Systemy odporne na błędy sprawozdanie z realizacji projektu

Maciej Bandura, Marcin Ślusarczyk; 2ID21B

6 maja 2025

1. Wstęp

Wykonywany temat: System składający się z 8 serwerów połączonych w graf. Serwer nadzorujący wysyła 16-bitową informację zabezpieczoną kodem korekcyjnym Hamminga, z możliwością symulowania błędów podczas transmisji danych.

1.1 Założenia projektowe oraz funkcjonalności

- System składa się z 8 serwerów połączonych w graf, umożliwiający komunikację między nimi.
- Jeden z serwerów pełni funkcję serwera nadzorującego, który zarządza przesyłaniem danych i kontrolą poprawności transmisji.
- Topologia grafu może być dowolna, np. pełne połączenie, pierścień, drzewo, siatka częściowa - struktura będzie definiowana przez użytkownika
- Serwer nadzorujący przesyła 16-bitowe bloki danych, które są zabezpieczone kodem korekcyjnym Hamminga
- Kody Hamminga pozwalają na korekcję pojedynczych błędów i detekcję podwójnych błędów, co zwiększa niezawodność systemu.
- Każdy serwer odbierający musi być wyposażony w mechanizm dekodowania i sprawdzania syndromu błędu. Po otrzymaniu danych, wykonuje dekodowanie Hamminga, aby sprawdzić poprawność transmisji. W przypadku wykrycia pojedynczego błędu, serwer koryguje go samodzielnie.
- Symulacja błędów jest realizowana przez użytkownika dla każdego serwera indywidualnie.
- Serwer odbierający raportuje raportuje wystąpienie błędu oraz skuteczność jego korekcji.

1.2 Wstęp teoretyczny

Celem realizowanego projektu jest stworzenie systemu transmisji danych z ośmioma serwerami połączonymi w grafie, w którym komunikacja odbywa się za pomocą 16-bitowych informacji zabezpieczonych kodem korekcyjnym Hamminga. Aby zapewnić niezawodność transmisji danych, w systemie wykorzystany zostanie kod **Hamminga z dodatkowym bitem parzystości SECDED** (Single Error Correction, Double Error Detection), który zapewni zarówno wykrywanie, jak i korekcję pojedynczych błędów, a także detekcję podwójnych błędów.

Kod Hamminga jest jedną z najczęściej stosowanych metod **wykrywania i korekcji błędów** w systemach komunikacyjnych i komputerowych. Jego główną zaletą jest zdolność do wykrywania i naprawiania pojedynczych błędów w danych bez konieczności ich retransmisji. Działa to na zasadzie dodania do danych dodatkowych bitów

parzystości, które są rozmieszczane w odpowiednich miejscach w danych, umożliwiając w ten sposób identyfikację, który bit uległ zmianie w wyniku błędu transmisji. Dzięki temu, w przypadku wykrycia błędu w jednym z bitów, system jest w stanie go skorygować i przywrócić poprawność danych.

Jednak klasyczny kod Hamminga, choć skuteczny w przypadku wykrywania i korekcji pojedynczych błędów, nie radzi sobie z wykrywaniem podwójnych błędów. W sytuacjach, gdy w przesyłanych danych wystąpią **dwa błędy**, klasyczny kod Hamminga **nie jest w stanie ich wykryć**, co może prowadzić do nieprawidłowej korekcji i w konsekwencji do błędnych danych. Aby rozwiązać ten problem, wprowadzono rozszerzenie kodu Hamminga, zwane SECDED. Jest to modyfikacja klasycznego kodu Hamminga, która dodaje **dodatkowy bit parzystości**, umożliwiający detekcję dwóch błędów jednocześnie, a jednocześnie pozwala na korekcję pojedynczych błędów. Dzięki temu, kod SECDED zapewnia wyższą niezawodność transmisji, umożliwiając wykrywanie błędów, które mogą wystąpić w wyniku zakłóceń w sieci.

W omawianym projekcie, serwer nadzorujący będzie odpowiedzialny za wysyłanie 16-bitowych bloków danych, które będą zabezpieczone tym kodem, oraz za symulowanie błędów w transmisji w celu testowania skuteczności tego mechanizmu korekcji. Symulowanie błędów w transmisji pozwala na przeprowadzenie testów w różnych warunkach, sprawdzając, jak system radzi sobie z błędami pojedynczymi oraz podwójnymi, a także w jaki sposób reaguje na ewentualne zakłócenia w sieci.

