WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI

POLITECHNIKA POZNAŃSKA

Praca dyplomowa - inżynierska

Protokół komunikacyjny do wymiany danych w grze sieciowej - przykład edukacyjny

Jan Biały, 144334
Piotr Stawski, 144336

Promotor
prof. dr hab. inż. Grzegorz Danilewicz

POZNAŃ 2023

Spis Treści

1. Wstęp

1.1. Kontekst

1.2. Cel Pracy

1.3. Skrót rozdziałów

2. Protokoły komunikacyjne

2.1. Modele Warstwowe

Pierwszym i zarazem najistotniejszym efektem prac normalizacyjnych w systemach otwartych jest wypracowanie jednolitego modelu sieciowego, przetwarzającego dane w rozproszonych systemach sieciowych od poziomu procesu aplikacji do przesłania danych przez linie transmisyjną [1].

2.2. Model ISO OSI

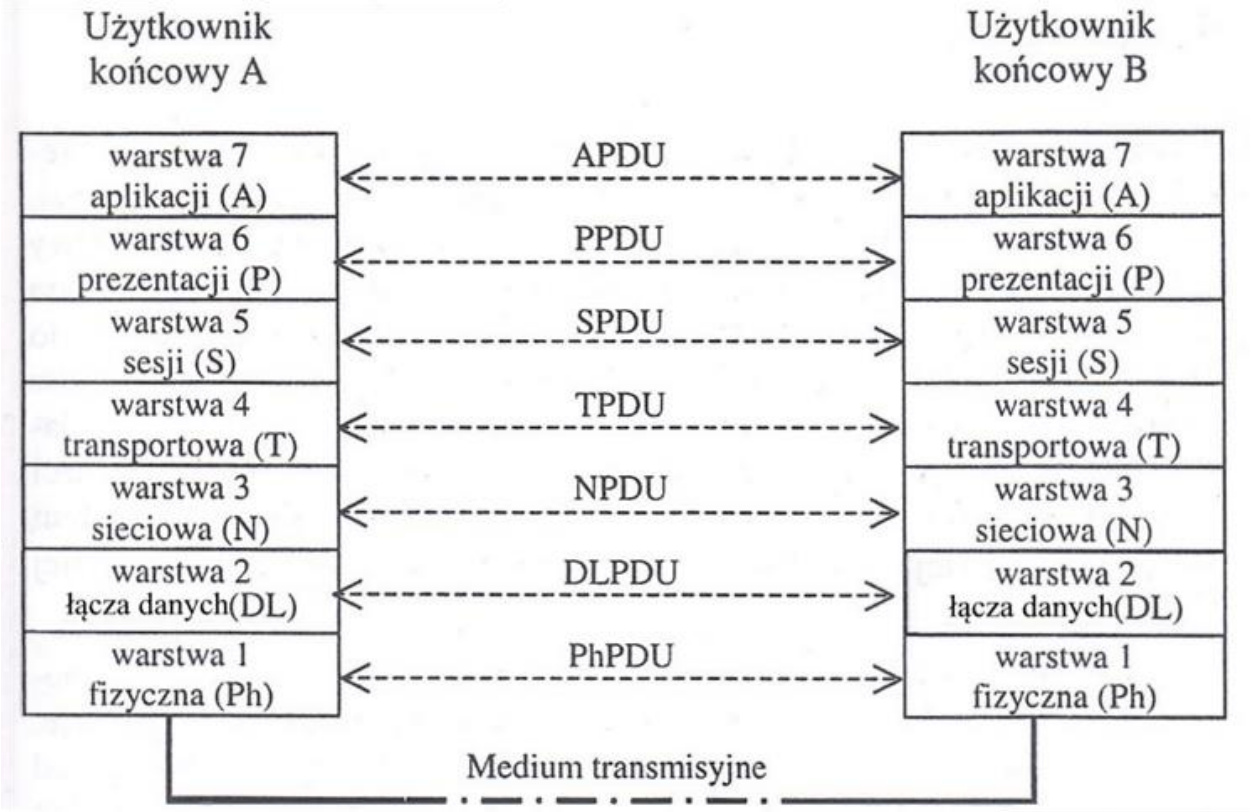
Jednym z modeli jest model OSI (pełna nazwa **ISO OSI RM** ang. *International Organization for Standardization Open Systems Interconnection Reference Model*), zwany **modelem odniesienia współdziałania systemów otwartych ISO OSI**. Został on zaprezentowany w 1983 roku przez przedstawicieli największych firm komputerowych i telekomunikacyjnych. Jako międzynarodowy standard został przyjęty w 1984 roku. Model ten definiuje architekturę i ogólne zasady przetwarzania danych w sieci oraz w środowisku przetwarzania otwartego, to oznacza, że do takiego systemu może podłączyć się każdy kto spełnia ustandaryzowane ogólne zasady [1].

Głównym pojęciem, które przewija się w strukturze modelu OSI jest **warstwa (ang. layer).** Architektura Systemu OSI składa się z podsystemów tego samego poziomu. Jako podsystem rozumiemy hierarchiczny element systemu otwartego, który współpracuje jedynie z sąsiednim wyższym, bądź niższym poziomem hierarchii modelu sieciowego [1].

Model ISO OSI składa się z siedmiu warstw o dokładnie określonych funkcjach i interfejsach łączących poszczególne warstwy (Rysunek 2.1). Każda warstwa posiada swoją nazwę oraz numer

(od 1 do 7) liczoną od warstwy fizycznej do warstwy aplikacji. Warstwowość modelu pozwala na grupowanie poszczególnych funkcjonalności w moduły, co czyni strukturę sieciową przejrzystą i dobrze zdefiniowaną.

Każdy system otwarty podłączony do sieci, funkcjonuje w niej jako zbiór stacji. Stacja jest reprezentacją procesu (lub zestawu procesów) w obrębie systemu otwartego świadczącego usługę w sieci. Stacja jest to aktywny, programowy lub sprzętowy moduł, który może występować pod różnymi postaciami. Każda taka stacja może reagować na otrzymywane wiadomości od innych jednostek bądź sama wysyłać wiadomości do innych stacji. Każda warstwa modelu może być rozpatrywana jako zbiór dowolnej ilości stacji znajdujących się w jednym systemie bądź w wielu systemach rozproszonych w sieci. Stacje ulokowane w najwyższej warstwie modelują procesy aplikacyjne. Wszystko co znajduje się poniżej tej warstwy modeluje procesy odpowiadające za dostarczenie usług OSI, a najniższa warstwa modelu umożliwia podłączenie systemu otwartego do medium transmisyjnego [1].



Rysunek 2.1 Model odniesienia ISO OSI [1]

PDU (ang. Protocol Data Unit) – jednostka danych protokołu

Usługa

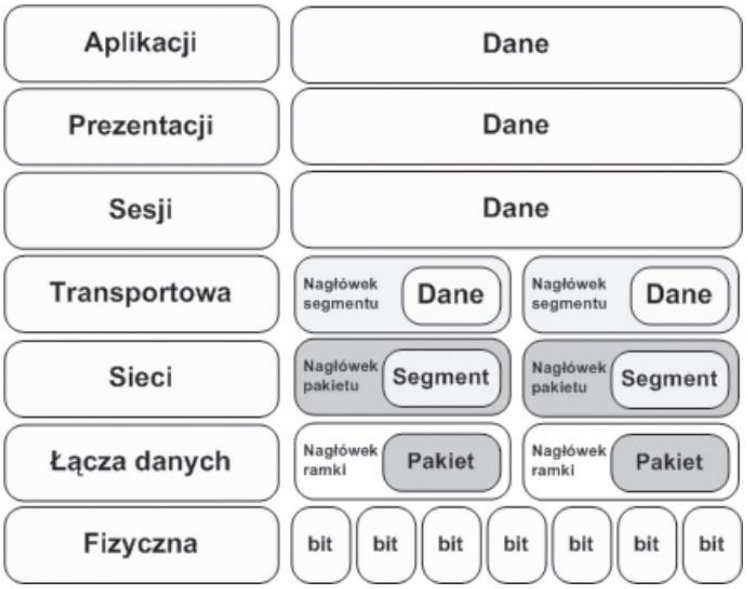
Usługą określamy zbiór funkcji realizowanych przez określoną warstwę. Każda warstwa posiada zbiór własnych funkcji, które świadczy warstwie wyższej. Każda warstwa świadczy usługi warstwie wyższej korzystając z usług dostarczonych jej przez warstwę bezpośrednio niższą, a co za tym idzie pośrednio świadczy usługi wszystkich warstw znajdujących się poniżej. Każda usługa jest odpowiedzialna za implementowanie własnej jasno określonej funkcjonalności. [1]

Protokół

Komunikacja między usługami tego samego poziomu, jeżeli obie usługi znajdują się w tym samym systemie otwartym (na przykład komputerze) jest stosunkowo prosta, ponieważ mogą bezpośrednio się ze sobą komunikować. Sytuacja zmienia się diametralnie, kiedy chcemy połączyć ze sobą usługi znajdujące się w różnych systemach otwartych niejednokrotnie oddalonych w stopniu znaczącym. Komunikacja między takimi systemami otwartymi odbywa się z wykorzystaniem ustanowionego zbioru reguł i formantów [1]. Taki zbiór reguł i formatów (semantyka, składnia i zależności czasowe) obowiązujących w danej warstwie nazywa się **protokołem**. Każda stacja, aby świadczyć usługi i implementować określone funkcjonalności musi działać wedle tych reguł. Stacje wymieniają informację między sobą za pomocą jednostek danych protokołu PDU (ang. *Protocol Data Unit),* dostosowanych do warstwy w których funkcjonują (Rysunek 2.1). PDU można podzielić na dwa zbiory informacji (Rysunek 2.2) [1]:

- Pierwszym z nich są *informacje sterujące protokołem PCI* (ang. *Protocol Control Information*), ten zbiór informacji podczas implementowania protokołu nazywamy nagłówkiem protokołu. Nagłówek ten jest tylko i wyłącznie interpretowany przez protokół, do którego należy, którego jest częścią. To on informuje jaka akcja powinna zostać podjęta w momencie, w którym otrzymaliśmy określone PDU.

- Drugim zbiorem informacji są *dane* które zostają przekazane do wyższej warstwy w celu obsłużenia usług znajdujących się wyżej w modelu systemu otwartego.



Rysunek 2.2 Proces kapsułkowania danych (ang. Encapsulation)

Warstwa aplikacji

Warstwa aplikacji (*ang. Application layer*) jest najwyższą warstwą modelu, a co za tym idzie jest zawsze najbliżej użytkownika. Dzieje się tak, ponieważ to właśnie w tej warstwie działają aplikacje, programy, z których interfejsu graficznego GUI (*ang. Graphical User Interface*) użytkownik ma bezpośredni wpływ na to co i na jakich warunkach zostanie przesłane przez zhierarchizowaną strukturę systemu otwartego. Przez programy działające w warstwie aplikacji użytkownik może wywołać akcje, które będą potrzebować różnorodnego obsłużenia na poziomie poszczególnych warstw modelu OSI. [2]

W warstwie aplikacji występuję wiele protokołów, jednakże często zdarza się tak, że protokół nie działa indywidualnie tylko potrzebuje do prawidłowego działania innych protokołów (*ang. Interaction between network protocols*), przykładem takiego protokołu jest protokół HTTP. [2]

Interakcja między protokołami często polega na wspomożeniu protokołu wyższej warstwy o funkcjonalność jakiej ten nie posiada. Przykładem takiego połączenia jest para protokołów HTTP i TCP. [2]

HTTP może być zaimplementowany zarówno po stronie klienckiej jak i serwerowej, ale HTTP nie posiada funkcjonalności dotarcia z jednego punktu do drugiego. Z pomocą przychodzą więc protokoły warstwy transportowej takie jak na przykład TCP, który zapewnia mechanizm

wymiany informacji docierający do miejsca docelowego. Dodatkowo pojawia się protokół adresowania IP z pomocą którego informacje w postaci PDU będą mogły być przenoszone między podsieciami i zarazem będą wiedziały z jakiego, i do jakiego miejsca chcą dotrzeć. W procesie poprawnej komunikacji interakcja między protokołami jest bardzo ważna do prawidłowego funkcjonowania systemu komunikacji. [2]

Do podstawowych protokołów warstwy aplikacji można zaliczyć takie protokoły jak [2]:

- HTTP (*ang. Hypertext Tranfer Protocol*) – dzięki współpracy z siecią WWW (ang. World Wide Web) umożliwia przeglądanie stron internetowych. Klient w warstwie aplikacji wyposażony w przeglądarkę internetową dzięki protokołowi HTTP może połączyć się z serwerem udostępniającym treść i przeglądać je na własnym komputerze. W praktyce proces ten wygląda następująco:
  - użytkownik wysyła żądanie PDU w nagłówku, którego znajduje się polecenie *GET* co oznacza dla serwera docelowego, że użytkownik zażądał danych znajdujących się na jego serwerze.
  - Następnie serwer odsyła użytkownikowi informację najczęściej z kodem 200, który oznacza, że żądanie użytkownika zostało obsłużone w sposób prawidłowy. Serwer przesyła także zawartość jaką użytkownik zechciał otrzymać wysyłając polecenie *GET*.
- DHCP (ang. Dynamic Host Configuration Protocol) – protokół służący do automatycznego przydzielania adresu IP urządzeniowi sieciowemu oraz innych adresów niezbędnych do prawidłowego funkcjonowania urządzenia w sieci. Takimi adresami są adres bramy domyślnej czy adres serwerów DNS (ang. Domain Name System).
- SMTP (*ang. Simple Mail Transport Protocol*) – protokół służący do przesyłania poczty elektronicznej, wyłącznie w postaci tekstowej, najczęściej współpracuje z protokołem POP3.
- POP3 (*ang. Post Office Protocol*) – odpowiada za odbiór poczty z serwera, współpracuje z protokołem TCP.
- SSL (*ang. Secure Sockets Layer*) – wprowadza funkcjonalność szyfrowania komunikacji między urządzeniami końcowymi.

