Desing Patterns – Dokumentacja

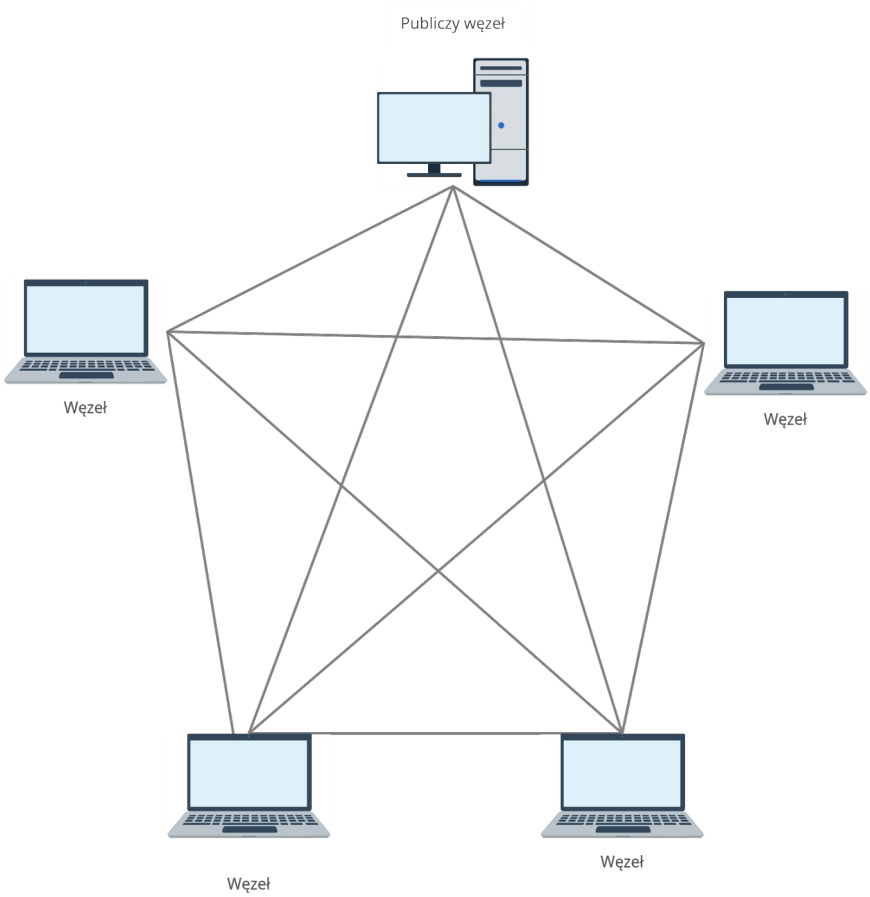
Kacper Piastowicz, Szymon Sumara

Opis projektu

Temat: Projekt oraz implementacja systemu przetwarzania danych w architekturze peer to peer. Wyniki obliczeń powinny być dostępne on-line podczas przetwarzania. Problem obliczeniowy może zostać dowolnie wybrany przez grupę, jednak powinien wymagać min 1 godziny obliczeń na 5 komputerach klasy PC. Technologia Node.js.

Założenia: Ograniczyliśmy dostępny alfabet do: dużych i małych liter alfabetu łacińskiego oraz cyfr.

