliwice: Wyd. Helion.
http://2045.com/ [dostęp: 28.01.2021].
https://aws.amazon.com/ [dostęp: 28.01.2021].
https://aws.amazon.com/cloudwatch/ [dostęp: 28.01.2021].
https://aws.amazon.com/dynamodb/ [dostęp: 28.01.2021].
https://aws.amazon.com/freertos/?c=i&sec=srv [dostęp: 28.01.2021].
https://aws.amazon.com/greengrass/ [dostęp: 28.01.2021].
https://aws.amazon.com/iot-analytics/?c=i&sec=srv [dostęp: 28.01.2021].
https://aws.amazon.com/iot-core/?c=i&sec=srv [dostęp: 28.01.2021].
https://aws.amazon.com/iot-device-defender/?c=i&sec=srv [dostęp: 28.01.2021].
https://aws.amazon.com/iot-device-management/?c=i&sec=srv [dostęp: 28.01.2021].
https://aws.amazon.com/iot-events/?c=i&sec=srv [dostęp: 28.01.2021].

https://aws.amazon.com/iot-sitewise/?c=i&sec=srv [dostęp: 28.01.2021].
https://aws.amazon.com/iot-things-graph/?c=i&sec=srv [dostęp: 28.01.2021].
https://aws.amazon.com/lambda/ [dostęp: 28.01.2021].
https://aws.amazon.com/s3/ [dostęp: 28.01.2021].
https://aws.amazon.com/sumerian/ [dostęp: 28.01.2021].
https://azure.microsoft.com/pl-pl/ [dostęp: 28.01.2021].
https://cloud.google.com/ [dostęp: 28.01.2021].
https://cloud.google.com/functions [dostęp: 28.01.2021].
https://cloud.google.com/iot-core [dostęp: 28.01.2021].
https://cloud.google.com/pubsub/docs [dostęp: 28.01.2021].
https://docs.microsoft.com/pl-pl/azure/azure-maps/about-azure-maps [dostęp: 28.01.2021].
https://docs.microsoft.com/pl-pl/azure/digital-twins/concepts-models [dostęp: 28.01.2021].
https://docs.microsoft.com/pl-pl/azure/digital-twins/overview [dostęp: 28.01.2021].
https://docs.microsoft.com/pl-pl/azure/iot-dps/about-iot-dps [dostęp: 28.01.2021].
https://docs.microsoft.com/pl-pl/azure/iot-edge/about-iot-edge?view=iotedge-2020-11 [dostęp: 28.01.2021].
https://docs.microsoft.com/pl-pl/azure/iot-hub/about-iot-hub [dostęp: 28.01.2021].
https://docs.microsoft.com/pl-pl/azure/iot-pnp/overview-iot-plug-and-play [dostęp: 28.01.2021].
https://docs.microsoft.com/pl-pl/azure/time-series-insights/overview-what-is-tsi [dostęp: 28.01.2021].
https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Introduction.html [dostęp: 28.01.2021].
https://firebase.google.com/docs [dostęp: 28.01.2021].
https://jupyter.org/ [dostęp: 28.01.2021].
https://numpy.org/ [dostęp: 28.01.2021].
https://pandas.pydata.org/ [dostęp: 28.01.2021].
https://scikit-learn.org/stable/ [dostęp: 28.01.2021].
https://www.anaconda.com/products/individual [dostęp: 28.01.2021].
https://www.python.org/ [dostęp: 28.01.2021].
McIlwraith D., Marmanis H., Babenko D., 2017, *Inteligentna sieć. Algorytmy przyszłości*, Gliwice: Wyd. Helion.

## A self-configuring IoT network

**Abstract.** This article describes steps in the process of creating a self-configuring network of IoT devices. The author analyses available cloud solutions and presents the idea of virtual simulations using a digital twin. For the purpose of the actual experiment, a machine learning model was created and described, which was then used to demonstrate that it was possible to use cloud solutions and machine learning model in order to create a self-configuring IoT network.

**Keywords:** machine learning, Internet of things, cloud, digital twin