- DNS (ang. Domain Name System – wprowadza funkcjonalność odwzorowywania nazw domenowych na adresy IP.
- FTP (ang. File Transfer Protocol) – wprowadza funkcjonalność przesyłania i odbierania plików z urządzeń znajdujących się w sieci.

Warstwa aplikacji jest więc, początkiem, ale też i końcem komunikacji między aplikacjami użytkowanych przez użytkowników końcowych.

Warstwa Prezentacji

Warstwa prezentacji (*ang. Presentation layer*) odpowiada za prezentowanie danych w sieci. Warstwa aplikacji przesyła dane warstwie prezentacji, która jest odpowiedzialna za przetłumaczenie tych danych do postaci zgodnej ze specyfikacją OSI RM. Następnie, jeżeli jest to wymagane szyfruje te dane lub kompresuje je. Celem warstwy prezentacji jest doprowadzenie do sytuacji, aby dane przekazywane w komunikacji miały wspólny format. [2]

Warstwa Sesji

Warstwa sesji (*ang. Session layer*) jest odpowiedzialna za zestawianie sesji między urządzeniami końcowymi oraz jej zarządzaniem, jeżeli sesja jest już nawiązana. Warstwa sesji wykorzystując odpowiednie protokoły zapewnia dwa rodzaje komunikacji: *połączeniową* i *bezpołączeniową*. W warstwie sesji dochodzi też do zabezpieczenia przed ponownym rozpoczęciem transmisji przez wprowadzenie do przesyłanych danych swego rodzaju punktów kontrolnych. Kiedy sesja zostaje zerwana, dane nie muszą być retransmitowane w całości, ale od miejsca ostatniego punktu kontrolnego. [2]

Warstwa transportowa

Warstwa transportowa (*ang. Transport layer*) należy do najważniejszych warstw modelu ISO OSI. To ona odpowiada za sposób w jaki dane są przesyłane. Dane wysyłane przez sieć nigdy nie są wysyłane w całości. Taka transmisja byłaby nieefektywna, ponieważ w przypadku zerwania

połączenia dane musiały by być transmitowane od początku. Dlatego też dane dzielone są na mniejsze kawałki. Proces ten nazywamy *segmentowaniem (ang. Segmenting).* Dzięki temu mechanizmowi znacznie poprawia się szybkość przesyłania danych w sieci, poprawia się wydajność całego systemu oraz zwiększa się bezpieczeństwo przesyłanych danych. Podczas stosowania mechanizmu segmentacji, pakiety powinny być po stronie odbiorczej odpowiednio odebrane a następnie przetworzone.

W sieciach komputerowych a zwłaszcza w sieci Internet, gdzie występuję bardzo duża złożoność połączeń, może się zdarzyć tak, że pakiety docierają do odbiorcy w kolejności innej niż zostały wysłane przez nadawcę. Zjawisko takie jest dość powszechne, dlatego też stworzono mechanizm porządkowania pakietów po stronie odbiorczej. Dlatego też każdy pakiet wysyłany przez nadawcę jest oznaczany numerem sekwencyjnym (*ang. Sequence number*). Warstwa transportowa jest odpowiedzialna ta to, aby każdy pakiet wysłany przez nadawcę został dostarczony do miejsca docelowego w odpowiedniej kolejności. W warstwie transportowej występują numery portów są to swego rodzaju adresy, na których pracują aplikacje. To właśnie w tej warstwie zostaje podjęta decyzja którego protokołu wybrać TCP (ang. *Transmission Control Protocol*) czy UDP (ang. *User Datagram Protocol*). [2]

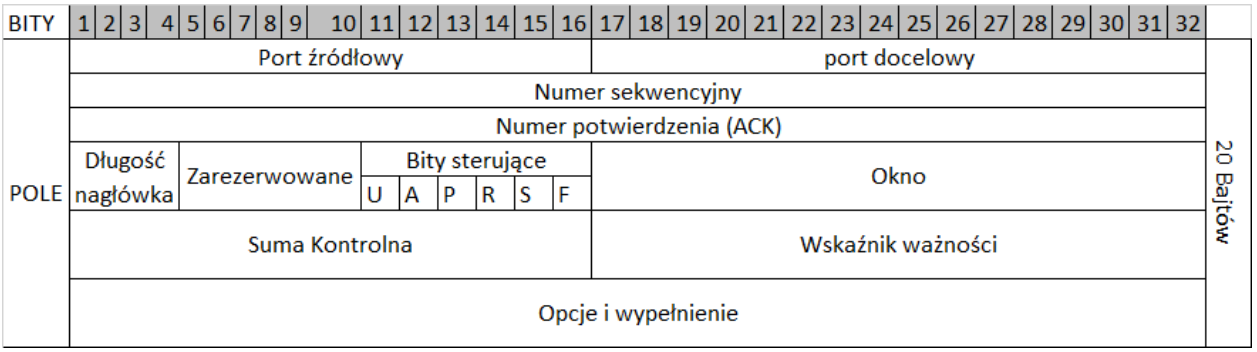
TCP

Priorytetowym zadaniem protokołu TCP (ang. *Transmission Control Protocol*) jest poprawne dostarczenie pakietu do miejsca docelowego. TCP jest uznawany za protokół pewny. Oznacza to, że TCP ma zaimplementowane mechanizmy zapewniające dotarcie danych do odbiorcy. Podczas segmentowania każdy segment dostaje indywidualny numer sekwencyjny wykorzystywany do prawidłowego złączenia ich po stronie odbiorcy. TCP na samym początku nawiązuje połączenie z odbiorcą, a co za tym idzie jest właśnie protokołem połączeniowym, który najpierw zestawia połączenie i uzgodnienie wszystkich parametrów transmisji na przykład prędkości wysyłania dostosowanej do prędkości akceptowalnej dla odbiorcy. [2]

Podczas przesyłania danych przy pomocy protokołu TCP dane po każdym wysłaniu muszą zostać potwierdzone. Jeżeli zostały potwierdzone, oznacza to, że zostały prawidłowo dostarczone do adresata. Protokół ten zapewnia mechanizm retransmisji danych w przypadku, jeżeli nie zostanie przesłane potwierdzenie od adresata. Mechanizm ten jest bardzo ważny podczas

przesyłania wrażliwych danych na przykład komunikacji email, korespondencji bankowej. Podczas komunikacji TCP dwa urządzenia końcowe zestawiają między sobą sesję, dzięki której oba urządzenia przygotowują się do wymiany informacji między sobą. W momencie przekazywania danych, jeżeli jedno z urządzeń nie nadąża z przetwarzaniem odbieranych danych zostaje aktywowany mechanizm kontroli przepływu (ang. Flow control) w momencie działania mechanizmu urządzenia dostosowują nowe warunki połączenia. [2]

TCP jest tzw. protokołem stanowym (ang. stateful) ustanawiając sesje, może on kontrolować które informacje zostały już przez niego wysłane oraz które zostały już potwierdzone. TCP posiada 20-bajtowy nagłówek (Rysunek 2.3), w którym znajdują się informację niezbędne do prawidłowego funkcjonowania komunikacji. [2]



Rysunek 2.3 Nagłówek TCP

Pole *port źródłowy* (*ang. Source port*) ma długość 16 bitów. W tym polu znajduje się informacja dotycząca portu aplikacji z którego zostały wysłane dane.

Kolejnym polem jest *port docelowy (ang. Destination port),* w tym polu znajduje się port urządzenia odbiorcy, na który mają zostać wysłane dane. Dzięki temu polu warstwa transportowa stacji docelowej będzie wiedziała, gdzie przekazać odebrane dane.

Pole *numer sekwencyjny (ang. sequence number)* ma długość 32 bitów i jest to jedno z najważniejszych pól nagłówka TCP, ponieważ odgrywa istotną rolę w porządkowaniu przychodzących pakietów do odbiorcy. Numer ten określa numer porządkowy pierwszego odebranego bajtu każdego odebranego segmentu danych.

Kolejnym polem jest 32 bitowy *Numer potwierdzenia (ang. ACK acknowledgement number).* To pole zawiera informację na temat kolejnego bajtu jaki jest oczekiwany przez odbiorcę. Dzięki tym dwóm polom cały mechanizm porządkowania segmentów odbywa się w sposób niemylący się.

Długość nagłówka (ang. header length) jest to pole 4 bitowe, które zawiera informację o długości nagłówka TCP z tego powodu, iż nagłówek może mieć zmienną długość, dlatego to pole informuję odbiorcę jakiej długości pakietu ma się spodziewać. Długość nagłówka TCP zależy od ewentualnych dodatkowych danych w polu *Opcje*.

Zarezerwowane bity (ang. reserved bits) to pole o długości 6 bitów które nie jest używane.

Pole Bity sterujące (flag) (ang. control bits – flags) zawiera tak naprawdę sześć pól, które mogą zajmować wartości 0 lub 1. Znaczenie kolejnych flag są następujące:

- U (URG) oznacza ważność (ang. urgent). Oznacza to czy pole ma większy priorytet w kolejności przetwarzania pakietów przez odbiorcę.

- A (ACK) oznacza potwierdzenie (ang. acknowledgment) otrzymanego segmentu. Po wysłanym segmencie bądź grupie segmentów następuje potwierdzenie ich otrzymania.

- P (PSH) informuje, że odbiorca powinien przekazać dane *(ang. push)* od razu do warstwy wyższej.

- R (RST) to flaga resetująca *(ang. reset)*, która jest wykorzystywana do natychmiastowego resetowania danej sesji.

- S (SYN) to flaga, która występuje podczas nawiązywania połączenia i rozpoczyna proces synchronizacji *(ang. synchronization)* klienta i serwera. Jeżeli klient chce rozpocząć połączenie, wysyła segment z ustawioną flagą SYN, informując drugą stronę, że chce rozpocząć nawiązywanie sesji za pomocą TCP.

- F (FIN) to flaga zakończenia (*ang. finish*) która jest ustawiana w momencie jak klient kończy połączenie TCP. Druga strona też odpowiada w tym momencie pakietem FIN i połączenie między nimi zostaje zakończone.

Pole *Okno* (ang. window) informuje o ilości danych jakie mogą zostać przesłane bez konieczności ich potwierdzenia. Oznacza to więc również ilość danych jaka może być przesłana do odbiorcy.

Pole *suma kontrolna (ang. checksum)* jest polem kontrolnym 16-bitowym w którym TCP umieszcza obliczoną wartość podsumowującą segment. Strona odbierająca w momencie odebrania segmentu również wykonuje obliczenia, aby zweryfikować czy przesyłana wiadomość nie zawiera błędów.

Wskaźnik *ważność (ang. urgent pointer)* to pole wykorzystywane do nadania priorytetu danych wysyłanych za pomocą TCP.

Ostatnie pole nagłówka TCP to pole, *Opcje i wypełnienie (ang. options and completion),* służy do umieszczania dodatkowych informacji związanych z przesyłanymi danymi. [2]

Kiedy jest ustanawiana sesja TCP ustawiany jest także początkowy *numer sekwencyjny (ang. Initial Sequence Number - ISN).* Jest to liczba losowa, reprezentująca wartość początkową bajtów przesyłanych do aplikacji odbierającej dane. W momencie przyjścia danych do odbiorcy, liczba ta jest zwiększana o liczbę przesyłanych bajtów. Dane następnie są składowane w buforze odbiorczym a następnie przekazywane do warstwy aplikacji. Żaden protokół nie jest w stanie zagwarantować, że dane dotrą do odbiorcy. W tym celu TCP ma mechanizmy, dzięki którymi jest w stanie zarządzać utraconymi pakietami. Jeżeli segment nie dotarł do celu to TCP ponownie przesyła je do odbiorcy. [2]

UDP

UDP *(ang. User Datagram Protocol)* jest protokołem zawodnym i niepewnym, gdyż jest protokołem bezpołączeniowym co oznacza, że w momencie jak dane nie dotrą do adresata a nadawca nie dowie się, że dane które wysłał nigdy nie dotarły. W przypadku UDP nie jest to jednak wadą, ponieważ niektóre rodzaje komunikacji nie mogą otrzymywać potwierdzenia po każdorazowym wysłaniu wiadomości. Przykładem takiego rozwiązania jest telefonia IP. Jeżeli podczas rozmowy telefonicznej, któraś z danych zostanie utracona, protokół je pominie. UDP nie prześle ich ponownie do odbiorcy. Odbiorca może odczuć ich brak, ale jeżeli problem ten pojawia się rzadko, to w zasadzie nic drożnego się nie dzieje. W momencie wykorzystania protokołu TCP do takiego celu utrata niewielkiej ilości danych spowodowałaby konieczność przesłania powtórnie całej transmisji. Wywołałoby to wieli zamęt w rozmowie a nawet uniemożliwiałoby to rozmowę. Typowym przykładem wykorzystania protokołu UDP to TFTP (ang. *Trivial File Transfer Protocol)*, SNMP (*ang. Simple Network Management Protocol*), DNS (ang. Domain Name System). [2]

Protokół UDP posiada 64-bitowy nagłówek (Rysunek 2.4) natomiast nie zapewnia niezawodności i kontroli przepływu jak TCP. Nie zapewnia dostarczenia pakietów do odbiorcy, nie wymaga potwierdzenia otrzymania danych. Jest więc protokołem bezpołączeniowym. Jest jednak szybszy i bardziej wydajny od TCP, dlatego jest przeznaczony do innych zastosowań. [2]

BITY	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32	
POLE	Port źródłowy	port docelowy	8-bajtów
	Długość	Suma kontrolna	
	Dane warstwy aplikacji		

Rysunek 2.4 Nagłówek UDP

Pole *port źródłowy* i *port docelowy* pełnią identyczne funkcje jak ich odpowiedniki w protokole TCP.