Komunikacja pomiędzy węzłami

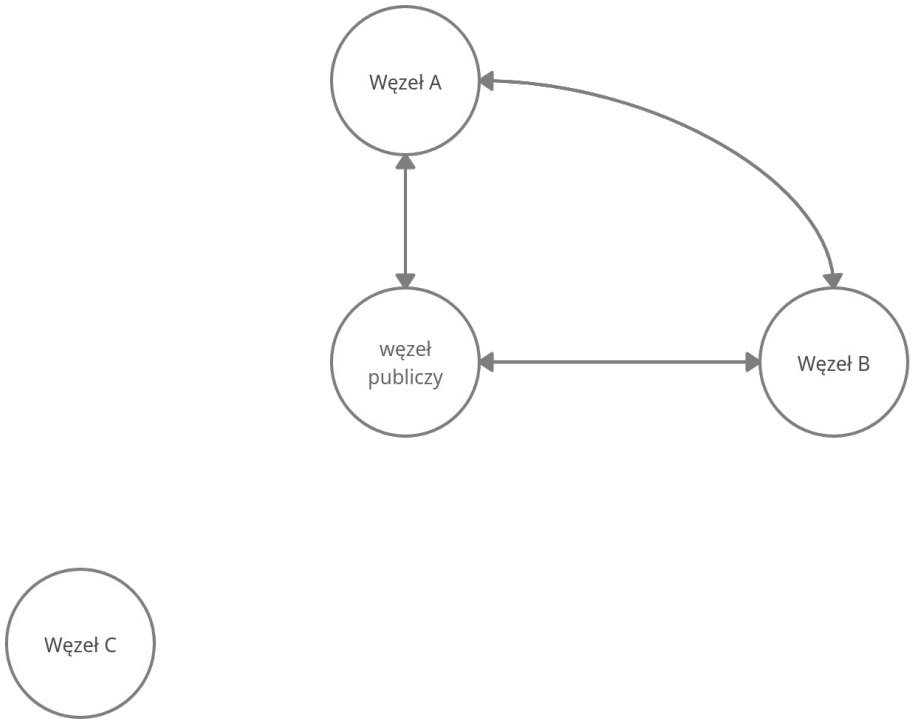


Schemat połączenia pomiędzy węzłami

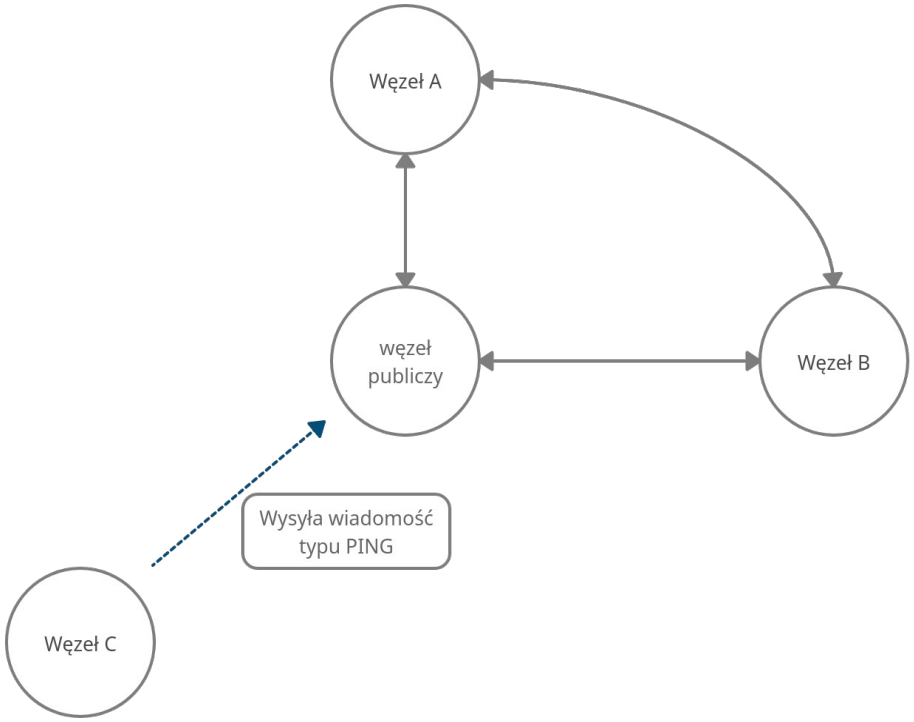
W naszym rozwiązaniu każdy węzeł może wysyłać wiadomości do wszystkich pozostałych. Całą komunikacje oparliśmy o protokół UDP (by móc zastosować metodę UDP Hole Puchning).

Podłączenie węzła do sieci

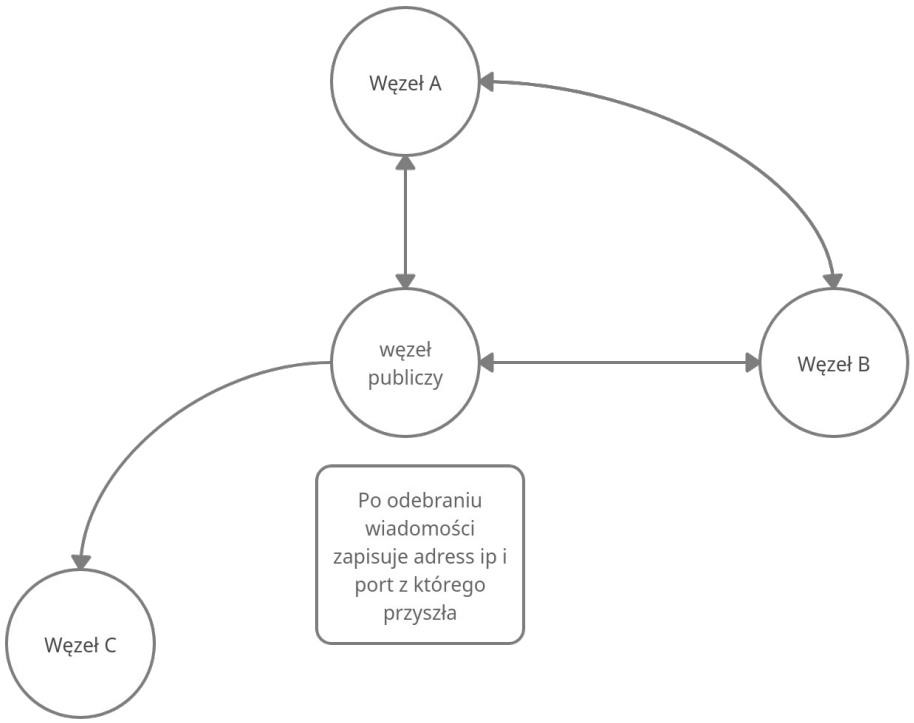
Węzeł, który chce podłączyć się do sieci wysyła wiadomość typu PING do publicznego węzła. Publiczny węzeł zapisuje w tablicy połączeń jego publiczny adres IP oraz jego port skojarzony z tym adresem. Następnie, gdy publiczny węzeł będzie rozsyłać rutynową wiadomość typu PING rozpropaguje to nowe połączenie po całej sieci, przez co zarówno nowo dołączony host jak i wcześniej podłączone mogą komunikować się ze sobą bezpośrednio.



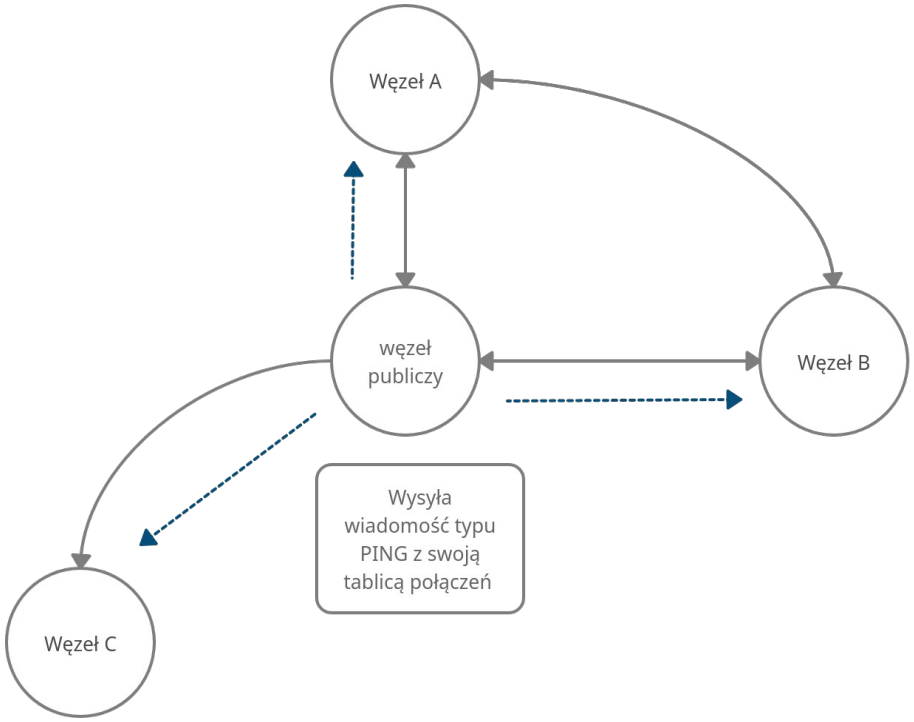
Węzły A, B i węzeł publiczny są w całości połączone między sobą.