1.3 Wybór technologii z stosownym uzasadnieniem

W projekcie zastosowane zostaną technologie **Docker** oraz **PHP** w celu zapewnienia wysokiej niezawodności, elastyczności oraz łatwego zarządzania serwerami i aplikacjami.

1.3.1 Docker

Docker to narzędzie do konteneryzacji, które pozwala na uruchamianie aplikacji w odizolowanych środowiskach zwanych kontenerami. Dzięki temu możliwe jest stworzenie spójnego środowiska pracy, niezależnie od infrastruktury serwera. Docker pozwala na łatwe tworzenie, wdrażanie i skalowanie aplikacji w różnych środowiskach.

Zalety użycia Dockera w tym projekcie:

- **Izolacja środowisk** – Każdy serwer w systemie może być uruchomiony w osobnym kontenerze, co zapewnia pełną izolację aplikacji i minimalizuje ryzyko konfliktów między zależnościami.
- **Łatwość skalowania** – Docker umożliwia szybkie tworzenie nowych instancji serwerów (kontenerów), co jest istotne, jeśli w przyszłości projekt będzie musiał obsługiwać większą liczbę serwerów w grafie lub innych rozproszonych systemach.

- **Niezależność od systemu operacyjnego** – Docker zapewnia, że aplikacja będzie działać w ten sam sposób na różnych systemach operacyjnych, co znacząco upraszcza wdrożenia i zarządzanie infrastrukturą.

Wybór technologii **Docker** i **PHP** jest uzasadniony potrzebą stworzenia skalowalnego, niezawodnego i elastycznego systemu, który będzie łatwy do wdrożenia, zarządzania i testowania. PHP, dzięki swojej elastyczności i wsparciu dla różnych bibliotek, jest odpowiednim wyborem do budowy aplikacji backendowej obsługującej logikę systemu i interakcję pomiędzy serwerami. Te technologie zapewnią stabilność, skalowalność oraz łatwość zarządzania systemem.

2. Realizacja projektu

Całość projektu działa w dockerze. Serwer centralny uruchomiony jest na porcie 8000, a pozostałe serwery na kolejnych portach (8001...8008). Wszystkie serwery fizycznie mogą komunikować się między sobą bezpośrednio, ale w rzeczywistości komunikacja ta jest ograniczona grafem połączeń między mniejszymi serwerami. Każdy serwer może komunikować się bezpośrednio z serwerem centralnym i na odwrót.

```
version: '3.8'
services:
central:
image: php:8.2-cli
container_name: php_central
volumes:
- ./central:/var/www/html
working_dir: /var/www/html
command: php -S 0.0.0.0:8000
ports:
- "8000:8000"

php1:
image: php:8.2-cli
container_name: php_instance_1
volumes:
- ./instance:/var/www/html
working_dir: /var/www/html
command: php -S 0.0.0.0:8001
ports:
- "8001:8001"

...

php8:
image: php:8.2-cli
container_name: php_instance_8
volumes:
- ./instance:/var/www/html
working_dir: /var/www/html
command: php -S 0.0.0.0:8008
ports:
- "8008:8008"

```

Listing 1: Plik konfiguracyjny docker composer'a

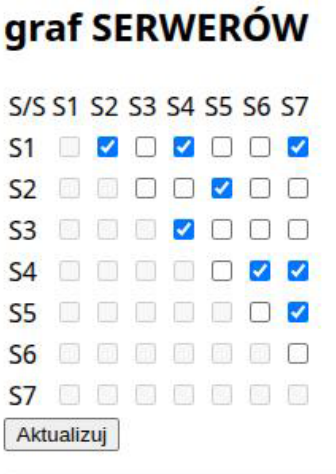
Serwer centralny wykorzystuje dwie funkcje do komunikacji: **cmd_broadcast()** pozwalająca wysyłać komunikat do wszystkich serwerów oraz **cmd_send()** wysyłająca komunikat do wskazanego serwera.

```php
function cmd_broadcast ($cmd)
{
  for ($i = 1; $i <= 7; $i++) {
    cmd_send($cmd, $i);
  }
}

function cmd_send ($cmd, $serverId)
{
  $url = "http://php{$serverId}:800{$serverId}";

  $ch = curl_init($url);
  curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
  curl_setopt($ch, CURLOPT_POST, true);
  curl_setopt($ch, CURLOPT_POSTFIELDS, http_build_query($cmd));
  curl_setopt($ch, CURLOPT_TIMEOUT_MS, 100);
  curl_setopt($ch, CURLOPT_NOSIGNAL, 1);

  $response = curl_exec($ch);

  curl_close($ch);
}
```