Pole Długość informuję o długości całego segmentu UDP. Mówi ono o ilości bajtów jakie zawiera segment.

Pole suma kontrolna podobnie jak w TCP służy do określania czy wystąpił błąd podczas przesyłania segmentu. [2]

O tym którego protokołu użyć – TCP czy UDP – decyduje warstwa wyższa. Wybór ten jak widać może mieć decydujące znaczenie dla jakości transmisji.

Ponieważ urządzenie jednocześnie korzysta z wielu usług pracujących na różnych portach stąd właśnie protokół transportowy potrafi rozróżnić do jakiej aplikacji przekazać odebrane dane.

Warstwa sieci

Warstwa sieci (*ang. Network layer*) odbiera segmenty z warstwy transportowej, a następnie są przeprowadzane działania przygotowujące wysłanie danych do adresata. W warstwie sieci spotykamy się z innym typem adresowania niż w warstwie transportowej. W warstwie sieci występuję m.in. protokół IP, który działa z wykorzystaniem adresów IP nadawcy oraz odbiorcy. Adresy te są wykorzystywane podczas przesyłania pakietów między sieciami i podsieciami w celu dotarcia pakietu do miejsca docelowego. W warstwie wyższej został przygotowany segment określający numery portów z jakiego i na jaki ma dotrzeć segment. Aby segment mógł zostać prawidłowo dostarczony do odbiorcy potrzebuje dodatkowych informacji adresowych. [2]

Najważniejszymi protokołami służącymi do adresowania segmentów w sieci są IPv4 oraz IPv6 (*ang. Internet Protocol*). Nie są to jednak jedyne protokoły działające w warstwie sieciowej. Aby urządzenia sieciowe znajdujące się w różnych podsieciach mogły komunikować się między sobą wykorzystują one inne protokoły, na przykład protokoły trasowania (ang. Routing protocols). Do takich protokołów należą m.in. OSPF, EIGRP. Rozwiązaniem wspierającym cała komunikację jest również ICMP (*ang. Internet Control Message Protocol*). [2]

Każde urządzenie chcące się komunikować z urządzeniami znajdującymi się w innych sieciach czy też podsieciach potrzebuje adresu IP urządzenia docelowego. Informacje te znajdują się w nagłówku IP.

Protokół IP został zaprojektowany do adresowania urządzeń i jest on protokołem bezpołączeniowym (*ang. connectionless*) co oznacza, że nie gwarantuje dostarczenia pakietu do

miejsca docelowego. Pakiet IP zawiera w sobie zasadniczo dwa pola nagłówek (*ang. header*) oraz ładunek (*ang. payload*). [2]

Nagłówek IP jest bardzo istotnym elementem w kontekście komunikacji w sieci. To właśnie on zawiera adres źródłowy nadawcy i docelowy adres odbiorcy oraz informację jak warstwa wyższa ma zareagować na otrzymane dane. [2]

IPv4

Adresowanie IPv4 jest adresowaniem wykorzystującym 32-bitowy adres IP składający się z czterech oktetów dzielących adres na mniejsze fragmenty przyjmujące wartości od 0 do 255 oddzielone od siebie kropkami. Nagłówek IPv4 (Rysunek 2.5) ma długość 20 bajtów. Każde urządzenie znajdujące się w sieci komputerowej potrzebuje adresu IP który je identyfikuje. Adres IPv4 składa się z dwóch części których na pierwszy rzut oka nie widać. Dopiero w momencie, gdy adres IP zostanie zestawiony z maską podsieci można zauważyć część adresu określającą fragment sieci oraz fragment adresu identyfikujący konkretne urządzenie znajdujące się w danej podsieci. Maska podsieci jest to ciąg jedynek który może mieć maksymalnie 32bity rozpoczynający się od lewej strony maski. [2]

<table>
<tr><th>BITY</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td colspan="4">Wersja</td><td colspan="4">Długość nagłówka</td><td colspan="6">Usługi zróżnicowane</td><td colspan="2">ECN</td><td colspan="16">Całkowita długość</td></tr>
<tr><td>32</td><td colspan="16">Numer identyfikacyjny</td><td colspan="3">Flagi</td><td colspan="13">Przesunięcie</td></tr>
<tr><td>64</td><td colspan="8">Czas życia</td><td colspan="8">Protokół warstwy wyższej</td><td colspan="16">Suma kontrolna nagłówka</td></tr>
<tr><td>96</td><td colspan="32">Adres źródłowy IP</td></tr>
<tr><td>128</td><td colspan="32">Adres docelowy IP</td></tr>
<tr><td>160</td><td colspan="19">Opcje IP</td><td colspan="13">Wypełnienie</td></tr>
<tr><td>192</td><td colspan="32">Dane</td></tr>
</table>

Rysunek 2.5 Nagłówek pakietu IPv4

Pierwszym polem nagłówka IP jest *Wersja (ang. Version)* zawiera ono informację o tym która wersja protokołu IP jest wykorzystywana.

Pole *długość nagłówka* (*ang. Header length*) określa długość nagłówka IPv4.

Ośmiobitowe pole *usługi zróżnicowane DS* (ang. Differentiated services) wykorzystywane jest do obsługi mechanizmu QoS (ang. Quality of service).

Szesnastobitowe pole *Długość pakietu (ang. Total length)* określa całkowitą długość pakietu liczoną łącznie z nagłówkiem i danymi znajdującymi się w pakiecie. Dzięki temu polu odbiorca wie, kiedy skończyć przetwarzanie otrzymanego pakietu.

Pole *Flagi (ang. flags)* określa, czy pakiet mógł zostać poddany fragmentacji. Jeżeli bit DF (ang. Don't Fragment) jest ustawione, oznacza to, że pakiet został odebrany w całości. Jeżeli pole MF (ang. More Fragment) jest ustawione oznacza to, że odebrany fragment nie jest ostatnią częścią odbieranego pakietu.

Pole *Przesunięcie (ang. Fragment Offset)* określa miejsce odebranego fragmentu w momencie, gdy odebrany fragment jest elementem pakietu podzielonego na mniejsze fragmenty.

Pole *TTL* (ang. Time To Live) oznacza maksymalną liczbę urządzeń, przez które może zostać przesłany pakiet. Przy każdym przesłaniu pakietu przez router pole to jest pomniejszane o 1. W momencie jak to pole osiągnie wartość 0 pakiet jest odrzucany i nie jest przesyłany po sieci w nieskończoność.

Pole *protokół* (ang. Protocol) oznacza charakter przesyłanych danych, dzięki tej wiadomości warstwa wyższa zostaje poinformowana w jaki sposób ma obsłużyć odebrane dane podczas procesu dekapsulacji.

Pole *Suma kontrolna nagłówka (ang. header checksum)* pozwala stwierdzić, czy nagłówek został przesłany poprawnie, jest sprawdzany przy każdorazowym analizowaniu nagłówka.

Pole *Adres źródłowy (ang. Source IP Address)* jest polem 32 bitowym określający adres IPv4 nadawcy.

Pole *Adres docelowy (ang. Destination IP Address)* jest polem 32 bitowym zawierający adres IPv4 odbiorcy.

Pole *Opcje (ang. Options)* nieobowiązkowe pole charakteryzujące dodatkowe zachowanie względem odebranego pakietu IP.

Pole wypełnienie (ang. Padding) jest polem opcjonalnym wypełniającym nagłówek tak aby jego wielkość była wielokrotnością 32, w tym celu wypełniane jest zerami. [2]

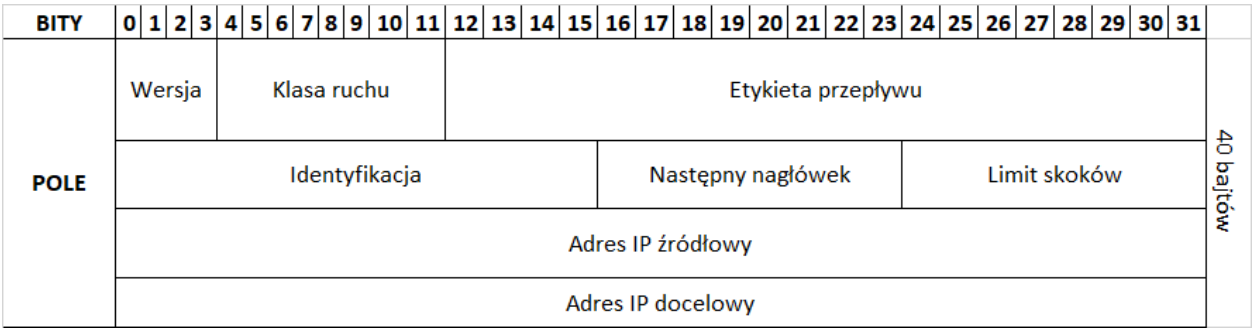
IPv6

Protokół IPv4 jest nadal najpowszechniejszym protokołem na świecie, jednakże specjaliści prorokowali już wiele razy jego koniec, ze względu na kończące się możliwości adresowania nowych urządzeń dodawanych do sieci. Ze względu na brak nowych adresów IPv4 konieczne jest omijanie tego problemu przez rozwiązania programowe takie jak *NAT (ang. Network Address Translation)* lub przez przekierowywanie portów. Zastosowanie tych dwóch mechanizmów zmniejsza szybkość transferu informacji, gdyż każdy nagłówek (Rysunek 2.6) pakietu musi zostać przeanalizowany przed przesłaniem dalej.

W związku z tym problemem między innymi został wynaleziony protokół IPv6, aby zabezpieczyć dostępność i możliwość dalszego adresowania urządzeń. Mechanizm NAT opóźnił przejęcie sieci przez nowy standard adresacji IPv6, ale przez to mechanizm ten uniemożliwia bezpośredniego komunikowania się urządzeń między sobą. Mechanizm ten polega na „ukryciu" adresów znajdujących się w danej podsieci pod jednym adresem publicznym.

Adresacja IPv6 została wynaleziona już w latach dziewięćdziesiątych przez *IETF (Internet Engineering Task Forces)*. Wielkim atutem adresacji IPv6 nad IPv4 jest zwiększenie pola adresu z 32 do 128 bitów.

Razem z wprowadzeniem adresacji IPv6 zmieniła się również forma nagłówka pakietu IP jest ona zdecydowanie prostsza niż w przypadku IPv4. [2]



Rysunek 2.6 Nagłówek IPv6

Pole *wersja (ang. Version)* identyfikuje jakiej wersji protokołu IP należy użyć podczas analizowania pakietu

Pole *Klasa ruchu (ang. Traffic class)* pełni identyczną funkcję co pole DS w nagłówku IPv4, jest więc związane z mechanizmem QoS.

Pole *następny nagłówek (ang. next header)* określa rodzaj danych jakie są przesyłane w pakiecie IP

Pole *limit skoków (ang. Hop limit)* jest to pole 8-bitowe zawierające informację o ilości skoków jakie może wykonać pakiet. Przy każdorazowym przeskoku pole to jest pomniejszane o 1. W momencie, gdy to pole osiągnie wartość 0 pakiet zostaje odrzucony.

Pole *adres źródłowy (ang. Source Address)* jest w polem 128 bitowym przechowującym adres nadawcy pakietu.

Pole *adres docelowy (and. Destination Address)* jest polem 128 bitowym przechowującym adres odbiorcy pakietu.

[2]

Warstwa łącza danych

Warstwa *łącza danych (ang. data link layer)* umieszcza pakiety w ramkach dodając do nich nagłówek (Rysunek 2.7) zawierający adresy MAC *(ang. Medium Access Control)* wykorzystywane do adresacji które są adresami fizycznymi urządzeń. Warstwę łącza danych dzielimy na dwie podwarstwy: *MAC (ang. Media Access Control)* oraz *LLC (ang. Logical Link Control).* [2]

Podwarstwa MAC określa sposób w jaki zostają przesyłane dane przez medium sieciowe i jest oparta na adresacji fizycznej natomiast podwarstwa LLC identyfikuje protokoły i występującą w nich enkapsulację danych.

Proces Enkapsulacji polega na dodaniu do fragmentu danych nagłówka oraz pola końcowego. W warstwie łącza danych dodawane są pola przechowujące źródłowe i docelowe adresy fizyczne (MAC). [2]

Podwarstwa MAC odpowiedzialna jest za dostęp do medium transmisyjnego tak aby w tym samym momencie korzystało z niego tylko jedno urządzenie. Mechanizmem sterującym jest tutaj *CSMA\CD (ang. Carrier Sense Multiple Access Collision Detection).* [2]

64 BITY	48 BITÓW	48 BITÓW	16 BITÓW	46 - 1500 BAJTÓW	32 BITY
PREAMBUŁA	ADRES DOCELOWY	ADRES ŹRÓDŁOWY	TYP/DŁUGOŚĆ	DANE	CRC

Rysunek 2.7 Ramka Ethernet

Preambuła (ang. preambule) jest to pole odpowiedzialne za wykrycie przez urządzenie rozpoczęcia komunikacji

Pole *Adres Docelowy (ang. Destination Address)* pole zawierające 48-bitowy adres fizyczny urządzenia, do którego jest wysyłana ramka

Pole Adres Źródłowy (ang. Source Address) pole zawierające 48-bitowy adres fizyczny urządzenia wysyłającego ramkę

Pole *typ/długość (ang. type/length)* ma długość 16 bitów odpowiedzialne jest za poinformowanie urządzenia, do którego jest wysyłana ramka, jakie dane są w niej przesyłane.