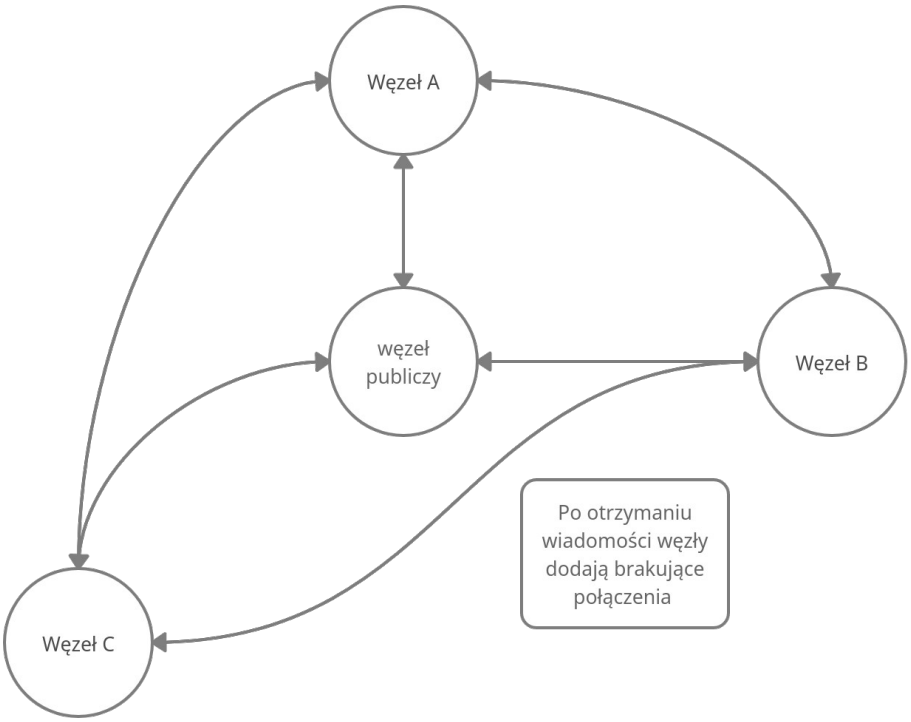
Węzeł C by się połączyć wysyła wiadomość typu PING do węzła publicznego



Węzeł publiczny zapisuje skąd przyszła wiadomość w swojej tablicy połączeń.



Gdy będzie wykonywać rutynową wiadomość PING roześle swoją tablice połączeń.

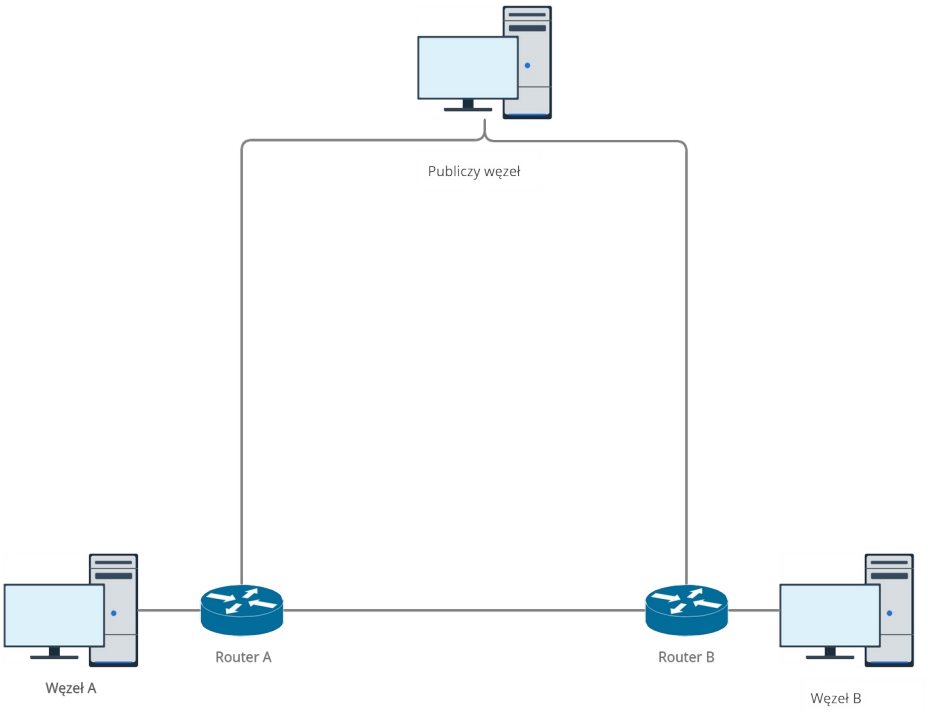


Gdy węzły otrzymają tablice połączeń węzła publicznego, to uzupełniają swoja tablicę o brakujące węzły. W tym momencie węzły mogą się komunikować między sobą bezpośrednio.

Awaria jednego z węzłów.