Listing 2: Broadcast konfiguracji grafu do serwerów

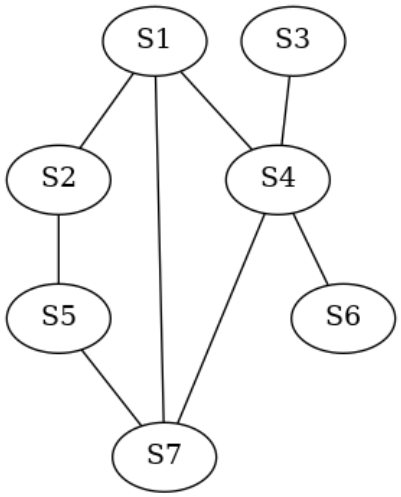
2.1 Graf połączeń między serwerami

System umożliwia pełną konfigurację połączeń między serwerami. Jest ona definiowana przez serwer centralny, który wysyła ją do mniejszych serwerów, aby zaktualizowały swoją konfigurację połączeń. Można ją również aktualizować w trakcie działania. Pomniejsze serwery bazują na jednym kodzie, który jest de facto wielokrotnym wdrożeniem tej samej mechaniki, na każdym z pomniejszych serwerów.



Rysunek 1: Konfiguracja połączeń między serwerami

Na podstawie powyższej konfiguracji możemy uzyskać poniższy schemat połączeń:



Rysunek 2: Schemat połączeń

```php
function convert_post_inputs_to_server_graph ()
{
  $graph = [];

  foreach ($_POST as $key => $val) {
    if (strpos($key, "connect_") === 0) {
      $pos = explode("_", $key);
      $pos = explode("x", $pos[1]);

      if (!isset($graph[$pos[0]])) {
        $graph[$pos[0]] = [];
      }

      $graph[$pos[0]][] = $pos[1];
    }
  }

  return (object) $graph;
}

function store_graph_configuration ($graph)
{
  file_put_contents("./graph.json", json_encode($graph));
}

function load_graph_configuration ()
{
  return json_decode(file_get_contents("./graph.json"));
}

if ($_SERVER['REQUEST_METHOD'] === 'POST' && $_POST['action'] == "graf") {
  $graph = convert_post_inputs_to_server_graph();
  store_graph_configuration($graph);
  cmd_broadcast([
    "action" => "graf",
    "graf" => $graph,
  ]);
  log_msg("Zakutalizowano strukturę grafu");
}

$graph = load_graph_configuration();
```

Listing 3: Broadcast konfiguracji grafu do serwerów

```
    {
      "1":["2", "4", "7"],
      "2":["5"],
      "3":["4"],
      "4":["6","7"],
      "5":["7"]
    }
```

Listing 4: Struktura grafu w formacie json

Graf ten rozgłaszany jest przez serwer centralny do mniejszych serwerów, serwery te odbierają go i zapisują sobie lokalne kopie w **/etc/graph.json**. Podczas ładowania struktura grafu jest rozwijana w drugą stronę tz. jeżeli serwer 3 widzi serwer 4, to w strukturze upewniamy się, że serwer 4 widzi też serwer 3.

```
function store_graph_configuration ($graph)
{
  file_put_contents("/etc/graph.json", json_encode($graph));
}

function load_graph_configuration ()
{
  $graph = (array) json_decode(file_get_contents("/etc/graph.json"));
  foreach ($graph as $key => $arr) {
    foreach ($arr as $val) {
      if (!isset($graph[$val])) {
        $graph[$val] = [];
      }
      if (!in_array($key, $graph[$val])) {
        $graph[$val][] = $key;
      }
    }
  }
  return $graph;
}

if ($_SERVER['REQUEST_METHOD'] === "POST" && $_POST['action'] === "graf") {
  store_graph_configuration($_POST['graf']);
  exit;
}

$graph = load_graph_configuration();
```