Pole *dane (ang. data)* ma minimum 46 bajtów a maksymalnie 1500 bajtów. W tym miejscu umieszczę są informację przesłane z warstwy wyższej modelu.

Ostatnie pole ramki CRC *(ang. Cyclic Redundancy Check)* jest to pole określające czy ramka została przesłana w sposób niezmieniony. Zapewnia ono sprawdzenie integralności przesyłanych danych. [2]

Warstwa fizyczna

Warstwa fizyczna (ang. Physical layer) jest ostatnią najniższą warstwą modelu OSI, która przesyła dane przez medium transmisyjne.

Warstwa ta jest odpowiedzialna jedynie za przesyłanie informacji przez dane medium transmisyjne. Warstwa ta obsługuje różne typy medium transmisyjnego, takie jak: *kabel miedziany*, *światłowód* czy *też fale radiowe.* Ponieważ transmisja odbywa się za pośrednictwem przesyłania zer i jedynek warstwa fizyczna konwertuje otrzymane dane do postaci strumieni binarnych. W momencie odbioru musi ona zamienić otrzymane impulsy czy to elektryczne, optyczne czy też radiowe na postać danych obsługiwanych przez warstwę wyższą modelu. [2]

2.2.1. Model TCP/IP

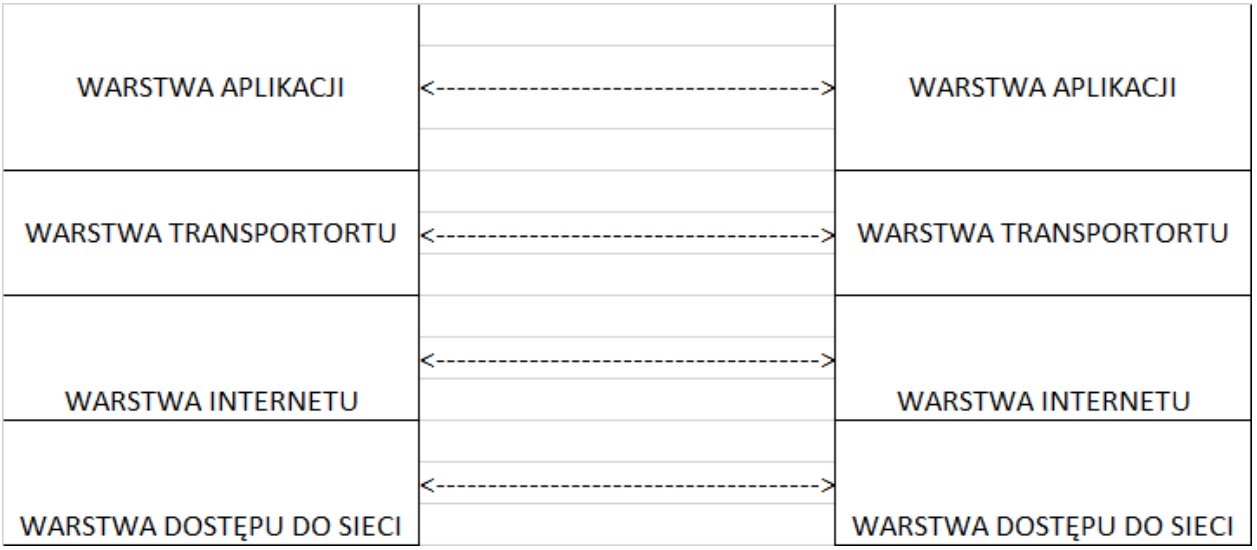
Model TCP/IP (ang. *Transmission Control Protocol/Internet Protocol)* jest swego rodzaju odzwierciedleniem sieci komputerowej oraz umiejscowienia danych urządzeń na odpowiednim poziomie modelu. [2]

Model TCP/IP złożony jest z czterech warstw (Rysunek 2.8), a każda warstwa opowiada co dzieje się na konkretnym etapie wymiany informacji między dwoma urządzeniami. Dzięki takiemu pogrupowaniu informacji można szybko przyporządkować dany problem w komunikacji do konkretnej warstwy i w tym miejscu szukać jego rozwiązania. [2]

Model TCP/IP jest modelem otwartym, który został opracowany we wczesnych latach siedemdziesiątych XX w. dla Departamentu Obrony Stanów Zjednoczonych. [2]

Model ten zawiera cztery warstwy, którymi są:

- Warstwa aplikacji (ang. appliacation layer)
- Warstwa transportu (ang. transport layer)
- Warstwa internetowa (ang. internet layer)
- Warstwa dostępu do sieci (ang. network access layer)



Rysunek 2.8 Model systemu otwartego TCP/IP

Warstwa aplikacji

Warstwa aplikacji (*ang. Application layer*) jest najwyższą warstwą modelu a co za tym idzie jest zawsze najbliżej użytkownika. Dzieje się tak ponieważ to właśnie w tej warstwie działają aplikacje, programy, z których GUI (*ang. Graphical User Interface*) użytkownik ma bezpośredni wpływ na to co i na jakich warunkach zostanie przesłane przez zhierarchizowaną strukturę systemu otwartego. Wybór aplikacji jest uzależniony od wyboru użytkownika czego on w danym momencie potrzebuje, może to być klient poczty w tym celu wykorzysta on protokół *POP3* (*ang. Post Office Protocol*) bądź też *IMAP (ang.* Internet Message Access Protocol*)* do odebrania wiadomości. Może on też użyć najpowszechniejszego protokołu tej warstwy protokołu *HTTP (ang. Hypertext Transfer Protocol)* w celu nawiązania komunikacji z serwerem *WWW (ang. World Wide Web)*. Pobranie zasobu odbywa się po podaniu prawidłowego adresu strony zwanego *URL (ang. Uniform Resource Loader)*. Następnie po wybraniu odpowiedniego protokołu warstwa

aplikacji przekazuje dane warstwie niższej w celu kontynuowania procesu komunikacji z urządzeniem docelowym. [2]

Warstwa transportu

Warstwa ta jest odpowiedzialna za transport danych między urządzeniami końcowymi i wszystkim co jest związane z prawidłowym dostarczeniem danych do odbiorcy. Na tym poziomie jest wybierany prawidłowy rodzaj komunikacji oraz protokół z jakiego będzie korzystać usługa wybrana w warstwie aplikacji. Do dwóch najpopularniejszych protokołów warstwy transportu zaliczają się *TCP (ang. Transmission Control Protocol)* i *UDP (*ang. *User Datagram Protocol).* Oba te protokoły zostały opisane w *podrozdziale 2.1.4.* [2]

Warstwa internetowa

Warstwa internetowa (ang. internet layer) jest to warstwa odpowiedzialna za tworzenie pakietów i adresowanie ich oraz podejmowania decyzji jaką drogą pakiet zostanie dostarczony do miejsca docelowego. Kontrolę nad tymi zadaniami sprawuje protokół IP. Warstwa internetowa w modelu TCP/IP odpowiada warstwie sieciowej (ang. network layer) w modelu odniesienia ISO OSI.

W celu dostarczenia pakietu z jednego miejsca do drugiego należy zaadresować dane, służy do tego protokół IP w wersji 4 bądź też w wersji 6.

To właśnie w tej warstwie podejmowana jest decyzja jaką ścieżką pakiet zostanie dostarczony do miejsca docelowego, służy do tego mechanizm Routingu.

Proces Routingu jest to proces budowania trasy od jednego routera do drugiego. Można wyróżnić dwa typy routingu: statyczny (ang. static routing) i dynamiczny (ang. dynamic routing). Routingiem statycznym nazywamy proces, w którym to administrator decyduje jaką trasą zostanie przesłany pakiet przez sieć. Natomiast routingiem dynamicznym nazywamy sytuacje, gdzie to routery same podejmują decyzje jak pakiet zostanie przesłany przez sieć. [2]

Warstwa dostępu do sieci

Warstwa dostępu do sieci (ang. network access layer) jest najniższą warstwą modelu TCP/IP. Odpowiada ona za prawidłowe zaadresowanie i przesłanie na poziomie fizycznym danych. Warstwa ta odpowiada także za prawidłowe dobranie technologii, aby dane zostały prawidłowo przesłane przez medium transmisyjne. To właśnie w tej warstwie dane zostają zamienione na bity. [2]

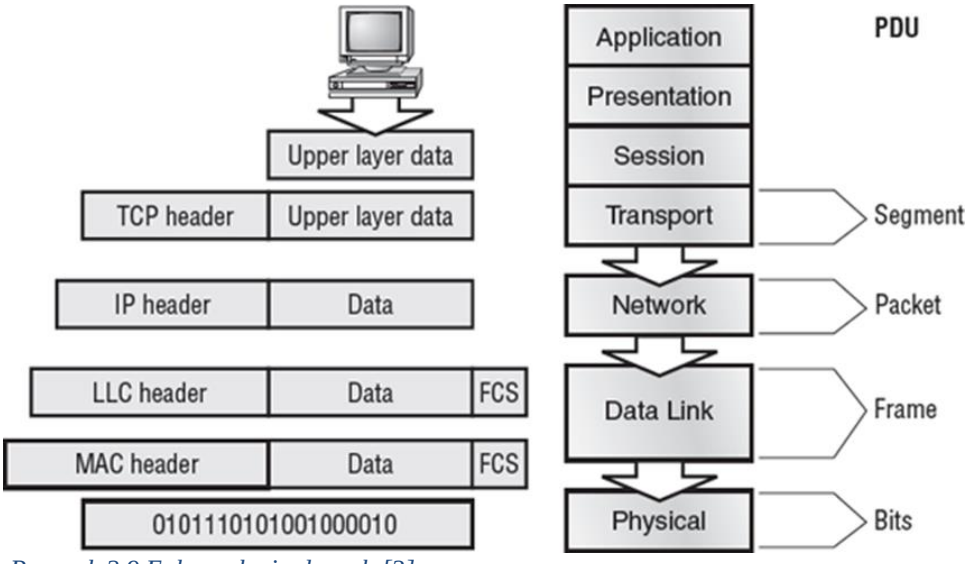
2.3. Definicja protokołu

2.3.1. Definicja jednostek danych protokołu

Jednostki danych protokołu najczęściej są spotykane pod skrótem PDU (ang. Protocol Data Units). Służą do komunikacji oraz wymiany informacji miedzy równorzędnymi elementami sieci komputerowej. Ich zadaniem jest przechowywanie danych oraz informacji kontrolnych, które są dodawane w kolejnych warstwach modelu OSI. [3][4]

2.3.2. Definicja zasad wymiany jednostek danych protokołu

Wymiana jednostek danych protokołu jest objęta pewnymi zasadami. Aby zapewnić przekazywanym informacją bezpieczeństwo, każda warstwa ma dostęp tylko do swojej części PDU. Jest to możliwe dzięki procesom zwanym enkapsulacją oraz dekapsulacją. Podczas przechodzenia w dół przez kolejne warstwy modelu OSI pierwotna zawartość PDU nigdy się nie zmienia, a zostają do niej tylko dodawane kolejne elementy. Jest to zilustrowane na rysunku 3.1.



Rysunek 2.9 Enkapsulacja danych [3]

3.2.1 Enkapsulacja i dekapsulacja

Enkapsulacja to proces umieszczania wewnątrz jednego formatu wiadomości inny format wiadomości. Podczas dekapsulacji następuję operacja odwrotna wykonywana przez odbiorcę. Dzięki temu konceptowi możemy projektować modułowe protokoły komunikacyjne, w których kolejne funkcje w sieci są od siebie oddzielone w sposób logiczny.

2.4. Specyfikacja ASN.1

Składnia ASN.1 (ang. Abstract Syntax Notation One) została zaprezentowana po raz pierwszy w 1984 roku przez CCIT (ang. Consultative Committee for International Telegraphy & Telephony) aktualnie *ITU-T (International Telecommunication Union – Telecommunication Standardization Sector).* Jest ono powszechnie wykorzystywane w szerokorozumianej telekomunikacji, sieciach komputerowych. Została wykorzystana między innymi do zdefiniowania protokołów w sieciach radiowych między innymi dla telefonii mobilnej trzeciej *(UMTS ang. Universal Mobile Telecommunications System)* i czwartek generacji *(LTE ang. Long Term Evolution).* [3]

2.4.1. Notacja abstrakcyjna typów danych

ASN.1 (ang. Abstract Syntax Notation One) jest narzędziem stworzonym do określania składni i zawartości jednostek wymiany informacji niezależnie od wykorzystywanych rozwiązań sprzętowych. Opracowana w 1995 roku norma ASN.1 narzucała obowiązek nadania etykiety każdemu typowi danych, która stanowiła identyfikator danej struktury. W późniejszych wersjach wydanych w latach 1995 oraz 1998 etykiety przestały być konieczne, ponieważ został wprowadzony mechanizm automatycznego etykietowania. Etykieta jest liczbą identyfikującą dany typ. Jedna etykieta może być nadana dla kilku typów, jeżeli występują one w różnych kontekstach, natomiast jeżeli ten sam typ jest wykorzystywany w obrębie jednej aplikacji należy nadać typom różne etykiety. Etykieta może należeć do jednej z czterech klas [1]:

- Klasa **Uniwersalna** – zawiera zbiór etykiet typów prostych bądź też złożonych określonych przez standard
- Klasa **Aplikacyjna** – etykieta jest przypisana do typów zdefiniowanych w innych niż międzynarodowe standardy i może być przypisana tylko do jednego typu.
- Klasa **prywatna** – etykiety należące do tej klasy nie są zawarte w żadnych standardach i są dostosowane do potrzeb użytkownika

- Klasa **kontekstowa** – implementacja etykiety zależy od kontekstu w jakim jest zastosowana.