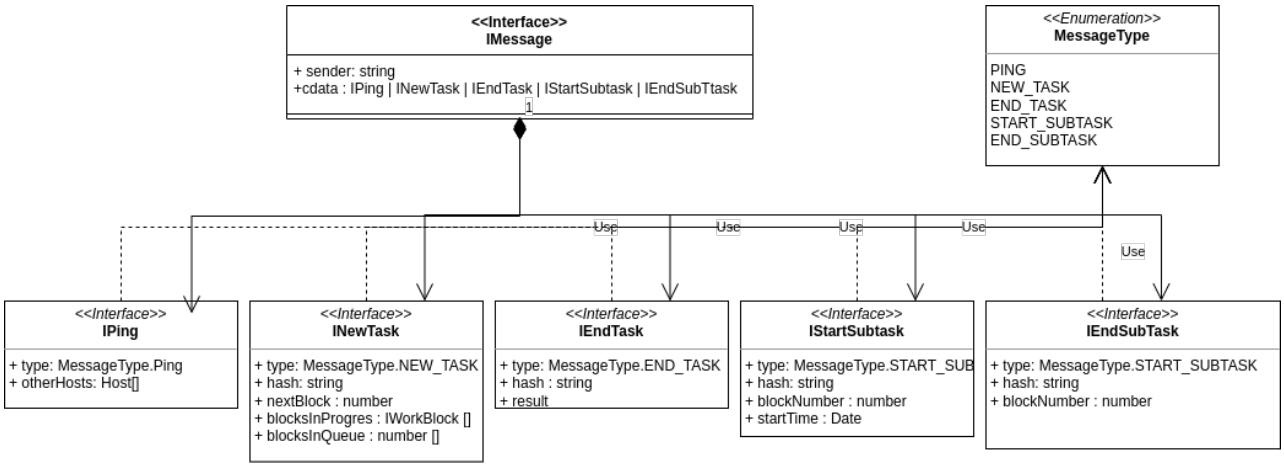
Każdy z węzłów sprawdza swoje połączenia indywidualnie. W przypadku awarii jednego z węzłów, inne węzły z sieci przestaną od niego otrzymywać wiadomości PING. Jeśli węzły nie będą otrzymywać od kogoś wiadomości PING przez 5 sekund, to zostanie on usunięty z tablic połączeń i jeśli będzie się chciał ponownie dołączyć do sieci będzie musiał zrobić to w taki sam sposób jak zupełnie nowy węzeł.

Komunikacja między różnymi sieciami lokalnymi.



Węzeł A i B nie może się bezpośrednio połączyć więc zastosowaliśmy technikę UDP Hole Puchning polegającą na tym, że na samym początku hosty komunikują się z węzłem, który jest dostępny publicznie, a ten zapisuje adres i port, z którego przyszła wiadomość (dla węzła A jest to adres routera A i skojarzony z nim port, a dla węzła B adres routera B i skojarzony z nim port). W ten sposób otrzymujemy drogę dojścia do węzłów i możemy ją rozpropagować na pozostałe węzły.

Przesyłane wiadomości pomiędzy węzłami.



Każda z przesyłanych wiadomości będzie zawierać id nadawcy oraz pole data które będzie miało jedną z wyszczególnionych na powyższym schemacie postać.

Opisy przesyłanych wiadomości:

PING: co określony czas węzły wysyłają ping do pozostałych węzłów by poinformować, że wciąż są podłączone. Jeśli przez określony czas węzeł nie wysyłał ping 'a to zostaje on usunięty z tablicy połączeń węzła

NEW_JOB: Broadcast o stworzeniu nowej pracy – poszukiwanie ciągu znaków zwracający dany hasz. Jest wysyłana również, gdy wykryto podłączenie nowego hosta i trzeba go poinformować a aktualnie wykonywanych zadaniach.

END_JOB: broadcast o zakończeniu wykonywania pracy – znaleziony został ciąg znaków odpowiadający haszowi

START_TASK: broadcast o rozpoczęciu nowego zadania – przeszukiwanie bloku zdań i sprawdzanie ich haszy

FINISH_TASK: broadcast o zakończeniu przeszukiwania bloku

Przebieg zadania

Kroki wykonywania zadania:

1. Węzeł rozsyła zadanie do wszystkich pozostałych węzłów
2. Węzły rezerwują porcje zadania przez broadcast
3. Węzły wykonują swoją porcje
4. Po skończeniu liczenia węzeł powiadamia pozostałe węzły, że skończył i rezerwuje kolejny blok
5. Gdy znajdziemy właściwy hasz powiadamiamy o tym pozostałe węzły

W dokumentacji używamy terminu Job (zlecenia) dla całego zlecenia złamania hasza, a terminu Task (pracy) pomniejszej jego części.

Podział zadania

Każde zadanie składa się z przedrostka (który można traktować jak identyfikator zadania) i z części, do obliczenia która ma stały rozmiar.

Wyjątkiem jest pierwsze zadanie pozbawione prefiks. Polega ono na sprawdzeniu wszystkich wyrazów krótszych lub równych ilości znaków obliczanej części. Jako że uwzględniane są napisy krótsze z natury ten blok będzie większy niż pozostałe.

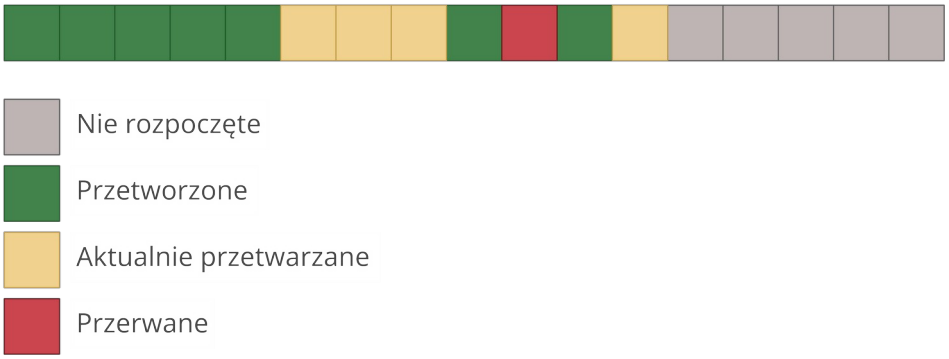
Przykład

Dla przykładu założę, że nasz alfabet jest równy {a,b,c}, a długość stałej części zadania jest równa 2. Próbujemy złamać hasz, który został wygenerowany z ciągu ‘accb’. Nasze bloki będą wyglądać następująco.

