WYKŁAD

TELEMETRIA INTERNETOWA

Mikroserwery TCP/IP

Autor: dr inż. Zbigniew Czaja

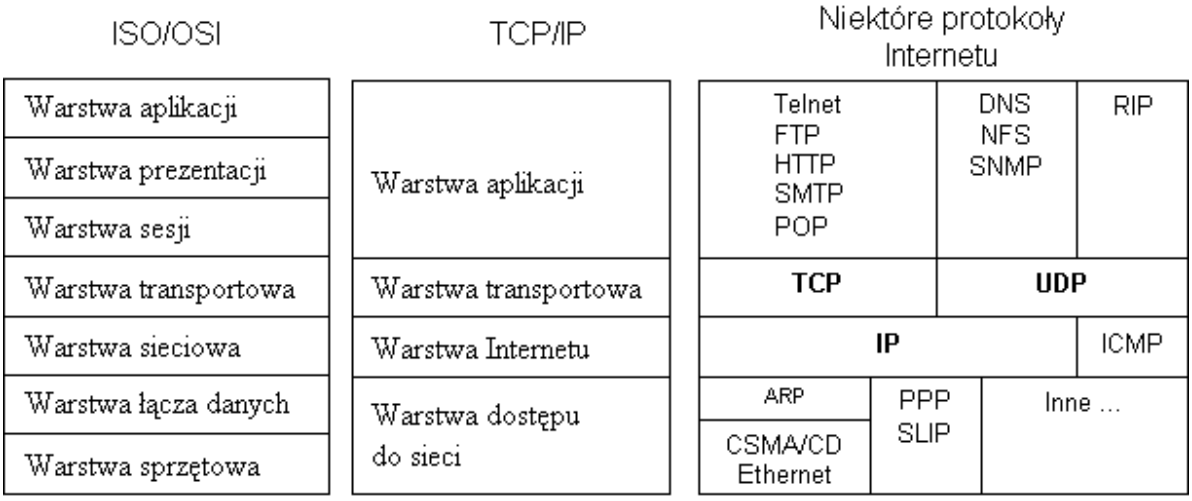
Gdańsk 2006

Spis treści

1. Model warstwowy protokołu TCP/IP

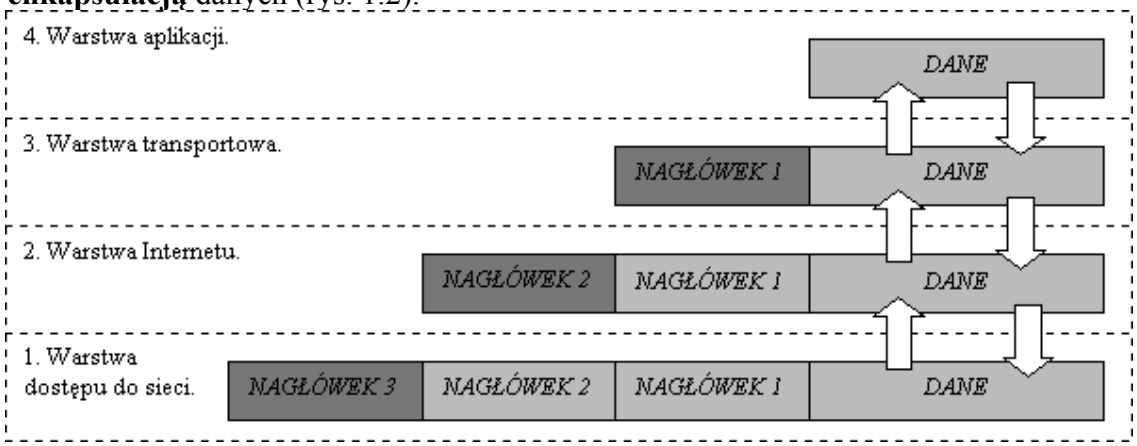
Protokół TCP/IP dzieli się na cztery warstwy (od najniższej):

- warstwa dostępu do sieci,
- warstwa Internet (nazywana inaczej sieciową, międzysieciową lub warstwą IP),
- warstwa transportowa,
- warstwa zastosowań (nazywana inaczej warstwą aplikacji).



Rys. 1.1. Porównanie modelu ISO – OSI i modelu TCP/IP

Podobnie jak w modelu OSI kolejne warstwy dołączają (bądź usuwają, w zależności w którą stronę przesuwają się dane na stosie protokołów) własne nagłówki. Taki proces nazywa się **enkapsulacją** danych (rys. 1.2).

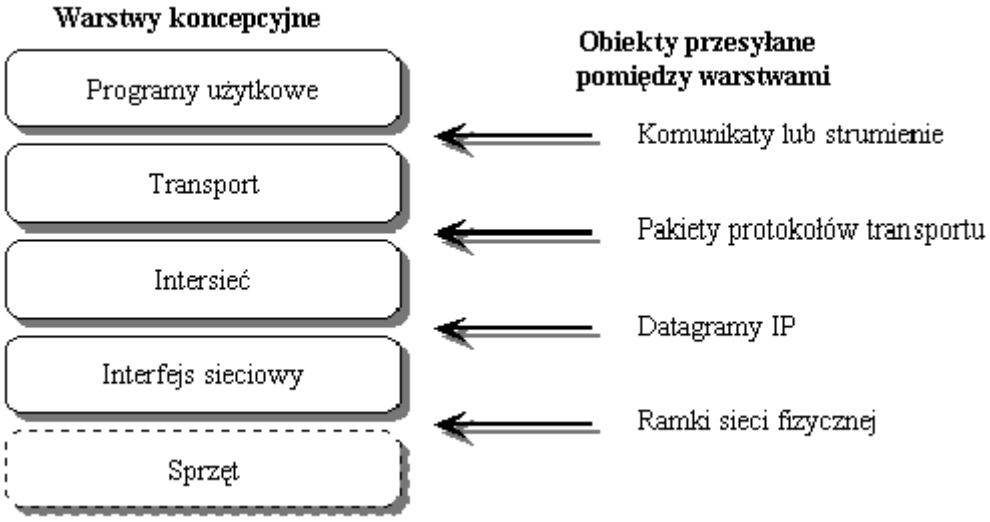


Rys. 1.2. Proces enkapsulacji danych

Każda warstwa ma swoją terminologię określającą dane aktualnie przez nią obrabiane. Ponieważ protokół TCP/IP składa się z dwóch głównych protokołów warstwy transportowej TCP i UDP, więc również w nazewnictwie wprowadzony został podział.

Tabela 1.1. Nazwy jednostek danych dla warstw modelu TCP/IP

Warstwa	**TCP**	**UDP**
Aplikacji	strumień	wiadomość
Transportowa	segment	pakiet
Internetu	datagram	
Dostępu do sieci	ramka	



Rys. 1.3. Obiekty przesyłane pomiędzy warstwami

Warstwa dostępu do sieci (fizyczna) odpowiada za dostarczanie danych do innych urządzeń bezpośrednio dołączonych do sieci. Współpracuje ona bezpośrednio ze sprzętem i sterownikami odpowiedzialnymi za współpracę z siecią Ethernet. W przypadku innych sieci mogą to być protokoły PPP, SLIP lub inne. Warstwa ta współpracuje więc z interfejsem sieciowym (kartą sieciową), modemem lub innym urządzeniem pozwalającym na bezpośrednie połączenie dwóch lub więcej komputerów i separuje resztę warstw od zastosowanych rozwiązań fizycznych (niskopoziomowych). Świadczy ona usługę warstwie wyższej polegającą na wysyłaniu i odbieraniu porcji danych (zwanych ramkami) z komputerów w danej sieci fizycznej.

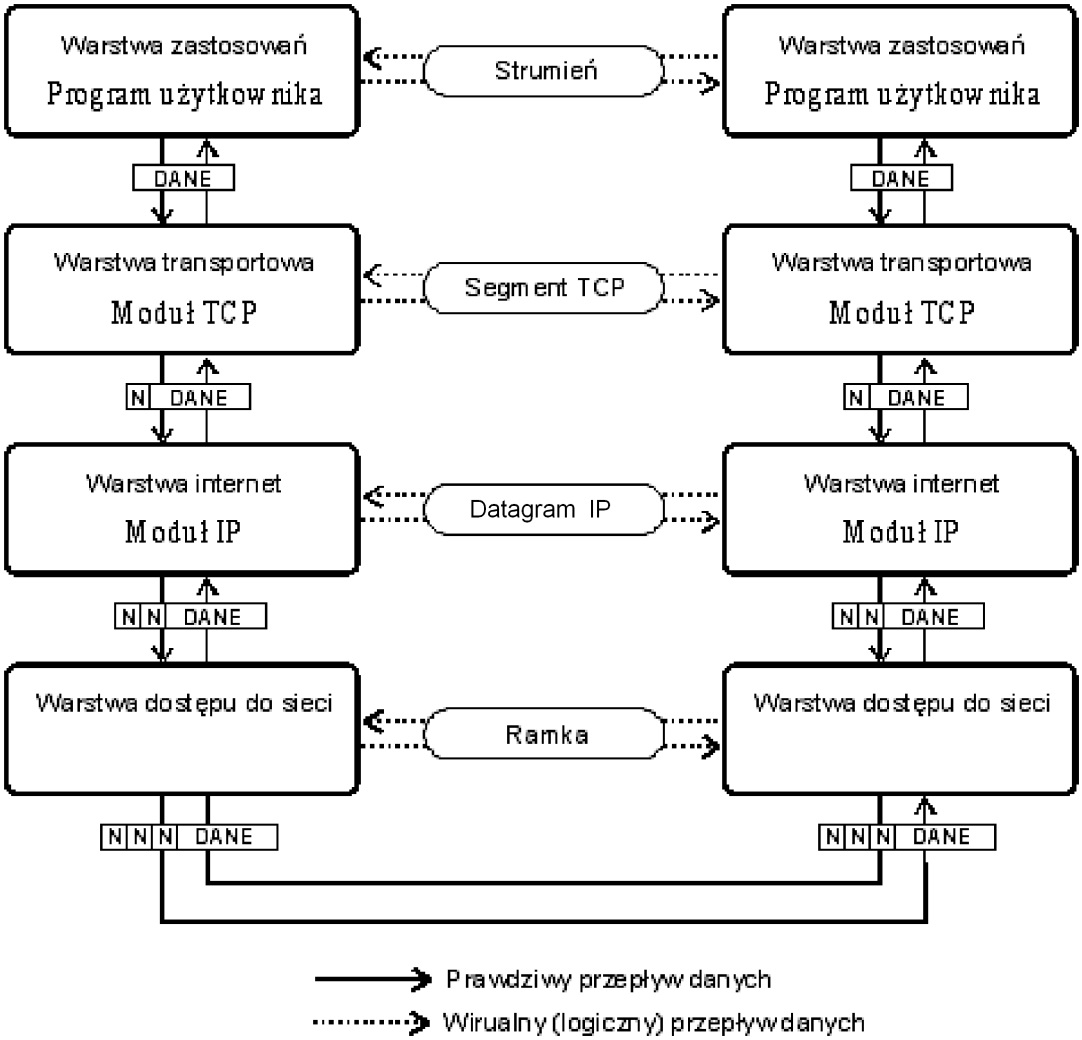
Warstwa Internet (IP) odpowiada za dostarczanie danych do urządzeń nie tylko w danej sieci fizycznej. Organizuje ona ruch tzw. datagramów IP między poszczególnymi sieciami fizycznymi połączonymi w intersieć. Korzysta z usług warstwy dostępu do sieci, sama zaś świadczy usługi dostarczania pakietu do dowolnego komputera w Internecie.

Warstwa transportowa odpowiedzialna jest za niezawodną wymianę danych z dowolnym komputerem w Internecie. Organizuje też i utrzymuje tzw. sesje, czyli wirtualne połączenia między komputerami. Korzysta z warstwy IP, sama zaś dostarcza usług niezawodnego transportu danych.

Warstwa zastosowań jest najwyżej położona. Tej warstwie odpowiadają wszelkie programy (aplikacje) internetowe korzystające z warstwy transportowej. Tu znajdują się wszelkie konkretne zastosowania Internetu - przesyłanie plików (FTP), poczty (SMTP) i inne.

Współpraca między warstwami polega na świadczeniu usług przez warstwy niższe warstwom wyższym. Związane to jest także z przepływem danych w dół sterty warstw (przy wysyłaniu danych) i w górę (przy odbieraniu). Moduł warstwy zastosowań (najczęściej program użytkownika) wysyła dane do warstwy transportowej. Ta odpowiednio formatuje je (dzieli lub łączy, dodaje nagłówek) i wysyła do warstwy IP. Ta z kolei dodaje swój nagłówek i wysyła do warstwy dostępu do sieci. I ta warstwa dołącza swój nagłówek (związany ze sprzętowym rozwiązaniem komunikacji, np. tzw. nagłówek MAC) i wysyła pakiet fizycznie do sieci.

Podobna droga, ale w drugą stronę, czeka dane w komputerze je odbierającym. Pakiet wędruje ku górze i jest pozbawiany odpowiednich nagłówków, by wreszcie dotrzeć do warstwy zastosowań w formie identycznej porcji danych jaką wysłała warstwa zastosowań w komputerze wysyłającym. Przedstawiony wyżej opis jest faktycznym obiegiem danych.



Rys. 1.4. Współpraca między warstwami w TCP/IP

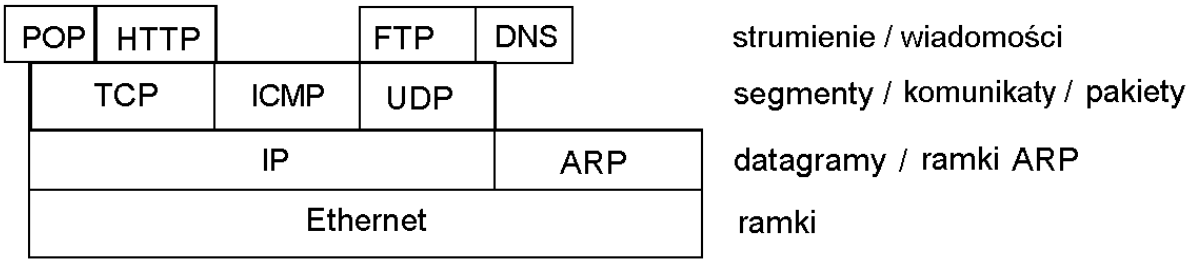
W koncepcji warstw istnieje jeszcze coś takiego jak wirtualny (logiczny) obieg danych. Występuje on pomiędzy odpowiadającymi sobie warstwami w odległych systemach. Warstwy dostępu do sieci wymieniają między sobą ramki. Warstwy IP wysyłają do siebie pakiety IP - choć w rzeczywistości muszą się ze sobą komunikować poprzez swoje niższe warstwy, to z logicznego punktu widzenia istnieje między nimi wirtualne połączenie, które pozwala na wymianę pakietów. Podobnie jest z warstwami transportowymi, które wysyłają między sobą poprzez swój wirtualny kanał pakiety danych (segmenty TCP) i inne komunikaty zapewniające utrzymanie sesji i niezawodne dostarczenia danych (potwierdzanie).

Między warstwami zastosowań istnieje również wirtualny kanał pozwalający na wysyłanie strumienia danych w obie strony. Tę warstwę stanowią najczęściej programy internetowe i z ich punktu widzenia istnieje bezpośrednie połączenie strumieniowe z innym programem działającym na komputerze odległym. Dzięki temu programista nie musi znać budowy i zasady działania warstw niższych - musi jedynie poznać usługi świadczone przez warstwę transportową.

2. Implementacja minimalna stosu TCP/IP w mikroserwerach wbudowanych

Na potrzeby mikroserwerów wbudowanych TCP/IP poszczególne elementy stosu protokołów TCP/IP są upraszczane, tak aby moc obliczeniowa mikrokontrolerów wystarczyła do ich implementacji.

W dalszej części rozdziału zostaną po kolei przedstawione poszczególne warstwy „minimalnego" stosu TCP/IP wraz z ich implementacją w mikroserwerze.



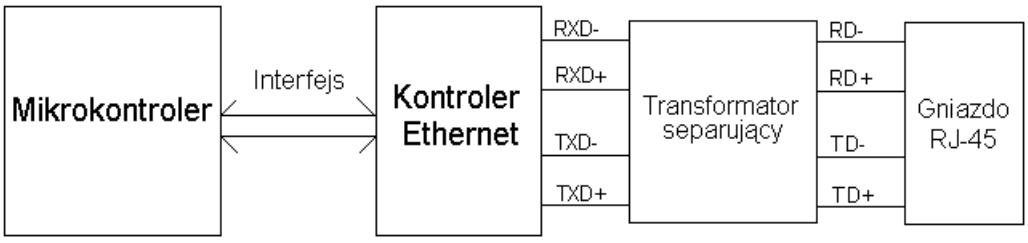
Rys. 2.1. Minimalny stos protokołów TCP/IP (poszczególne warstwy ułożone zgodnie z kapsulacją i enkapsulacją danych)

Warstwę Ethernet możemy podzielić na:

- **warstwę sprzętową** – sposób podłączenia kontrolera Ethernet do mikrokontrolera oraz sterowanie nim,
- **właściwą warstwę Ethernet** – obsługa ramek Ethernet, kodowanie danych kodem Manchester, protokół CSMA/CD, parametry elektryczne i czasowe sygnałów w sieci komputerowej.

2.2. Warstwa sprzętowa

Na rys. 2.2 pokazano ogólny schemat blokowy mikroserwera TCP/IP. Składa się on mikrokontrolera nadzorującego pracą kontrolera Ethernet i zaimplementowanymi protokołami IP, TCP i aplikacji, następnie transformatora separującego i gniazda RJ45.



Rys. 2.2. Ogólny schemat blokowy mikroserwera TCP/IP

Mikrokontroler za pośrednictwem dedykowanego interfejsu steruje pracą kontrolera sieci Ethernet. Obecnie do budowy mikroserwerów TCP/IP stosowane są kontrolery Ethernet z interfejsem ISA, który był stosowany w komputerach PC, oraz z niezależnym od medium fizycznego interfejsem MII.

Kontroler Ethernet może być również zaszyty w mikrokontrolerze, wówczas jest traktowany jako jeden z jego interfejsów (np. mikrokontroler MC9S12NE64 firmy Freescale Semiconductor) a jego obsługa jest taka, jak pozostałych urządzeń peryferyjnych mikrokontrolera, tzn. sterowanie interfejsem Ethernet odbywa się za pomocą wpisów i

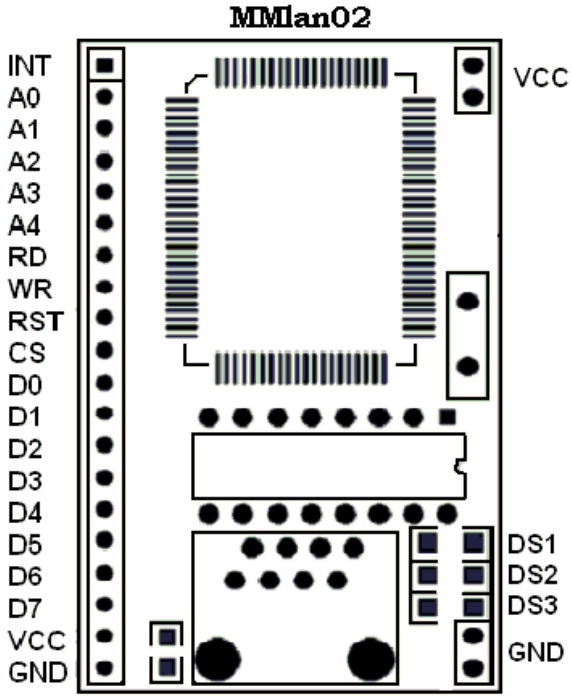
odczytów do/z jego rejestrów sterujących, statusowych i danych zawartych w obszarze adresowania pamięci danych lub obszarze urządzeń we/wy.

Transformator separujący ma za zadanie galwanicznie odizolować mikroserwer od sieci Ethernet oraz, o ile jest to wymagane, uformować przebieg trapezoidalny z prostokątnego podawanego na różnicowym wyjściu TxD+, TxD-. Do gniazda RJ-45 wpinamy kabel standardu 10BaseT łączący mikroserwer z siecią Internet.

Warstwa sprzętowa zostanie przedstawiona na przykładzie mikroserwera bazującego na module opartym na kontrolerze RLT8019AS (moduł **MMLan2**).

2.2.1. Moduł MMlan2

Moduł MMLan2 oparty jest na kontrolerze sieci Realtek 8019AS i jest przystosowany do 8-bitowej pracy z dowolnym mikrokontrolerem. Jego zastosowanie eliminuje potrzebę projektowania i montażu bloku interfejsu sieciowego mikroserwera. Do połączenia z tym modułem wykorzystywane jest 16 linii we/wy mikrokontrolera.