Struktury danych należące do danej aplikacji grupowane są w jednostki nazywane modułami (Rysunek 2.10). Identyfikatorem danego modułu jest jego nazwa a zawartością są wszystkie definicje typów powiązane z daną aplikacją. [1]

Przykładowy moduł ma postać:

```
Requests DEFINITIONS ::= BEGIN

    Connected ::= SEQUENCE{
        name      UTF8String,
        number      INTEGER,
        connected   BOOLEAN
    }

    Request ::= SEQUENCE{
        name       UTF8String,
        column     INTEGER,
        row        INTEGER
    }

    Response ::= SEQUENCE{
        name       UTF8String,
        column     INTEGER,
        row        INTEGER,
        hit        BOOLEAN,
        sunk       BOOLEAN
    }

    Ready ::= SEQUENCE{
        name      UTF8String,
        ready      BOOLEAN
    }

END
```

Rysunek 2.10 Przykładowy moduł ASN.1

Elementy DEFINITIONS, BEGIN, END wyżej zamieszonego przykładu są słowami kluczowymi zarezerwowanymi ASN.1.

2.4.2. Elementy języka

Identyfikatory mogą być zbudowane z dowolnej liczby znaków alfanumerycznych i myślników. Podstawowymi ograniczeniami w budowie identyfikatorów są:

- Myślnik nie możne znajdować się na końcu ciągu znaków
- Dwa myślniki nie mogą występować obok siebie
- Identyfikatory i nazwy wartości muszą rozpoczynać się z małej litery
- Nazwy typów i modułów muszą rozpoczynać się z dużej litery oraz nie mogą być słowem zarezerwowanym
- Liczba może zawierać jedną lub więcej cyfr ale pierwszą cyfrą w liczbie nie może być zero chyba że jest to jedyna cyfra liczby

Podstawowym atrybutem identyfikatora jest typ. Występują dwa rodzaje typów: *proste* i *złożone*. Typy proste to typy zdefiniowane przez bezpośrednie określenie zbioru ich wartości. Łącząc typy proste w struktury tworzy się typy złożone które mogą zawierać nieskończenie wiele wartości, ponieważ nie jest określona górna granica wielkości jednostek wymiany informacji. [1]

Przykładami prostych typów danych definiowanych w standardzie są:

- INTEGER – reprezentujący wszystkie wartości całkowite.
- BITSTRING – reprezentujący uporządkowany ciąg bitów.
- OCTETSTRING – reprezentujący uporządkowany ciąg słów bajtowych.
- BOOLEAN - reprezentujący wartości logiczne *Prawda* lub *Fałsz*
- UTF8String – reprezentujący łańcuchy znakowe wykorzystujące znaki unicode

Do standardowych typów złożonych zaliczają się:

- SEQUENCE – jest złożony z uporządkowanej sekwencji innych typów prostych.
- SEQUENCE OF – pozwala na uporządkowane zebranie wartości tych samych typów.
- SET – składa się z nieuporządkowanej sekwencji innych typów prostych.

- SET OF – składa się z nieuporządkowanych wartości tych samych typów.
- CHOICE – służy do wyboru zmiennej z określonej listy wartości

[1]

2.4.3. Zasady kodowania na przykładzie BER

ASN.1 – umożliwia definiowanie struktur danych oraz określania wartości poszczególnych elementów struktury, ale nie uwzględnia sposobu reprezentacji danych do celów transmisji. Od przyjętej metody kodowania wymaga się ścisłego powiązania z przesyłaną składnią ponieważ nadawca jak i odbiorca powinni mieć informację o wymienianych strukturach danych. W tym celu opracowano zbiór reguł kodowania. Z notacją ASN.1 zgodne są cztery składnie kodowań wydane w dwóch normach ISO/IEC 8825-1 (dotyczy BER, CER, DER) i ISO/IEC 8825-2 (dotyczy PER).

[1]

Standard kodowania składni abstrakcyjnej, inaczej mówiąc składni transferu, jest zdefiniowany w normie ISO/IEC 8825-1 pod nazwą *reguły podstawowego kodowania (ang. Basic Encodings Rules).* Składnia znana jako skrót BER jest ściśle powiązana z ASN.1 i dlatego nie stanowi odrębnego standardu. [1]

Zgodnie z regułami BER każda reprezentacja wartości abstrakcyjnej typu danych ANS.1 składa się z trzech pól:

- **Identyfikator –** określającą rodzaj danych
- **Długość** – określającej liczbę bajtów danych
- **Zawartość** – zawierające przesyłaną wartość

Każde pole wyrównane jest do całkowitej liczby oktetów. Kodowanie może występować w jednej z dwóch postaci:

- Kodowanie proste (ang. primitive) – taki sposób kodowania wartości danych, w którym zawartości oktetów reprezentują bezpośrednio wartość.
- Kodowanie złożone (ang. constructed) – taki sposób kodowania wartości danych w którym zawartość oktetów zawiera zakodowaną jedną lub więcej wartości danych

[1]

Pole identyfikatora

Pole identyfikatora zawiera informację identyfikujące wartości takie jak: klasa i numer etykiety typu wartości danych oraz formie kodowania. Do zakodowania czterech klas etykiet stosuje się bity 7 i 6 w następujący sposób:

- 00 - UNIVERSAL
- 01 - APPLICATION
- 10 - KONTEKSTOWA
- 11 - PRIVATE

Do zakodowania informacji o reprezentacji kodowanego typu wystarcza jeden bit. Przeznaczony jest do tego celu bit 5, który może przyjmować wartość:

- 0 – typ prosty
- 1 – typ złożony

Bity od 4 do 0 są przeznaczone do kodowania etykiet. Numery poszczególnych etykiet przedstawiono w tabeli poniżej.

TYP	Etykieta		
	prosta		złożona
BOOLEAN	01		
INTEGER	02		
BIT STRING	03		23
OCTET STRING	04		24
NULL	05		
OBJECT IDENTIFIER	06		
ObjectDescription	07		
REAL	08		
ENUMERATED	09		
NumericString	0A		
PrintableString	0B		
GeneralizedTime	0C		
SEQUENCE	18		30
SEQUENCE OF			30
SET			31
SET OF			31

Rysunek 2.11 Wartości identyfikatorów niektórych typów ASN.1 [2]

Numery etykiet zawarte w przedziale od 0 do 30 w połączeniu z polem zawierającym typ oraz dwoma bitami reprezentującymi klasę wypełnia dokładnie jeden oktet (Rysunek 2.12).

7	6	5	4	3	2	1	0
Klasa		P/Z	numer etykiety				

Rysunek 2.12 Oktet identyfikatora dla małych wartości etykiet

Pole długości

Drugim polem w postaci zakodowanej informacji jest ***pole długości*** określa ono długość pola zawartości. Pole długości może być zakodowane w trzech formach: *długiej, krótkiej* oraz *nieokreślonej*. Pierwsze dwie formy nazywa się formami określonymi ponieważ znana jest ich wartość, natomiast forma nieokreślona jest stosowana w momencie kiedy nie znamy długości pola zawartości [1].

Jeżeli pole zawartości ma rozmiar mniejszy od 128 bajtów to stosuje się formę krótką. Forma ta wykorzystuje pojedynczy oktet bitów z najbardziej znaczącym bitem ustawionym na zero a na pozostałych siedmiu bitach zapisana jest długość pola zawartości [1].

Jeżeli pole zawartości ma rozmiar większy od 128 bitów to stosowana jest forma długa. Forma ta wykorzystuje pierwszy oktet jako informację ile bajtów jest potrzebnych do zakodowania długości pola zawartości a na następnych oktetach zapisana jest wartość. Jeżeli na najbardziej znaczącym bicie pierwszego oktetu ustawiona jest 1 oznacza to że mamy do czynienia właśnie z formą długą. Przykładowo zakodowanie długości 480 bajtów będzie wymagało dwóch bajtów więc na całą informację o długości pola zawartości będą składały się trzy bajty. Kod taki będzie wyglądał następująco: **82 01 E0**$_{16}$. [1]

Forma nieokreślona pola długości zawartości polega na stosowaniu ograniczników umieszczanych na początku i końcu kodowanej wartości. Ogranicznikiem początku jest oktet 80_{16} natomiast znacznikiem końca kodowania pola są dwa bajty wypełnione samymi zerami. Wszystko co znajduje się między tymi trzema oktetami jest wartością określającą długość pola zawartości. [1]

Pole zawartości

Pole zawartości zawiera zakodowaną wartość przesyłanej informacji i jest zależna ona od tego jaką
informację użytkownik chce zakodować.

3. Modele komunikacji

3.1. Model klient-serwer

Model struktury sieci klient-serwer jest obecnie najbardziej popularnym sposobem komunikacji na świecie. Polega on na hierarchii, gdzie serwer, najczęściej maszyna zazwyczaj z dużymi możliwościami obliczeniowymi, pojemną pamięcią i szybkim łączem danych, udostępnia swoje zasoby dla klientów, którymi są najczęściej zwykłe komputery i smartphony. Jednak nie jest to tylko fizyczna topologia, termin też odnosi się do oprogramowania i protokołów, dzięki którym serwer i klient mają swoje wydzielone role, a komunikacja przepływa sprawniej. Oprogramowanie na urządzeniu prywatnym jest w głównej mierze odpowiedzialne za interfejs użytkownika, a strona serwera tak aby jak najefektywniej odpowiadać na wszystkie żądania. Model ten ma trzy główne zalety. Pierwszą z nich jest bezpieczeństwo, ponieważ wszystkie nazwy oraz hasła użytkowników są na serwerze, dzięki czemu system bezpieczeństwa może sprawdzić czy dana osoba ma dostęp

do zasobów, do których i w jakim stopniu. Dzięki temu modelowi sieć jest lepiej zorganizowana, użytkownik nie musi pamiętać, gdzie dokładnie znajdują się konkretne zasoby, ponieważ wszystko jest w jednym miejscu. Ostatnią główną zaletą jest skalowalność, gdyż w takiej strukturze może istnieć ogromna ilość urządzeń końcowych. Powodem dla którego model ten stał się tak dominujący, że jest on podstawowym sposobem w największej sieci świata jaką jest Internet. Architektura klient-serwer jest podstawą większości protokołów i usług TCP/IP. Zasadniczo jest to kwestia nazewnictwa ponieważ termin strona sieciowa to naprawdę serwer sieciowy, a przeglądarka internetowa to inna nazwa na klienta sieciowego [3] [4]

3.2. Alternatywne modele komunikacji

Drugim najpopularniejszym modelem komunikacji obecnie jest peer-to-peer. W takiej sieci każde urządzenie jest równorzędnym partnerem, który może dzielić zasoby z każdym innym urządzeniem w sieci. W takim modelu żadna maszyna nie ma przypisanej roli i uruchamia podobne oprogramowanie, a wszystkie żądania są wysyłane miedzy sobą. Główną zaletą tego rozwiązania jest prostota i niskie koszty, jednak wraz z upływem lat coraz więcej sieci powstawało i zmieniało się na model klient-serwer. W strukturze peer-to-peer wszystkie urządzenia, są same odpowiedzialne za przeprowadzanie kontroli bezpieczeństwa w zakresie praw dostępu do swoich zasobów. Działa to dość sprawnie, o ile w sieci nie ma zbyt wiele maszyn. Przez to nasza skalowalność jest mocno ograniczona, tym bardziej jeśli użytkownicy korzystają z różnych systemów operacyjnych. [3] [4]

4. Środowisko programistyczne

4.1. Język Python

Python to obiektowy język skryptowy wysokiego poziomu, ogólnego przeznaczenia, który został opracowany przez Guido van Rossuma i opublikowany w 1991 roku. Szybko stał się ona jednym z najpopularniejszych języków na świecie, a jest on szczególnie popularny w edukacji oraz z zastosowaniach naukowych. Stało się tak, ponieważ wiele osób uważa, że jest on łatwiejszy do nauczenia oraz czytania w porównaniu do wielu innych popularnych języków programowania. Python jest darmowy i posiada otwarty kod źródłowy, dzięki czemu zebrał wokół siebie dużą społeczność. Z tego względu powstało i nadal jest rozwijane wiele nowych bibliotek, zarówno

standardowych, jak i od stron trzecich, które zwiększają produktywność pisanego kodu. Język ten jest bardzo uniwersalny, ponieważ działa na wszystkich popularnych systemach operacyjnych oraz posiada swoje zastosowania w: pisaniu skryptów automatyzacji, uczeniu maszynowym, obliczeniach naukowych, big data, aplikacjach internetowych, programowaniu GUI oraz urządzeniach IoT. Dzieje się tak, ponieważ Python to język interpretowany, co oznacza że instrukcje są interpretowane w czasie jego działania i nie ma potrzeby prekompilacji programu do instrukcji języka maszynowego. Jedną z głównych zalet tego rozwiązanie jest to, że umożliwia szybkie tworzenie prototypów i eksperymentowanie. Python wspiera także popularne paradygmaty takie jak programowanie: obiektowe, funkcjonalne, imperatywne i proceduralne. Posiada również mechanizmy, które upraszczają programowanie współbieżne. [5] [6]

