ДОДАТОК А

Графічна частина атестаційної роботи

Харківський національний університет радіоелектроніки

Кафедра АПОТ

Атестаційна робота магістра

Системи управління реального часу віддаленими об’єктами

Студента групи СКСм-18-1
Євчука Костянтина Павловича

Керівник:
Доц. каф. АПОТ
Шкіль О.С.

Харків 2020

Вступ

Мета роботи – розробка методу побудови системи реального часу з використанням технології віддаленої комунікації DTN та паралельного програмування.

Актуальність даної роботи пов’язана зі зростаючою потребою в реалізації систем реального часу у віддаленому контексті, а відповідно і у тестуванні.

Реалізація системи повинна розв’язувати проблему втрати зв’язку між множиною об’єктів у розподіленій системі реального часу зі збереженням та синхронізацією даних. Сферами, що повинна покривати реалізація системи, може бути авіа, космічна, наземна інфракструктура з зовнішними катаклізмами і т.д

У концепції побудови систем керування віддалених об’єктів у реальному часі лежить декілька компонентів:

- безпосередньо віддалений об’єкт, відповідальний за проведення наукових досліджень, обчислювальних процесів;
- пульт (сервіс) керування, відповідальний за прийняття та доставку конфігурації віддалених об’єктів в залежності від отриманих даних.

2

Постановка задач

Об’єктом дослідження є моделі систем управління реального часу у віддаленому контексті.

Предметом дослідження є застосування інтеграції DTN протоколу при проектуванні системи управління реального часу за допомогою мови програмування Golang.

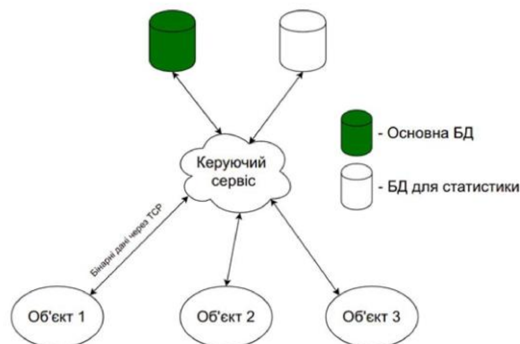
Поставлені наступні завдання:

- дослідження протоколів мережі DTN та способи інтеграції в системи реального часу;
- розробка системи управління реального часу віддаленими об’єктами;
- аналіз поведінки системи на збої та розриви у зв’язку;
- розробка алгоритму тестування конфігурації у контексті управління віддаленими об’єктами.

Технічне завдання полягає в розробці моделей віддалених об’єктів та сервісу керування, що можуть бути протестовані на втрату зв’язку, помилки та попередження.

3

Попередня архітектура системи. Недоліки



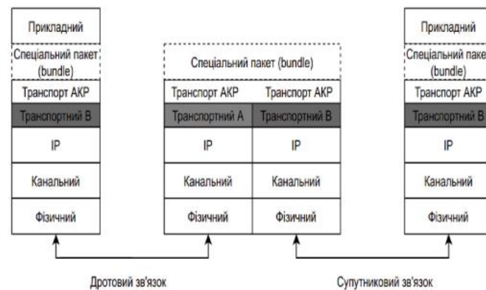
Серед недоліків попередньої системи важливо відзначити:

- неможливість використання системи у віддаленому контексті через використання TCP протоколу;
- відсутність обробки обривів зв’язку;
- відсутність єдиного протоколу обробки помилок та попереджень;
- відсутність резервного копіювання статистичних даних;
- відсутність зберігання помилок та попереджень об’єктів.

Неможливість використання системи у віддаленому контексті виникла з причини використання TCP протоколу як основного підходу до комунікацій у системі. TCP протокол не має підтримки багато-канального зв’язку та передбачає 2 вузла комунікації.

4

Програмна реалізація протоколу мережі DTN



Мережі толерантні до затримок та обривів (DTN) характеризуються стабільністю до відсутності зв'язку. У середовищах зі складними умовами таки популярним протоколам спеціального маршрутизації, як AODV та DSR або вже відомим протоколам TCP або HTTP (попередня реалізація системи), не вдається встановити маршрути. Це пов'язано з тим, що ці протоколи намагаються спочатку встановити повний маршрут, а потім, після встановлення маршруту, переслати фактичні дані.