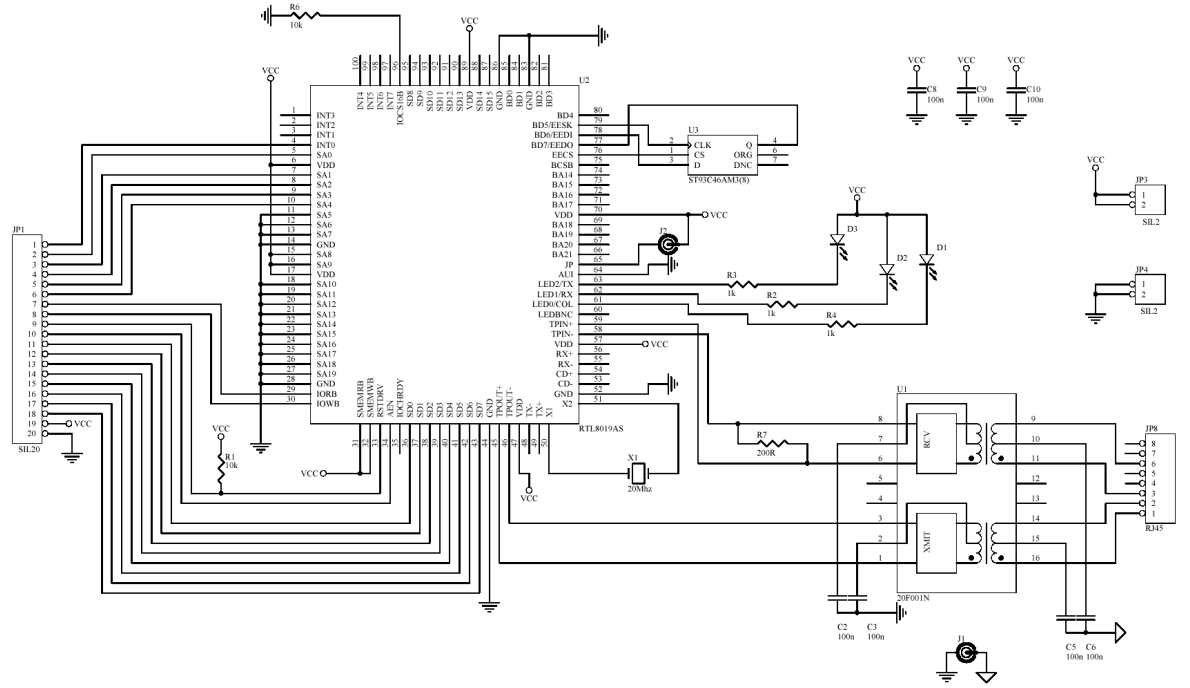
Rys. 2.3. Widok ogólny modułu MMLan2

Opis wyprowadzeń:

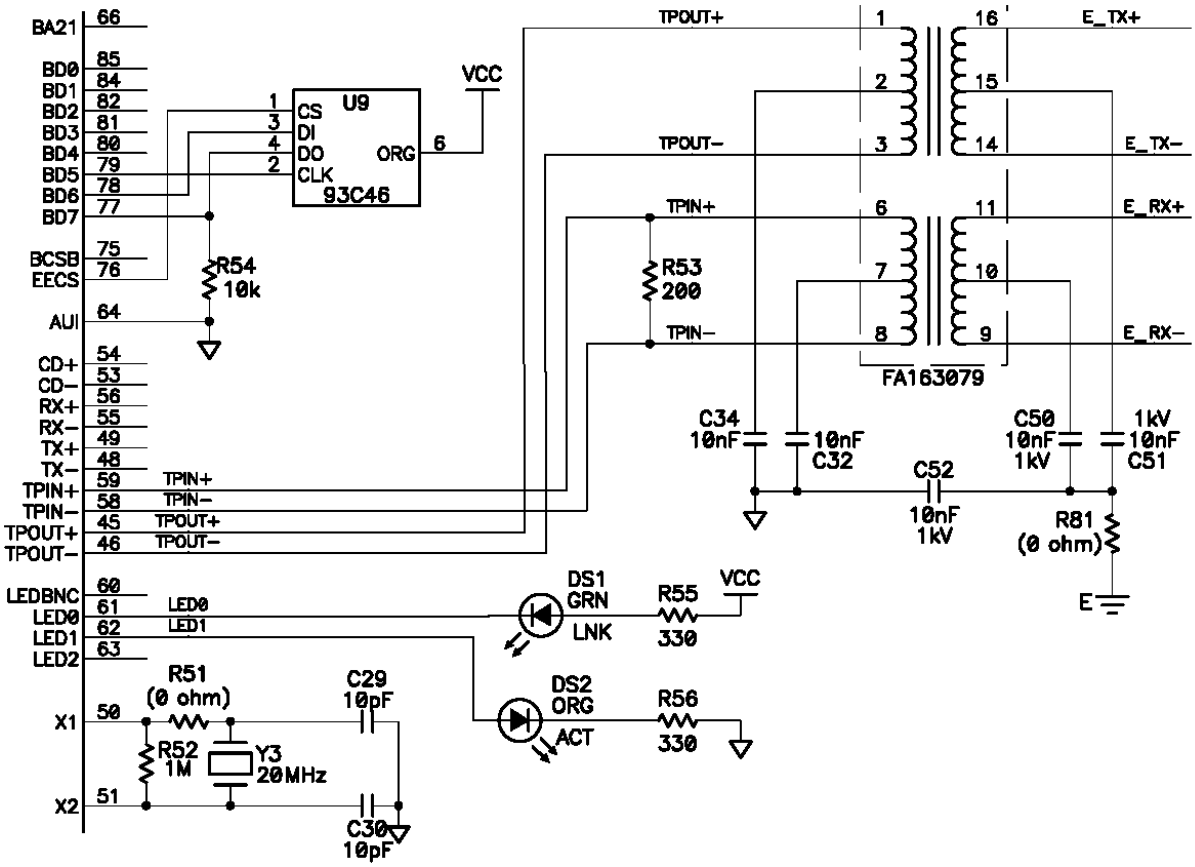
- INT – wyjście przerwanie (aktywne zboczem narastającym),
- A0-A4 –magistrala adresowa układu RTL8019AS,
- /RD – wejście strobujące zapis (aktywne stanem niskim),
- /WR – wejście strobujące odczyt (aktywny stanem niskim),
- D0-D7 – dwukierunkowa magistrala danych układu RTL8019AS,
- VCC –napięcia zasilania,
- GND –masa.

Dioda LED DS1 sygnalizuje kolizję w sieci, LED DS2 wysyłanie danych, a LED DS3 przychodzenie danych. Pin CS służący do wyboru układu podłączony jest na stałe do masy. Wyjście przerwania MMLan2 nie jest wykorzystywane. Moduł został wykonany w

technologii SMD na dwuwarstwowej płytce o wymiarach 51mm x 30mm. Piny zostały wyprowadzone z rastrem 0.1" (2.54mm).



Rys. 2.4. Schemat ideowy modułu MMlan2



Rys. 2.5. Fragment aplikacji układu RTL8019AS

2.2.2. Układ RTL8019AS firmy Realtek

RTL8019AS jest zintegrowanym kontrolerem sieci Ethernet umożliwiającym proste wykonanie karty zgodnej z NE2000 Plug and Play z możliwościami transmisji full-duplex i trybami obniżonego poboru mocy.

Komunikacja pomiędzy RTL8019AS a mikrokontrolerem odbywa się za pomocą magistrali ISA (w module MMLan2 wykorzystywane są: 8 linii danych i 5 najmłodszych linii adresowych, sygnały zapisu, odczytu, sterujące). Sterowanie kontrolerem polega na wpisywaniu odpowiednich wartości do rejestrów sterujących i konfiguracyjnych kontrolera oraz na odczycie jego rejestrów statusu i danych. Wybór danego rejestru do zapisu/odczytu odbywa się przez wystawienie jego adresu na 5-liniowej magistrali adresowej.

Rejestry RTL8019AS można podzielić na dwie grupy ze względu na ich adres i funkcje:
- zgodne z NE2000,
- związane z funkcją Plug and Play.

Grupa rejestrów związana z funkcją Plug and Play zostanie pominięta ponieważ nie jest wykorzystywana przez aplikacje mikroserwerów TCP/IP.

Grupa rejestrów zgodna z NE2000 składa się z 4 stron rejestrów. Strony są wybierane bitami PS0 i PS1 w rejestrze CR. Każda strona zawiera 16 rejestrów. Poza rejestrami zgodnymi z NE2000 RTL8019AS posiada rejestry służące do konfiguracji przyszłych rozszerzeń.

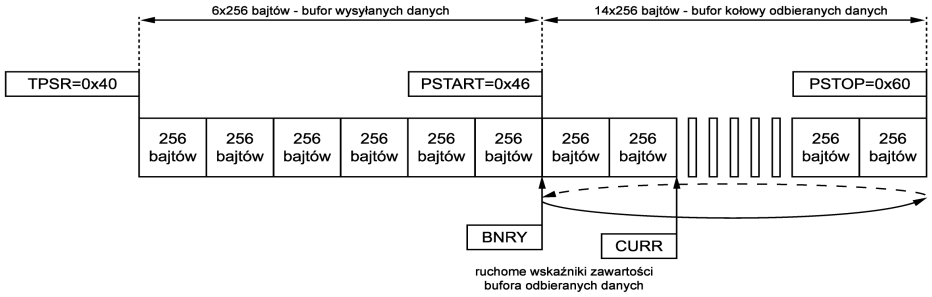
Tabela 2.1. Rejestry układu RTL8019AS

Nr [hex]	**Strona 0**		**Strona 1**	**Strona 2**	**Strona 3**	
	odczyt	zapis	odczyt/zapis	odczyt/zapis	odczyt	zapis
00	CR	CR	CR	CR	CR	CR
01	CLDA0	PSTART	PAR0	PSTART	9346CR	9346CR
02	CLDA1	PSTOP	PAR1	PSTOP	BPAGE	BPAGE
03	BNRY	BNRY	PAR2	-	CONFIG0	
04	TSR	TPSR	PAR3	TPSR	CONFIG1	CONFIG1
05	NCR	TBCR0	PAR4	-	CONFIG2	CONFIG2
06	FIFO	TBCR1	PAR5	-	CONFIG3	CONFIG3
07	ISR	ISR	CURR	-	-	TEST
08	CRDA0	RSAR0	MAR0	-	CSNSAV	-
09	CRDA1	RSAR1	MAR1	-	-	HTLCLK
0A	8019ID0	RBCR0	MAR2	-		
0B	8019ID1	RBCR1	MAR3	-	INTR	-
0C	RSR	RCR	MAR4	RCR	-	
0D	CNTR0	TCR	MAR5	TCR	CONFIG4	-
0E	CNTR1	DCR	MAR6	DCR	-	-
0F	CNTR2	IMR	MAR7	IMR	-	-
10-17	RDMAPORT – Remote DMA Port					
18-1F	Reset Port					

Charakterystyka rejestrów wykorzystywanych w mikroserwerach:
- **CR** – (Command Register) – przy pomocy tego rejestru wybierana jest odpowiednia strona rejestrów oraz włączany i wyłączany odbiór i wysłanie pakietów.
- **ISR** – (Interrupt Status Register) – wskazuje źródło wywołania przerwania

- **IMP** – (Interrupt Mask Register) – bity tego rejestru w zależności od ustawienia odblokowują lub blokują możliwość użycia przerwań
- **DCR** – (Data Configuration Register) – służy do wyboru 8 lub 16 bitowego trybu przesyłania danych do pamięci
- **TCR** – (Transmit Configuration Register) – służy do konfiguracji parametrów transmisji pakietów do sieci
- **TSR** – (Transmit Status Register) – wskazuje status transmisji pakietu
- **RCR** – (Receive Configuration Register) – ustala parametry układu dotyczące odbierania pakietów
- **RSR** – (Receive Status Register) – zawiera informacje o ostatnio odebranym pakiecie
- **CLDA0,1** – (Current Local DMA Register) – rejestry zawierają adres lokalnego DMA
- **TPSR** – (Transmit Page Start Register) – definiuje adres początkowy pamięci dla wysyłanych pakietów
- **PSTART** – (Page Start Register) – definiuje początkowy adres kołowego bufora odbioru danych
- **PSTOP** – (Page Stop Register) – definiuje końcowy adres kołowego bufora odbioru danych
- **BNRY** – (Boundary Register) – pierwszy z dwóch ruchomych wskaźników określających zawartość bufora odbioru danych
- **CURR** – (Current Page Register) – drugi z dwóch ruchomych wskaźników określających zawartość bufora odbioru danych
- **TBCR0,1** – (Transmit Byte Count Register) – rejestry zawierają liczbę bajtów do wysłania
- **CRDA0,1** – (Current Remote DMA Address Register) – rejestry zawierają bieżący adres zdalnego DMA
- **RSAR0,1** – (Remote Start Address Registers) – rejestry wskazują początek danych do wysłania
- **RBCR0,1** – (Remote Byte Count Registers) – rejestry określają ilość danych do wysłania zdalnego DMA
- **PAR0-5** (Physical Address Registers) – rejestry zawierają fizyczny adres kontrolera sieci

Pamięć kontrolera przeznaczona do wysyłania i odbierania datagramów została podzielona na strony o rozmiarze 256 bajtów każda. Dla pakietów przeznaczonych do wysłania zarezerwowanych zostało 6 stron pamięci natomiast dla pakietów przychodzących z sieci zarezerwowano 14 stron pamięci. Obszar pamięci przeznaczony do odbioru danych został logicznie zorganizowany w bufor kołowy. Oznacza to, że kiedy odbierane dane osiągną koniec pamięci przeznaczonej na ich odbiór to zostaną zapisywane na początku obszaru pamięci odbioru danych. Realizacja bufora kołowego wykonywana jest przy pomocy dwóch wskaźników ruchomych BNRY oraz CURR. Graficzna prezentacja podziału pamięci układu została przedstawiona na rys. 2.6.



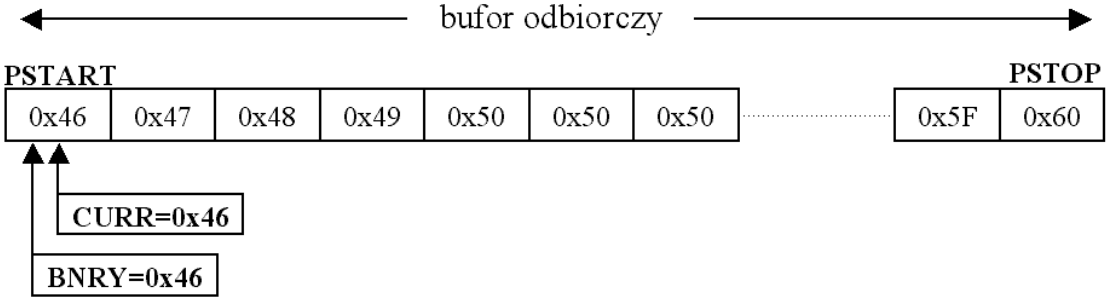
Rys. 2.6. Podział pamięci układu RTL8019AS

Rejestry przedstawione poniżej umożliwiają korzystanie z bufora wewnętrznego układu (podano także wartości po inicjalizacji):

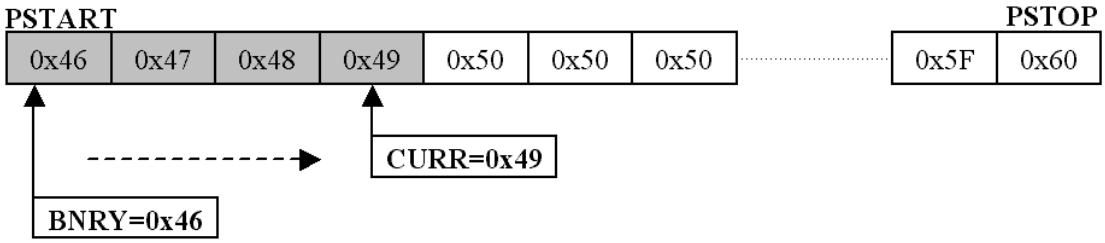
- **PSTART** - adres pierwszej strony bufora odbioru = 0x46
- **PSTOP** - adres ostatniej strony bufora odbioru = 0x60
- **BNRY** - adres wskazujący ostatnią odczytaną stronę bufora odbioru = 0x46
- **CURR** - adres bieżącej strony odbioru; wskazuje, gdzie będą wpisywane dane przychodzące = 0x46
- **RBCR0,1** - licznik danych odczytywanych przez DMA = 0
- **RSAR0,1** - bieżący adres odczytu z bufora
- **TPSR** - adres pierwszej strony bufora transmisji = 0x40
- **RDMAPORT-** adres, gdzie zapisuje/odczytuje się transmitowany/odbierany bajt z bufora kontrolera Ethernetu

Para ruchomych wskaźników **BNRY** (*boundary*) i **CURR** (*current*) umożliwia kontrolę nad danymi wpisywanymi do bufora z sieci i odczytywanymi przez mikrokontroler. Stan, kiedy **BNRY** = **CURR** świadczy o pustym buforze. Ten warunek jest sprawdzany w celu wykrycia pojawienia się nowej ramki w buforze. Kolejne stadia odbierania ramki z sieci do pamięci mikrokontrolera za pośrednictwem kontrolera RTL8019AS przedstawia rys. 2.7.

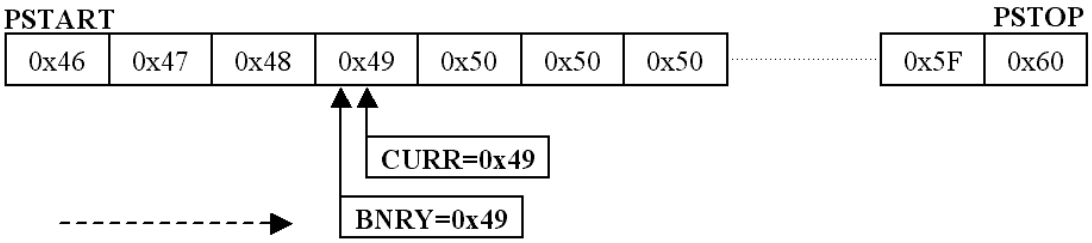
Odczyt ramki z sieci przez mikrokontroler, za pośrednictwem układu RTL8019AS podano na poniższych rysunkach.



Rys. 2.7a. Położenie wskaźników po inicjalizacji kontrolera RTL8019AS



Rys. 2.7b. Odebranie przez kontroler ramki o rozmiarze 896 bajtów



Rys. 2.7c. Odczytanie buforowanej ramki przez mikrokontroler

Wskaźniki BNRY oraz CURR po inicjalizacji układu mają wartość równą 0x46, co odpowiada początkowi bufora kołowego. W momencie odebrania przez układ ramki z sieci wartość wskaźnika CURR jest zwiększana o liczbę odebranych danych. Przy czym jeżeli rozmiar ramki nie jest wielokrotnością 256 bajtów i jej końcówka zajmuje tylko część strony pamięci to wskaźnik CURR przyjmie wartość wskazującą na pierwszą wolną stronę pamięci. Po odczytaniu przez mikrokontroler danych odebranych przez kontroler sieci wskaźnik BNRY zwiększany jest o liczbę odczytanych danych. Równość wskaźników BNRY oraz CURR oznacza, że w pamięci kontrolera sieci nie ma danych do odebrania.

Wysłanie ramki sprowadza się do umieszczenia danych do wysłania w przestrzeni bufora wysyłanych danych i wpisania do rejestru CR odpowiedniej wartości inicjującej transmisję ramki. Wysyłana ramka nie musi zawierać początkowych bajtów synchronizacji, bajta sygnalizującego początek ramki, ani sumy kontrolnej na końcu, gdyż te wszystkie pola są automatycznie uzupełniane przez układ RTL8019AS.

2.2.3. Sterowanie układem RTL8019AS

Sterowanie kontrolerem sieci RTL8019AS zostanie przedstawione na przykładzie mikroserwera sterowanego mikrokontrolerem ATmega32 firmy Atmel. Port C mikrokontrolera służy do transmisji danych z/do kontrolera sieci, na porcie B są ustawiane adres, pod który mają być zapisywane dane, a na porcie D sygnały strobujące zapis i odczyt danych.

Przy odczycie danych port C konfigurowany jest jako port wejściowy, a na linii RD (PD7) ustawiany jest stan niski, powoduje to odczytanie bajta danych z kontrolera sieci z pod adresu podanego na liniach adresowy A0-A4 (PB0-PB4).

Zapis do kontrolera sieci odbywa się w podobny sposób. Port C jest ustawiany na wyjście, na liniach adresowych A0-A4 (PB0-PB7) jest podawany adres, pod który ma zostać zapisany bajt danych, kiedy bajt jest już wystawiony na porcie C to na linii WR (PD6) kontrolera RTL8019AS podawany jest stan niski, co powoduje zapis 8 bitów danych pod wskazany adres.

Na listingu 2.1. podano deklarację sygnałów sterujących kontrolerem sieci.

```
#define RTL_ADDRESS_PORT        PORTB
#define RTL_ADDRESS_DDR         DDRB

#define RTL_DATA_PORT           PORTC
#define RTL_DATA_DDR            DDRC
#define RTL_DATA_PIN            PINC

#define RTL_CONTROL_PORT        PORTD
#define RTL_CONTROL_DDR         DDRD
#define RTL_CONTROL_READPIN     PD7
#define RTL_CONTROL_WRITEPIN    PD6

#define RTL_CLEAR_READ   cbi(RTL_CONTROL_PORT, RTL_CONTROL_READPIN)
#define RTL_SET_READ     sbi(RTL_CONTROL_PORT, RTL_CONTROL_READPIN)
#define RTL_CLEAR_WRITE  cbi(RTL_CONTROL_PORT, RTL_CONTROL_WRITEPIN)
#define RTL_SET_WRITE    sbi(RTL_CONTROL_PORT, RTL_CONTROL_WRITEPIN)

#define RTL_RESET_PORT  PORTD
#define RTL_RESET_DDR   DDRD
#define RTL_RESET_PIN   PD5
```

Listing. 2.1. Deklaracje sygnałów sterujących kontrolerem RTL8019AS

Sterowanie oraz konfiguracja kontrolera sieci RTL8019AS odbywa się przez wpisywanie odpowiednich wartości do rejestrów układu. Lista rejestrów oraz ich opis zostały przedstawione we wcześniejszej części pracy.