4.2. Środowisko IDE

Środowiskiem programistycznym wykorzystanym do stworzenia aplikacji i serwera był Pycharm wyprodukowany przez firmę programistyczną JetBrains. Wspiera ono proces implementacji aplikacji w języku Python oraz pozwala na sprawne debugowanie rozwijanej aplikacji. Do odizolowania aplikacji od globalnego środowiska Python posłużyło wirtualne środowisko (*ang. Virtual Environments*) które pozwala na odizolowane instalowanie pakietów dla każdej implementowanej aplikacji. Do zarządzania bibliotekami niezbędnymi do prawidłowego działania programu został wykorzystany PIP oficjalny system zarządzania pakietami dla środowiska języka Python, który korzysta z dedykowanego repozytorium pakietów o nazwie Python Package Index lub innych zdalnych oraz lokalnych repozytoriów. [3] [4] [5]

4.3. Biblioteki języka Python

Biblioteka w kontekście programistycznym jest zbiorem prekompilowanych kodów, które mogą zostać użyte w naszym programie, ale również odnosi się takich rzeczy jak na przykład dane konfiguracyjne, dokumentacje, szablony wiadomości, klasy, czy wartości. Jako zbiór kodów lub modułów kodów ich głównym celem jest sprawienie, aby programiści nie musieli ponownie pisać istniejących już kodów, które są ogólnie dostępne. Korzystanie z bibliotek w wielu przypadkach jest wskazane, ponieważ minimalizuje to ilość kodu oraz poprawia jego przejrzystość. [7]
W naszym programie korzystamy z następujących bibliotek:

- pygame
- pygame_menu
- asn1tools
- _thread
- socket

4.3.1. Pygame

Pygame to biblioteka języka Python, składająca się z zestawu modułów, umożliwiające łatwiejsze tworzenie pełni funkcjonalnych gier i programów multimedialnych. Pozwala ona na korzystanie z wielu rozwiązań innej biblioteki zwanej Simple DirectMedia Layer (SDL), na której się opiera oraz dodaje wiele swoich własnych rozwiązań. Swoją popularność zyskał dzięki swojej darmowości, współpracy z wieloma systemami operacyjnymi oraz prostocie użycia. Z tego narzędzia korzystają zarówno dzieci stawiające swoje pierwsze kroki w programowaniu, jak i dorośli. Rdzeń pygame jest utrzymywany w prostocie, dzięki czemu ilość kodu jest mała. Dodatkowe rzeczy takie jak efekty i biblioteki GUI są rozwijane osobno. Pygame jest biblioteką modułową, w której wiele z podstawowych modułów może być używana i inicjalizowana oddzielnie. [8][9]

4.3.2. Pygame_menu

Pygame-menu to biblioteka python-pygame, która pozwala nam tworzyć GUI oraz menu. Obsługuje szereg widżetów takich jak ramki, obrazy, selektory, etykiety, przyciski, obiekty zegara, pola wprowadzania tekstu oraz wiele innych. Każdy z nich można dopasować do zapotrzebowania, ponieważ posiadają w tym celu wiele opcji. [10]

4.3.3. Asn1tools

Ans1tools to pakiet Pythona do parsowania, kodowania i dekodowania w standardzie ASN.1. Posiada zmapowane wszystkie typy na odpowiednie typy języka Python. Biblioteka ta jest prosta w użyciu, ale wspiera najważniejsze kodeki takie jak:

- Basic Encoding Rules (BER)
- Distinguished Encoding Rules (DER)

- Generic String Encoding Rules (GSER)
- JSON Encoding Rules (JER)
- Basic Octet Encoding Rules (OER)
- Aligned Packed Encoding Rules (PER)
- Unaligned Packed Encoding Rules (UPER)
- XML Encoding Rules (XER)

[11]

4.3.4. _thread

_thread to moduł pozwala nam na prace z wieloma wątkami, które dzielą ze sobą globalną przestrzeń danych. Zapewnia on łatwiejsze używanie wyżej poziomowych API opartych na wątkach. Synchronizacja jest uzyskiwana dzięki prostym mechanizmom takim jak muteksy i semafory binarne. Biblioteka ta jest niskopoziomowa, więc często zalecane jest korzystanie z wysokopoziomowego modułu threading, który jest oparty na _thread. [11][12]

4.3.5. Socket

Socket (ang. gniazdo) to biblioteka, która zapewnia dostęp do interfejsu BSD. Pozwala na przesyłanie danych pomiędzy programami na maszynach w tej samej sieci, jak i na tym samym komputerze. Do komunikacji używają globalnego numeru portu, lub adresu IP oraz numeru portu. Gniazda są globalne w całej maszynie oraz nie wymagają pamięci współdzielonej między wątkami lub procesami. [13][14]

5. Specyfikacja gry sieciowej

5.1. Klient

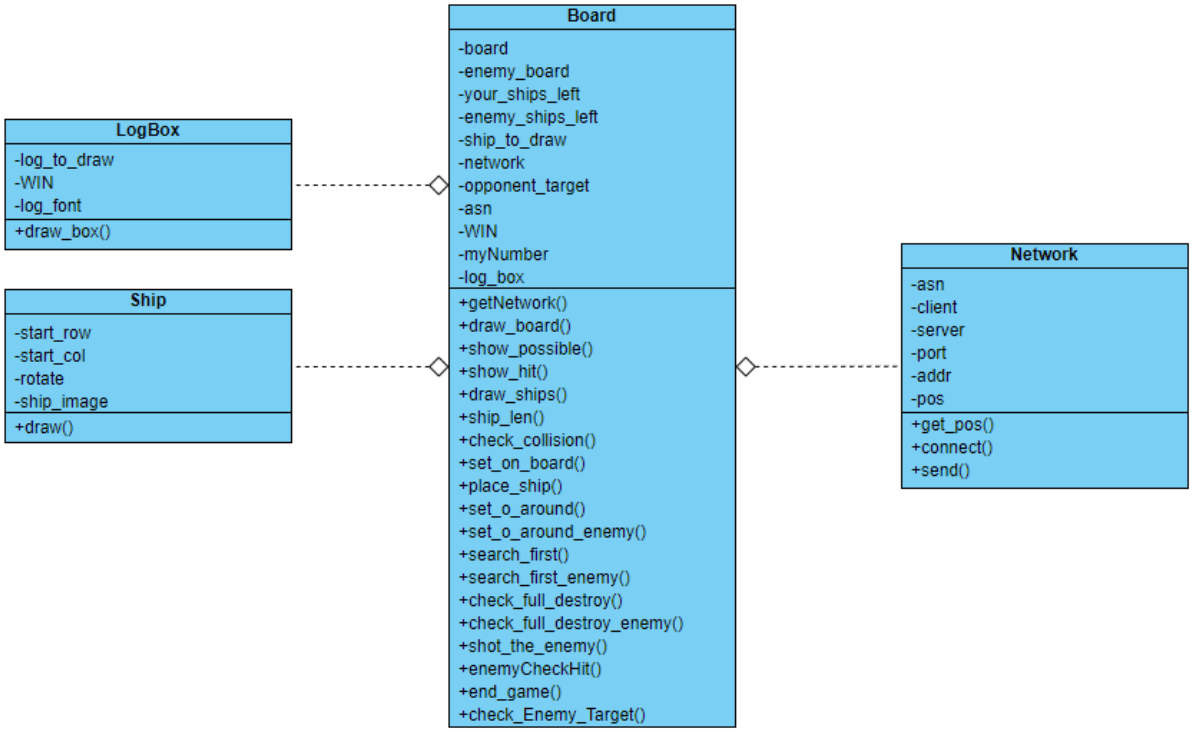
5.1.1. Wymagania funkcjonalne

Aplikacja powinna spełniać następujące wymagania funkcjonalne:

- Użytkownik powinien mieć możliwość rozpoczęcia gry po połączeniu z serwerem oraz drugim graczem
- Użytkownik powinien mieć możliwość rozmieścić statki na swojej części planszy

- Użytkownik powinien mieć możliwość oddania strzału w plansze przeciwnika oraz otrzymać informację zwrotną, czy trafił w statek przeciwnika
- Użytkownik powinien otrzymać informację, że cały statek przeciwnika został zatopiony
- Użytkownik powinien dostać stosowny komunikat o zakończeniu gry oraz jej rezultacie

5.1.2. Używane diagramy UML



Rysunek 5.1Klient - schemat UML

5.2. Serwer

5.2.1. Wymagania funkcjonalne

Wymaganiami funkcjonalnymi serwera są:

- Tworzenie osobnego wątku dla każdej rozgrywki
- Podłączenie się użytkownika do serwera
- Przesłanie informacji o rozpoczęciu rozgrywki
- Przesłanie informacji o gotowości obu użytkowników
- Przesłanie informacji o sprawdzeniu pola o zadanych koordynatach
- Przesłanie informacji o sprawdzonym polu

5.2.2. Używane diagramy UML

5.3. Protokół warstwy aplikacji

5.3.1. Wymagania

Wymaganiami funkcjonalnymi protokołu są:

- Wymiana informacji o prawidłowym podłączeniu użytkownika do serwera
- Wymiana informacji o gotowości użytkownika
- Wymiana informacji o rozpoczęciu rozgrywki
- Wymiana informacji w celu sprawdzenia pola o zadanych koordynatach
- Wymiana informacji o sprawdzonym polu

5.3.2. Specyfikacja jednostek i wymiany danych w ASN.1

Jednostkami protokołu wykorzystywanymi w naszym systemie są:

- Connected

- Ready
- Request
- Response

Jednostki danych protokołu są kodowane w formacie BER dzięki temu zakodowane jednostki są jednoznacznie zdekodowane po stronie odbiorcy.

Connected

Pierwszą jednostką protokołu wymienianą po podłączeniu się obu użytkowników do serwera jest jednostka **Connected** jest ona odpowiedzialna za przekazanie użytkownikom informacji o rozpoczęciu przygotowania rozgrywki. Jednostka ta składa się z następujących pól:

- name – pole typu łańcucha znaków unicode zawierające nazwę przesyłanego PDU.
- number – pole typu całkowitego zawierające informację o kolejności rozpoczęcia rozgrywki.
- connected – pole typu logicznego oznaczające prawidłowe podłączenie się użytkowników.

```
3    Connected ::= SEQUENCE{
4        name        UTF8String,
5        number      INTEGER,
6        connected   BOOLEAN
7    }
```

Rysunek 5.2 Jednostka Connected

Ready

Kolejną wiadomością protokołu wymienianą między klientem a serwerem jest jednostka **Ready,** odpowiada ona za informowanie serwera jak i klienta o gotowości do rozpoczęcia rozgrywki. Jednostka ta składa się z następujących pól:

- name – pole typu łańcucha znaków unicode zawierające nazwę przesyłanego PDU.
- ready – pole typu logicznego zawierające informację o gotowości do rozpoczęcia rozgrywki.

```
23      Ready ::= SEQUENCE{
24          name        UTF8String,
25          ready       BOOLEAN
26      }
```

Rysunek 5.3 Jednostka Ready

Request

Pierwszą wiadomością przekazywaną w etapie rozgrywki jest wiadomość **Request,** odpowiada ona za przekazanie informacji o chęci sprawdzenia pola o koordynatach X,Y na planszy przeciwnika. Jednostka ta składa się z następujących pól:

- name – pole typu łańcucha znaków unicode zawierające nazwę przesyłanego PDU.
- column – pole typu całkowitego zawierające index kolumny planszy przeciwnika jakie klient chce sprawdzić.
- row – pole typu całkowitego zawierające index wiersza planszy przeciwnika jakie klient chce sprawdzić.

```
 9      Request ::= SEQUENCE{
10          name        UTF8String,
11          column      INTEGER,
12          row         INTEGER
13      }
```

Rysunek 5.4 Jednostka Request

Response

Kolejną jednostką informacji przekazywaną w etapie rozgrywki jest wiadomość **Response,** odpowiedzialna za przekazanie informacji o sprawdzonym polu na planszy przeciwnika. Jednostka protokołu składa się z następujących pól:

- name - pole typu łańcucha znaków unicode zawierające nazwę przesyłanego PDU.
- column – pole typu całkowitego zawierające index kolumny planszy przeciwnika jakie klient chce sprawdzić.
- row – pole typu całkowitego zawierające index wiersza planszy przeciwnika jakie klient chce sprawdzić.
- hit – pole typu logicznego zawierające informację o trafieniu statku przeciwnika
- sunk – pole typu logicznego zawierające informacje o tym czy trafiony statek został zatopiony

```
15    Response ::= SEQUENCE{
16        name        UTF8String,
17        column      INTEGER,
18        row         INTEGER,
19        hit         BOOLEAN,
20        sunk        BOOLEAN
21    }
```

Rysunek 5.5 Jednostka Response

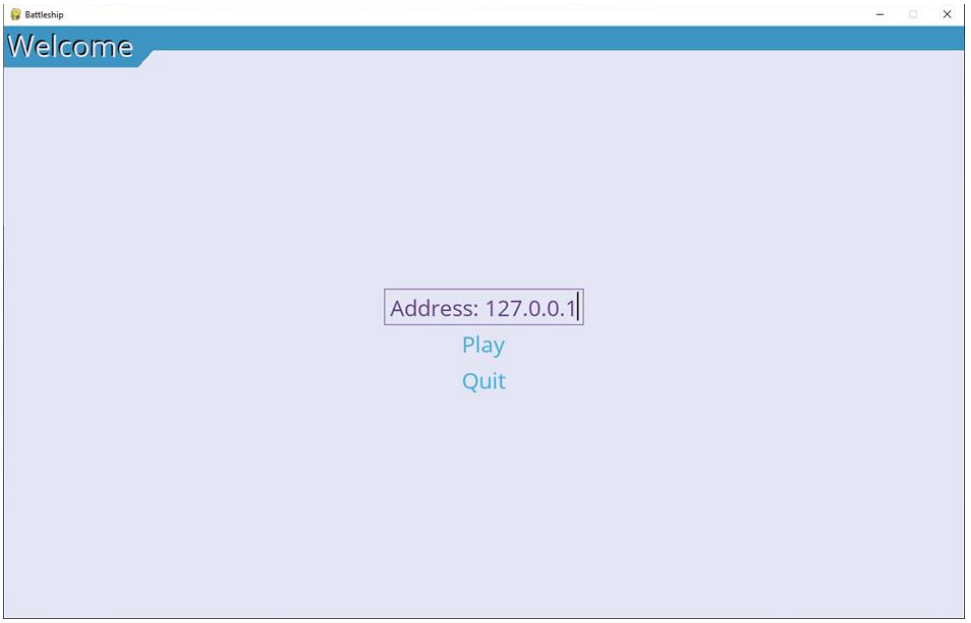
6. Implementacja gry sieciowej

6.1. Klient

6.1.1. Przykład realizacji specyficznej funkcjonalności

6.1.2. Elementy interfejsu graficznego

W stworzonej aplikacji do wykreowania menu (Rysunek 6.1) wykorzystywana jest biblioteka pygame_menu. Pozwala ona na łatwe i szybkie stworzenie prostego ekranu wstępnego. Aplikacja jako przykład edukacyjny, powinna zapewnić tylko niezbędną funkcjonalność. Użytkownik może wpisać dowolny adres IP, rozpocząć grę lub zamknąć program.