1. bez prefixu. (sprawdzimy ciągi “”,”a”,”b”,…,”cc”)
2. Z prefixem a . (sprawdzimy ciągi “aaa”,”aab”,”aac”,…,”acc”)
3. Z prefixem b. (sprawdzimy ciągi “aaa”,”aab”,”aac”,…,”acc”)
4. .
5. .
6. Z prefixem ab. (sprawdzimy ciągi “abaa”,”abab”,”abac”,…,”abcc”)
7. Z prefixem ac. (sprawdzimy ciągi “acaa”,”acab”,”acac”,…,”accb”)

Jak widać rozwiązanie zostanie znalezione w bloku siódmym.

Sposób przechowywania stanu zadania:



W każdym zleceniu przechowywane są informacje o aktualnie przetwarzanych zadaniach, zadaniach przerwanych oraz numer następnego bloku do wykonania (zadanie o najmniejszym numerze nie rozpoczętym przez żaden węzeł). Zadania przetworzone nie są przechowywane.

Gdy chcemy rozpocząć przetwarzanie nowego bloku to w pierwszej kolejności upewniamy się, że tablica z przerwanymi zadaniami jest pusta.

Po otrzymaniu wiadomości, że inny węzeł rozpoczął wykonywanie jakiegoś zadania, zadanie jest umieszczane w tablicy z aktualnie wykonywanymi zadaniami. Sprawdzamy czy numer tego zadania jest większy lub równy numerowi następnego zadania i w razie potrzeby zwiększamy ten numer.

Jeśli otrzymamy informacje, że jeden z węzłów nie odpowiada to sprawdzamy czy nie wykonywał jakiegoś zadania (sprawdzając tablice aktualnie wykonywanych zadań). Jeśli tak to umieszczamy zadania przez niego wykonywane w tablicy zadań do wykonania.

Utrzymywanie spójności sieci.

Zadanie będzie przechowywać swój czas rozpoczęcia. W monecie, gdy dwa węzły zgłoszą chęć wykonania zadania, porównują czas rozpoczęcia tych zadań i węzeł z późniejszym czasem odstąpi. Pozostałe węzły też w momencie otrzymania ogłoszenia sprawdzają tablice aktualnie wykonywanych zadań. Jeśli przysłane zadanie zostało utworzone wcześniej to aktualizują tablice zadań, a jeśli później to odrzucają przysłane zadanie. Musimy również aktualizować wskaźnik na następne zadanie, w taki sposób, że jeśli dostaniemy zadanie z późniejszym prefiksem to zwiększamy to przypisujemy naszemu przedrostkowi zwiększony o 1 krok przysłany przedrostek.

Diagramy sekwencyjne możliwych zdarzeń.

Diagram przedstawiający w jaki sposób będzie odbywać się dodanie nowego zlecenia.

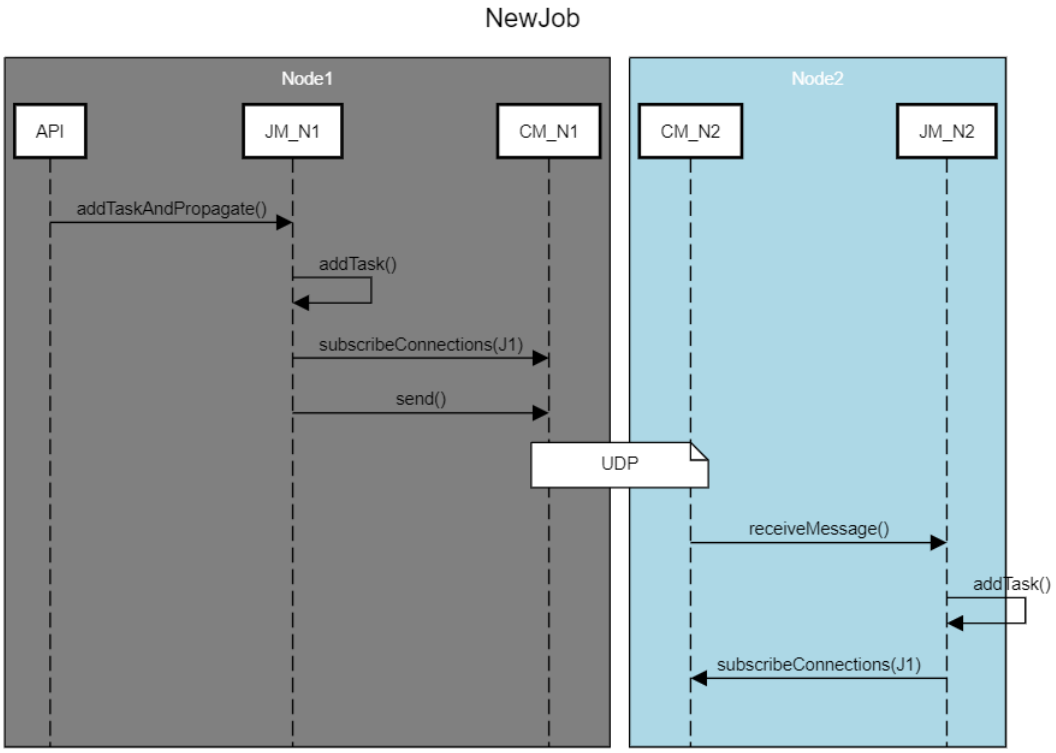


Diagram przedstawiający w jaki sposób będzie komunikowane zakończenie danego zlecenia.