Inicjalizacja kontrolera sieci

Przed rozpoczęciem pracy kontroler sieci wymaga inicjalizacji. Procedura inicjalizacji kontrolera sieci RTL8019AS:

- **CR**, **0x41** – wybrana zostaje pierwsza strona rejestrów, praca układu jest zatrzymana,
- **CURR**, **0x46** – ustawiana jest początkowa wartość wskaźnika odczytu danych,
- **PAR0-5**, **MyMac0-5** – zapis adresu MAC do rejestrów kontrolera sieci,
- **CR**, **0x21** – zatrzymanie kontrolera, wybrana zerowa strona rejestrów,
- **DCR**, **0x58** – ustawiany tryb pracy kontrolera: dane przesyłane po 8 bitów, Little Endian
- **RCR**, **0x04** – ustawienie parametrów akceptowanych pakietów: akceptowanie pakietów rozgłoszeniowych oraz pakietów z adresem MAC kontrolera, odrzucane są pakiety z błędami,
- **TCR**, **0x02** – ustawienie kontrolera w testowy tryb loopback,
- **RBCR0-1**, **0x00** – wyzerowanie rejestrów określających długość danych do wysłania,
- **TPSR**, **0x40** – ustawiany jest początkowy adres bufora do wysyłania danych,
- **PSTART**, **0x46** – ustawiany jest początkowy adres bufora odbioru danych,
- **BNRY**, **0x46** – ustawiana jest początkowa wartość wskaźnika odczytu danych,
- **PSTOP**, **0x60** – ustawiany jest końcowy adres bufora odbioru danych,
- **ISR**, **0xff** – zerowanie przerwań,
- **IMR**, **0x11** – zezwolenie na przerwanie od pakietu przychodzącego
- **TCR**, **0x00** – konfigurowane są parametry wysyłania pakietu: normalny tryb wysyłania, generowana jest suma kontrolna ramki oraz dodawane są początkowe bity synchronizacyjne
- **CR**, **0x22** – uruchomienie kontrolera sieci RTL8019AS, konfiguracja jest zakończona, układ jest gotowy do pracy

```
#define CR              0x00
#define PSTART          0x01
#define PAR0            0x01
#define CR9346          0x01
#define PSTOP           0x02
#define BNRY            0x03
#define TSR             0x04
#define TPSR            0x04
#define TBCR0           0x05
#define NCR             0x05
#define TBCR1           0x06
#define ISR             0x07
#define CURR            0x07
#define RSAR0           0x08
#define CRDA0           0x08
#define RSAR1           0x09
#define CRDA1           0x09
#define RBCR0           0x0A
#define RBCR1           0x0B
#define RSR             0x0C
#define RCR             0x0C
#define TCR             0x0D
#define CNTR0           0x0D
#define DCR             0x0E
#define CNTR1           0x0E
```

```
#define IMR             0x0F
#define CNTR2           0x0F
#define RDMAPORT        0x10
#define RSTPORT         0x18

#define ISR_PRX 0
#define ISR_PTX 1
#define ISR_OVW 4
#define ISR_RDC 6
#define ISR_RST 7

#define TXSTART_INIT    0x40
#define RXSTART_INIT    0x46
#define RXSTOP_INIT     0x60

#define ETHERNET_MIN_PACKET_LENGTH      0x3C    //60
#define ETHERNET_HEADER_LENGTH          0x0E    //14

#define IP_TCP_HEADER_LENGTH 40
#define TOTAL_HEADER_LENGTH (IP_TCP_HEADER_LENGTH+ETHERNET_HEADER_LENGTH)
```

Listing. 2.2. Deklaracje wartości wpisywanych do rejestrów RTL8019AS

Poniższa funkcja konfiguruje porty mikrokontrolera. Linie adresowe A0-A4 są ustawiane jako wyjścia (Port B konfigurowany jako port wyjściowy), linie RD (PD7), WR (PD6), RST (PD5) są liniami wyjściowymi ustawionymi przez funkcję w stan wysoki. Jest ona wykonywana na początku procedury inicjalizacyjnej kontrolera sieci.

```
void RTLsetup_ports(void)
{
        RTL_ADDRESS_DDR = 0xFF;
        RTL_DATA_PORT = 0xFF;

        sbi( RTL_CONTROL_DDR, RTL_CONTROL_READPIN );
        sbi( RTL_CONTROL_DDR, RTL_CONTROL_WRITEPIN );
        sbi( RTL_CONTROL_PORT, RTL_CONTROL_READPIN );
        sbi( RTL_CONTROL_PORT, RTL_CONTROL_WRITEPIN );

        sbi( RTL_RESET_DDR, RTL_RESET_PIN );
}
```

Listing 2.3. Konfiguracja linii portów mikrokontrolera

```
void RTLhw_reset(void)
{
        sbi(RTL_RESET_PORT, RTL_RESET_PIN);
        Delay10ms(1);
        cbi(RTL_RESET_PORT, RTL_RESET_PIN);
        Delay10ms(1);
}
```

Listing 2.4. Reset kontrolera sieci

Poniżej przedstawiono podstawowe funkcje związane z wpisywaniem i odczytywaniem danych do/z rejestrów kontrolera sieci.

```
void RTLwrite(unsigned char address, unsigned char data)
{
        RTL_ADDRESS_PORT = address;
        RTL_DATA_DDR = 0xFF;
        RTL_DATA_PORT = data;

        RTL_CLEAR_WRITE;
        RTL_SET_WRITE;

        RTL_DATA_DDR = 0x00;
        RTL_DATA_PORT = 0xFF;
}
```

Listing. 2.5. Zapis danej bajtowej `data` do rejestru kontrolera sieci umieszczonego pod adresem `address`.

```
unsigned char RTLread(unsigned char address)
{
        unsigned char byte;

        RTL_ADDRESS_PORT = address;
        asm volatile("nop\n\t"::);

        RTL_CLEAR_READ;
        asm volatile("nop\n\t"::);

        byte = RTL_DATA_PIN;

        RTL_SET_READ;

        return byte;
}
```

Listing. 2.6. Odczyt danej bajtowej `byte` z rejestru kontrolera sieci umieszczonego pod adresem `address`.

```
void RTLinit(void)
{
        RTLsetup_ports();
        RTLhw_reset();
        RTLwrite(RSTPORT,0);
        Delay10ms(3);

        RTLwrite(CR, 0x41);
        Delay1ms(2);

        RTLwrite(CURR, RXSTART_INIT);
        RTLwrite(PAR0+0, MyMac0);     // Zapis adresu MAC
        RTLwrite(PAR0+1, MyMac1);
        RTLwrite(PAR0+2, MyMac2);
        RTLwrite(PAR0+3, MyMac3);
        RTLwrite(PAR0+4, MyMac4);
        RTLwrite(PAR0+5, MyMac5);

        RTLwrite(CR,0x21);
        Delay1ms(2);

        RTLwrite(DCR, 0x58);

        RTLwrite(RCR, 0x04);
```

```
        RTLwrite(TCR, 0x02);

        RTLwrite(RBCR0,0x00);
        RTLwrite(RBCR1,0x00);
        RTLwrite(TPSR, TXSTART_INIT);
        RTLwrite(PSTART, RXSTART_INIT);
        RTLwrite(BNRY, RXSTART_INIT);
        RTLwrite(PSTOP, RXSTOP_INIT);

        RTLwrite(ISR, 0xFF);
        RTLwrite(IMR, 0x11);

        RTLwrite(TCR, 0x00);
        RTLwrite(CR, 0x22);
}
```

Listing 2.7. Pełna inicjalizacja kontrolera sieci RTL8019AS

Kompletna ramka (zawierająca datagram IP) przeznaczona do wysłania jest już wcześniej skompletowana i zapisana w buforze wysyłania danych – tablica bajtowa w SRAM mikrokontrolera zdefiniowana jako `unsigned char EthFrame[1514]`. Do rejestrów TBCR1, TBCR0 wpisywana jest ilość bajtów ramki, która ma być wysłana – $05EA_h$ (1514_d).Następnie zawartość bufora jest bajt po bajcie wpisywana do rejestru RDMAPORT (do bufora kołowego wysyłanych danych). Jeżeli długość ramki jest mniejsza od 60, to pozostała część bufora jest uzupełniana zerami, tak aby ramka miała 60 bajtów. Na zakończenie następuje uruchomienie wysyłania przez kontroler sieci ramki zawierającej nasz datagram IP przez wpisanie do rejestru CR wartości 0x24.

Wysyłanie ramki jest zrealizowane przez funkcję RTLsend_packet(), której kod pokazano na listingu 2.8.

```
void RTLsend_packet(void)
{
        unsigned int i;

        RTLwrite(CR,0x22);

        while( RTLread(CR) & 0x04 );

        RTLwrite(TPSR,TXSTART_INIT);

        RTLwrite(RSAR0,0x00);
        RTLwrite(RSAR1,TXSTART_INIT);

        RTLwrite(ISR,(1<<ISR_PTX));


        RTLwrite(RBCR0, 0xEA);
        RTLwrite(RBCR1, 0x05);

        RTLwrite(CR,0x12);


        for(i=0; i<PacketSize; i++)
        {
```

```
                RTLwrite(RDMAPORT, EthFrame[i]);
        }

        while(PacketSize<60)
        {
                RTLwrite(RDMAPORT, 0);
                PacketSize++;
        }

        RTLwrite(TBCR0, (unsigned char)(PacketSize));
        RTLwrite(TBCR1, (unsigned char)((PacketSize)>>8));


        RTLwrite(CR,0x24);


        RTLwrite(ISR, (1<<ISR_RDC));

}
```

Listing 2.8. Wysyłanie ramki (pakietu danych) do kontrolera sieci Ethernet

Procedura odbioru ramki jest zdecydowanie bardziej skomplikowana. Kontroler sieci trzeba cyklicznie odpytywać i sprawdzać czy znajdują się w nim odebrane dane. Zatem funkcja odczytująca ramkę `RTLreceive_packet()` musi być cyklicznie wywoływana w pętli głównej programu mikroserwera.

Funkcja ta korzysta z dwóch prostszych funkcji. Pierwszej , sprawdzającej czy przyszła ramka (Listing 2.9) i drugiej kończącej odbieranie pakietu danych.

```
unsigned char RTLreceive_empty_check(void)
{
        unsigned char curr;

        RTLwrite(CR,0x62);
        curr = RTLread(CURR);

        RTLwrite(CR,0x22);

        return ( curr == RTLread(BNRY) );

}
```

Listing 2.9. Sprawdzenie czy w buforze kontrolera sieci jest nowy pakiet danych

Ramki odbierane z sieci zapisywane są w buforze kołowym kontrolera sieci. Równość wskaźników CURR i BNRY oznacza, że nie ma w pamięci kontrolera ramki do odczytania, wówczas funkcja zwraca wartość „1”, w przeciwnym wypadku dane należy odczytać – funkcja zwraca wartość „0”.

```
void RTLend_retreive(void)
{
        unsigned int i;
```

```
    RTLwrite(CR, 0x22);
    for(i = 0; i <= 20; i++)
        if(RTLread(ISR) & 1<<6)
            break;
    RTLwrite(ISR, 1<<6);

    RTLwrite(BNRY, nextPacketPtr);
}
```

Listing 2.10. Funkcja kończąca odczyt ramki z kontrolera sieci

Odczyt danych kończony jest wpisaniem do rejestru CR wartości 0x22, a wartość wskaźnika BNRY jest zmniejszana o ilość odebranych danych i wskazuje na następny pakiet danych znajdujący się w buforze kołowym kontrolera RTL8019AS.

Na początku funkcji odczytu pakietu danych (Listing 2.11) jest wywoływana funkcja `RTLreceive_empty_check()`, która sprawdza czy przyszła nowa ramka do odczytu. Jak nie to funkcja `RTLreceive_packet()` kończy działanie i zwraca wartość „0”. Jak tak, to następuje odczyt rejestru BNRY i inicjalizacja zmiennych wskazujących na numer strony i adres bajta do odczytu. Jeśli aktualny numer strony znajduje się poza obszarem bufora odczytu również następuje zatrzymanie odczytu ramki i wyjście z funkcji z wartością „0”.

Następnie następuje odczyt 4 bajtów statusu dołączanych przez układ RTL8019AS na początku bufora odczytu zawierających długość odbieranych danych. Po sprawdzeniu poprawności wartości długości danych następuje odczyt adresu MAC odbiornika (naszego kontrolera sieci i adresu MAC nadajnika) oraz dwóch bajtów informujących o typie pakietu: czy datagram IP, czy ramka ARP.

Na zakończenie bajt po bajcie odczytywana jest pozostała część pakietu danych. Gdy cały pakiet jest odczytany, to kończony jest odczyt i następuje wyjście z funkcji z wartością „1”.

```
unsigned char RTLreceive_packet(void)
{
        unsigned int i;

        if ( RTLreceive_empty_check() ) return 0;

        RTLwrite(CR, 0x22);

        currentPacketPtr = RTLread(BNRY);
        currentRetreiveAddress = (currentPacketPtr<<8) + 4;

        RTLwrite(ISR, (1<<ISR_PRX));

        if( (currentPacketPtr >= RXSTOP_INIT) || (currentPacketPtr <
RXSTART_INIT) )
        {
                RTLwrite(BNRY, RXSTART_INIT);
                RTLwrite(CR, 0x62);
                RTLwrite(CURR, RXSTART_INIT);
                RTLwrite(CR, 0x22);
                return 0;
        }


    RTLwrite(RBCR0, 4 + ETHERNET_HEADER_LENGTH);
```

```
    RTLwrite(RBCR1, 0);
    RTLwrite(RSAR0, 0);
    RTLwrite(RSAR1, currentPacketPtr);

        RTLwrite(CR, 0x0A);

        RTLread(RDMAPORT);
        nextPacketPtr = RTLread(RDMAPORT);

        if( (nextPacketPtr >= RXSTOP_INIT) || (nextPacketPtr <
RXSTART_INIT) )
                return 0;

        PacketSize = RTLread(RDMAPORT);
        PacketSize |= ((unsigned int)RTLread(RDMAPORT))<< 8;

        if (PacketSize > 4)
                PacketSize -= 4;
        else
        {
                RTLend_retreive();
                return 0;
        }

        if( PacketSize > 1514 )
        {
                RTLend_retreive();
                return 0;
    }

        EthFrame[EthDestMac0]=RTLread(RDMAPORT);
        EthFrame[EthDestMac1]=RTLread(RDMAPORT);
        EthFrame[EthDestMac2]=RTLread(RDMAPORT);
        EthFrame[EthDestMac3]=RTLread(RDMAPORT);
        EthFrame[EthDestMac4]=RTLread(RDMAPORT);
        EthFrame[EthDestMac5]=RTLread(RDMAPORT);

        EthFrame[EthSourceMac0]=RTLread(RDMAPORT);
        EthFrame[EthSourceMac1]=RTLread(RDMAPORT);
        EthFrame[EthSourceMac2]=RTLread(RDMAPORT);
        EthFrame[EthSourceMac3]=RTLread(RDMAPORT);
        EthFrame[EthSourceMac4]=RTLread(RDMAPORT);
        EthFrame[EthSourceMac5]=RTLread(RDMAPORT);

        EthFrame[EthTypeLen0]=RTLread(RDMAPORT);
        EthFrame[EthTypeLen1]=RTLread(RDMAPORT);



        currentRetreiveAddress += 6+6+2;

    RTLwrite(CR, 0x22);
    for(i = 0; i <= 20; i++)
        if(RTLread(ISR) & 1<<6)
            break;
    RTLwrite(ISR, 1<<6);

    RTLwrite(RBCR0, (unsigned char)PacketSize);
    RTLwrite(RBCR1, (unsigned char)(PacketSize>>8));
    RTLwrite(RSAR0, (unsigned char)currentRetreiveAddress);
    RTLwrite(RSAR1, (unsigned char)(currentRetreiveAddress>>8));
```

```
    RTLwrite(CR, 0x0A);

        for(i=0;i<PacketSize;i++)
                EthFrame[14+i] = RTLread(RDMAPORT);

    currentRetreiveAddress += PacketSize;
    if( currentRetreiveAddress >= 0x6000 )
        currentRetreiveAddress = currentRetreiveAddress - (0x6000-0x4600) ;

        RTLend_retreive();
        return 1;

}
```

Listing 2.11. Funkcja odczytująca ramkę z kontrolera sieci

Ostatni listing pokazuje bezpośrednie wykorzystanie powyższych funkcji w głównym programie obsługi mikroserwera. Jak widać z Listingu 2.12 bezpośrednio korzysta się z funkcji inicjalizującej kontroler sieci oraz z funkcji odczytującej zawartość bufora odczytu. Funkcja wysyłająca pakiet danych do kontrolera sieci jest wywoływana przez funkcje wyższych warstw stosu TCP/IP.

```
int main(void)
{
   ...
RTLinit();
   ...
SetIp();  // ustawienie adresu IP mikroserwera
   ...
while(1)
{
         ...
           // sprawdzenie np. naciśnięcia klawisza i wysłanie pakietu IP
           // przez funkcje warstw IP, TCP
         ...
       if(RTLreceive_packet())
            {
             ...
             // obsługa protokołów ARP, IP, ICMP, TCP
             ...
            }
}
return 0;
}
```

Listing 2.12. Fragment programu głównego mikroserwera

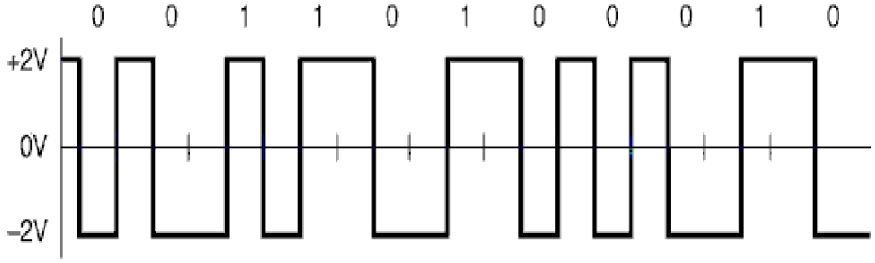
2.3. Warstwa Ethernet

Warstwa ta definiuje typ okablowania, zastosowanych złącz oraz określa sposób kodowania informacji i poziomy sygnałów elektrycznych, jak i opisuje postać ramki Ethernet.

Mikroserwery pracują zazwyczaj w standardzie sieci 10BaseT, która charakteryzuje się następującymi parametrami:

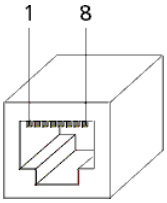
- topologia gwiazdy,
- jako medium transmisyjne wykorzystywana jest skręcona nieekranowana para przewodów miedzianych – skrętka (UTP) o impedancji 100 Ω. Stosuje się skrętkę co najmniej 4-tej kategorii o maksymalnym paśmie przenoszenia 20 MHz,
- złącza typu RJ45, przy czym kable zakończone są wtykami, gniazda zaś posiadają urządzenia sieciowe np. routery, karty sieciowe, huby, switche,
- maksymalna odległość pomiędzy dwoma urządzeniami sieciowymi wynosi 100 m,
- maksymalna prędkość transmisji dochodzi do 10 Mb/s,
- transmisja różnicowa dwiema parami przewodów,
- kodowanie *Manchester*.

Na rys. 2.8 przedstawiono zasadę pracy kodera *Manchester* oraz typowy poziom sygnału w sieci Ethernet 10BaseT. Na wyjściach TXD+ i TXD- wystawiany jest różnicowy sygnał o wartości piku około 2V do 2,5V. Jest to sygnał prostokątny o częstotliwości 20MHz (2*10MHz) z danymi kodowanymi metodą Manchester.



Rys. 2.8. Dane kodowane metodą Manchester

Na rys. 2.9 pokazano widok gniazda RJ45.



Styk	Lobe (ustawione od początku)	Trunc
3,6	para RX (wejście)	para TX (wyjście)
4,5	para TX (wyjście)	para RX (wejście)
1,2,7,8	nie używane	nie używane

Rys. 2.9. Widok gniazda RJ45 z opisem styków

2.3.1. Ramka sieci Ethernet, adres MAC

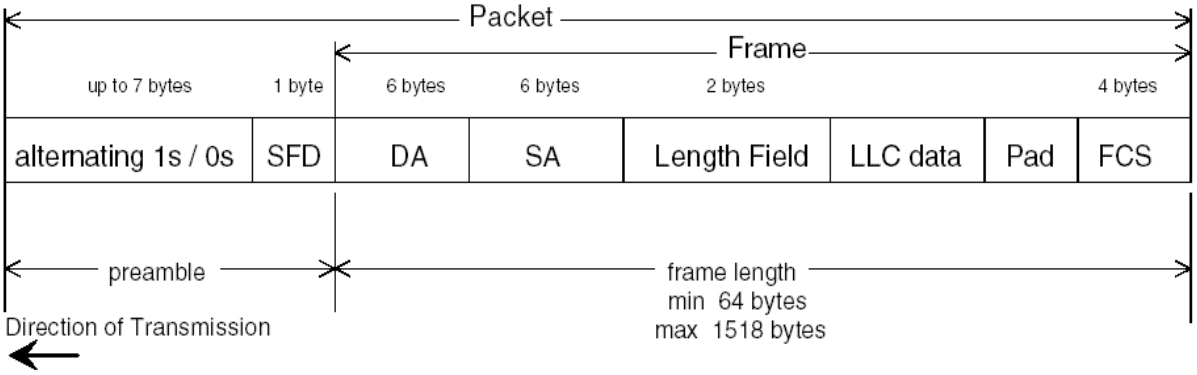
W sieci Ethernet wszystkie urządzenia posiadają unikalne (niepowtarzalne) fizyczne adresy 48-bitowe, zapisywane szesnastkowo w postaci 6 bajtów, oddzielonych dwukropkami lub myślnikami. Są to tzw. adresy MAC (ang. *Media Access Code*). Każdy producent komputerowej karty sieciowej lub niezależnego urządzenia sieciowego nadaje mu adres należący do przyznanej danemu producentowi puli adresów. Przykładowe adresy MAC mają postać:

```
00-00-1F-15-AA-C8
00-E0-7D-00-68-58
```

Dodatkowo zdefiniowany jest tzw. rozgłoszeniowy adres MAC. Wysłanie ramki Ethernet do odbiorcy z tym adresem powoduje, iż odbiorą ją wszystkie komputery działające w danym segmencie sieci. Adres rozgłoszeniowy składa się z 48 bitów o wartości '1', czyli 6 bajtów o wartości 255:

```
FF-FF-FF-FF-FF-FF
```

Wszystkie 16- i 32-bitowe liczby w protokołach sieciowych zapisywane są w konwencji Big-Endian, tzn. najstarszy bajt pierwszy. Dane zaś przesyłane są w pakietach o długości od 72 do 1526 bajtów. Rys. 2.10 przedstawia wygląd ramki ethernetowej (ang. *frame*) w standardzie *Ethernet II*.



Rys. 2.10. Ramka sieci Ethernet.

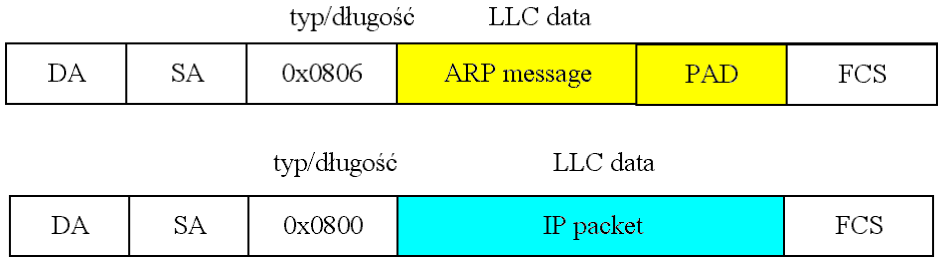
Budowa ramki jest następująca:

- Na początku wysyłana jest preambuła (ang. *preamble*) – stały ciąg siedmiu zer i jedynek naprzemiennie (1010101) mający na celu zsynchronizowanie odbiornika umieszczonego w zdalnym urządzeniu sieciowym z nadajnikiem, a następnie jeden bajt o wartości 10101011, wskazujący na początek ramki – SFD (ang. *Start-of-Frame Delimiter*).
- Tuż po SFD przesyłane są adresy MAC odbiorcy (ang. *Destination Address* – DA) i nadawcy ramki (ang. *Source Address* – SA), każdy po 6 bajtów,
- Następnie wysyłane jest pole LF (ang. *Length Field*) zawierające typ protokołu wyższej warstwy lub długość danych
- oraz blok danych o długości maksimum 1500 bajtów (ang. *Logical Link Control* – LLC), zakończony 32-bitową sumą kontrolną (ang. *Frame Check Sequence* – FCS).

Minimalna długość ramki Ethernet to 64 bajty. W przypadku, gdy pole danych jest mniejsze od 46 bajtów istnieje potrzeba jego wydłużenia, tak by cała ramka miała długość 64 bajty. Wówczas na końcu pola danych zostaje dodane pole PAD wypełnione odpowiednią liczbą zer.

Interpretacja wspomnianego wyżej 16-bitowego pola długość/typ (na rys. *Length Field*) zależy od jego wartości. Jeżeli zawiera się w nim liczba mniejsza lub równa maksymalnemu rozmiarowi pola danych (1500 bajtów), określa ono długość pola danych. Było to wykorzystywane w starszych protokołach rodziny IEEE 802.2. Jeżeli natomiast to pole ma wartość większą niż 1500, jest ona interpretowana jako identyfikator protokołu wyższej warstwy, zawartego (zagnieżdżonego) w polu danych. Przykładowo liczba `0x0806` (dziesiętnie 2054) jednoznacznie identyfikuje protokół ARP, a liczba `0x0800` (dziesiętnie 2048) protokół IP. Taka budowa ramki sieci Ethernet określona jest jako standard *Ethernet II*.