Listing 5: Przejęcie grafu na serwerach

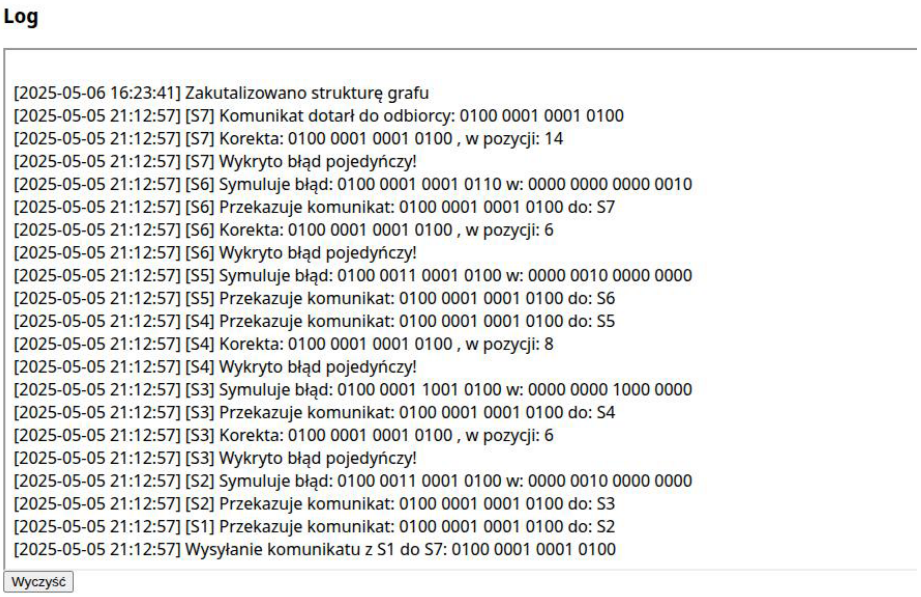
Kiedy serwery otrzymują polecenie o przesłaniu komunikatu do innego serwera, muszą one ustalić drogę, którą ten komunikat prześlą, innymi słowy, muszą ustalić, który z ich sąsiedzkich serwerów będzie mógł przekazać ten komunikat dalej. Poniżej znajduje się funkcja ustalająca taki serwer/drogę, jeżeli połączenie w grafie nie istnieje to serwer poddaje się zgłaszając serwerowi centralnemu stosowny komunikat. Algorytm nie szuka najbardziej optymalnej ścieżki, lecz pierwszej pasującej.

```
1  $visited = [];
2
3  function route_graph_traverse ($ptr, $dest)
4  {
5    global $graph;
6    global $visited;
7    $visited[] = $ptr;
8
9    foreach ($graph[$ptr] as $edge) {
10
11     if ($edge == $dest) {
12       return $edge;
13     }
14
15     if (in_array($edge, $visited)) {
16       /** już odwiedzony */
17       continue;
18     }
19
20     if (route_graph_traverse($edge, $dest) != null) {
21       return $edge;
22     }
23   }
24
25   return null;
26 }
27
28 function route_graph ($dest)
29 {
30   global $graph;
31   global $serverId;
32   return route_graph_traverse($serverId, $dest);
33 }
```

Listing 6: Trasowanie grafu

2.2 Logowanie operacji

Istotną częścią projektu jest logowanie wszystkich operacji wykonywanych zarówno na serwerze centralnym jak i pomniejszych serwerach.



Rysunek 3: Log widoczny dla użytkownika

Log przechowywany jest w pliku **log.txt**, na serwerze centralnym, jego wyświetlanie i automatyczne odświeżanie jest zrobione na zasadzie strony internetowej w **iframe**.

```
1  <html>
2    <head>
3      <meta http-equiv="refresh" content="3">
4    </head>
5    <body>
6    <?php
7      $lines = explode("\n", file_get_contents("./log.txt"));
8
9      for ($i = count($lines) - 1; $i >= 0; $i--) {
10       echo $lines[$i] . "<br/>";
11     }
12   ?>
13   </body>
14 </html>
```

Listing 7: Wyświetlenie log'u

```php
if ($_SERVER['REQUEST_URI'] === "/getlog") {
  include "./log.php";
  exit;
}

function msg_pp ($msg)
{
  return substr($msg, 0, 4) . " " . substr($msg, 4, 4) . " " . substr($msg, 8, 4)
  ↪  . " " . substr($msg, 12, 4);
}

function log_msg ($msg)
{
  file_put_contents("./log.txt", "[" . date('Y-m-d G:i:s') . "] " . $msg . "\n",
  ↪  FILE_APPEND);
}

if ($_SERVER['REQUEST_METHOD'] === "POST" && $_POST['action'] == "clear_log") {
  file_put_contents("./log.txt", "");
}

if ($_SERVER['REQUEST_METHOD'] === "POST" && $_POST['action'] == "log") {
  log_msg($_POST['msg']);
  exit;
}
```