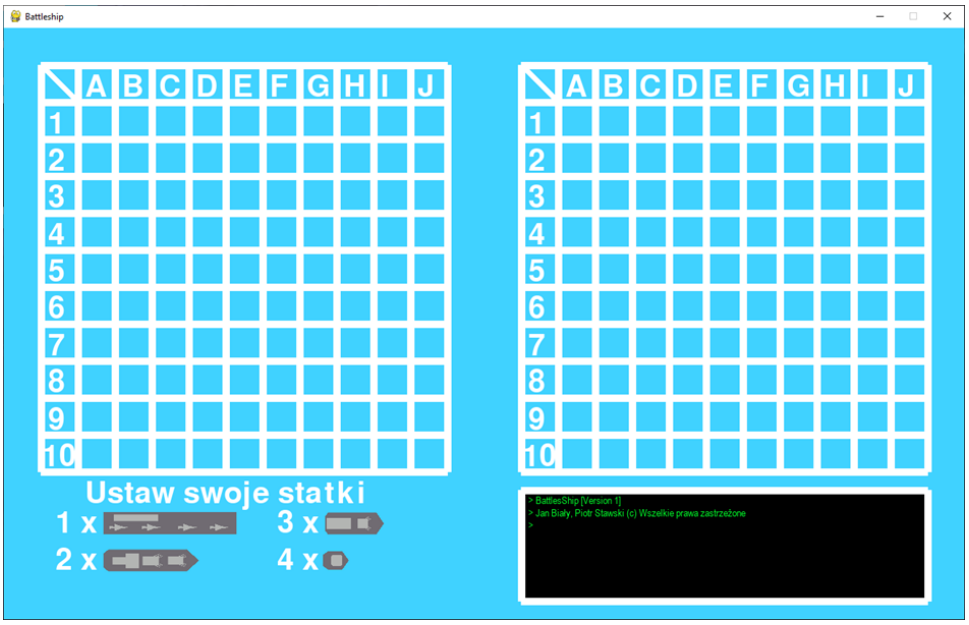


Rysunek 6.1 Aplikacja – menu

Po zestawieniu połączenia i gry z drugim graczem ukazuję się obu użytkownikom plansza, na której będzie prowadzona rozgrywka. Wszystkie jej elementy są stworzone za pomocą narzędzi wbudowanych w sam język lub biblioteki, poza własnoręcznie zrobionymi prostymi modelami statków (Rysunek 6.2). Zostały one wykonane techniką wektorową, aby zapewnić możliwość swobodnego zmienienia wielkości okna aplikacji.

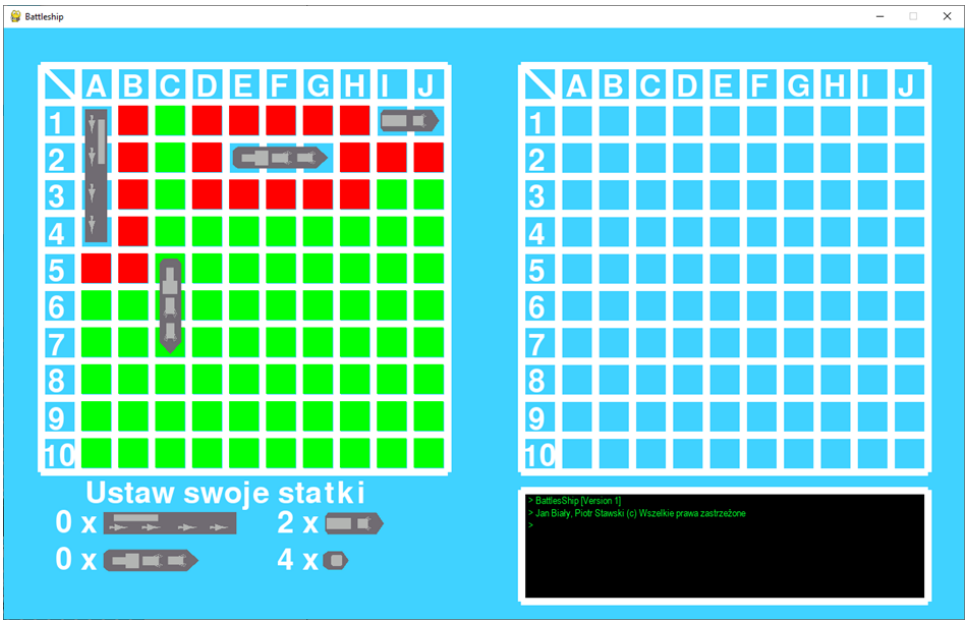
Rysunek 6.2 Aplikacja – statki

Reszta planszy została stworzona korzystając jedynie z narzędzi, które oferuje biblioteka pygame. Jest to zarówno tablica użytkownika, przeciwnika, wszystkie napisy oraz okno w którym wyświetlają się dane wysłane i odebrane z serwera. Okno aplikacji, które ukazuje się gdy zostanie zestawiona gra, zostało zilustrowane na rysunku niżej.



Rysunek 6.3 Aplikacja - faza rozmieszczenia okrętów

Podczas fazy ustawiania statków, gdy użytkownik chce umieścić konkretną jednostkę na planszy, musi kliknąć na jej reprezentacje pod tablicą. Zostaje ona przyczepiona wtedy automatycznie do jego kursora. Pola na których możliwe jest umieszczenie statku są pokazane na zielone, a te na których nie, na czerwono. Pozwala to graczowi w łatwy sposób zinterpretować zasady układania statków na planszy oraz pokazuje mu, która plansza należy do niego. Statki można obracać prawym przyciskiem myszy. Licznik przy konkretnej jednostce pokazuje, ile użytkownik musi ich jeszcze rozstawić na swojej planszy aby rozpocząć grę. Proces rozstawiania statków został zilustrowany na rysunku niżej.

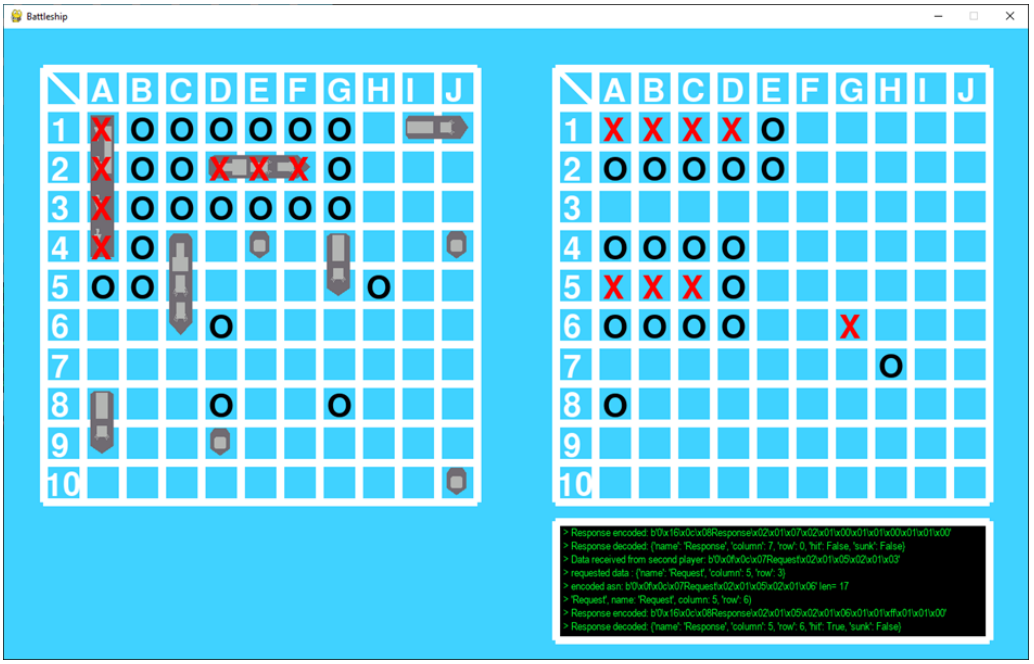


Rysunek 6.4 Aplikacja – faza rozmieszczenia okrętów (wybrany okręt)

Faza gry polega na naprzemiennym oddawaniu strzałów ze swoim przeciwnikiem. Gdy gracz zdecyduje się które pole chce wybrać, klika na nie lewym przyciskiem myszy. Po otrzymaniu informacji z serwera czy został trafiony statek przeciwnika, czy też nie, zarówno na planszy przeciwnika w jego aplikacji jak i użytkownika pojawia się stosowne oznaczenie. Aby korzystać jedynie z zasobów wbudowanych, udany strzał zostaje oznaczony czerwonym znakiem „X”, a chybiony czarnym „O”. Jeśli cała jednostka została zatopiona, wokół niej zostają ustawione symbole niepoprawnego trafienia, aby gracz nie mógł tych pól już niepotrzebnie typować. Ułatwia to użytkownik automatycznie zawęzić poszukiwania kolejnych statków przeciwnika. Bardzo ważnym elementem jest jednak mały i nie pozorny czarny prostokąt, który reprezentuję większość komunikacji z serwerem, a co za tym idzie z drugim graczem. Przedstawia on wiadomości:

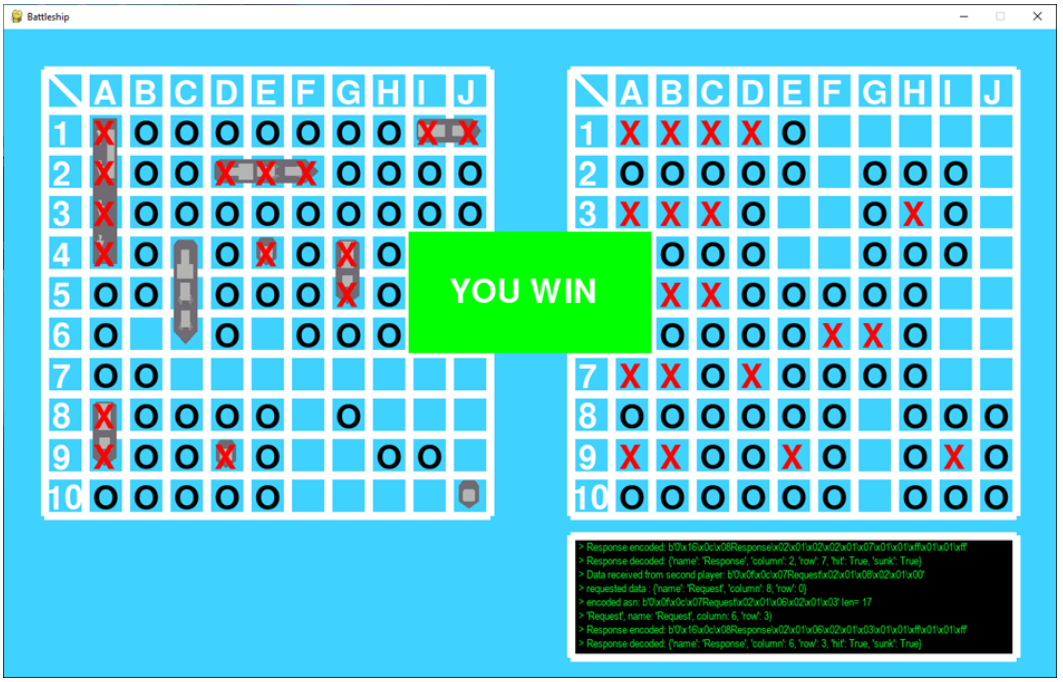
- Zakodowaną i zdekodowana wiadomość Request wysyłaną do serwera w celu sprawdzenia czy statek przeciwnika znajduje się na zadanych koordynatach.
- Zakodowana i zdekodowana wiadomość Response wysyłana do serwera określająca czy na zadanych koordynatach znajdował się statek.

Na TUTAJ JANEK DAJ ODNOSNIK DO RYSUNKU przedtsawiono moment z gry, na którym można zaobserować wszytskie wymienione wyżej elementy.