Na rysunku 2.11 przedstawiono przykłady ramek Ethernet niosących ramkę ARP i datagram IP. Pole PAD zostało dodane aby spełniony był warunek minimalnej długości ramki równej 64 bajty.



Rys. 2.11. Ramka ARP oraz datagram IP osadzone w ramce Ethernet II.

Zatem po odczytaniu bufora odczytu z danymi odebranymi z sieci przez kontroler należy zawsze sprawdzić, czy jest to ramka ARP, czy datagram IP poprzez testowanie pola typ/długość jak pokazano na listingu 2.13.

```
unsigned char PacketCheck(void)
{
        if(EthFrame[EthTypeLen0]==0x08 && EthFrame[EthTypeLen1]==0x06)
        {
                return 1;     // wiadomosc ARP
        }

        if(EthFrame[EthTypeLen0]==0x08 && EthFrame[EthTypeLen1]==0x00)
        {
                return 2;     // pakiet IP
        }

        return 0;            // bledne dane
}
```

Listing 2.13. Sprawdzenie czy otrzymane dane to ramka ARP, czy datagram IP

2.4. Protokół ARP

ARP (*Address Resolution Protocol*) jest protokołem najniższej warstwy modelu ISO/OSI – warstwy fizycznej. Jego zadaniem jest mapowanie adresu logicznego IP na adres fizyczny MAC. Komputer chcący wysłać datagram IP zna jedynie adres IP odbiorcy. Natomiast w sieci Ethernet wszystkie przesyłane ramki powinny być opatrzone adresem MAC docelowego urządzenia sieciowego (pole DA). Protokół ARP definiuje sposoby wyszukiwania i konwersji 32-bitowych adresów IP na adresy wymagane przez warstwę dostępu do medium – w przypadku sieci Ethernet będą to 48-bitowe adresy MAC.

Protokół ARP zakłada dynamiczne pozyskiwanie informacji o poszukiwanym adresie MAC. W tym celu w sieci Ethernet rozgłaszane są ramki o określonej strukturze, zawierającej m.in. adresy MAC i IP nadawcy oraz adres IP poszukiwanego urządzenia. Urządzenie, które rozpozna swój adres IP, odsyła do nadawcy odpowiedź, zawierającą również jego adres MAC.

W celu uzyskania odpowiedzi, komputer wysyła ramkę ARP stanowiącą zapytanie (ang. *ARP Request*), pozostawiając wyzerowane pole adresu sprzętowego stacji poszukiwanej. Ramka Ethernet zawierająca ramkę ARP jest rozgłaszana do wszystkich odbiorców w sieci lokalnej poprzez podanie adresu MAC odbiorcy równego `FF-FF-FF-FF-FF-FF`.

Urządzenie, które rozpozna swój adres IP, odsyła do nadawcy ramkę ARP z odpowiedzią (ang. *ARP Reply*). W tym celu zamienia blok adresów nadawcy i odbiorcy, wypełniając dodatkowo swój adres MAC. Odpowiedź ARP jest kierowana bezpośrednio do nadawcy.

Mechanizm pozyskiwania adresów sprzętowych oparty na protokole ARP jest prosty i wydajny. W celu zmniejszenia ilości wysyłanych zapytań ARP, każdy komputer dysponuje tablicą przypisań, odwzorowującą adresy IP na adresy MAC urządzeń, do których się ostatnio odwoływał. W celu zapobieżenia nieskończonemu przyrostowi ilości magazynowanych adresów, są one usuwane z tablicy, jeżeli nie były przez pewien czas potrzebne.

Dla przykładu - jeśli komputer A o adresie sprzętowym MAC `00-00-1F-15-AA-C8` i adresie IP 192.168.0.5 chce nawiązać połączenie z komputerem B o adresie IP 192.168.0.8, i nie zna adresu MAC komputera B, to wysyła ramkę zapytania ARP, której postać przedstawia tabela 2.2. Zacieniowane wiersze stanowią nagłówek ramki *Ethernet II*, pozostałe tworzą ramkę ARP.

Z kolei komputer B, o ile jest aktywny, natychmiast odsyła odpowiedź ARP zawierającą jego adres MAC `00-0C-76-8A-11-DE`. Postać odpowiedzi ARP przedstawia tabela 2.3.

Tabela 2.2. Postać ramki zapytania ARP.

długość pola	**opis**	**zawartość**
6 bajtów	adres MAC odbiorcy (DA)	FF-FF-FF-FF-FF-FF
6 bajtów	adres MAC nadawcy (SA)	00-00-1F-15-AA-C8
2 bajty	typ/długość	0x0806 – ARP
2 bajty	protokół sprzętowy	0x0001 – Ethernet
2 bajty	protokół logiczny	0x0800 – IP
1 bajt	długość adresu fizycznego (MAC)	0x06 – 6 bajtów
1 bajt	długość adresu logicznego (IP)	0x04 – 4 bajty
2 bajty	kod operacji	0x0001 – zapytanie ARP
6 bajtów	adres sprzętowy (MAC) nadawcy	00-00-1F-15-AA-C8
4 bajty	adres logiczny (IP) nadawcy	C0-A8-00-05 - 192.168.0.5
6 bajtów	adres sprzętowy (MAC) odbiorcy	00-00-00-00-00-00
4 bajty	adres logiczny (IP) odbiorcy	C0-A8-00-08 - 192.168.0.8

Tabela 2.3. Postać ramki odpowiedzi ARP.

długość pola	**opis**	**zawartość**
6 bajtów	adres MAC odbiorcy (DA)	00-00-1F-15-AA-C8
6 bajtów	adres MAC nadawcy (SA)	00-0C-76-8A-11-DE
2 bajty	typ/długość	0x0806 – ARP
2 bajty	protokół sprzętowy	0x0001 – Ethernet
2 bajty	protokół logiczny	0x0800 – IP
1 bajt	długość adresu fizycznego (MAC)	0x06 – 6 bajtów
1 bajt	długość adresu logicznego (IP)	0x04 – 4 bajty
2 bajty	kod operacji	0x0002 – odpowiedź ARP
6 bajtów	adres sprzętowy (MAC) nadawcy	00-0C-76-8A-11-DE
4 bajty	adres logiczny (IP) nadawcy	C0-A8-00-08 - 192.168.0.8
6 bajtów	adres sprzętowy (MAC) odbiorcy	00-00-1F-15-AA-C8
4 bajty	adres logiczny (IP) odbiorcy	C0-A8-00-05 - 192.168.0.5

Na poniższym listingu pokazano definicje stałych i zmiennych związanych z obsługą ramek protokołu ARP.

```
#define EthData                          14

#define ArpMediumType0          EthData+0
#define ArpMediumType1          EthData+1

#define ArpProtocolType0        EthData+2
#define ArpProtocolType1        EthData+3

#define ArpHardLen              EthData+4

#define ArpLogicLen             EthData+5

#define ArpOpType0              EthData+6
#define ArpOpType1              EthData+7

#define ArpSourceMac0           EthData+8
#define ArpSourceMac1           EthData+9
#define ArpSourceMac2           EthData+10
#define ArpSourceMac3           EthData+11
#define ArpSourceMac4           EthData+12
#define ArpSourceMac5           EthData+13

#define ArpSourceIp1            EthData+14
#define ArpSourceIp2            EthData+15
#define ArpSourceIp3            EthData+16
#define ArpSourceIp4            EthData+17

#define ArpDestMac0             EthData+18
#define ArpDestMac1             EthData+19
#define ArpDestMac2             EthData+20
#define ArpDestMac3             EthData+21
#define ArpDestMac4             EthData+22
#define ArpDestMac5             EthData+23

#define ArpDestIp1              EthData+24
#define ArpDestIp2              EthData+25
#define ArpDestIp3              EthData+26
#define ArpDestIp4              EthData+27

  …

unsigned char ArpReplyMac0;
unsigned char ArpReplyMac1;
unsigned char ArpReplyMac2;
unsigned char ArpReplyMac3;
unsigned char ArpReplyMac4;
unsigned char ArpReplyMac5;

unsigned char ArpReplyIp1=0;
unsigned char ArpReplyIp2=0;
unsigned char ArpReplyIp3=0;
unsigned char ArpReplyIp4=0;
```

Listing 2.14. Definicje stałych i zmiennych do obsługi protokołu ARP

Zgodnie z zasadami przedstawionymi w tabelach 2.2 i 2.3 tworzone jest zapytanie ARP w sposób pokazany na Listingu 2.15. Zmienna `EthFrame` zawiera dane, które zostaną wysłane do bufora zapisu kontrolera sieciowego.

```
void MakeArpRequest(unsigned char destrIp1, unsigned char destrIp2,
                    unsigned char destrIp3, unsigned char destrIp4)
{
        MakeEthHeader(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x08, 0x06);

        EthFrame[ArpMediumType0]=0x00;
        EthFrame[ArpMediumType1]=0x01;

        EthFrame[ArpProtocolType0]=0x08;
        EthFrame[ArpProtocolType1]=0x00;

        EthFrame[ArpHardLen]=0x06;

        EthFrame[ArpLogicLen]=0x04;

        EthFrame[ArpOpType0]=0x00;
        EthFrame[ArpOpType1]=0x01;

        EthFrame[ArpSourceMac0]=MyMac0;  // Adres MAC mikroserwera
        EthFrame[ArpSourceMac1]=MyMac1;
        EthFrame[ArpSourceMac2]=MyMac2;
        EthFrame[ArpSourceMac3]=MyMac3;
        EthFrame[ArpSourceMac4]=MyMac4;
        EthFrame[ArpSourceMac5]=MyMac5;

        EthFrame[ArpSourceIp1]=ServIp1;  // Adres IP mikroserwera
        EthFrame[ArpSourceIp2]=ServIp2;
        EthFrame[ArpSourceIp3]=ServIp3;
        EthFrame[ArpSourceIp4]=ServIp4;

        EthFrame[ArpDestMac0]=0x00;
        EthFrame[ArpDestMac1]=0x00;
        EthFrame[ArpDestMac2]=0x00;
        EthFrame[ArpDestMac3]=0x00;
        EthFrame[ArpDestMac4]=0x00;
        EthFrame[ArpDestMac5]=0x00;

        EthFrame[ArpDestIp1]=destrIp1;
        EthFrame[ArpDestIp2]=destrIp2;
        EthFrame[ArpDestIp3]=destrIp3;
        EthFrame[ArpDestIp4]=destrIp4;

        PacketSize=PacketSize+28;
}
```

Listing 2.15. Tworzenie zapytania ARP, gdzie kod funkcji `MakeEthHeader` umieszczono na listingu 2.16.

```
void MakeEthHeader(unsigned char dest0, unsigned char dest1, unsigned char
                   dest2, unsigned char dest3, unsigned char dest4,
             unsigned char dest5, unsigned char type0, unsigned char type1)
{
        PacketSize=0;

        EthFrame[EthDestMac0]=dest0;
        EthFrame[EthDestMac1]=dest1;
        EthFrame[EthDestMac2]=dest2;
```

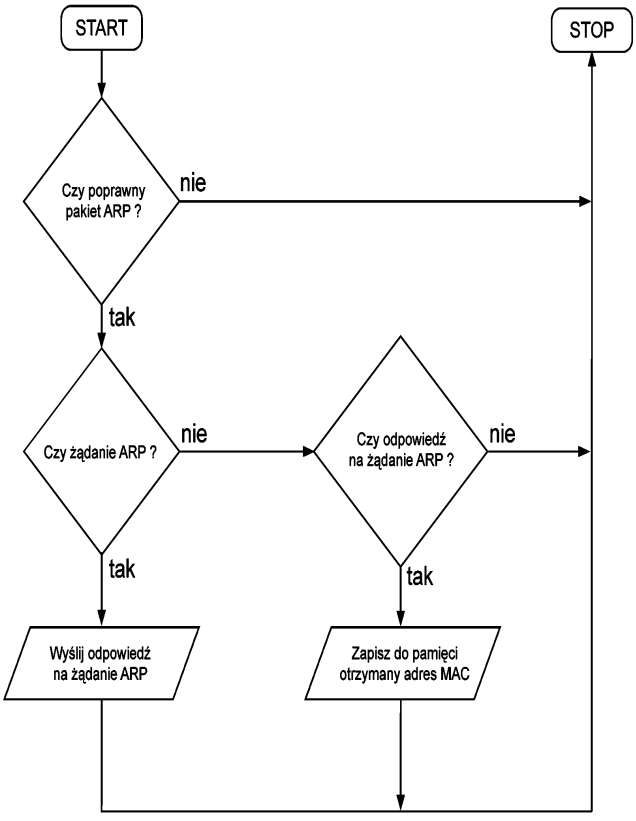
```
        EthFrame[EthDestMac3]=dest3;
        EthFrame[EthDestMac4]=dest4;
        EthFrame[EthDestMac5]=dest5;

        EthFrame[EthSourceMac0]=MyMac0;
        EthFrame[EthSourceMac1]=MyMac1;
        EthFrame[EthSourceMac2]=MyMac2;
        EthFrame[EthSourceMac3]=MyMac3;
        EthFrame[EthSourceMac4]=MyMac4;
        EthFrame[EthSourceMac5]=MyMac5;

        EthFrame[EthTypeLen0]=type0;
        EthFrame[EthTypeLen1]=type1;

        PacketSize=PacketSize+14;
}
```

Listing 2.16. Tworzenie nagłówka ramki Ethernet

Algorytm obsługi ramki ARP został przedstawiony na rys. 2.12. Obsługa zapytania ARP rozpoczyna się od sprawdzenia jego poprawności. Jeżeli ramka nie jest poprawna procedura jest kończona, w przeciwnym przypadku sprawdzany jest rodzaj zapytania ARP. Jeżeli jest on żądaniem ARP wysyłana jest odpowiedź z adresem MAC mikroserwera. W przypadku otrzymania odpowiedzi na żądanie ARP zapisywany jest w pamięci otrzymany adres MAC oraz jest on przypisywany do adresu IP urządzenia, które odpowiedziało na żądanie ARP.

Mikroserwer przechowuje w pamięci adres MAC urządzenia, które jako ostatnie odpowiedziało na żądanie ARP oznacza to, że ponowne nawiązanie połączenia nie wymaga wysyłaniaza pytania ARP gdyż docelowy adres MAC jest znany.

Sprawdzanie czy ramka jest żądaniem czy odpowiedzią ARP odbywa się niezależnie. W związku z tym algorytm uwzględnia sytuację, w której żaden z tych wariantów nie wystąpi.



Rys. 2.12. Algorytm obsługi pakietu ARP

Algorytm ten został zaimplementowany w funkcji ArpCheck(), której listing pokazano poniżej.

```
void ArpCheck(void)
{
        if(EthFrame[ArpMediumType0]==0x00 && EthFrame[ArpMediumType1]==0x01
    && EthFrame[ArpProtocolType0]==0x08 && EthFrame[ArpProtocolType1]==0x00
            && EthFrame[ArpHardLen]==0x06 && EthFrame[ArpLogicLen]==0x04 &&
                EthFrame[ArpOpType0]==0x00 && EthFrame[ArpOpType1]==0x01 &&
          EthFrame[ArpDestIp1]==ServIp1 && EthFrame[ArpDestIp2]==ServIp2 &&
            EthFrame[ArpDestIp3]==ServIp3 && EthFrame[ArpDestIp4]==ServIp4)
        {
                EthFrame[EthDestMac0]=EthFrame[EthSourceMac0];
                EthFrame[EthDestMac1]=EthFrame[EthSourceMac1];
                EthFrame[EthDestMac2]=EthFrame[EthSourceMac2];
                EthFrame[EthDestMac3]=EthFrame[EthSourceMac3];
                EthFrame[EthDestMac4]=EthFrame[EthSourceMac4];
                EthFrame[EthDestMac5]=EthFrame[EthSourceMac5];

                EthFrame[EthSourceMac0]=MyMac0;
                EthFrame[EthSourceMac1]=MyMac1;
                EthFrame[EthSourceMac2]=MyMac2;
                EthFrame[EthSourceMac3]=MyMac3;
                EthFrame[EthSourceMac4]=MyMac4;
                EthFrame[EthSourceMac5]=MyMac5;

                EthFrame[ArpOpType0]=0x00;
                EthFrame[ArpOpType1]=0x02;

                EthFrame[ArpSourceMac0]=MyMac0;
                EthFrame[ArpSourceMac1]=MyMac1;
                EthFrame[ArpSourceMac2]=MyMac2;
                EthFrame[ArpSourceMac3]=MyMac3;
                EthFrame[ArpSourceMac4]=MyMac4;
                EthFrame[ArpSourceMac5]=MyMac5;

                EthFrame[ArpDestIp1]=EthFrame[ArpSourceIp1];
                EthFrame[ArpDestIp2]=EthFrame[ArpSourceIp2];
                EthFrame[ArpDestIp3]=EthFrame[ArpSourceIp3];
                EthFrame[ArpDestIp4]=EthFrame[ArpSourceIp4];

                EthFrame[ArpSourceIp1]=ServIp1;
                EthFrame[ArpSourceIp2]=ServIp2;
                EthFrame[ArpSourceIp3]=ServIp3;
                EthFrame[ArpSourceIp4]=ServIp4;

                EthFrame[ArpDestMac0]=EthFrame[EthDestMac0];
                EthFrame[ArpDestMac1]=EthFrame[EthDestMac1];
                EthFrame[ArpDestMac2]=EthFrame[EthDestMac2];
                EthFrame[ArpDestMac3]=EthFrame[EthDestMac3];
                EthFrame[ArpDestMac4]=EthFrame[EthDestMac4];
                EthFrame[ArpDestMac5]=EthFrame[EthDestMac5];

                RTLsend_packet();

        }

        if(EthFrame[ArpMediumType0]==0x00 && EthFrame[ArpMediumType1]==0x01
                                    && EthFrame[ArpProtocolType0]==0x08 &&
             EthFrame[ArpProtocolType1]==0x00 && EthFrame[ArpHardLen]==0x06
               && EthFrame[ArpLogicLen]==0x04 && EthFrame[ArpOpType0]==0x00
```

```
            && EthFrame[ArpOpType1]==0x02 && EthFrame[ArpDestIp1]==ServIp1
          && EthFrame[ArpDestIp2]==ServIp2 && EthFrame[ArpDestIp3]==ServIp3
                                          && EthFrame[ArpDestIp4]==ServIp4)
        {
                ArpReplyMac0=EthFrame[EthSourceMac0];
                ArpReplyMac1=EthFrame[EthSourceMac1];
                ArpReplyMac2=EthFrame[EthSourceMac2];
                ArpReplyMac3=EthFrame[EthSourceMac3];
                ArpReplyMac4=EthFrame[EthSourceMac4];
                ArpReplyMac5=EthFrame[EthSourceMac5];

                ArpReplyIp1=EthFrame[ArpSourceIp1];
                ArpReplyIp2=EthFrame[ArpSourceIp2];
                ArpReplyIp3=EthFrame[ArpSourceIp3];
                ArpReplyIp4=EthFrame[ArpSourceIp4];

                ArpReply=1;
                ArpCount=60;
                pingtimeout=100;
        }
}
```

Listing. 2.17. Funkcja obsługująca wiadomości ARP

2.5. Warstwa Internetu – protokół IP

IP jest bezpołączeniowym protokołem komunikacyjnym, generującym usługi **datagramowe**. Znaczy to, że sieć oparta na tym protokole jest siecią z przełączaniem pakietów. Pakiet rozumiemy tu jako blok danych uzupełniony o informacje niezbędne do jego prawidłowego dostarczenia (nagłówki i końcówki). Sieć z przełączaniem pakietów wykorzystuje informację adresową do przełączania pakietów z jednej sieci fizycznej do drugiej, aż do miejsca przeznaczenia. Każdy pakiet jest przesyłany po sieci w sposób niezależny. Należy sobie uświadomić, że jeśli pewna porcja danych została podzielona na pakiety i wysłana do pewnego adresata, to droga każdego z tych pakietów przez sieć do adresata może okazać się całkiem inna. Przepływ pakietów (datagramów) w sieci odbywa się bez kontroli kolejności dostarczania ich do miejsca przeznaczenia, kontroli błędów i bez potwierdzania odbioru. Dzięki takiemu ograniczeniu funkcji, jakie musi spełniać IP, powoduje jego szybkość i efektywność.

W sieciach z protokołem IP przepływem datagramów sterują routery IP. Routery IP łączą sieci lokalne lub zdalne przesyłając datagramy pomiędzy nimi.

Przekierowanie datagramu odbywa się ze względu na numer logiczny jego adresata – 32-bitowy adres IP.

32-bitowe adresy IP są zwykle przedstawiane w formie 4 bajtów zapisanych dziesiętnie, oddzielonych kropkami. Przykładowe adresy IP mają postać:

```
100.200.100.253          80.50.38.114
```

Urządzenia pogrupowane są w podsieci, określane również przez 32-bitowe adresy. Każde urządzenie z przypisanym adresem IP przynależy do jednej podsieci i musi prócz swego adresu znać również tzw. maskę podsieci (ang. *subnet mask*). Maska podsieci jest 32-bitowa i zapisujemy ją podobnie jak adres IP. Przykładowa maska podsieci:

```
255.255.255.0
```

Grupa bitów ustawionych w masce sieciowej (w przykładzie: 24 najstarsze bity adresu) definiuje zakres bitów określający podsieć. Grupa bitów wyzerowanych (tutaj 8 najmłodszych bitów) definiuje zakres bitów określający numer hosta – komputera lub innego urządzenia sieciowego.

Dla pierwszego z przykładowych adresów IP na podstawie maski podsieci można obliczyć numer podsieci mnożąc logicznie (operacja AND) maskę podsieci z adresem IP. Należy pamiętać, że wszystkie adresy są po prostu 32-bitowymi liczbami, tyle że zapisanymi w specyficzny sposób:

```
100.200.100.253  AND  255.255.255.0  =  100.200.100.0
```

Wyróżnia się grupę adresów IP, które nie mogą być przyznawane urządzeniom, ani nie określające numerów podsieci. Jest wśród nich m.in. adres rozgłoszeniowy (ang. *broadcast*) – używany w celu wysłania informacji do wszystkich komputerów odbierających nadaną ramkę (połączonych co najwyżej koncentratorami lub przełącznikami). Adres rozgłoszeniowy składa się z 32 ustawionych bitów:

```
255.255.255.255
```

Istnieje również specjalny adres rozgłoszeniowy podsieci, używany w celu transmisji informacji jedynie do hostów działających w danej podsieci. Tworzy się go poprzez ustawienie wszystkich bitów adresu podsieci przypisanych do numeru hosta. Matematycznie ujmując, w celu obliczenia adresu rozgłoszeniowego podsieci należy do numeru podsieci dodać logicznie (operacja OR) zanegowaną maskę podsieci. Na przykład:

```
100.200.100.0  OR  (NOT 255.255.255.0)  =  100.200.100.255
```

W omawianym wcześniej standardzie *Ethernet II*, datagramy protokołu IP są enkapsulowane w polu danych przesyłanej ramki. Pole określające typ zawiera natomiast wartość 0x0800, co jednoznacznie identyfikuje protokół IP.