Listing 8: Funkcje obsługujące log na serwerze centralnym

```php
function log_central ($msg)
{
  global $serverId;
  cmd_send([
    "action" => "log",
    "msg" => "[S{$serverId}] " . $msg
  ], 0);
}
```

Listing 9: Funkcja umożliwiająca wysłanie log'a z pomniejszego serwera

2.3 Symulacja błędu

Widok symulacji błędu jest załadowaną w iframe stroną serwera mniejszego, przyciski S1 do S7 zmieniają podgląd na konkretny serwer. Ustawienie konkretnego bitu sprawi, że w trakcie przekazywania komunikatu dalej przez ten serwer bit w ustawionej pozycji zostanie odwrócony, co pozwala w kontrolowany sposób symulować błędy.

Rysunek 4: Log widoczny dla użytkownika

```php
function toggler_sim ($msg, $bits)
{
  for ($i = 0; $i < 16; $i++) {
    if ($bits[$i] == '1') {
      $msg[$i] = ($msg[$i] == '1') ? '0' : '1';
    }
  }

  return $msg;
}

function store_toggler ($bits)
{
  file_put_contents("/etc/toggler.json", json_encode(["bits" => $bits]));
}

function load_toggler ()
{
  return json_decode(file_get_contents("/etc/toggler.json"))->bits ??
  ↪  "0000000000000000";
}

if ($_SERVER['REQUEST_METHOD'] === "POST" && $_POST['action'] === "set") {
  $bits = "";

  for ($i = 0; $i < 16; $i++) {
```

```
26      $bits .= isset($_POST["bit_{$i}"]) ? "1" : "0";
27    }
28
29    store_toggler($bits);
30    log_central("S{$serverId} przestawia bity: " . msg_pp($bits));
31  }
32
33  $bits = load_toggler();
34
35  ?>
36
37  <form action="/" method="POST">
38    <input type="hidden" name="action" value="set" />
39
40    (S<?= $serverId ?>) Zamień:
41    <? for ($i = 0; $i < 16; $i++): ?>
42      <input type="checkbox" name="bit_<?= $i ?>" <?= $bits[$i] == "1" ? "checked"
      ↪  : "" ?> />
43
44      <? if ($i % 4 == 3): ?>
45        <span style="margin-left: 20px;"></span>
46      <? endif ?>
47    <? endfor ?>
48
49    <button>Ustaw</button>
50  </form>
```

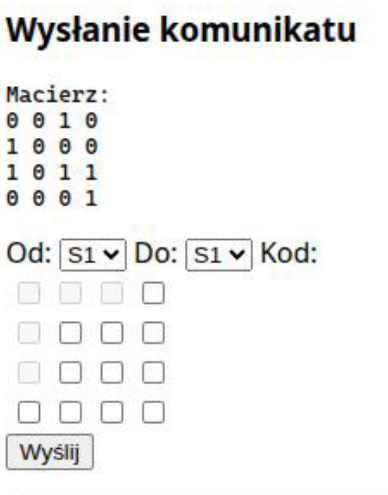
Listing 10: Obsługa symulacji błędów

2.4 Wysłanie komunikatu

Opcja wysłania komunikatu pozwala nam ustawić 11 bitów danych (pozostałe 5 to redundancja - bity parzystości). Pola do ustawiania bitów rozmieszczone są w wyglądzie macierzowym 4x4, z zaszarzonymi bitami parzystości - ich nie można ustawić. Ponadto możemy wybrać z którego do którego serwera komunikat ma zostać wysłany.

Rysunek 5: Formularz do wysłania komunikatu

Po wysłaniu komunikatu wyświetla się podgląd macierzowy z obliczonymi bitami parzystości.



Rysunek 6: Formularz do wysłania komunikatu

```php

if ($_SERVER['REQUEST_METHOD'] === "POST" && $_POST['action'] === "komunikat") {

  $hamming = [
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
  ];

  for ($i = 0; $i < 4; $i++) {
    for ($j = 0; $j < 4; $j++) {
      $hamming[$i][$j] = isset($_POST["bit_{$i}x{$j}"]) ? 1 : 0;
    }
  }

  for ($i = 0; $i < 4; $i++) {
    $hamming[0][1] += $hamming[$i][1] + $hamming[$i][3];
    $hamming[0][2] += $hamming[$i][2] + $hamming[$i][3];

    $hamming[1][0] += $hamming[1][$i] + $hamming[3][$i];
    $hamming[2][0] += $hamming[2][$i] + $hamming[3][$i];
  }

  $hamming[0][1] %= 2;
  $hamming[0][2] %= 2;
```

```
27
28    $hamming[1][0] %= 2;
29    $hamming[2][0] %= 2;
30
31    for ($i = 0; $i < 4; $i++) {
32      for ($j = 0; $j < 4; $j++) {
33        $hamming[0][0] += $hamming[$i][$j];
34      }
35    }
36
37    $hamming[0][0] %= 2;
```