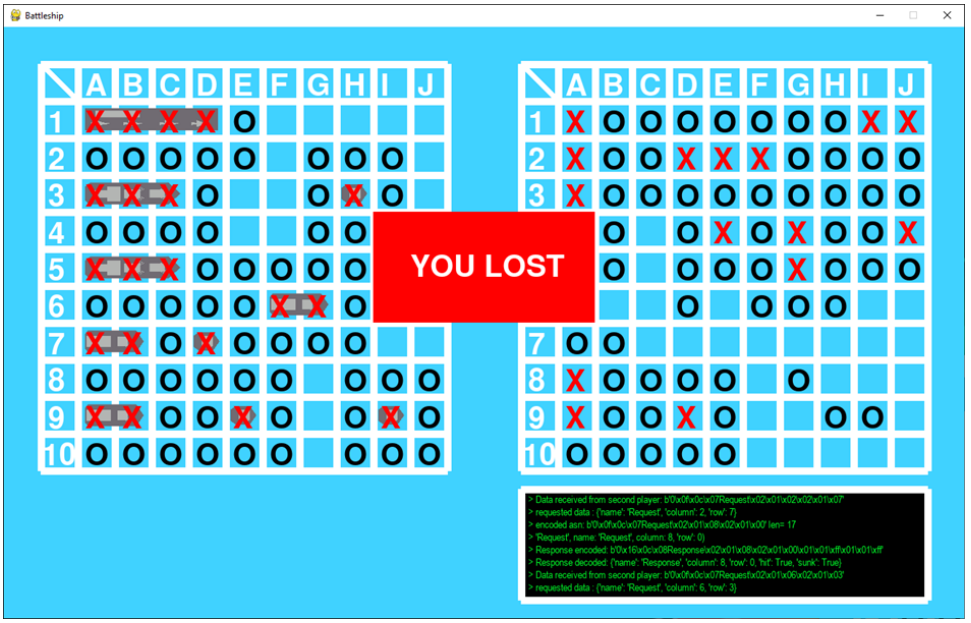


Rysunek 6.5 Aplikacja – faza rozgrywki

Gdy wszystkie statki jednego z graczy zostaną zatopione, gra kończy się. Obaj użytkownicy otrzymują informacje o rezultacie gry w formie prostego okna. Wygranemu ukazuje komunikat „YOU WIN” na zielonym tle, a przegranemu napis „YOU LOSE” na czerwonym.



Rysunek 6.6 Aplikacja – ekran końcowy, wygrana



Rysunek 6.7 Aplikacja – ekran końcowy, przegrana

6.2. Serwer

6.2.1. Przykład realizacji specyficznej funkcjonalności

6.2.2. Elementy interfejsu graficznego

Serwer w momencie uruchomienia wyświetla w konsoli (Rysunek 6.8) na jakich adresach wewnętrznych zostaje uruchomione nasłuchiwanie na połączenie.

```
Server Lan IP:
    192.168.1.212
    192.168.56.1
    169.254.233.98
    169.254.134.72
    192.168.1.230
    169.254.131.173
    127.0.0.1
Waiting for connection, Server Started
```

Rysunek 6.8 Rozpoczęcie działania serwera

W momencie podłączenia klienta do serwera serwer wyświetla w konsoli (Rysunek 6.9) informacje o użytkownikach oraz o wysłaniu powitalnej wiadomości do każdego z użytkowników.

```
Connected to: ('127.0.0.1', 56182) ('127.0.0.1', 56184)
Sent welcome data to ('127.0.0.1', 56182)
Sent welcome data to ('127.0.0.1', 56184)
```

Rysunek 6.9 Serwer - podłączenie dwóch użytkowników

Serwer w momencie otrzymania wiadomości Request wyświetla w konsoli (Rysunek 6.10) jej zakodowaną w formacie BER wartość. A następnie zostaje wyświetlona jej zdekodowana wartość.

```
b'0\x0f\x0c\x07Request\x02\x01\x04\x02\x01\x03'
{'name': 'Request', 'column': 4, 'row': 3}
```

Rysunek 6.10 Serwer - odebranie jednostki Request

Serwer w momencie otrzymania wiadomości Response wyświetla w konsoli (Rysunek 6.11) jej zakodowaną w formacie BER wartość. A następnie zostaje wyświetlona jej zdekodowana wartość.

```
b'0\x16\x0c\x08Response\x02\x01\x04\x02\x01\x03\x01\x01\x00\x01\x01\x00'
{'name': 'Response', 'column': 4, 'row': 3, 'hit': False, 'sunk': False}
```

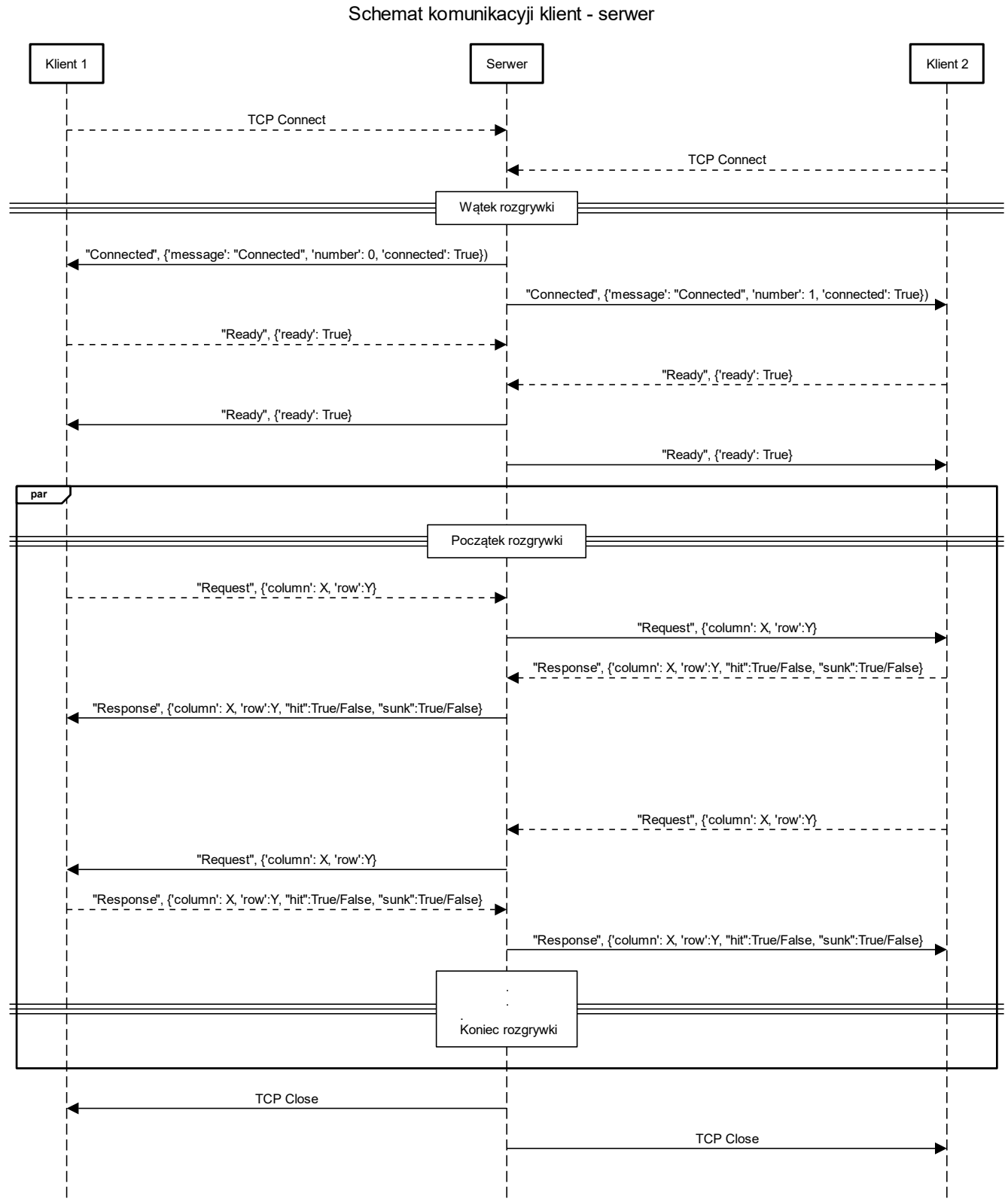
Rysunek 6.11 Serwer - odebranie jednostki Response

6.3. Protokół komunikacyjny

6.3.1. Przykład realizacji specyficznej funkcjonalnościs

Celem protokołu warstwy aplikacji jest wymiana informacji między klientami a serwerem w systemie sieciowym (Rysunek 6.12). Odbywa się ona poprzez połączenie się klienta do serwera, klient oraz serwer oczekują na połączenie się drugiego klienta. W momencie gdy drugi użytkownik połączy się do serwera, uruchamiany jest oddzielny wątek odpowiedzialny za

daną rozgrywkę a następnie wysyłana jest wiadomość zwrotna **Connected** do obu klientów, która świadczy o przejściu do fazy rozmieszczania okrętów. W tym momencie serwer oczekuje na wiadomości potwierdzające gotowość klientów do przejścia do fazy rozgrywki, w momencie gdy serwer otrzyma wiadomości **Ready** od obu klientów następuje odesłanie wiadomości **Ready** do obu klientów. W fazie rozgrywki użytkownik który jako pierwszy podłączył się do serwera rozpoczyna rozgrywkę wysyłając do serwera wiadomość **Request** zawierającą koordynaty X, Y. Następnie serwer przekazuje otrzymaną wiadomość do klienta numer dwa w celu sprawdzenia czy na zadanych koordynatach znajduje się statek przeciwnika. Klient numer dwa odsyła do serwera wiadomość **Response** zawierającą wcześniej otrzymane koordynaty oraz dodatkowe pola hit oraz sunk określające czy na zadanych koordynatach znajdował się statek oraz czy został on zatopiony. Serwer przekazuje do klienta numer jeden dane otrzymane od klienta numer dwa. Następnie klient numer dwa wysyła zapytanie **Request** które jest przesyłane do serwera. Następnie serwer przekazuje otrzymane dane do klienta numer jeden który sprawdza otrzymane koordynaty i odsyła wiadomość **Response** do serwera a serwer przekazuje ją do klienta numer dwa. Wyżej opisany cykl wysyłania wiadomości **Request** i **Response** jest kontynuowany do momentu aż jeden z klientów nie zniszczy wszystkich statków przeciwnika. Następnie następuje zakończenie połączenia między serwerem a klientami.



Rysunek 6.12 Schemat komunikacji klient – serwer

6.3.2. Przykład kodowania i dekodowania jednostek protokołu

```
Connected ::= SEQUENCE{
    name        UTF8String,
    number      INTEGER,
    connected   BOOLEAN
}
```

Rysunek 6.13 Jednostka protokołu - Connected

```
asn.encode("Connected",{"name":"Connected","number":3,"connected":True})
```

```
9    b'0\x11\x0c\tConnected\x02\x01\x03\x01\x01\xff'
```

```
b'0\x11\x0c\tConnected\x02\x01\x03\x01\x01\xff'
```

7. Testowanie i wykorzystanie gry

8. Wnioski

Bibliografia

[1] E. Kosmulska-Bochenek, Wymiana informacji w heterogenicznych systemach sieciowych, Wrocław: Oficyna Wydawnicza Politechniki Wrocławskiej, 2002.

[2] A. Józefiok, CCNA 200-301 Zostań administratorem sieci komputerowych CISCO, Gliwice: Helion, 2020.

[3] „Abstract Syntax Notation One," [Online]. Available: https://pl.wikipedia.org/wiki/Abstract_Syntax_Notation_One. [Data uzyskania dostępu: 30 Grudzień 2022].

[4] „obj-sys ASN.1," [Online]. Available: https://obj-sys.com/asn1tutorial/node124.html. [Data uzyskania dostępu: 5 Styczeń 2023].

[5] „Python 3 Documentation," [Online]. Available: https://docs.python.org/3/installing/index.html. [Data uzyskania dostępu: 12 Styczeń 2023].

[6] „JetBrains - Pycharm," [Online]. Available: https://www.jetbrains.com/pycharm/. [Data uzyskania dostępu: 12 Styczeń 2023].

[7] „Python - installing-packages," [Online]. Available: https://packaging.python.org/en/latest/tutorials/installing-packages/. [Data uzyskania dostępu: 12 Styczeń 2023].

[8] T. Lammle, „CompTIA Network+® Study Guide," Indianapolis, Wiley Publishing, 2009.

[9] C. M. Kozierok, „TCP/IP Guide," San Francisco, No Starch Press, 2005.

[10] D. Paul i D. Harvey, „Python for Programmers," Boston, Pearson, 2019.

[11] R. Marvin, M. Ng'ang'a i A. Omondi, „Python Fundamentals," Birmingham, Packt Publishing, 2018.

[12] „Libraries in Python," [Online]. Available: https://www.geeksforgeeks.org/libraries-in-python/. [Data uzyskania dostępu: 2 Styczeń 2023].

[13] „pygame," [Online]. Available: https://www.pygame.org/wiki/about. [Data uzyskania dostępu: 2 Styczeń 2023].

[14] K. Harrison i M. Will, Beginning Python Games Development : With Pygame, Second Edition, New York: Apress, 2015.

[15] „readthedocs," [Online]. Available: https://pygame-menu.readthedocs.io/en/4.3.4/. [Data uzyskania dostępu: 2 Styczeń 2023].

[16] M. Alex, R. Anna i H. Steve, Python in a Nutshell, 3rd Edition, Sebastopol: O'Reilly Media, Inc., 2017.

[17] „docs.python.org _thread,” [Online]. Available: https://docs.python.org/3/library/_thread.html. [Data uzyskania dostępu: 2 Styczeń 2023].

[18] „docs.python.org socket,” [Online]. Available: https://docs.python.org/3/library/socket.html. [Data uzyskania dostępu: 4 Styczeń 2023].

[19] L. Mark, Programming Python, 4th Edition, Sebastopol: O'Reilly Media, Inc., 2010.

Spis Ilustracji

****
****
PROMOTOR
****
****

Spis treści

Spis zagadnień