Poniżej na rys. 2.13 przedstawiono budowę nagłówka datagramu IP. W pierwszym wierszu i pierwszej kolumnie umieszczono numery bitów nagłówka.

<table>
<tr><th></th><th>00</th><th>01</th><th>02</th><th>03</th><th>04</th><th>05</th><th>06</th><th>07</th><th>08</th><th>09</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>00</td><td colspan="4">wersja</td><td colspan="4">IHL</td><td colspan="8">ToS</td><td colspan="16">długość całkowita</td></tr>
<tr><td>32</td><td colspan="16">identyfikator</td><td colspan="3">flagi</td><td colspan="13">przesunięcie fragmentu</td></tr>
<tr><td>64</td><td colspan="8">TTL</td><td colspan="8">protokół</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>
</table>

Rys. 2.13. Nagłówek datagramu IP

Pole „**wersja**” (ang. *version*) zawiera, w datagramach protokołu IP wersji czwartej, wartość 4 (binarnie: 0100). Istnieje również szósta wersja protokołu IP, znacznie bardziej rozwinięta i do tej pory bardzo mało rozpowszechniona w Internecie.

Pole „**IHL**” (ang. *Internet Header Length***)** specyfikuje długość nagłówka liczoną w słowach 32-bitowych. Nagłówki datagramów IP mogą zawierać dodatkowe pola opcji,

zapisywane bezpośrednio za polem adresu docelowego. W opcjach są zapisywane parametry bezpieczeństwa, specjalne informacje na temat wyznaczania trasy datagramu (ang. *routing*) czy też dane o czasie wysłania datagramu (ang. *timestamp*). Najczęściej jednak dodatkowe opcje nie są doklejane do nagłówków datagramów IP i 4-bitowe pole „IHL" ma wartość 5 (długość nagłówka wynosi 20 bajtów).

Pole „**ToS**" czyli „typ usługi" (ang. *Type of Service*) pozwala na określenie parametrów pomocnych w wyznaczaniu trasy datagramu. Podzielone jest na 5 fragmentów o długości od 1 do 3 bitów, których znaczenie podano w tabeli 2.4.

Tabela 2.4. Zawartość pola ToS.

bit	znaczenie
0–2	pierwszeństwo
3	opóźnienie: 0 – normalne, 1 – niskie
4	przepustowość: 0 – normalna, 1 – wysoka
5	pewność transmisji: 0 – normalna, 1 – wysoka
6–7	bity zarezerwowane

W praktyce wykorzystuje się bity 3–5 pola „ToS" w takich usługach jak VoIP (ang. *Voice over IP*) czy wideokonferencjach internetowych, choć do niedawna produkowane routery nie analizowały zawartości pola „ToS" i nie uwzględniały go w procesie określania trasy transmisji datagramu. Zapewnienie jakości usług poprzez m.in. specyfikację typu usługi dla przekazywanego datagramu okazało się niezbędne do poprawnego przekazu głosu przez sieć tak rozległą, jak Internet. Dlatego większość obecnie produkowanego osprzętu sieciowego analizuje pole „ToS".

16-bitowe pole „**długość całkowita**" (ang. *total length*) datagramu IP zawiera liczoną w bajtach sumę długości nagłówka (podaną w polu „IHL") i ilości danych występujących tuż za nagłówkiem. Każde urządzenie sieciowe musi być przygotowane do obróbki datagramów o całkowitej długości nie mniejszej niż 576 oktetów. Z wielkości pola wynika, iż najdłuższe transmitowane datagramy nie mogą przekraczać 65535 oktetów. Jednak w praktyce maksymalna długość datagramu jest ograniczona do rozmiaru pola danych ramki Ethernet o maksymalnej długości (1518 bajtów) i jest nie większa niż 1500 bajtów.

„**Identyfikator**" (ang. *identification*) datagramu IP stanowi liczba 16-bitowa, inna dla każdego nadawanego datagramu. Bardzo ważna jest dla prawidłowego złożenia datagramu podzielonego na fragmenty, którego każdy przesyłany fragment otrzymuje ten sam identyfikator.

3-bitowe pole „**flagi**" (ang. *flags*) specyfikuje, czy niższa warstwa protokołów sieciowych ma nie dzielić datagramu na fragmenty (zapalony bit 17 drugiego słowa datagramu – flaga „*Don't Fragment*") oraz, dla datagramu podzielonego na fragmenty, czy będą przesyłane kolejne fragmenty (zapalony bit numer 18, flaga „*More Fragments*"). Pierwszy bit pola „flagi" nie jest używany i w wysyłanych datagramach musi miećwartość 0.

13-bitowe pole „**przesunięcie fragmentu**" (ang. *fragment offset*) określa, do którego miejsca datagramu dany fragment należy. Przesunięcie fragmentu jest mierzone w

jednostkach po 8 bajtów a pierwszy z wysyłanych fragmentów musi mieć w tym polu wstawioną wartość 0.

Pole „**TTL**” (ang. *Time To Live*) datagramu IP określa czas życia datagramu. Jest on liczony w ilości węzłów, przez które musi przejść datagram aż zostanie zatrzymany – nie przesłany dalej. Taki mechanizm zabezpiecza przed nieskończoną transmisją datagramu w razie niemożliwości znalezienia właściwej drogi do odbiorcy oraz pozwala ograniczyć transmisję do małego obszaru geograficznego, np. jednego państwa. Nadanie polu „TTL” zbyt niskiej wartości spowoduje, że nie dotrze on do zbyt odległego, w sensie ilości pośrednich węzłów, odbiorcy. Najczęściej polu „TTL” przy wysyłaniu datagramu nadawana jest stała wartość, np. 64.

8-bitowe pole „**protokół**” (ang. *protocol*) specyfikuje protokół wyższej warstwy komunikacyjnej, którego pakiet jest enkapsulowany w polu danych datagramu IP. Oprogramowanie mikroserwera będzie wymagało implementacji protokołów: ICMP (o identyfikatorze 1) oraz TCP (o identyfikatorze 6).

Integralność informacji zawartych w nagłówku datagramu IP zapewnia 16-bitowa „**suma kontrolna nagłówka**” (ang. *header checksum*). Jest ona obliczana według prostego algorytmu, zakładającego zsumowanie wszystkich słów nagłówka jako liczb 16-bitowych a następnie dopełnienie do jedności otrzymanej liczby. Otrzymana liczba jest 24-bitowa i nie można jej umieścić w 16-bitowym polu sumy kontrolnej. Dokonuje się tutaj prostego zabiegu – najstarszy bajt tej liczby jest „odcinany” a następnie dodawany do otrzymanej w ten sposób liczby 16-bitowej. Na czas obliczeń pole sumy kontrolnej jest zerowane. Poniżej zamieszczono przykładowy nagłówek IP oraz sposób obliczenia sumy kontrolnej nagłówka (tabela 7).

Ostatnie dwa 32-bitowe słowa nagłówka datagramu IP stanowią: **adres źródłowy IP** (adres nadawcy datagramu) oraz **adres docelowy** IP (adres odbiorcy).

Poniżej pokazano przykład nagłówka IP wysyłanego przez mikroserwer o adresie 192.168.0.5 do komputera o adresie 192.168.0.8 datagramu IP oraz sposób obliczania sumy kontrolnej.

00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
0100 (4)				0101 (5)				00000000								0000000001011100 (92 bajty)															
1100110111001101 (52685)																000			0000000000000000											
10000000 (128)								00000110 (6 – TCP)								1110101101110001 (EB70)															
11000000101010000000000000000101 (192.168.0.5)																															
11000000101010000000000000001000 (192.168.0.8)																															

Rys. 2.14. Przykładowy nagłówek datagramu IP

Nagłówek z rys. 2.14 zakodowany heksalnie ma postać:

```
45 00 00 5C
CD CD 00 00
80 06 EB 70
C0 A8 00 05
C0 A8 00 08
```

By obliczyć sumę kontrolną należy najpierw zsumować przedstawione liczby jako 16-bitowe (po 2 bajty):

```
  45 00
  00 5C
  CD CD
  00 00
  80 06
  00 00   <-- wyzerowane pole sumy kontrolnej
  C0 A8
  00 05
  C0 A8
+ 00 08
-------
03 14 8C
```

W kolejnym kroku najstarszy bajt liczby 24-bitowej trzeba „odciąć” i dodać do uzyskanej liczby 16-bitowej:

```
   14 8C
+  00 03
-------
   14 8F
```

Teraz pozostaje tylko uzyskaną liczbę `148F` odjąć od liczby `FFFF`, co daje liczbę `EB70`, która jest szukaną sumą kontrolną nagłówka IP.

Na listingu 2.18 pokazano implementację obliczania sumy kontrolnej nagłówka datagramu IP.

```
unsigned int IpCheckSumCalc(void)
{
        EthFrame[IpCheckSum0]=0x00;
        EthFrame[IpCheckSum1]=0x00;

        unsigned int word16;
        unsigned int i;
        unsigned long sum=0;

        for(i=0;i<20;i=i+2)
        {
                word16=((EthFrame[EthData+i]<<8)&0xFF00)
                                        +(EthFrame[EthData+i+1]&0xFF);
                sum=sum+(unsigned long)word16;
        }
        while(sum>>16)
        {
                sum=(sum&0xFFFF)+(sum>>16);
        }
        sum=~sum;
        return ((unsigned int) sum);
}
```

Listing 2.18. Obliczanie sumy kontrolnej nagłówka datagramu IP

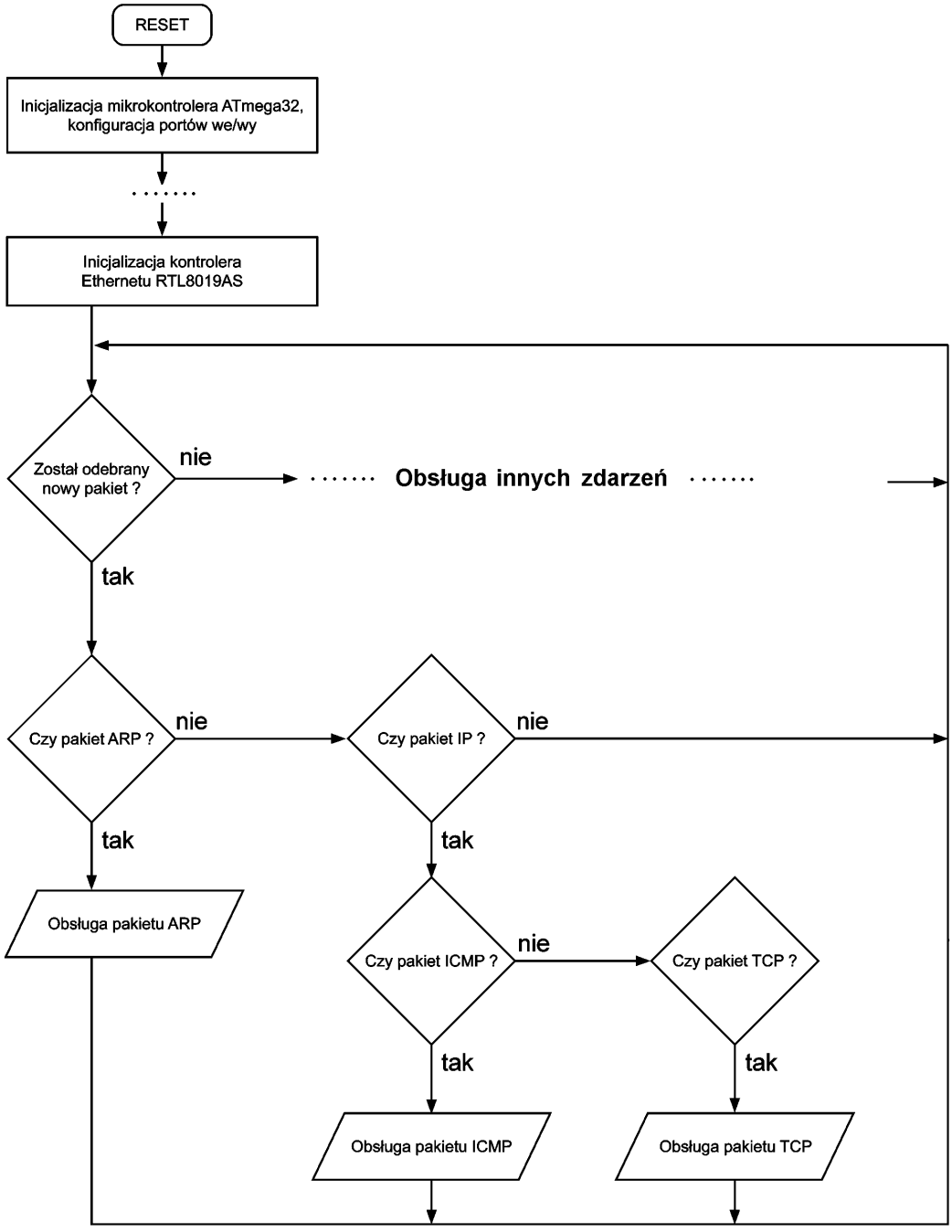
W układach mikroserwerów, ze względu na ograniczone zasoby sprzętowe i moc obliczeniową mikrokontrolerów, warto wprowadzić pewne założenia odnośnie protokołu IP:

- pole „wersja” zawsze przyjmuje wartość dziesiętną 4, zaś pole „IHL” wartość 5,
- pola: „ToS”, „flagi”, „przesunięcie fragmentu” są wypełnione zerami, jeżeli nie przewiduje się fragmentacji,
- długość całkowita nie może przekraczać maksymalnego rozmiaru pola danych ramki *Ethernet II*, który wynosi 1500 bajtów,

- pole „identyfikator” może przyjmować dowolną wartość stałą, jeżeli nie przewiduje się fragmentacji,
- pole „TTL” posiada stałą, niską wartość (np. TTL=32),
- w polu „protokół” wartości dozwolone to: 1 – ICMP, 6 – TCP (tylko te dwa protokoły zostały zaimplementowane). Datagramy IP z innymi wartościami tego pola będą ignorowane.

Przy powyższych założeniach można obsługę protokołu IP (analiza nagłówka datagramu IP, sprawdzanie i obliczanie sumy kontrolnej) zawrzeć w funkcjach obsługujących protokoły wyższej warstwy, czyli w obsłudze protokołu ICMP i protokołu TCP.

Koncepcja taka została przedstawiona na rys. 2.15. Oczywiście jest to rozwiązanie najprostsze nie pozwalające na późniejszą rozbudowę obsługi protokołu IP.



Rys. 2.15. Algorytm obsługi stosu protokołów TCP/IP

Warunki „Czy pakiet ARP?" i „Czy pakiet IP?" są sprawdzane przez funkcję przedstawioną na listingu 2.13, natomiast warunki „Czy pakiet ICMP" i „Czy pakiet TCP" są realizowane przez funkcję pokazaną na listingu 2.19. Funkcję tę można zaliczyć do funkcji obsługujących protokół IP.

```
unsigned char IpCheck(void)
{
        if(EthFrame[IpProt]==0x01) return 3;
        if(EthFrame[IpProt]==0x06) return 4;
        return 0;
}
```

Listing. 2.19. Funkcja testująca czy datagram IP zawiera komunikat ICMP, czy segment TCP

Na listingach 2.20, 2.21 i 2.22 pokazano inne podejście obsługi protokołu IP. Listing 2.20 zawiera deklaracje nagłówka IP, listing 2.21 funkcję przetwarzającą odebrany datagram IP, funkcję formującą datagramy IP zawierające komunikat ICMP lub segment TCP.

```
struct ip_packet
{
struct eth_hdr ip_eth_hdr;
unsigned char verIHL;                // version - 4 bits, Internet Header
Length - 4 bits
unsigned char TOS;                   // Type of Service - 8 bits
unsigned int totlen;                 // Total Length of the datagram - 16
                                     // bits
unsigned int id;                     // Identification of datagram - 16 bits
unsigned int fragoff;                // Flags - 3 bits, Fragment Offset - 13
                                     //bits
unsigned char TTL;                   // Time to Live - 8 bits
unsigned char proto;                 // Protocol - 8 bits
unsigned int hdrchksum;              // Header Checksum - 16 bits
unsigned char srcIP[ip_size];        // Source IP Address - 32 bits
unsigned char destIP[ip_size];       // Destination IP Address - 32 bits
};

#define IP_ICMP 0x0001
#define IP_TCP 0x0006
#define ip_destIP_offset     30
#define ip_totlen_offset     16
```

Listing 2.20. Deklaracje nagłówka IP

```
#define TxTTL 0x20
unsigned char rx_ip_packet(unsigned char *rx_buffer) small
{

struct ip_packet *rx_ip_hdr = (struct ip_packet *) rx_buffer;

// Make sure that the packet is destined for me or for broadcast
if ((!memcmp(myIP, &rx_ip_hdr->destIP, sizeof(unsigned char) * ip_size)) ||
   (!memcmp(myBCAST, &rx_ip_hdr->destIP, sizeof(unsigned char) * ip_size)))
{
       // Save the source MAC and IP for when I reply
       memcpy(srcMAC, rx_ip_hdr->ip_eth_hdr->shost, sizeof(unsigned char)
                                                          * mac_size);
       memcpy(srcIP, rx_ip_hdr->srcIP, sizeof(unsigned char) * ip_size);
return rx_ip_hdr->proto;
} }
```

Listing 2.21. Funkcja przetwarzająca odebrany datagram IP

```
void tx_ip_packet(unsigned char *tx_buffer, unsigned int tx_len, unsigned
char cProto) small
{

unsigned char data i = 0;
unsigned long data chksum = 0;
struct ip_packet *tx_ip_hdr = (struct ip_packet *) tx_buffer;

memcpy(tx_ip_hdr->ip_eth_hdr->dhost, targetMAC, sizeof(unsigned char)
                                                        * mac_size);
memcpy(tx_ip_hdr->ip_eth_hdr->shost, myMAC, sizeof(unsigned char)
                                                      * mac_size);

    tx_ip_hdr->ip_eth_hdr->type = ETHER_IP;
    tx_ip_hdr->verIHL = 0x45;
    tx_ip_hdr->TOS = 0;
    tx_ip_hdr->totlen = tx_len + (sizeof(struct ip_packet) - eth_hdr_len);
    tx_ip_hdr->id = 0x0a;
    tx_ip_hdr->fragoff = 0;
    tx_ip_hdr->TTL = TxTTL;
    tx_ip_hdr->proto = cProto;
    tx_ip_hdr->hdrchksum = 0;

memcpy(tx_ip_hdr->srcIP, myIP, sizeof(unsigned char) * ip_size);
memcpy(tx_ip_hdr->destIP, targetIP, sizeof(unsigned char) * ip_size);

// Compute the checksum
for (i=0; i<ip_len; i = i + 2)

chksum += (unsigned int) ((tx_buffer[i + eth_hdr_len] << 8) | tx_buffer[i +
eth_hdr_len + 1]);

tx_ip_hdr->hdrchksum = (unsigned int) ~((chksum >> 16) + (chksum
                                                        & 0xffff));
     tx_buffer[tx_len + sizeof(struct ip_packet)] = 0;
     tx_packet(tx_buffer, tx_len + sizeof(struct ip_packet));
}
```

Listing 2.22. Funkcja formująca datagramy IP zawierające komunikat ICMP lub segment TCP.

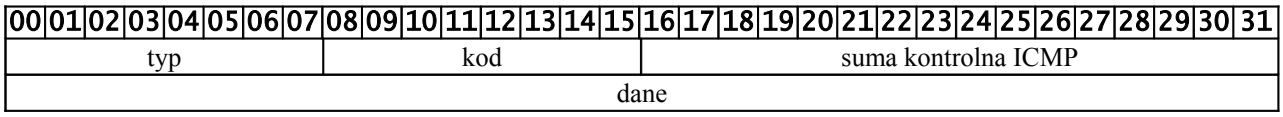
2.6. Protokół ICMP

Protokół komunikatów kontrolnych Internetu ICMP powstał aby umożliwić routerom oznajmianie o błędach oraz udostępnianie informacji o niespodziewanych sytuacjach. Protokół ICMP jest traktowany jako wymagana część IP i musi być realizowany przez każdą implementację IP, ponieważ wiadomość ICMP jest enkapsulowana w datagramie IP, którego pole „protokół" przyjmuje wartość 1.

Chociaż protokół ICMP powstał, aby umożliwić routerom wysyłanie komunikatów, to każda maszyna może wysyłać komunikaty ICMP do dowolnej innej. Z technicznego punktu widzenia ICMP jest mechanizmem powiadamiania o błędach. Udostępnia on routerom, które rozpoznają błąd, sposób powiadomienia o nim nadawcy, którego błąd dotyczy. Chociaż w specyfikacji protokołu są określone zamierzone sposoby korzystania z ICMP i sugestie na temat tych działań, które mogą być podejmowane w odpowiedzi na komunikaty o błędach, ICMP nie dla każdego możliwego błędu wyszczególnia działania, jakie mają być zapoczątkowane.

Tak więc, gdy datagram powoduje błąd, ICMP może jedynie powiadomić pierwotnego nadawcę o przyczynie. Nadawca musi otrzymaną informację przekazać danemu programowi użytkownika, albo podjąć inne działanie mające na celu uporanie się z tym problemem.

Każdy komunikat ICMP ma własny format, ale wszystkie zaczynają się trzema takimi samymi polami. Budowę wiadomości ICMP przedstawia rys. 2.16.



Rys. 2.16. Komunikat ICMP

- Pole „**typ**” (ang. *type*) jednoznacznie określa typ wiadomości ICMP (Tabela 2.5).

Tabela 2.5. Opis pola Typ wiadomości

Numer typu	**Typ wiadomości**
0	Odpowiedź na Echo
3	Adresat nieosiągalny
4	Łącze zajęte
5	Przełączenia
8	Żądanie Echo
11	Czas przekroczony
12	Zły parametr
13	Żądanie znacznika czasowego
14	Odpowiedź na żądanie znacznika czasowego
15	Żądanie informacyjne
16	Odpowiedź na żądanie informacyjne
17	Żądanie maski
18	Odpowiedź na żądanie maski

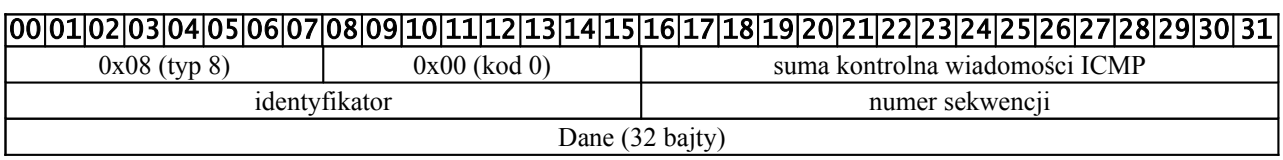
- Pole „**kod**” (ang. *code*) specyfikuje dokładniej wiadomość w obrębie danego typu (Tabela 2.6).