5

Маршрутизація на основі реплікацій

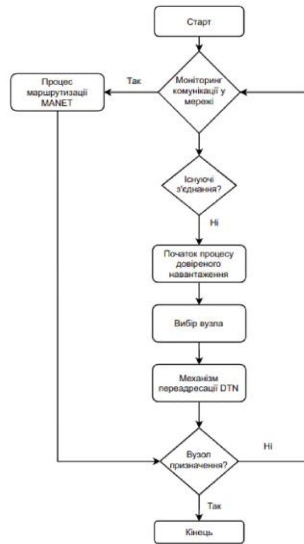
Протоколи на основі реплікації останнім часом привернули багато уваги в науковому співтоваристві, оскільки вони можуть забезпечити значно кращі коефіцієнти доставки повідомлень, ніж протоколи, що базуються на переадресації. Ці типи протоколів маршрутизації дозволяють реплікувати повідомлення; кожну репліку, як і саме оригінальне повідомлення, зазвичай називають копіями повідомлень або репліками повідомлення. Можливі проблеми з маршрутизацією на основі реплікації означають:

- перевантаженість мережі в кластерних районах;
- марні витрати на мережеві ресурси (включаючи пропускну здатність, накопичувач та енергію);
- масштабованість мережі.

Оскільки мережеві ресурси можуть швидко обмежуватися, вирішуючи, які повідомлення передавати першими, а які повідомлення потрібно скинути, відіграють важливу роль у багатьох протоколах маршрутизації.

6

Визначення вузлу призначення



Протокол DTN використовується у системі, як основний спосіб комунікації та передачі даних. За замовчуванням кожний компонент системи конфігурується як окремий мережевий bundle.

Конфігурація об’єкта у мережі DTN

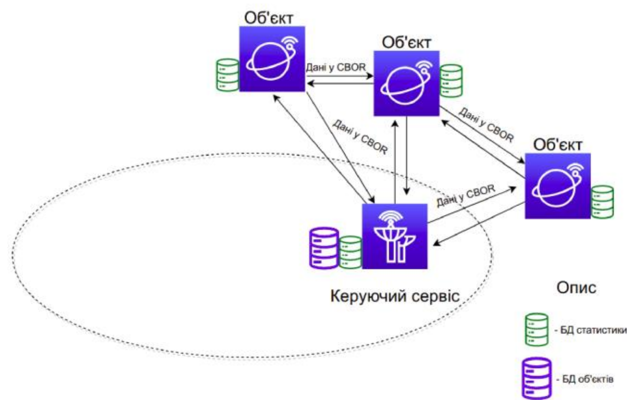
UID_LISTEN_HOST=localhost
UID_LISTEN_PORT=3001
UID_NAME=object2001
UID_NETWORK_NAME=bundle1

MAIN_SEND_HOST=localhost
MAIN_SEND_PORT=4000

NEIGHBOUR_SEND_HOST=localhost
NEIGHBOUR_SEND_PORT=3002
NEIGHBOUR_NETWORK_NAME=bundle2

7

Запропонована архітектура системи



Програмні компоненти поточної системи включають:

- модель віддаленого об’єкта, у космічному контексті – супутник, з назвою *object2000*;
- модель віддаленого об’єкта, у космічному контексті – супутник, з назвою *object2001*;
- сервіс керування – система управління віддалених об’єктів, назва – *air-management*.

8

Інструменти програмної реалізації

Розробка програмного продукту була виконана за допомогою такої мови програмування Go (Golang).

Go використовує горутіни замість потоків. Вони споживають майже 2кб пам'яті з heap. Тому, є можливість запускати сотні горутін в будь-який час.

Горутіни також мають такі переваги:

- використовуються сегментовані стеки – це означає, що горутіни будуть використовувати пам'ять тільки в разі потреби;
- горутіни мають більш швидкий час запуску, ніж потоки;
- горутіни поставляються з вбудованими примітивами для безпечного обміну даними між собою (канали);
- горутіни дозволяють уникнути необхідності вдаватися до блокування м'ютексів при спільному використанні структур даних;
- також, горутіни і потоки ОС не мають зіставлення 1: 1. Одна горутіна може запускатися в безлічі потоків. Горутіни об'єднані (multiplexed) в малу кількість потоків ОС.