Listing 11: Obliczanie bitów parzystości

Tak przygotowany komunikat wysyłany jest do wybranego serwera, łącznie z informacją do którego serwera docelowo ma trafić.

```
1   $msg = "";
2   for ($i = 0; $i < 4; $i++) {
3     for ($j = 0; $j < 4; $j++) {
4       $msg .= $hamming[$i][$j] ? "1" : "0";
5     }
6   }
7
8   log_msg("Wysyłanie komunikatu z S{$_POST['from']} do S{$_POST['to']}: " .
    ↪  msg_pp($msg));
9
10  cmd_send([
11    "action" => "komunikat",
12    "to" => (int) $_POST['to'],
13    "msg" => $msg
14  ], $_POST['from']);
```

Listing 12: Wysłanie komunikatu do wybranego serwera

Serwer odbiera żądanie, weryfikuje poprawność komunikatu, jeżeli wystąpił błąd podwójny, to nie ma możliwości korekcji, na serwer centralny wysyłana jest stosowna informacja. Jeżeli wystąpił błąd pojedynczy to jest on korygowany. Dalej jeżeli serwer jest odbiorcą to przesyłanie zakańcza się, jeżeli nie to wyznaczana jest trasa, dokonywana jest ewentualna symulacja błędu, komunikat przekazywany jest do kolejnego węzła w grafie.

```php
if ($_SERVER["REQUEST_METHOD"] === "POST" && $_POST['action'] === "komunikat") {

  $msg = $_POST['msg'];

  /** Sprawdzeneie czy komuniakt posiada błąd */
  $msg = hamming_check($msg);
  if ($msg === false) {
    exit;
  }

  if ($_POST['to'] == $serverId) {
    /** Jesteśmy odbiorcą! */
    log_central("Komunikat dotarł do odbiorcy: " . msg_pp($msg));
    exit;
  }

  $route = route_graph($_POST['to']);

  if ($route == null) {
    /** Droga nie istnieje */
    log_central("Droga do S{$_POST['to']} przez S{$serverId} nie istnieje!");
    exit;
  }

  log_central("Przekazuje komunikat: " . msg_pp($msg) . " do: S{$route}");

  /** Symulacja błędu na podstawie ustawień */
  $bits = load_toggler();
  if (strpos($bits, "1") !== false) {
    $msg = toggler_sim($msg, $bits);
    log_central("Symuluje błąd: " . msg_pp($msg) . " w: " . msg_pp($bits));
  }

  /** Przekazujemy komunikat do kolejnego serwera */
  cmd_send([
    "action" => "komunikat",
    "msg" => $msg,
    "to" => $_POST['to']
  ], (int) $route);

  exit;
}
```