Tabela 2.6. Opis pola Kod błędu

Kod błędu	**Opis**
0	Sieć niedostępna
1	Stacja niedostępna
2	Protokół nie jest obsługiwany
3	Port zajęty
4	Konieczna fragmentacja
5	Zła specyfikacja ścieżki

- **Suma kontrolna wiadomośc**i ICMP (ang. *ICMP checksum*) wyznacza się w podobny sposób jak dla nagłówka IP, lecz obejmuje ona całą wiadomość ICMP, tzn. łącznie z polem „dane”.

Format komunikatu ICMP nakazującego wysłanie echa (ang. *Echo Request*) przedstawia rys. 2.17.



Rys. 2.17. Komunikat „Echo Request”

Komunikat „Echo Request” zawiera, prócz wcześniej omówionych pól, 16-bitowy identyfikator oraz 16-bitowy numer sekwencji. Te pola mogą przyjmować wartość zerową lub inną dowolnie wybraną i są pomocne w dopasowaniu wysyłanych komunikatów „Echo Reply” do odebranych odpowiedzi.

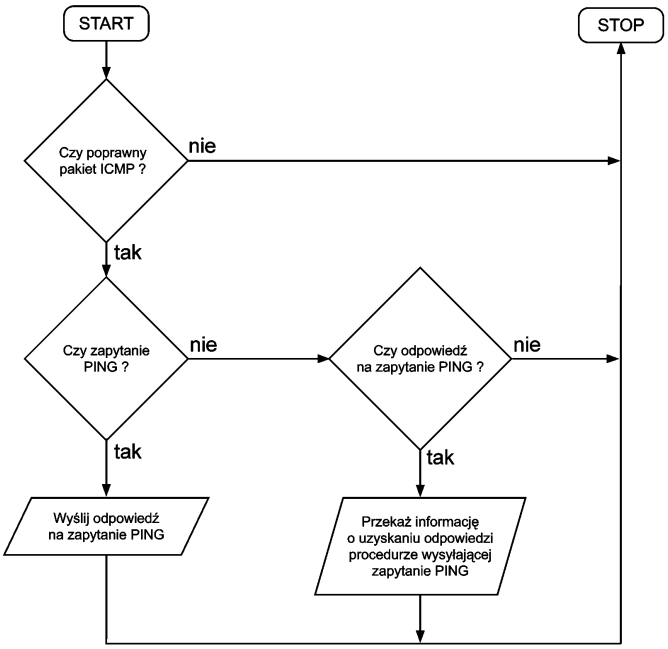
Po odebraniu skierowanej do niego komunikatu ICMP „Echo Request”, urządzenie sieciowe ma obowiązek odesłać nadawcy potwierdzenie w postaci odpowiedzi ICMP „Echo Reply”. Musi ona zawierać dane identyczne z odebranymi: ten sam identyfikator i numer sekwencji, inną zawartość pola „typ”, które tym razem przyjmuje wartość 0 oraz właściwą sobie sumę kontrolną.

Mechanizm automatycznego odsyłania odpowiedzi na komunikat ICMP „Echo Request” daje użytkownikowi komputera wygodną możliwość sprawdzenia, czy dane urządzenie sieciowe działa poprawnie i czy można się z nim skomunikować. W każdym sieciowym systemie operacyjnym istnieje polecenie *ping*, w którego wywołaniu należy podać nazwę lub numer IP komputera, do którego chcemy wysłać komunikat ICMP „Echo Request”.

Podczas implementacji protokołu ICMP warto przyjąć następujące założenia:

- gdy nie ma takiej potrzeby, mikroserwer powinien reagować wyłącznie na żądanie echa,
- minimalny rozmiar pola danych w wiadomości ICMP „Echo Reply” wynosi 0 bajtów, maksymalny jest ograniczony do 128 bajtów,
- suma kontrolna wiadomości ICMP jest obliczana w uproszczony sposób. Wykorzystano tu fakt, iż odpowiedź na żądanie echa różni się jedynie wartością pola „typ” (typ=0) i sumy kontrolnej. Z zasady tworzenia sumy kontrolnej wynika, że jej wartość w odpowiedzi ICMP „Echo Reply” jest o `0x8000` większa niż komunikatu ICMP „Echo Request”. Dwubajtowa liczba `0x8000` wzięła się z bajtów pola „typ” i „kod”. Zatem w momencie odsyłania odpowiedzi na żądanie echa do wartości sumy kontrolnej wiadomości żądania echa, dodawana jest liczba `0x8000`, przy czym jeśli wystąpi przepełnienie, to do LSB sumy kontrolnej dodawana jest liczba 1 (przeniesienie z MSB).

Algorytm obsługi komunikatu ICMP został przedstawiony na rys. 2.18. Obsługa komunikatu ICMP rozpoczyna się od sprawdzenia poprawności odebranych danych. Jeżeli komunikat nie jest poprawny to procedura jest zakańczana w przeciwnym wypadku sprawdzane jest czy komunikat ICMP zawiera zapytanie PING kierowane do mikroserwera albo odpowiedź na zapytanie PING wysłane przez mikroserwer. W pierwszym przypadku wysyłana jest odpowiedź na zapytanie PING, natomiast w drugim informacja o uzyskaniu odpowiedzi na PING jest przekazywana procedurze realizującej wysyłanie zapytania PING.



Rys. 2.18. Algorytm obsługi komunikatu ICMP

Procedura wysyłania zapytania PING powinna rozpocząć się od sprawdzenia czy znany jest adres MAC urządzenia, do którego chcemy wysłać zapytanie PING. Jeżeli nie jest znany adres fizyczny odpowiadający adresowi IP docelowego urządzenia to wysyłane jest żądanie ARP w celu ustalenia tego adresu. W przypadku braku odpowiedzi na żądanie ARP jest ono ponownie wysyłane. Jeżeli po wysłaniu np. 5 razy żądania ARP nie uzyskamy odpowiedzi to urządzenie zostaje uznane za nieosiągalne. Po uzyskaniu odpowiedzi na żądanie ARP, kiedy znany jest już docelowy adres fizyczny, wysyłany jest pakiet z zapytaniem PING. Jeżeli upłynie czas oczekiwania na odpowiedź od urządzenia świadczy to o braku odpowiedzi na PING. Gdy mikroserwer uzyska odpowiedź na zapytanie PING, to znaczy że urządzenie jest dostępne w sieci.

2.7. Protokół TCP

Protokół TCP tworzy wirtualne połączenie pomiędzy dwoma użytkownikami umożliwiając na bezpieczny (bezbłędny) transfer danych. Do wyspecyfikowania odbiorcy za pomocą 32-bitowej adresacji, wspieranej przez protokół IP. TCP dodaje mechanizm wyróżniający w każdym urządzeniu 65536 tzw. portów. Port jest logiczną jednostką, do której kierowane są informacje. Do każdego portu może być przypisany program odbierający wyłącznie informację skierowaną do danego portu. Istnieje grupa wyróżnionych numerów portów, dla których podano standardowy program obsługujący i rodzaj przesyłanych danych.

Do portów o numerach poniżej 1024 są przypisane konkretne usługi np. HTTP – port 80. Ich użycie do innych celów jest utrudnione lub wręcz uniemożliwione w zwykłym oprogramowaniu użytkowym działającym pod kontrolą niektórych sieciowych systemów operacyjnych. Korzystanie przez aplikacje z portów o wysokich numerach (czasami dopiero tych przekraczających 16383) jest dozwolone i zalecane dla niestandardowych usług systemowych.

2.7.1. Nagłówek segmentu TCP

Na rys. 2.19 pokazano format nagłówka segmentu TCP.

<table>
<tr><th>00</th><th>01</th><th>02</th><th>03</th><th>04</th><th>05</th><th>06</th><th>07</th><th>08</th><th>09</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 colspan="16">port źródłowy (nadawcy)</td><td colspan="16">port docelowy (odbiorcy)</td></tr>
<tr><td colspan="32">numer porządkowy</td></tr>
<tr><td colspan="32">numer potwierdzenia</td></tr>
<tr><td colspan="8">długość nagłówka</td><td colspan="8">zarezerwowane</td><td colspan="8">bity kontrolne</td><td colspan="8">rozmiar okna</td></tr>
<tr><td colspan="16">suma kontrolna</td><td colspan="16">wskaźnik pilnych danych</td></tr>
<tr><td colspan="32">dane</td></tr>
</table>

Rys. 2.19. Nagłówek segmentu TCP

- Pole „**port źródłowy**” (ang. *source port*) zawiera numer portu, jakiego użył nadawca przy wysyłaniu strumienia TCP, zaś „**port docelowy**” (ang. *destination port*) zawiera numer portu odbiorcy, do którego jest skierowany ów strumień. Np. użytkownik przegląda strony WWW (ang. *World Wide Web*). Przeglądarka internetowa w jego komputerze wysyła odpowiednio segmenty TCP z zapytaniami do serwera WWW, w których „port docelowy” zawiera liczbę 80. Podobnie serwer HTTP odpowiada segmentami TCP, w których pole „port źródłowy” jest równe 80. Numer portu źródłowego w segmentach użytkownika oraz portu docelowego w segmentach serwera WWW jest również taki sam. Zatem przy wzajemnej wymianie danych pomiędzy hostami, każdy z nich musi zamieniać miejscami pola „port źródłowy” i „port docelowy” w nagłówku TCP.
- 32-bitowe pole „**numer porządkowy**” (ang. *sequence number*) identyfikuje bajty w przesyłanym strumieniu danych. Wszystkie bajty przesyłane strumieniem TCP są numerowane. Numer SEQ odpowiada numerowi pierwszego bajta przesyłanego w danym segmencie TCP.
- „**Numer potwierdzenia**” (ang. *acknowledgment number*) jest numerem następnego bajta, jaki powinien dotrzeć do odbiorcy. ACK jest o jeden większy od numeru sekwencyjnego ostatniego poprawnie odebranego bajta. Pole to jest ważne, gdy ustawiona jest flaga ACK.
- **Długość nagłówka** (ang. *data offset*) zawiera liczbę całkowitą, która określa długość nagłówka segmentu mierzoną w wielokrotnościach 32 bitów. Dla typowego segmentu TCP pole ma wartość 5.
- Ponieważ niektóre segmenty mogą przenosić tylko potwierdzenia, inne dane, inne zaś zawierają prośby o ustanowienie lub zamknięcie połączenia - pole „**bity kontrolne**” (ang. *control bits*) zawiera informację o przeznaczeniu zawartości segmentu:
  - **Flaga URG** – flaga pilności, kiedy jest ustawiona pole wskaźnika pilności jest ważne,
  - **Flaga ACK** – flaga potwierdzenia, kiedy jest ustawiona pole numeru potwierdzenia jest ważne,
  - **Flaga PSH** – ustawiona flaga PSH informuje odbiornik, że powinien jak najszybciej przekazać dane aplikacji,
  - **Flaga RST** – ustawienie flagi RST oznacza zerwanie połączenia,
  - **Flaga SYN** – flaga ustawiana przy synchronizacji numerów sekwencyjnych w trakcie nawiązywania połączenia,
  - **Flaga FIN** – flaga ustawiana przy realizacji zakończenia połączenia.
- Przy każdym wysłaniu segmentu oprogramowanie TCP klienta proponuje ile danych może przyjąć od serwera, umieszczając rozmiar swojego bufora w polu „**rozmiar okna**” (ang. *window*).
- **Sumę kontrolną** w segmencie TCP oblicza się w podobny sposób jak dla nagłówka IP, lecz tu sumowany jest tzw. pseudo-nagłówek IP, nagłówek TCP oraz dane, przy czym

dane są dopełniane zerami tak, by ich rozmiar w bajtach był liczbą parzystą. Na rys. 2.20 przedstawiono postać pseudo-nagłówka IP.

- **Wskaźnik pilnych danych** zawiera dodatnie przesunięcie, które musi być dodane do numeru sekwencyjnego, aby wyznaczyć numer sekwencyjny ostatniego bajta pilnych danych

00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
źródłowy adres IP																															
docelowy adres IP																															
00000000								typ protokołu								długość całkowita															

Rys. 2.20. Pseudo nagłówek IP

Pole „długość całkowita” w pseudo-nagłówku IP zawiera rozmiar całego segmentu TCP, czyli jest sumą długości jego nagłówka oraz danych przez niego niesionych. Dla odróżnienia w nagłówku IP pole „długość całkowita” jest sumą długości nagłówka IP i danych po nim następujących.

Poniżej przedstawiono sposób obliczenia sumy kontrolnej segmentu TCP maszyny o adresie IP 192.168.0.5, która chce nawiązać połączenie z serwerem FTP w sieci lokalnej o adresie IP 192.168.0.8. Na rys. 2.21 przedstawiono postać pseudo-nagłówka IP, a na rys. 2.22 postać nagłówka segmentu TCP, dla opisanej sytuacji.

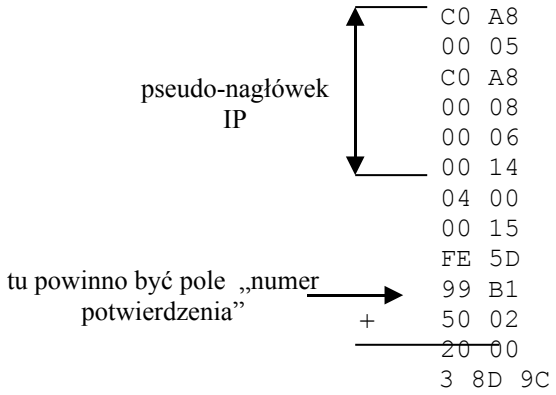
00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
0xC0A80005 (192.168.0.5)																															
0xC0A80008 (192.168.0.8)																															
00000000								0x06 (TCP)								0x0014 (20 bajtów)															

Rys. 2.21. Przykładowy pseudo nagłówek IP

00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
0x0400 (port 1024)																0x0015 (port 21 – FTP)															
0xFE5D99B1																															
0x00000000																															
0101 (5)				0x0												000010 (SYN)								0x2000 (8192)							
suma kontrolna																0x0000															

Rys. 2.22. Przykładowy nagłówek segmentu TCP

Dane zawarte w pseudo-nagłówku IP oraz nagłówku TCP należy zsumować jako liczby dwubajtowe. Na czas sumowania nagłówka TCP pole sumy kontrolnej jest wyzerowane.



W słupku nie zapisano pola „numer potwierdzenia” i „suma kontrolna” ponieważ oba pola są puste (wypełnione zerami). Strzałką oznaczono liczby powstałe z pseudo-nagłówka IP, pozostałe liczby powstały z segmentu TCP. W podanym przykładzie nie występuje również pole z danymi, bo jest to segment rozpoczynający negocjację otwarcia połączenia TCP.

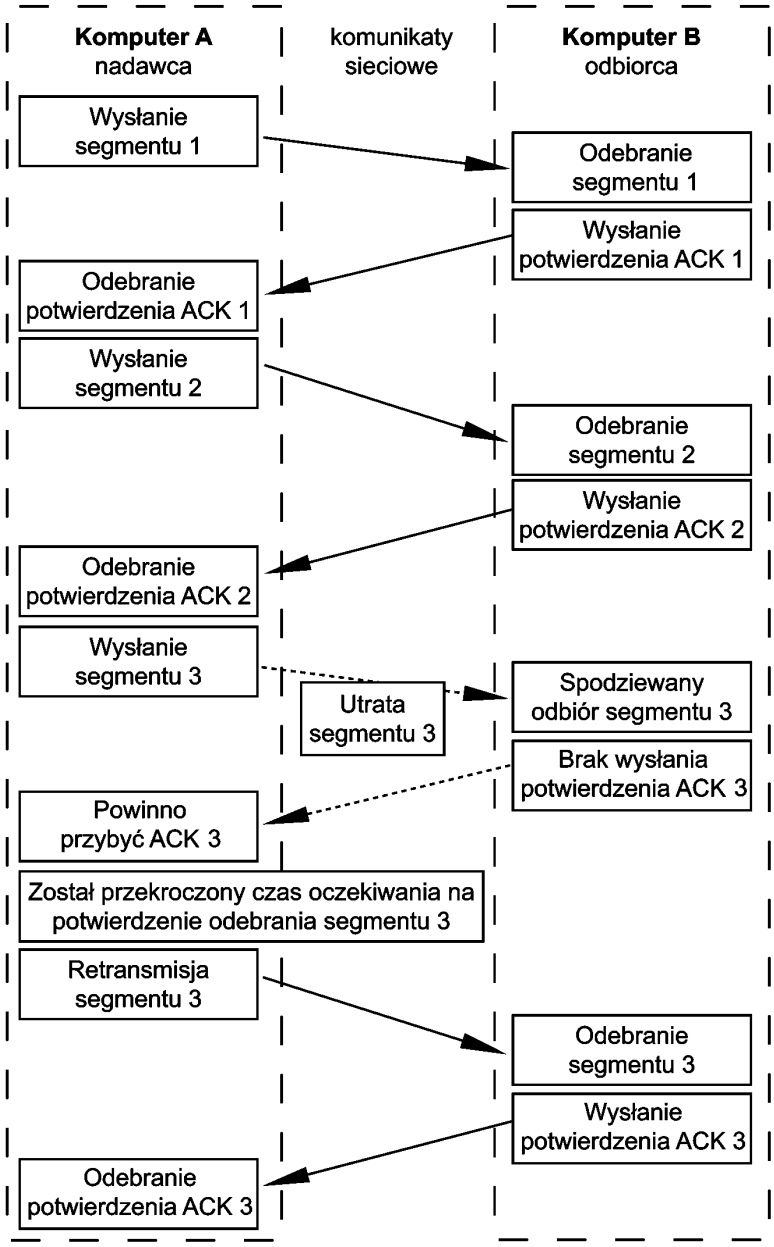
W kolejnym kroku najstarszy bajt liczby 24-bitowej trzeba „odciąć” i dodać do uzyskanej liczby 16-bitowej:

```
   8D 9C
+  00 03
-------
   8D 9F
```

Na koniec pozostało jeszcze uzyskaną liczbę dopełnić do jedności, czyli przeprowadzić operację: `0xFFFF - 0x8D9F = 0x7260`. Liczba `0x7260` jest poszukiwaną sumą kontrolną nagłówka TCP.

2.7.2. Graf przejść stanów TCP

Aby zagwarantować, że dane przesyłane z jednej maszyny do drugiej nie są ani tracone, ani duplikowane używa się podstawowej metody znanej jako pozytywne potwierdzanie z retransmisją. W tym przypadku, gwarancję dotarcia poszczególnych danych do odbiorcy zapewnia mechanizm potwierdzania odebrania segmentu przez odbiorcę. W przypadku odebrania segmentu odbiorca wysyła do nadawcy potwierdzenie jego odebrania. Nadawca po otrzymaniu potwierdzenia odbioru może wysłać następny segment danych. Przy każdym wysyłaniu segmentu nadawca uruchamia zegar odliczający czas od jego wysłania. W przypadku, gdy zostanie przekroczony czas przeznaczony na otrzymanie potwierdzenia odbioru danego segmentu jest on retransmitowany. Każdy kolejny wysyłany segment TCP jest kolejno oznaczany. Przy czym numerowany nie jest sam segment, ale kolejne bajty danych przesyłanych w segmencie. Taki sposób numeracji pozwala realizować transmisje strumieniową. Po stronie odbiorcy numeracja pozwala poprawnie złożyć odbierane dane oraz odrzucić powtarzające się segmenty. Dodatkowo odbiorca sprawdza sumę kontrolną każdego odebranego segmentu. W przypadku odebrania segmentu z błędną sumą kontrolną jest on odrzucany i nie zostaje wysłane potwierdzenie odebrania tego segmentu. Ilustracja graficzna potwierdzenia odbioru segmentu oraz jego retransmisji została pokazana na rys. 2.23.



Rys. 2.23. Przykład transmisji z potwierdzeniem odbioru danych

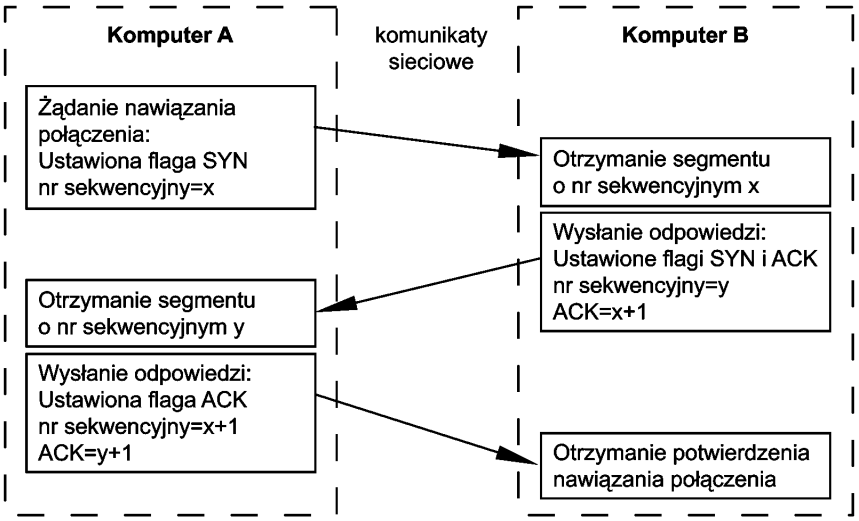
Mechanizm potwierdzania odbioru danych jest skuteczny jednak mało efektywny. Kanał transmisyjny w czasie oczekiwania na potwierdzenie od odbiorcy nie jest wykorzystywany. W celu uniknięcia takiej sytuacji wprowadzono w protokole TCP metodę przesuwanego okna. Nadawca określa ile bajtów danych jest w stanie odebrać. Ilość danych, którą jest w stanie przetworzyć odbiorca jest nazywana szerokością okna. Nadawca wysyła taką ilość danych, która zmieści się w oknie. W praktyce oznacza to, że okno określa liczbę segmentów, którą nadawca wysyła do odbiorcy nie oczekując na potwierdzenie odbioru. Po uzyskaniu potwierdzenia odbioru pierwszego segmentu nadawca może wysłać kolejny segment oczekujący na wysłanie, co jest równoznaczne z przesunięciem się okna o jedną jednostkę. Metoda przesuwającego się okna pozwala dostosować ilość wysyłanych segmentów do aktualnego obciążenia sieci oraz możliwości odbiornika.

Realizacja mechanizmu potwierdzania danych wymaga ustanowienia połączenia między nadawcą i odbiorcą. Połączenie TCP zawsze odbywa się między dwoma końcówkami, z których każda zdefiniowana jest przez tzw. gniazdo. Gniazdo składa się z zestawienia adresu

IP oraz numeru portu. Numer portu określa proces, do którego kierowane są dane. Część numerów portów została na stałe przypisana do określonych zadań.