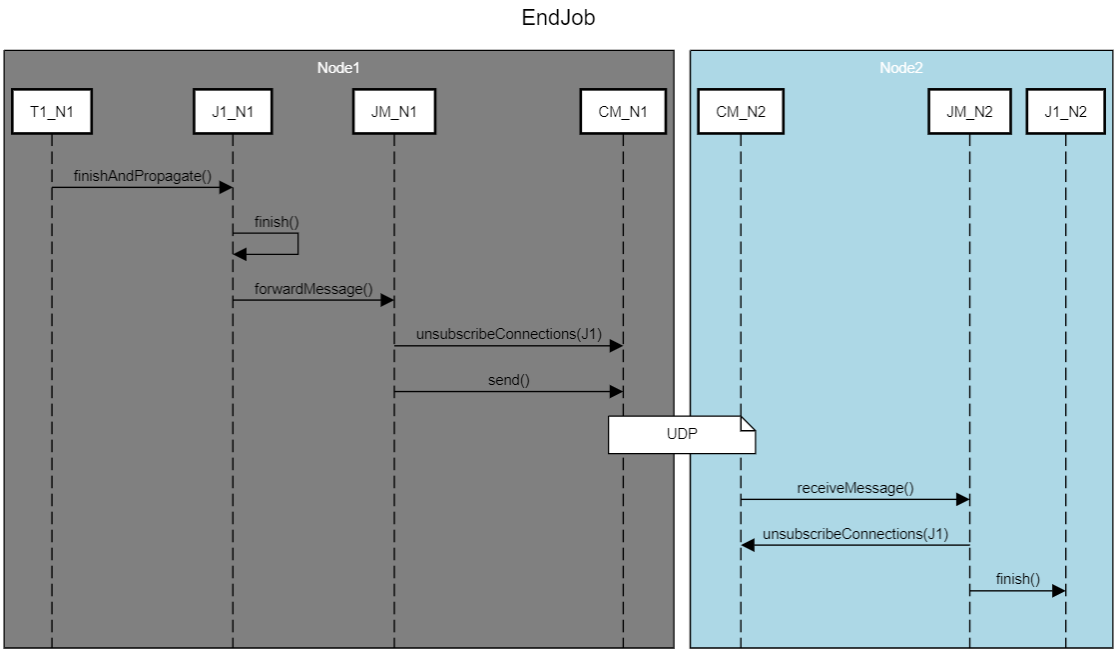
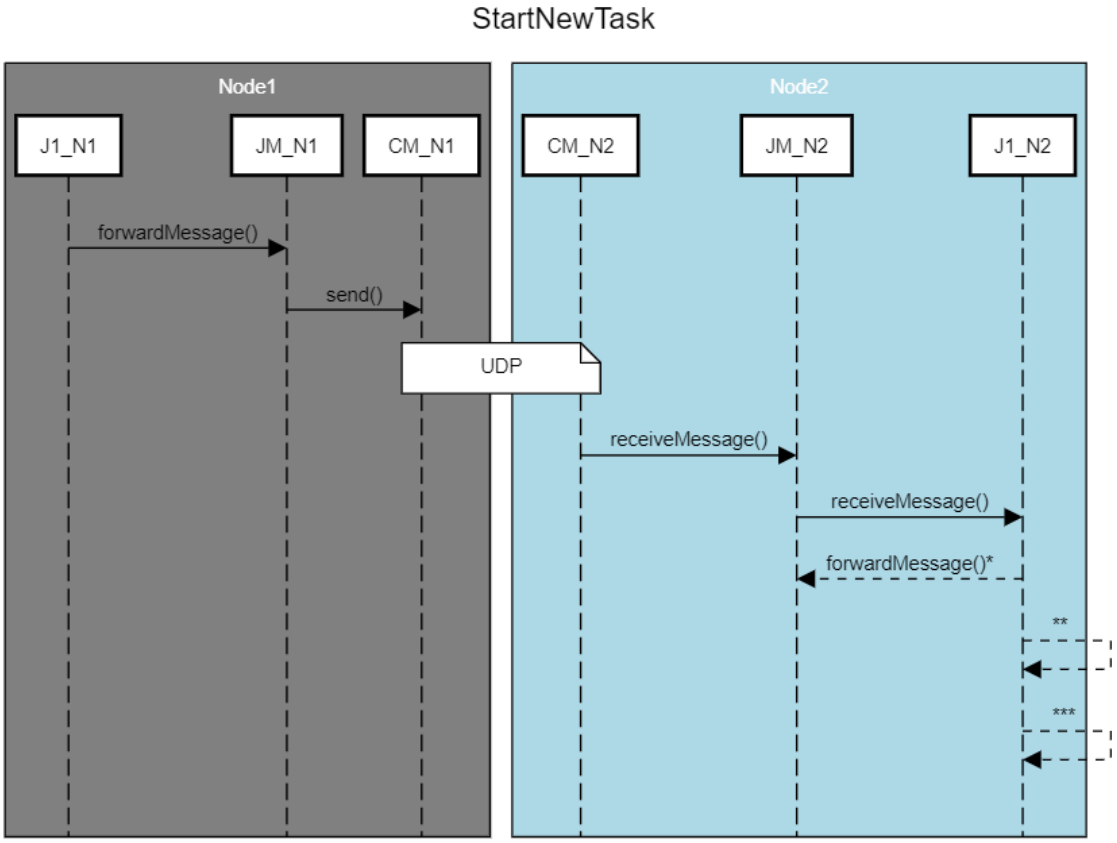


Diagram przedstawiający w jaki sposób będzie przesyłana wiadomości o rozpoczęciu przetwarzania nowego bloku



- *- przypadek, w którym node2 wykonuje już to zadanie i ma wcześniejszy czas rozpoczęcia niż node1. Ponownie rozesłany jest sygnał StartNewTask z oryginalną datą rozpoczęcia zadania przez node2.
- **- przypadek, w którym node2 wykonuje już to zadanie i ma późniejszy czas rozpoczęcia niż node1. Node2 pozbywa się obecnie wykonywanego zadania oraz dodaje zadanie ogłoszone przez node1 do tablicy zadań obecnie przetwarzanych.
- ***- przypadek, w którym node2 nie wykonuje rozgłaszanego zadania. Jeśli nie posiada je w tablicy zadań aktualnie przetwarzanych, to dodaje je. W przeciwnym wypadku zachowywane ogłoszenie, które ma wcześniejszy czas rozpoczęcia.

Diagram przedstawiający jak będzie wyglądać komunikacja po odnaleznieniu rozwiązania zlecenia.