Listing 13: Obsługa komunikatu przez serwery

Funkcja **hamming_check()** sprawdzająca, korygująca komunikat i zwracająca prawidłowy komunikat lub NULL w przypadku błędu podwójnego.

```
function hamming_check ($msg)
{
  /** OMP */
  $omp = 0;
  for ($i = 0; $i < 16; $i++) {
    $omp += $msg[$i] == '1' ? 1 : 0;
  }
  $omp %= 2;

  $mx = [
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0]
  ];

  for ($i = 0; $i < 4; $i++) {
    for ($j = 0; $j < 4; $j++) {
      $mx[$i][$j] = ($msg[$i * 4 + $j] == '1') ? 1 : 0;
    }
  }

  /**
  * Liczenie bitów parzystości, i sprawdzenie z orginalnymi
  * oraz z dodatkowym w celu określenia ilości błędów, i
  * ewentualnej możliwości korekty
  */

  $op1 = $mx[0][1];
  $op2 = $mx[0][2];
  $op4 = $mx[1][0];
  $op8 = $mx[2][0];

  $mx[0][1] = 0;
  $mx[0][2] = 0;
  $mx[1][0] = 0;
  $mx[2][0] = 0;

  $p1 = 0;
  $p2 = 0;
  $p4 = 0;
  $p8 = 0;
```

```
43
44    for ($i = 0; $i < 4; $i++) {
45      $p1 += $mx[$i][1] + $mx[$i][3];
46      $p2 += $mx[$i][2] + $mx[$i][3];
47
48      $p4 += $mx[1][$i] + $mx[3][$i];
49      $p8 += $mx[2][$i] + $mx[3][$i];
50    }
51
52    $p1 %= 2;
53    $p2 %= 2;
54    $p4 %= 2;
55    $p8 %= 2;
56
57    $areParityBitsCorrect = ($p1 == $op1 && $p2 == $op2 && $p4 == $op4 && $p8 ==
      ↪  $op8);
58    $isOveralParityCorrect = $omp == 0;
59
60    /** Nie da się poprawić, dwa błędy */
61    if (!$areParityBitsCorrect && $isOveralParityCorrect) {
62      log_central("Wykryto błąd podwójny, poddaje się!");
63      return false;
64    }
65
66    /** Nie ma co porpawiać, parzystość się zgadza */
67    if ($areParityBitsCorrect && $isOveralParityCorrect) {
68      return $msg;
69    }
70
71    log_central("Wykryto błąd pojedyńczy!");
72
73    /**
74    * Jak weźmiemy {p8, p4, p2, p1}, to jest to
75    * dosłownie index bitu który był flipnięty
76    */
77
78    $pos = "0000";
79
80    if ($p1 != $op1) {
81      $pos[3] = '1';
82    }
83
84    if ($p2 != $op2) {
85      $pos[2] = '1';
86    }
```

```php

  if ($p4 != $op4) {
    $pos[1] = '1';
  }

  if ($p8 != $op8) {
    $pos[0] = '1';
  }

  $pos = bindec($pos);
  $msg[$pos] = ($msg[$pos] == '1') ? '0' : '1';

  log_central("Korekta: " . msg_pp($msg) . " , w pozycji: " . $pos);

  return $msg;
}

```

Listing 14: Funkcja implementująca sprawdzanie i korekcję Hamminga

Rysunek 7: Formularz do wysłania komunikatu

```
Wysyłanie komunikatu z S7 do S1: 0000 0000 0000 0000
[S7] Przekazuje komunikat: 0000 0000 0000 0000 do: S6
[S7] Symuluje błąd: 0001 0000 0000 0000 w: 0001 0000 0000 0000
[S6] Wykryto błąd pojedyńczy!
[S6] Korekta: 0000 0000 0000 0000 , w pozycji: 3
[S6] Przekazuje komunikat: 0000 0000 0000 0000 do: S5
[S6] Symuluje błąd: 0000 0000 0000 0010 w: 0000 0000 0000 0010
[S5] Wykryto błąd pojedyńczy!
[S5] Korekta: 0000 0000 0000 0000 , w pozycji: 14
[S5] Przekazuje komunikat: 0000 0000 0000 0000 do: S4
[S5] Symuluje błąd: 0000 0010 0000 0000 w: 0000 0010 0000 0000
[S4] Wykryto błąd pojedyńczy!
[S4] Korekta: 0000 0000 0000 0000 , w pozycji: 6
[S4] Przekazuje komunikat: 0000 0000 0000 0000 do: S3
```

```
[S3] Przekazuje komunikat: 0000 0000 0000 0000 do: S2
[S3] Symuluje błąd: 0000 0000 1000 0000 w: 0000 0000 1000 0000
[S2] Wykryto błąd pojedyńczy!
[S2] Korekta: 0000 0000 0000 0000 , w pozycji: 8
[S2] Przekazuje komunikat: 0000 0000 0000 0000 do: S1
[S2] Symuluje błąd: 0000 0010 0000 0000 w: 0000 0010 0000 0000
[S1] Wykryto błąd pojedyńczy!
[S1] Korekta: 0000 0000 0000 0000 , w pozycji: 6
[S1] Komunikat dotarł do odbiorcy: 0000 0000 0000 0000
```