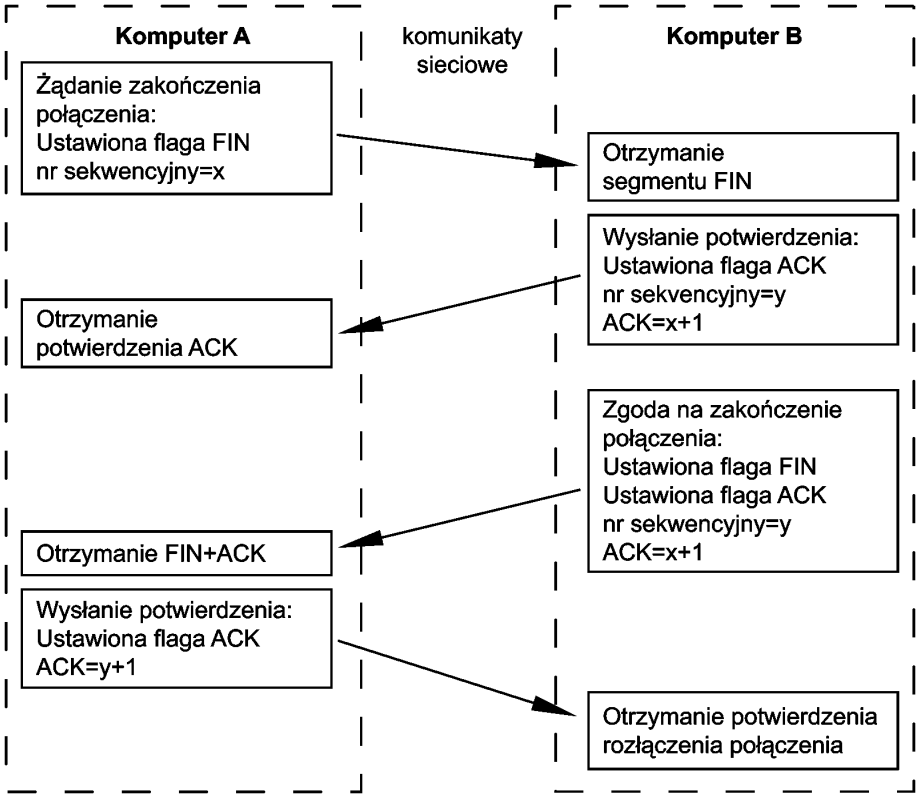
Połączenie TCP między dwoma gniazdami jest połączeniem full-duplex, które pozwala na przesyłanie danych w dwóch kierunkach pomiędzy połączonymi końcówkami. W trakcie nawiązywania połączenia przekazywana jest informacja o numerach portów używanych przez stronę przeciwną, numerach sekwencyjnych wykorzystywanych do kontroli kolejności przesyłanych danych oraz opcjonalnie uzgadniana jest maksymalna wielkość przesyłanych segmentów. Jedna z końcówek biorących udział w połączeniu pełni rolę klienta i jest końcówką aktywną, która inicjuje połączenie natomiast druga końcówka jest pasywna i pełni rolę serwera, który oczekuje na nadchodzące połączenia. Protokół TCP przy nawiązywaniu połączenia korzysta z mechanizmu trzyetapowego porozumienia (three-way handshake).

Mechanizm nawiązywania połączenia zostanie opisany na przykładzie łączenia się komputera A z komputerem B. Komputer A wysyła do komputera B segment z ustawioną flagą SYN oraz początkowym numerem sekwencyjnym ISN (Initial Sequence Number) równym x, gdzie x jest losowo wybraną liczbą z przedziału od 0 do 2^{32}-1. Odebrany segment jest informacją dla komputera B, że komputer A chce nawiązać z nim połączenie. W odpowiedzi komputer B wysyła do komputera A segment z ustawioną flagą SYN, własnym numerem ISN równym y oraz potwierdzeniem dla komputera A (ACK=x+1 oraz ustawiona flaga ACK). Następnie komputer A odpowiada wysyłając potwierdzenie dla komputera B (ACK=y+1 oraz ustawiona flaga ACK). Odebranie przez komputer B segmentu potwierdzającego oznacza, że komputery A i B nawiązały połączenie, posiadają uzgodnione numery sekwencyjne i są gotowe do wymiany danych. Ilustrację nawiązania połączenia przedstawia rys. 2.24.



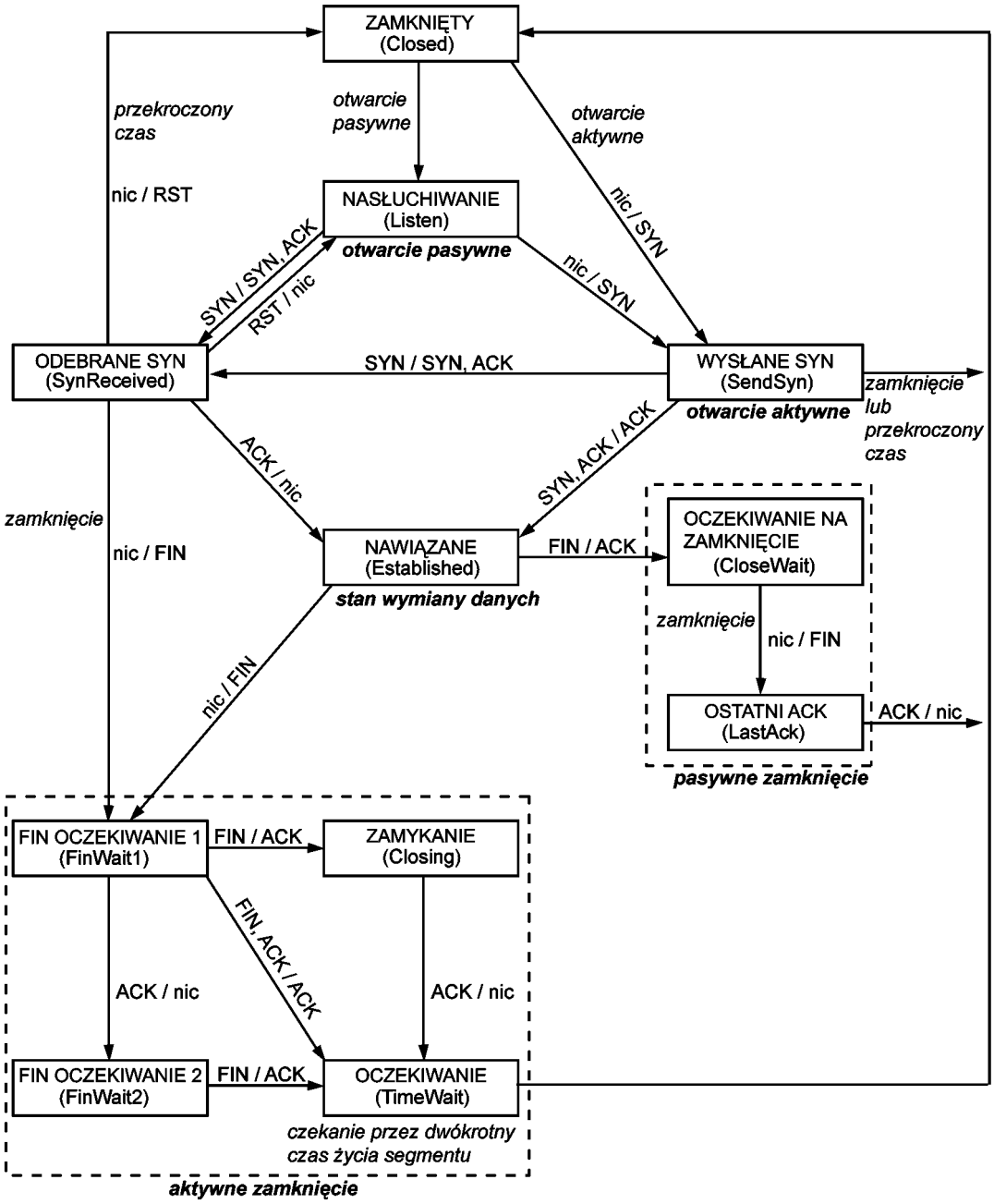
Rys. 2.24. Mechanizm nawiązywania połączenia protokołu TCP

Rozłączanie połączenia odbywa się na podobnej zasadzie, co połączenie. Komputer A wysyła do komputera B segment z ustawioną flagą FIN, co oznacza chęć przerwania połączenia. Komputer B wysyła potwierdzenie otrzymania segmentu FIN oraz następnie jak uzna, że może zakończyć połączenie wysyła do komputera A segment z ustawioną flagą FIN. Po otrzymaniu od komputera A potwierdzenia otrzymania segmentu FIN połączenie jest rozłączane. Taki mechanizm kończenia połączenia gwarantuje, że wszystkie dane zostaną dostarczone do odbiorcy gdyż żeby zakończyć połączenie dwie strony muszą wyrazić na to zgodę. Ilustrację rozłączenia połączenia przedstawia rys. 2.25.



Rys. 2.25. Mechanizm rozłączania połączenia TCP

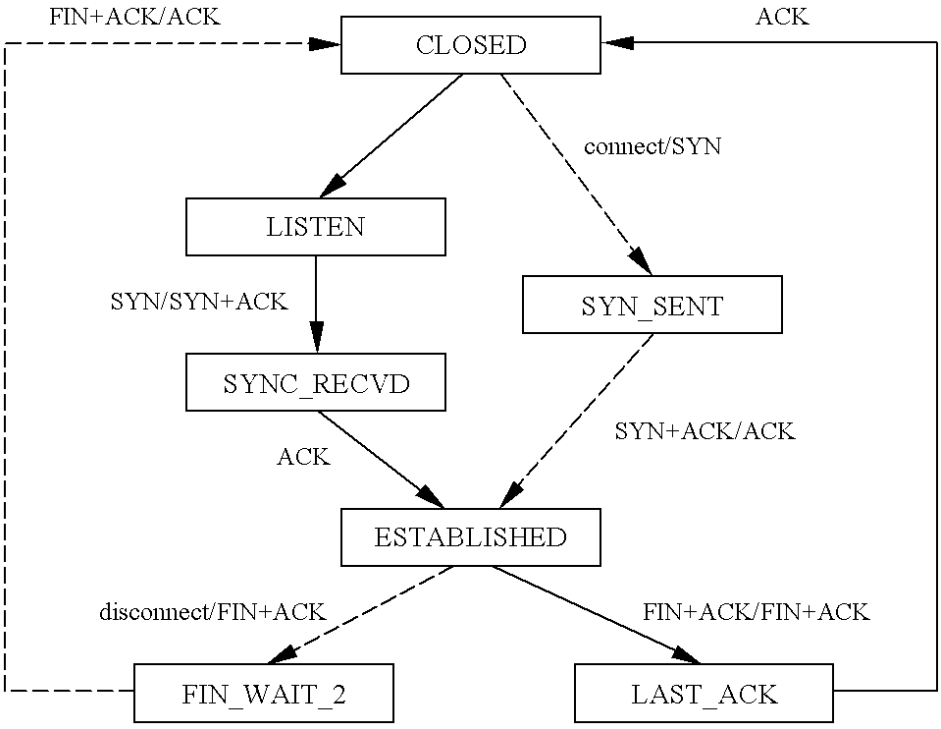
Wszystkie stany, w jakich mogą się znajdować końcówki biorące udział w połączeniu TCP oraz warunki przejścia między tymi stanami przedstawia graf przejść stanów TCP na rys. 2.12. Na grafie zostały przedstawione zarówno stany dla serwera oraz klienta. Przy strzałkach pokazane są warunki, jakie muszą być spełnione żeby przejść między poszczególnymi stanami. Format zapisu warunku przejścia między stanami jest następujący: dane odebrane przez serwera / dane wysłane przez serwer. Przykładowo przejście ze stanu „WYSŁANE SYN" do stanu „NAWIĄZANE" nastąpi, kiedy zostanie odebrany segment z ustawionymi flagami SYN i ACK oraz wysłany segment potwierdzający z ustawioną flagą ACK.



Rys. 2.26. Pełny graf przejść protokołu TCP

2.7.3. Implementacja protokołu TCP w mikroserwerze

Uwzględniając niewielkie moce obliczeniowe mikroserwerów bazujących ma prostych 8-bitowych mikrokontrolerach należy upraszczać graf przejść stanów TCP. Przykład uproszczonego grafu TCP zawierającego obsługę serwera jak i klienta przedstawiono na rys. 2.27.



Rys 2.27. Przykład minimalnego grafu TCP.

Przejścia pomiędzy stanami w grafie realizowane przez klienta oznaczono przerywanymi strzałkami.

Dla serwera kolejność przejść stanów jest odpowiednia:

- Po włączeniu zasilania, restarcie mikrokontrolera lub przy nieaktywnym połączeniu, TCP znajduje się w stanie LISTEN (ang. nasłuch). Serwer oczekuje na nadchodzące połączenia tylko na konkretnych portach (np. 80).
- Jeśli odebrany zostanie segment TCP z flagą SYN, to wysłany zostaje segment z ustawioną flagą SYN i flagą ACK. TCP przechodzi do stanu SYN_RECVD (ang. odebrano SYN).
- Nadejście segmentu z flagą ACK powoduje otwarcie wirtualnego połączenia. TCP przechodzi do stanu ESTABLISHED (ang. ustanowione).
- Klient chcąc zakończyć połączenie z serwerem wysyła segment TCP z ustawionymi flagami FIN i ACK. Serwer odpowiada takim samym segmentem, zaś TCP przechodzi do stanu LAST_ACK (ang. oczekiwanie na ostatnie ACK). W tym grafie nie przewidziano możliwości zamykania połączenia przez serwer.
- Po odebraniu segmentu z flaga ACK połączenie zostaje prawidłowo zakończone. TCP przechodzi do stanu CLOSED (ang. zamknięte), a następnie do stanu LISTEN.

W przypadku klienta przejścia między stanami są następujące:

- Np. naciśnięcie przycisku początkuje proces nawiązywania połączenia. Do serwera danej usługi o znanym adresie IP i MAC, wysyłany jest segment o ustawionej fladze SYN. TCP przechodzi do stanu SYN_SENT (ang. wysłano SYN).
- Serwer odpowiada segmentem z flagami SYN i ACK. Wówczas zostaje odesłany segment z flagą ACK. Wirtualne połączenie zostaje otwarte, zaś TCP jest w stanie ESTABLISHED.
- Połączenie zostaje zamknięte przez serwer lub klienta. Zamknięcie przez serwer omówiono wyżej. Zamknięcie połączenia przez klienta jest zapoczątkowane wysłaniem

przez niego segmentu z ustawionymi flagami FIN i ACK. TCP przechodzi do stanu FIN_ WAIT_2.
- Po odebraniu z serwera segmentu z ustawionymi flagami FIN i ACK, zostaje wysłany segment z flaga ACK. Połączenie zostało pomyślnie zamknięte, zaś TCP przechodzi do stanu CLOSED, a następnie LISTEN.

Oczywiście w dowolnym stanie grafu TCP następuje odpowiednia, przedstawiona wcześniej, wymiana numeru potwierdzenia (ACK) i porządkowego (SEQ).

Podczas konstruowania mikroserwera można przyjąć następujące założenia odnośnie protokołu TCP:
- pole „numer potwierdzenia" w trakcie nawiązywania połączenia z klientem przyjmuje zawsze tą samą wartość (np. `0x1000`),
- „rozmiar okna" przyjąć np. równy `0xFF`,
- pole „numer porządkowy" w trakcie nawiązywania połączenia z innym serwerem ma stałą wartość (np. `0x1000`),
- serwer nie ma możliwość zamknięcia aktywnego połączenia,
- nie ma możliwość automatycznego powrotu do stanu CLOSED jeśli próba wysłania danych przez mikroserwer w trybie klienta nie powiodła się. Jedynym wyjściem ze tego stanu jest restart mikrokontrolera,
- mikroserwer wykorzystuje tylko ustalone porty: np. 80, 1024,
- brak implementacji retransmisja przy braku potwierdzenia. W takiej sytuacji mikroserwer zakleszcza się w odpowiednim stanie grafu TCP dopóty transmisja nie zostanie wznowiona lub mikroserwer zostanie zrestartowany.

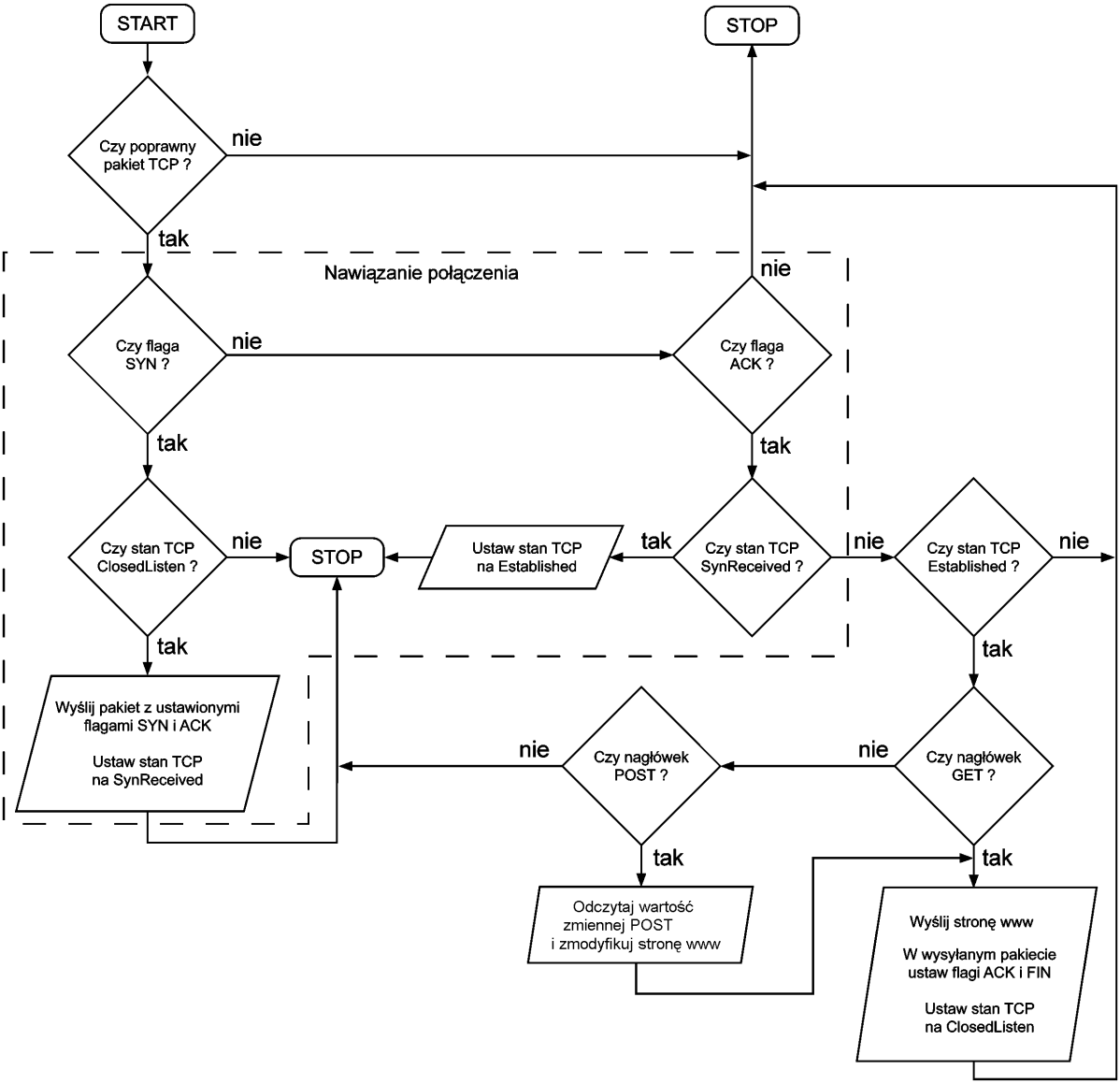
Oczywiście są to ostre ograniczenia funkcjonalności grafu TCP, lecz znacznie upraszczające obsługę protokołu TCP.

Drugi sposób uproszczenia grafu TCP podano poniżej. W tym przypadku procedura obsługi protokołu TCP jest następująca:
- Na początku sprawdzana jest poprawność segmentu TCP. Jeżeli segment nie jest poprawny to następuje zakończenie procedury, w przeciwnym wypadku następuje jego obsługa.
- Przyjęto założenie, że wszystkie wysyłane pakiety dochodzą do odbiorcy, w związku z tym nie została zaimplementowana retransmisja utraconych pakietów. Stany Closed i Listen (rys. 2.26) zostały połączone w jeden stan ClosedListen.
- W pełni zostało zaimplementowane nawiązywanie połączenia - tzw. trzy etapowy handshake.
- Pominięte zostało poprawne kończenie ustanowionego połączenia. Segment z danymi wysyłanymi z mikroserwera ma ustawioną flagę FIN i po wysłaniu przechodzi w stan ClosedListen co jest jednoznaczne z zakończeniem połączenia.
- Po inicjalizacji mikroserwera graf przejść TCP znajduje się w stanie ClosedListen.
- Odebranie pakietu z ustawioną flagą SYN rozpoczyna procedurą nawiązania połączenia.
- W odpowiedzi na SYN wysyłany jest pakiet z ustawionymi flagami SYN i ACK, a stan TCP jest ustawiany na SynReceived. Na tak wysłany pakiet mikroserwer otrzymuje odpowiedz ACK i przechodzi do stanu Established. W tym miejscu grafu TCP obydwa urządzenia są ze sobą połączone i mogą wymieniać się danymi.
- Zakłada się komunikację mikroserwera wyłącznie z przeglądarką internetową, zatem w stanie TCP Established pakiety z ustawioną flagą ACK są wyłącznie sprawdzane pod

kątem komunikatów przeglądarki internetowej. W przypadku odebrania pakietu z komendą GET mikroserwer wysyła do przeglądarki pakiet ze stroną internetową.

- Po przejściu z jednego stanu TCP do drugiego rozpoczyna się odliczanie czasu. Jeżeli mikroserwer zbyt długo przebywa w jednym stanie, np. z powodu zerwanego połączenia w stanie Established, automatycznie przechodzi do stanu ClosedListen. Zapobiega to zawieszeniu się mikroserwera w którymś ze stanów TCP.

Na poniższym rysunku pokazano graf TCP zmodyfikowany zgodnie z powyższymi założeniami. Warto zauważyć fakt, iż zawarto również w nim obsługę protokołu warstwy aplikacji HTTP. Takie rozwiązanie jest do przyjęcia, gdy w mikroserwerze implementujemy jeden protokół warstwy aplikacji i w danej chwili mikroserwer obsługuje tylko jedno połączenie oraz cała strona www wraz z nagłówkiem mieści się w jednym segmencie TCP.