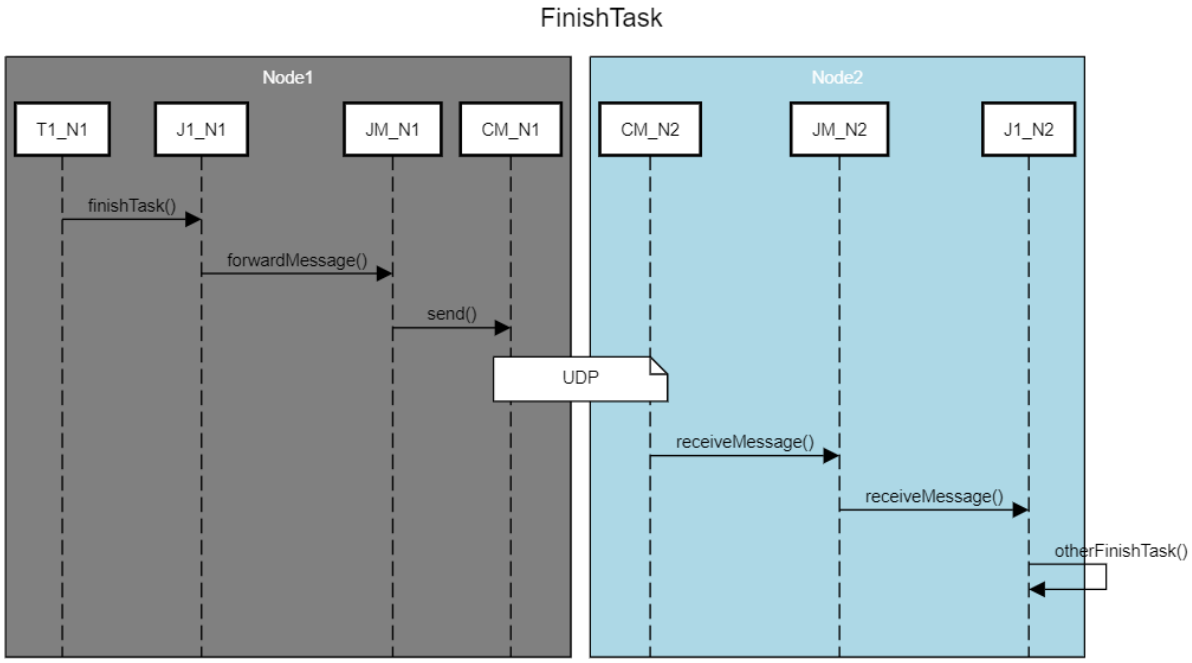
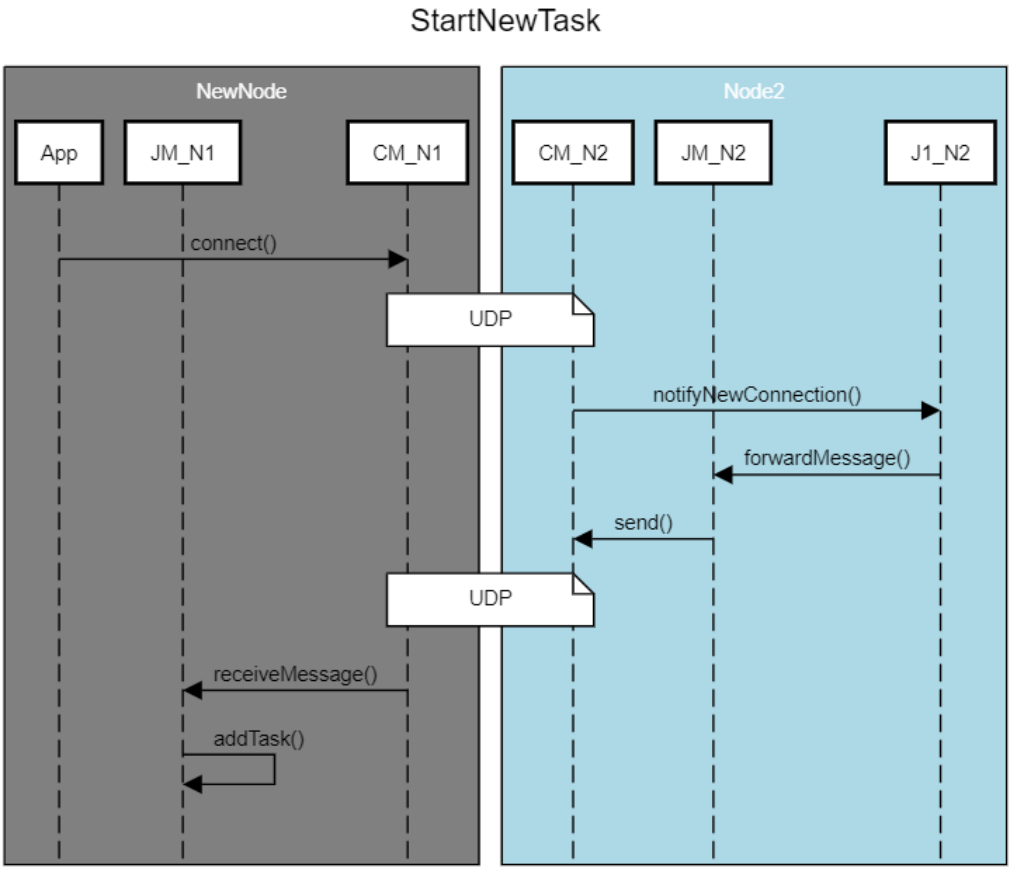
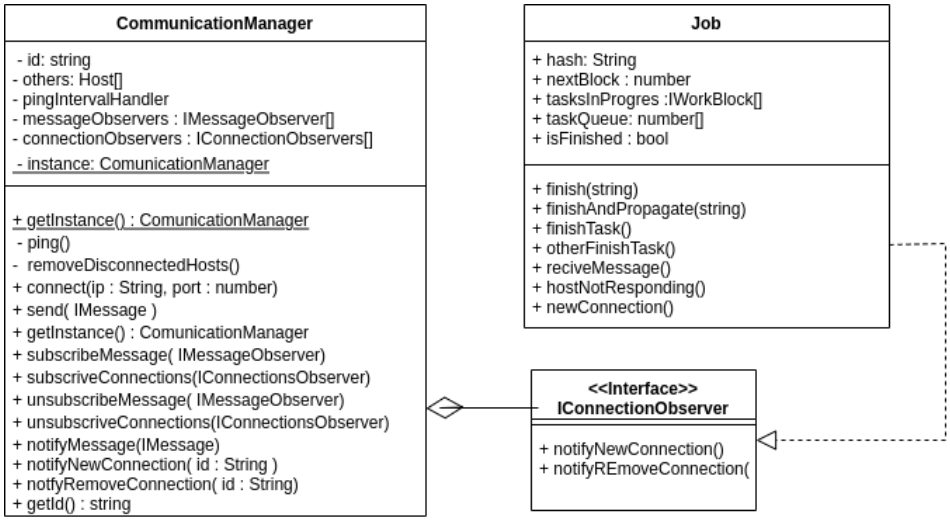