Listing 15: Przebieg komunikacji między S7 i S1 z pustym komunikatem

```
Wysyłanie komunikatu z S1 do S7: 0100 0001 0001 0100
[S1] Przekazuje komunikat: 0100 0001 0001 0100 do: S2
[S1] Symuluje błąd: 0100 0101 0001 0100 w: 0000 0100 0000 0000
[S2] Wykryto błąd pojedyńczy!
[S2] Korekta: 0100 0001 0001 0100 , w pozycji: 5
[S2] Przekazuje komunikat: 0100 0001 0001 0100 do: S3
[S2] Symuluje błąd: 0100 0011 0001 0100 w: 0000 0010 0000 0000
[S3] Wykryto błąd pojedyńczy!
[S3] Korekta: 0100 0001 0001 0100 , w pozycji: 6
[S3] Przekazuje komunikat: 0100 0001 0001 0100 do: S4
[S3] Symuluje błąd: 0100 0001 1001 0100 w: 0000 0000 1000 0000
[S4] Wykryto błąd pojedyńczy!
[S4] Korekta: 0100 0001 0001 0100 , w pozycji: 8
[S4] Przekazuje komunikat: 0100 0001 0001 0100 do: S5
[S5] Przekazuje komunikat: 0100 0001 0001 0100 do: S6
[S5] Symuluje błąd: 0100 0011 0001 0100 w: 0000 0010 0000 0000
[S6] Wykryto błąd pojedyńczy!
[S6] Korekta: 0100 0001 0001 0100 , w pozycji: 6
[S6] Przekazuje komunikat: 0100 0001 0001 0100 do: S7
[S6] Symuluje błąd: 0100 0001 0001 0110 w: 0000 0000 0000 0010
[S7] Wykryto błąd pojedyńczy!
[S7] Korekta: 0100 0001 0001 0100 , w pozycji: 14
[S7] Komunikat dotarł do odbiorcy: 0100 0001 0001 0100
```

Listing 16: Przebieg komunikacji między S1 i S7 z testowym komunikatem

```
[S4] S4 przestawia bity: 0000 1100 0000 0000
Wysyłanie komunikatu z S1 do S7: 1101 1000 1110 0100
[S1] Przekazuje komunikat: 1101 1000 1110 0100 do: S2
[S1] Symuluje błąd: 1101 1100 1110 0100 w: 0000 0100 0000 0000
[S2] Wykryto błąd pojedyńczy!
[S2] Korekta: 1101 1000 1110 0100 , w pozycji: 5
[S2] Przekazuje komunikat: 1101 1000 1110 0100 do: S3
[S2] Symuluje błąd: 1101 1010 1110 0100 w: 0000 0010 0000 0000
[S3] Wykryto błąd pojedyńczy!
[S3] Korekta: 1101 1000 1110 0100 , w pozycji: 6
[S3] Przekazuje komunikat: 1101 1000 1110 0100 do: S4
[S3] Symuluje błąd: 1101 1000 0110 0100 w: 0000 0000 1000 0000
[S4] Wykryto błąd pojedyńczy!
[S4] Korekta: 1101 1000 1110 0100 , w pozycji: 8
[S4] Przekazuje komunikat: 1101 1000 1110 0100 do: S5
[S4] Symuluje błąd: 1101 0100 1110 0100 w: 0000 1100 0000 0000
[S5] Wykryto błąd podwójny, poddaje się!
```

Listing 17: Przebieg komunikacji między S1 i S7 z podwójnym błędem

3. Podsumowanie

Celem projektu było stworzenie systemu transmisji danych o podwyższonej niezawodności, złożonego z ośmiu serwerów połączonych w graf, z centralnym serwerem nadzorującym. Kluczowym elementem projektu było zastosowanie 16-bitowych bloków danych zabezpieczonych kodem Hamminga z dodatkowym bitem parzystości (SECDED), umożliwiającym korekcję pojedynczych błędów i detekcję podwójnych.

Do realizacji systemu wykorzystano kontenery Docker, co zapewniło izolację środowisk serwerów i ułatwiło zarządzanie nimi. Serwer centralny oraz serwery podrzędne komunikują się poprzez interfejs HTTP, a konfiguracja połączeń między nimi jest dowolna i dynamicznie definiowana. Logika trasowania wiadomości oparta jest na rekursywnym przeszukiwaniu grafu, bez optymalizacji ścieżek.

Zaimplementowano również mechanizm symulacji błędów, pozwalający użytkownikowi ręcznie wprowadzać błędy w przesyłanych bitach na poziomie każdego serwera. Dzięki temu możliwe było testowanie działania kodu Hamminga w różnych scenariuszach: bez błędów, z pojedynczymi błędami (automatycznie korygowanymi) oraz z podwójnymi błędami (wykrywanymi, ale niekorygowanymi).

Projekt zawierał też panel do tworzenia i wysyłania komunikatów oraz system logowania operacji, widoczny dla użytkownika w czasie rzeczywistym. Testy wykazały skuteczność systemu w korekcji błędów pojedynczych oraz poprawność obsługi błędów podwójnych, co potwierdziło odporność systemu na zakłócenia transmisji.