9

Інструменти зберігання даних

Панель моніторингу статистичних даних об'єктів

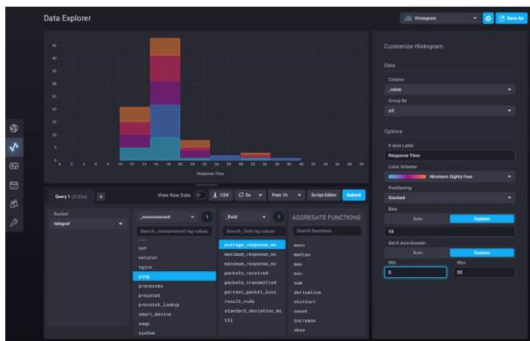
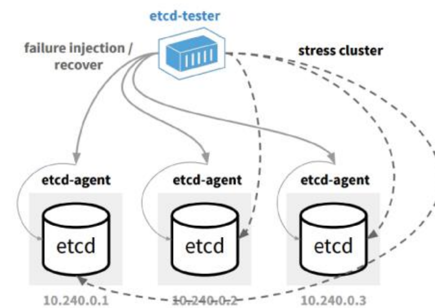


Схема аварійного зберігання помилок та попереджень



10

Конфігурація зберігання даних

Запуск Influx за допомогою Docker

```
docker run -p 8086:8086 \
    -v $PWD:/var/lib/influxdb \
    influxdb
```

Запуск ETCD за допомогою Docker

```
docker run -p 8086:8086 \
    -v $PWD:/var/lib/etcd \
    etcd
```

Для запуску моделей віддалених об'єктів також потребується запуск 2 баз даних на середовищі кожного з об'єктів.

Швидким способом запуску є використання docker-середовища. Docker – спеціальний інструмент для запуску та управління ізольованим Linux-контейнером.

У контексті запуску баз даних системи Docker дозволяє не перейматись вмістом контейнера та запустити довільні процеси баз даних і ізолювати їх від логіки сервісів реалізованої системи.

Також Docker реалізований за допомогою мови програмування Golang, що сприяє швидкої інтеграції.

Конфігурація сервісу керування

Параметри сервісу

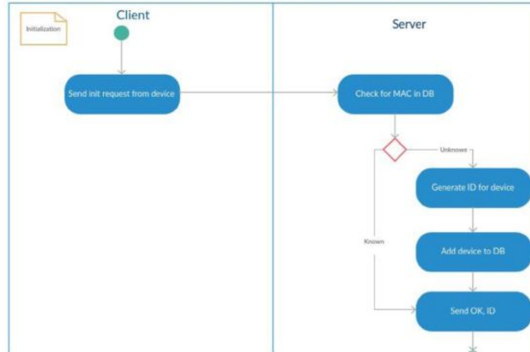
```
PORT=5000
MYSQL_USER=test
MYSQL_PASSWORD=password
MYSQL_HOST=localhost
MYSQL_PORT=3306
MYSQL_NAME=air-manage
INFLUX_USER=test
INFLUX_PASSWORD=password
INFLUX_HOST=localhost
INFLUX_PORT=8089
INFLUX_NAME=air-manage
TRANSPORT_RECEIVE_HOST=localhost
TRANSPORT_RECEIVE_PORT=4000
TRANSPORT_SEND_HOST=localhost
TRANSPORT_SEND_PORT=4040
```

Порт 5000 відкриває доступ до HTTP API, за допомогою котрого здійснюється симуляція обривів зв'язку. Конфігураційні параметри з префіксами MYSQL та INFLUX містять необхідні налаштування для баз даних. У свою чергу, керуючий сервіс має 2 інтерфейси – прийому та відправлення даних, але сконфігурований як 1 bundle (Main). Це дозволяє ідентифікувати керуючий сервіс та заборонити репліки (у майбутньому такі налаштування можуть бути відкинуті для масштабування). Запуск баз даних також може бути здійснений за рахунок інструменту Docker.

Останнім пунктом конфігурації системи є запуск DTN образу, що зможе виконати маршрутизацію та з'єднати кожен з об'єктів у мережі (bundle). Для цього може бути використаний також образ Docker та контейнер. Зазначимо, що для локального запуску та моделювання достатньо контейнеру, але для повного тестування бажано запустити повноцінний механізм маршрутизації на кожному середовищі для першого, другого об'єктів та керуючого сервісу.

API-інтерфейс управління сервісом

Алгоритм створення віддаленого об'єкта у сервісі керування



Користувач (інженер, розробник) повинен відправити POST запит у вигляді `http POST :5000/object id="object2000" name="object0" status:=true`. Таким чином, в основній базі даних буде створений запис *object2000* і коли модель віддаленого об'єкта розпочне роботу, керуючий сервіс матиме можливість ідентифікувати об'єкт та його конфігурацію.