Diagram przedstawiający sytuacje, gdy do sieci, w której wykonywane jest już zlecenie dołączy się nowy węzeł.

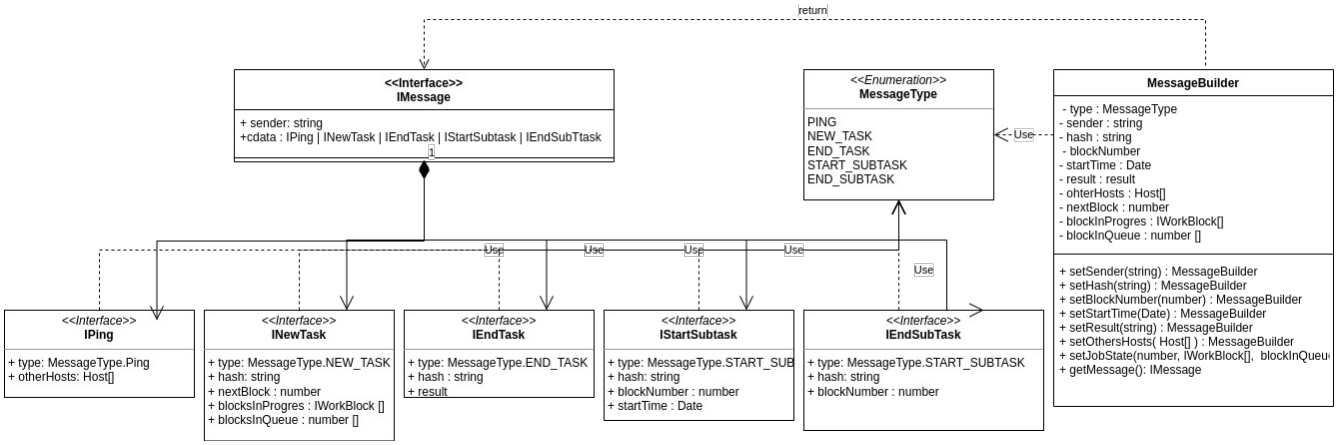


Zastosowane wzorce projektowe

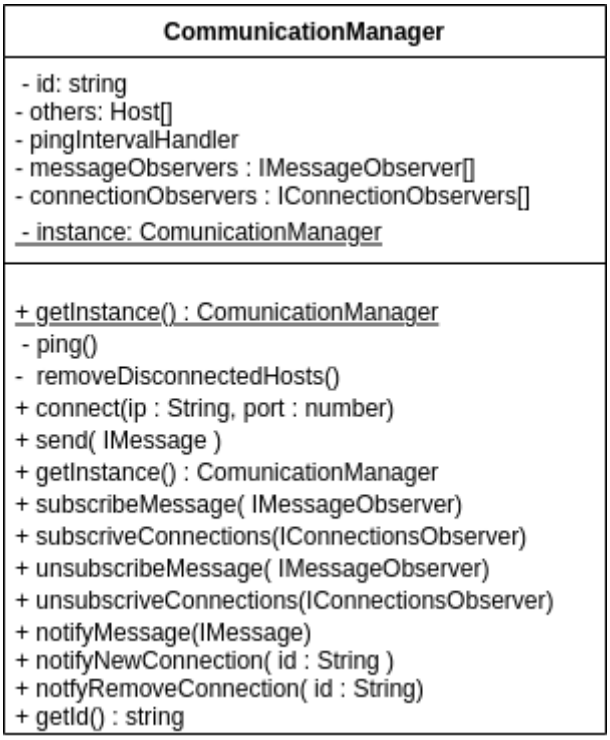
Obserwator: będzie upraszczał nam komunikacje pomiędzy ConnectionManager i Job. Obserwatorzy będą mogli obserwować zarówno nowo dodane połączenia oraz usunięcia starych połączeń.



Budowniczy: dzięki zastosowaniu tego wzorca będziemy mogli w prosty sposób tworzyć wiadomości różnych typów nie zagłębiając się w ich strukturę.



Singleton: w naszym systemie nie ma potrzeby tworzenia wielu instancji klasy CommunicationMenagera, więc skorzystaliśmy z tego wzorca. Na wydajność całego systemu nie wpłynie to negatywnie. ponieważ tylko jeden wątek będzie korzystał z tej klasy. Tworzenie wielu instancji tej klasy byłoby o tyle kłopotliwe, że każda musiałaby posiadać osobne gniazdo i każda musiałaby przechowywać aktualne “połączenia” i mogłyby być one niespójne względem pozostałych instancji



Interface użytkownika

Z każdym węzłem jest zintegrowany serwer http, przez który użytkownik będzie mieć możliwość dodawania nowych zadań, śledzenia połączeń i stanu aktualnie wykonywanych zadań z poziomu przeglądarki.

API węzła

Nasze Api będzie miało trzy endpointy.

\new - Dodanie taska

\stat - Pobranie statystyk odnośnie aktualnych połączeń oraz wykonanych zadań

\ - przesłanie interfejsu użytkownika