Rys. 2.28. Uproszczony graf TCP wraz z obsługą protokołu HTTP

Na listingu 2.23 pokazano przykładowe kodowanie stanów grafu TCP oraz deklaracje flag nagłówka segmentu TCP.

```
// flagi TCP

#define FIN  0x01
#define SYN  0x02
#define RST  0x04
#define PSH  0x08
#define ACK  0x10

// stany TCP

#define ClosedListen            0
#define SynReceived             1
#define Established             2
#define FinWait1                3
#define FinWait2                4
#define Closing                 5
#define TimeWait                6
#define CloseWait               7
#define LastAck                 8
```

Listing 2.23. Deklaracje opisu stanów grafu TCP

Na poniższych listingach pokazano funkcje obsługi protokołu TCP dla mikroserwera PICDEM.net napisane w języku C18 dla mikrokontrolerów rodziny PIC16 i PIC18. Funkcje te obsługują pełny graf TCP.

```
#define IPHDR_LEN   20      // długość nagłówka IP
#define TCPHDR_LEN  20      // długość nagłówka TCP

WORD locport, remport;      // numery portów TCP
LWORD rseq, rack;           // numer porządkowy i numer potwierdzenia ACK

BOOL get_tcp(void)
{
    int hlen, n;
    BOOL ret=0;

    checkhi = checklo = 0;
    if (get_word(remport) && get_word(locport) &&   // pobierz numery
                                                    // portów
        get_lword(rseq.l) && get_lword(rack.l) &&   // pobierz nr porządkowy
                                                    // i potwierdz.
        get_byte(hlen) && get_byte(rflags) &&       // pobierz długość
                                                    //i flagi
        skip_word() && skip_lword())                // pomiń rozmiar okna,
                                                    // sumę kontrolną
                                                    //i wskaźnik pilności
    {
        iplen -= IPHDR_LEN;                         // określ długość
                                                    // segmentu TCP
        check_byte(iplen>>8);                       // uwzględnij
                                                    // pseudonagłówek IP
        check_byte(iplen);
        check_lword(local.l);
        check_lword(remote.l);
        check_byte(0);
        check_byte(PCOL_TCP);
        rxout = (hlen>>2) + IPHDR_LEN;
```

```
        rpdlen = iplen - rxout + IPHDR_LEN;
        checkhi += rdcheckhi;
        checklo += rdchecklo;
        ret = (checkhi==0xff) && (checklo==0xff);
    }
    return(ret);
}
```

Listing 2.24. Procedura pobierająca i sprawdzająca nagłówek TCP

```
void put_tcp(void)
{
    WORD len;

    checkflag = 0;                  // zeruj flagę wyboru bajtu w sumowaniu
kontrolnym
    put_word(locport);              // ustaw lokalny i zdalny port
    put_word(remport);
    put_lword(rack.l);              // ustaw potwierdzenie i nr porządkowy
    put_lword(rseq.l);
    put_byte(TCPHDR_LEN*4);         // długość nagłówka (dlaczego *4 ?)
    put_byte(tflags);
    put_byte(0x0b);                 // rozmiar okna na 3000 bajtów
    put_byte(0xb8);
    check_lword(local.l);           // uwzględnij pseudonagłówek IP
    check_lword(remote.l);
    check_byte(0);
    check_byte(PCOL_TCP);
    len = tpdlen + TCPHDR_LEN;
    check_byte(len>>8);
    check_byte(len);
    checkflag = 0;
    put_byte(~checkhi);             // wyślij sumę kontrolną
    put_byte(~checklo);
    put_nullw();                    // wyślij zerowy wskaźnik pilności
    if (!txi2c)                     //
        txin += tpdlen;             // ...uwzględnij plik danych
    tx_end();                       // wyślij dane...
}
```

Listing 2.25. Procedura generująca i wysyłająca nagłówek TCP

```
void tcp_rx(void)
{
    BYTE *p, *q;
    BOOL tx=1;

    tpdlen = 0;                      // Na początek bez danych
                                     // użytkownika
    tflags = TACK;                   // ... i ustaw flagę ACK
    if (txflag || (rflags & TRST))   // Jeśli odebrano RESET, lub zajęty?
        tx = 0;                      // …opuść procedurę
    else if (rflags & TSYN)          // Odebrano ustawioną flagę SYN?
    {
        inc_lword(rseq.l);           // Wyznacz numer potwierdzenia ACK
```

```
        if (locport==DAYPORT || locport==HTTPORT)
        {                                   // Czy port jest obsługiwany?
            rack.w[0] = 0xffff;
            rack.w[1] = concount++;
            tflags = TSYN+TACK;             // Ustaw flagi wyjściowe SYN i ACK
        }
        else                                // Port nie jest obsługiwany
            tflags = TRST+TACK;             // …więc wyślij RESET
    }
    else if (rflags & TFIN)                 // Odebrano ustawioną flagę FIN?
        add_lword(rseq.l, rpdlen+1);        // ...więc wyślij dane i ustaw
                                            // flagę FIN
    else if (rflags & TACK)                 // Odebrano ustawioną flagę ACK?
    {
        if (rpdlen)                         // ...więc ustaw odpowiedź dla danych
                                            // wyjściowych
            add_lword(rseq.l, rpdlen);
        else                                // Jeśli nie ma danych nie wysyłaj
                                            // ACK
            tx = 0;
        if (locport==DAYPORT && rack.w[0]==0)
        {                                   // Przyszło żądanie DAYTIME
            daytime_rx();                   // ...wygeneruj odpowiedź DAYTIME
            tx = 0;                         // ...i zamknij połączenie

        }
        else if (locport==HTTPORT && rpdlen)
        {                                   // Przyszło żądanie HTTP
            if (http_rx())                  // ...wygeneruj dane HTTP i zamknij
                                            // połączenie
                tx = 0;
            else                            // …lub tylko zamknij połączenie
                tflags = TFIN+TACK;
        }
    }
    if (tx)                                 // Jeśli segment do wysłania (ACK)
                                            //   to...
    {
        put_ip(TCPHDR_LEN);                 // ...wyślij nagłówek IP
        checkhi = checklo = 0;              // ...zeruj symy kontrolne
        put_tcp();                          // ...wyślij nagłówek TCP
    }
}
```

Listing 2.26. Procedura przetwarzająca segment TCP

2.8. Warstwa aplikacji na przykładzie protokołu HTTP

Protokół HTTP jest protokołem warstwy aplikacyjnej. Jego podstawowym zastosowaniem jest pobieranie z serwera strony WWW oraz innych zasobów znajdujących się na serwerze. Umożliwia on również wysyłanie do serwera WWW informacji zwrotnych np. haseł dostępowych itp. Działanie protokołu opiera się na połączeniu TCP. Dialog między klientem a serwerem WWW oparty jest o jednokrotną wymianę komunikatów. Klient po nawiązaniu połączenia wysyła do serwera komunikat zapytania i przechodzi w stan oczekiwania na odpowiedź. Serwer po otrzymaniu pytania wysyła stosowną odpowiedź i się rozłącza.

Komunikat zapytania kierowany do serwera składa się z identyfikatora metody, identyfikatora zasobu oraz numeru wersji protokołu HTTP.

Identyfikator metody określa rodzaj usługi, której domaga się klient. W praktyce stosuje się trzy metody: GET, HEAD i POST. Najczęściej stosowaną jest metoda GET. W wyniku jej działania pobierana jest z serwera cała zawartość zasobu określonego identyfikatorem zasobu. Komunikat zapytania GET składa się z nagłówka i pustej linii. W odpowiedzi serwer wysyła nagłówek odpowiedzi, pustą linię i ciało zasobu. Metoda HEAD jest podobna do GET z tą różnicą, że w odpowiedzi serwer nie wysyła ciała zasobu tylko sam nagłówek odpowiedzi. Metoda ta jest stosowana przy sprawdzaniu dostępności oraz atrybutów zasobu. Metoda POST jest wykorzystywana do przekazywania danych od klienta do serwera. Identyfikator zasobu określa lokalizacje i rodzaj zasobu, którego dostarczenia domaga się klient. Przeważnie jest to bezwzględna ścieżka dostępu do pliku na serwerze, zaczynająca się od znaku „/”. W przypadku, gdy występuje sam znak „/” oznacza to domyślny zasób na serwerze. Przykład komunikatu zapytania dla trzech metod przedstawia tabela 2.7.

Tabela 2.7. Przykładowe komunikaty zapytania

Metoda	**Przykładowy komunikat zapytania**
GET	GET /dane/ HTTP/1.0
HEAD	HEAD /archiwum/dane.zip HTTP/1.0
POST	POST /cgi-bin/formularz.pl HTTP/1.0

Nagłówek odpowiedzi, którą wysyła serwer składa się z numeru wersji protokołu, trzycyfrowego kodu odpowiedzi oraz komentarza opisującego dany kod odpowiedzi oraz ewentualnej zawartości przesyłanej do klienta. Przykładowe linie statusu odpowiedzi (protokół, kod i komentarz) przedstawia tabela 2.8.

Tabela 2.8. Przykładowe linie statusu odpowiedzi serwera WWW

HTTP/1.0 200 OK
HTTP/1.0 404 Not Found
HTTP/1.0 401 Unauthorized

Podstawowymi portami wykorzystywanymi przez protokół HTTP są porty o numerach 80 i 8080.

2.8.1. Implementacja protokołu HTTP w mikroserwerach

Protokół HTTP jest protokołem warstwy aplikacji prosto dającym się zaimplementować w mikroserwerach. Wynika to z faktu, iż jest on protokołem bezpołączeniowym, czyli dane (przy założeniu, że mieszczą się w jednym segmencie TCP) są przesyłane tylko na żądanie od klienta i to w jednym segmencie TCP. Nie ma potrzeby prowadzenia dialogu, podtrzymywania połączenia, realizacji jakiegokolwiek uzgadniania czy handshake.

Jeżeli strona www ma być statyczna można ją w prosty sposób zadeklarować jako tablicę char w pamięci programu mikrokontrolera. W przeciwnym przypadku w pamięci programu deklaruje się szablon strony, a jej poszczególne fragmenty są trzymane w zmiennych w pamięci danych również w postaci tablic typu char.

Na listingu 2.28 pokazano przykład deklaracji statycznej strony www.

```
unsigned char content[] PROGMEM= "HTTP/1.1 200 OK\r\nContent-type:
                                                  text/html\r\n\r\n";

unsigned char index[] PROGMEM =
                    "<HTML>\r\n<HEAD>\r\n<TITLE>Mikroserwer LAN</TITLE>\r\n"
                    "<meta http-equiv=\"content-type\" content=\"Text/Html;
                                                 charset=windows-1250\">\r\n"
                    "</HEAD>\r\n"
                    "<BODY >\r\n"
                    "<h1> Tytuł strony </h1><br>\r\n"
                    "<br>\r\n"
                    "<p> Tekst na stronie</p>\r\n"
                    "</BODY>\r\n"
                    "</HTML>";
```

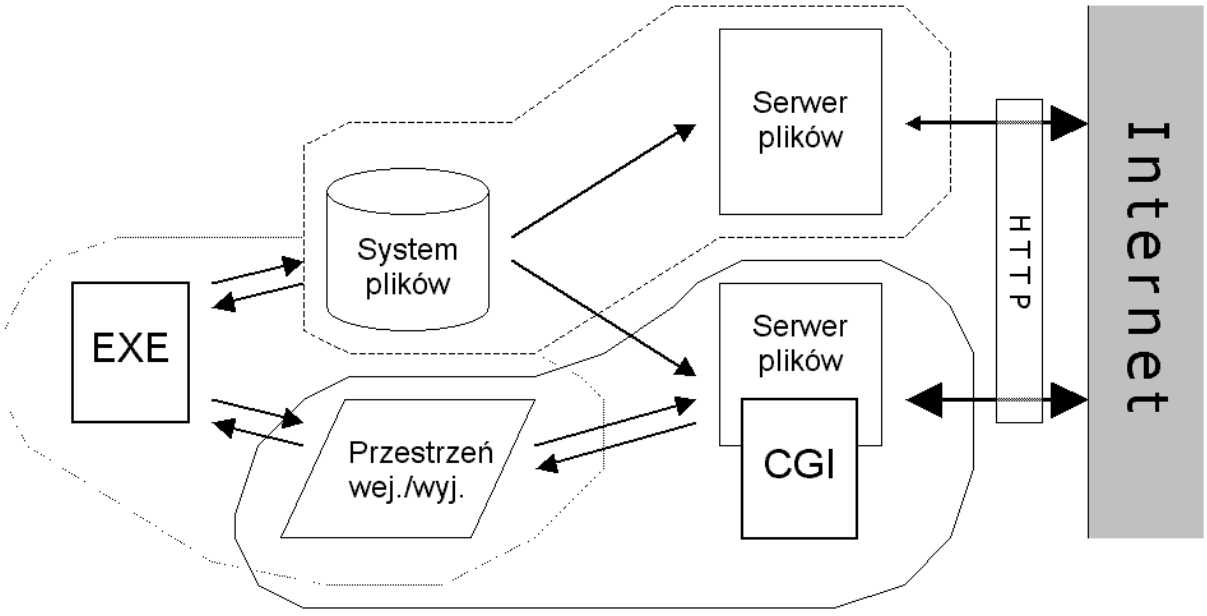
Listing 2.28. Deklaracja statycznej strony www w pamięci programu mikrokontrolera

Tak zadeklarowaną stronę przed wysłaniem umieszcza się w ramce wysyłanej do kontrolera sieci w sposób podany poniżej.

```
        ...
strncpy_P(&EthFrame[14+20+20],content,strlen_P(content));
strncpy_P(&EthFrame[14+20+20+strlen_P(content)],index,strlen_P(index));
        ...
```

Listing 2.29. Umieszczenie strony www w ramce przesyłanej do kontrolera sieci

Poniżej pokazano mikro-implementację protokołu HTTP w mikroserwerze PICDEM.net. Schemat funkcjonalny mikroserwera przedstawiono na rys. 2.29.



Rys. 2.29. Schemat funkcjonalny mikroserwera PICDEM.net

Strony www są umieszczone w bloku „System plików” (zewnętrzna szeregowa pamięć EEPROM 24LC256). Maksymalny rozmiar dokumentu wynosi 950 bajtów. Nagłówek HTTP musi zawierać:

```
HTTP/1.0 200 OK<CRLF>
Content-type: text/html<CRLF>
<CRLF>
```

Kolejne dwa listingi przedstawiają funkcje obsługujące żądania dostępu do stron www mikroserwera.

```
#define HTTP_FAIL       "HTTP/1.0 200 OK\r\n\r\nMicroServer ROM error\r\n"
#define HTTP_FAIL_LEN   42

BOOL http_rx(void) {
    int len, i;
    BOOL ret=0;
    char c;
    tpdlen = 0;                          // Czy przyszło żądanie GET?
    if (match_byte('G') && match_byte('E') && match_byte('T')) {
        ret = 1;
        skip_space();
        match_byte('/');                 // Początek URL zasobu
        for (i=0; i<ROM_FNAMELEN; i++) {   // Przekopiuj nazwę zasobu
            c = rxbuff[rxout];             // do bufora systemu plików
            if (c>' ' && c!='?') rxout++;  // Czy już koniec nazwy zasobu?
            else c = 0;
            romdir.f.name[i] = c;
        }
        if (find_file())            // Czy taki plik istnieje w ROM?
            check_formargs();       // Pobierz wartości z formularza
        else {                      // Brak pliku! Wyślij "index.htm"
            romdir.f.name[0] = 0;
            find_file();
        }
        if (!fileidx) {                  // Brak plików w ROM - błąd!
            tpdlen = HTTP_FAIL_LEN;      // Długość komunikatu o błędzie
            put_ip(TCPHDR_LEN+tpdlen);   // Wstaw nagłówek IP do bufora wyj.
            checkhi = checklo = 0;       // Resetuj sumy kontrolne
            strcpy(&txbuff[IPHDR_LEN+TCPHDR_LEN], HTTP_FAIL); // Kopiuj komunikat
            txin = IPHDR_LEN + TCPHDR_LEN;
            check_txbytes(tpdlen);       // Oblicz sumę kontrolną dla
                                         // komunikatu
            txin = IPHDR_LEN;            // Wróć do końca nagłówka IP
            tflags = TFIN+TPUSH+TACK;    // Zamknij połączenie po wysłaniu
            put_tcp();                   // Wstaw nagłówek TCP do bufora wyj
        }
        else {                           // Plik ROM odnaleziony
            tpdlen = romdir.f.len;       // Pobierz długość danych
            put_ip(TCPHDR_LEN+tpdlen);   // Wstaw nagłówek IP do bufora wyj.
            checkhi = checklo = 0;       // Resetuj sumy kontrolne
            check_byte(romdir.f.check); // Dodaj sumę kontrolną pliku ROM
            check_byte(romdir.f.check >> 8);
            tflags = TFIN+TPUSH+TACK;    // Zamknij połączenie po wysłaniu
            txi2c = 1;                   // Ustaw flagę wysyłania dla pliku
                                         // ROM
            put_tcp();                   // Wstaw nagłówek TCP do bufora wyj
        }
    }
    return(ret);
}
```

Listing 2.30. Przetwarzanie przychodzącego strumienia danych:

```
void check_formargs(void) {
    BOOL update=0;
    while (rxout < rxcount) {        // Argument zaczyna się od '?' lub '&'
        c = rxbuff[rxout++];
        if (c=='?' || c=='&') {      // Rozpoznaj zmienną i pobierz jej
                                     // wartość
            strcpy(temps, "nazwa_zmiennej=");  // Jakiej zmiennej szukamy..
            if (match_str(temps)) {                  // Czy znaleziono?
                update |= get_...(nazwa_zmiennej);   // Pobierz wartość
                continue;
            }
            ...
            ...
        }
    }
    if (update) {                    // Są nowe parametry!
        ...                          // Wymagana aktualizacja plików ROM
        ...
    }
}
```

Listing 2.31. Przetwarzanie danych z formularza:

2.8.2. Interfejs EGI – interaktywność na stronach www mikroserwerów

Firma Microchip zaproponowała interfejs EGI (Embedded Gateway Interface) zwiększający interaktywność stron www mikroserwerów sieciowych. Składa się on z kontrolek (obrazków) obrazujących pracę mikroserwera oraz przeznaczonych do jego sterowania. Imitują one diody LED, przełączniki, suwaki. Na rys. 2.30 pokazano przykłady elementów interfejsu EGI.

a) b)

Rys. 2.30. Przykłady elementów EGI: a) przełącznik dwustanowy, b) wskaźnik typu LED

Elementy graficzne mogą występować w dokumencie WWW w funkcji hiperlinków do aplikacji EGI z parametrem identyfikującym kontrolkę, np:

```
<a href="appl.egi?switch1"><img src="switch1.gif"></a>
<a href="appl.egi?switch0"><img src="switch0.gif"></a>
```

lub w funkcji kontrolek formularza, których stan jest przesyłany metodą GET:

```
<form action="appl.egi">
<input type="image" name="switch1" src="switch1.gif">
<input type="image" name="switch0" src="switch0.gif">
</form>
```

generując zapytanie zawierające koordynaty, np.:

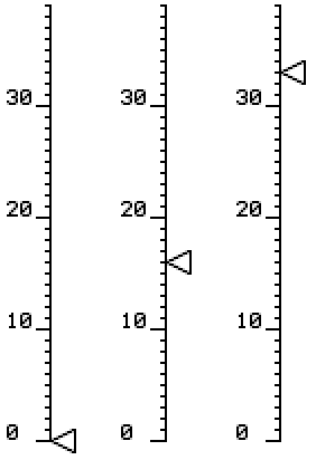
```
GET /appl.egi?switch1.x=6&switch1.y=7 HTTP/1.0
```

gdzie:

name.**x** = *X*
name.**y** = *Y*

stąd można wykorzystać elementy graficzne w funkcji suwaków (rys. 2.31). Ogólna postać kodu HTML dla pojedynczego elementu EGI pełniącego funkcję kontrolki zapytania jest następująca:

<input type=”image” name=”*name***” src=”***plik***”>**



Rys. 2.31. Suwaki analogowe

Na listingu 2.32 przedstawiono kod HTML opisujący suwaki analogowe (rys. 2.31).

```
<table border=0 cellpadding=0 cellspacing=0>
<tr align=center  valign=bottom >
<td><img src="v40.gif"></td>
<td><img src="lptr.gif"><br>
    <img src="clr.gif"  height=0  width=10></td>
<td><img src="v40.gif"></td>
<td><img src="lptr.gif"><br>
    <img src="clr.gif"  height=85  width=10></td>
<td><img src="v40.gif"></td>
<td><img src="lptr.gif"><br>
    <img src="clr.gif"  height=170  width=10></td>
</tr>
</table>
```

Listing 2.32. Kod HTML opisu suwaków analogowych

Tag odpowiedzialny za wyświetlenie wartości *n* w kodzie HTML może być reprezentowany znacznikiem EGI zapisanym zgodnie z formą:

<!--#$zmienna -->

zdefiniowanym w tym przypadku następująco:

<!--#$ptr**$_i$ **--> ≡ <img src="clr.gif" height= $5n_i$ **width=10>**

wówczas kod HTML wskaźników będzie wyglądał następująco:

```
<table border=0 cellpadding=0 cellspacing=0>
<tr align=center  valign=bottom >
<td><img src="v40.gif"></td>
<td><img src="lptr.gif"><br><!--#$ptr1--></td>
<td><img src="v40.gif"></td>
<td><img src="lptr.gif"><br><!--#$ptr2--></td>
<td><img src="v40.gif"></td>
<td><img src="lptr.gif"><br><!--#$ptr3--></td>
</tr>
</table>
```

Listing 2.33. Kod HTML obsługi suwaków analogowych wraz z elementami EGI

Poniżej podano kilka przykładów elementów EGI:

<!--#ptr_i$ --> ≡ **<img src="clr.gif" height=** $5n_i$ **width=10>**

<!--#led_i$ --> ≡ **<img src="ledon.gif">**
lub **<img src="ledoff.gif">**

<!--#$switch$_i$ --> ≡ **<input type=image name="switch** $_i$**" src="switchu.gif">**
lub **<input type=image name="switch** $_i$**" src="switchd.gif">**

<!--#$scale$_i$ --> ≡ **<input type=image name="slider** $_i$**" src="v40.gif">**

Oraz sposób przetwarzania elementów EGI:

Nastawa	**GET /appl.egi?slider1.x=6&slider1.y=50 HTTP/1.0**
Przetrwarzanie WebSerwer	n_1 **=** (200−**slider1.y**) / 5
Wynik	**<img src="clr.gif" height=** $5n_i$ **width=10>**

3. Bibliografia

Niniejszy manuskrypt powstał na podstawie materiałów pochodzących z następujących pozycji literaturowych:

1. Rzeszotarski R.: „Mikroserwer wspomagający diagnostykę sieci LAN", Praca dyplomowa, kierujący pracą mgr inż. Adamczyk A., Politechnika Gdańska, Wydział ETI, Katedra Optoelektroniki i Systemów Elektronicznych, Gdańsk 2006.
2. Załęski D.: „Mikroserwer TCP/IP oparty na mikrokontrolerze AT89S53 i kontrolerze sieci Ethernet CS8900A", Praca dyplomowa, kierujący pracą dr inż. Czaja Z., Politechnika Gdańska, Wydział ETI, Katedra Metrologii i Systemów Elektronicznych, Gdańsk 2004.
3. Bentham J.: „TCP/IP Lean. Web servers for embedded systems", CMP Books, Lawrence, Kansas, USA, 2000.