Конфігурація моделі віддаленого керованого об'єкта

Параметри об'єкту

UID_LISTEN_HOST=localhost

UID_LISTEN_PORT=3000

UID_NAME=object2000

UID_NETWORK_NAME=bundle1

MAIN_SEND_HOST=localhost

MAIN_SEND_PORT=4000

NEIGHBOUR_SEND_HOST=localhost

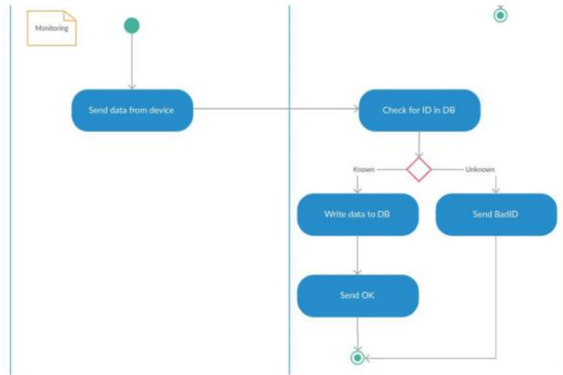
NEIGHBOUR_SEND_PORT=3001

NEIGHBOUR_NETWORK_NAME=bundl e2

У даному контексті UID – спеціальний ідентифікатор, що допомагає ідентифікувати об'єкт як bundle у DTN мережі. Network name (ім'я в мережі) для object2000 буде bundle1. Конфігураційні параметри з префіксом MAIN позначають ідентифікацію керуючого сервісу, а префікс NEIGHBOUR – об'єкта-сусіда, за допомогою якого буде здійснюватися зв'язок у разі обриву.

Конфігурація наступної моделі віддаленого об'єкту object2001 буде аналогічною до object2000, адже за технічним завданням – об'єкти повинні мати однакові кодові бази, але різну обчислювальну мету.

Створення зв’язку об’єктів та сервісу керування



Модель віддаленого об’єкта виконує з’єднання зі сервісом керування та починає передавати обчислювальні дані з поточною конфігурацією сервісу. В цей час сервіс керування публікує повідомлення про успішне отримання даних від моделі віддаленого об’єкту object2000.

15

Результати роботи системи під час запуску

Повідомлення про успішний запуск сервісу керування

```
Terminal: Земля Супутник1 Супутник2 Командний пульт +
2020/04/20 23:31:24 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:25 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:26 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:27 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:28 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:29 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:30 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:31 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:32 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:33 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:34 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:35 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:36 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:37 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:38 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:39 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:40 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:41 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:42 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:43 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:44 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:45 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:46 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:47 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:48 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:49 [INFO] received data from object - (object2000), bundle - (bundle0)
2020/04/20 23:31:50 [INFO] received data from object - (object2001), bundle - (bundle1)
2020/04/20 23:31:51 [INFO] received data from object - (object2000), bundle - (bundle0)
^CMakefile:24: recipe for target 'run' failed
make: *** [run] Interrupt

kostiantyn@kostiantyn-Inspiron-5370:~/Projects/yevchuk-kostiantyn/management$
```

Повідомлення про успішний запуск віддаленого об’єкту object2020

```
Terminal: Земля Супутник1 Супутник2 Командний пульт +
2020/04/20 23:31:25 [INFO] collected - id: object2000, temp: 0.753161, time: 2020-04-20 23:31:25.365212118 +0300 EEST m=+274.001310088
2020/04/20 23:31:25 [INFO] data successfully sent &{object2000 bundle0 0.7531612777567586 2020-04-20 23:31:25.365212118 +0300 EEST m=+274.001310088}
2020/04/20 23:31:27 [INFO] collected - id: object2000, temp: 0.150964, time: 2020-04-20 23:31:27.365285752 +0300 EEST m=+276.001383738
2020/04/20 23:31:27 [INFO] data successfully sent &{object2000 bundle0 0.150964844979607 2020-04-20 23:31:27.365285752 +0300 EEST m=+276.001383738}
2020/04/20 23:31:29 [INFO] collected - id: object2000, temp: 0.355767, time: 2020-04-20 23:31:29.365266286 +0300 EEST m=+278.001364176
2020/04/20 23:31:29 [INFO] data successfully sent &{object2000 bundle0 0.35576726540923664 2020-04-20 23:31:29.365266286 +0300 EEST m=+278.001364176}
2020/04/20 23:31:31 [INFO] collected - id: object2000, temp: 0.831931, time: 2020-04-20 23:31:31.365301628 +0300 EEST m=+280.001399597
2020/04/20 23:31:31 [INFO] data successfully sent &{object2000 bundle0 0.8319388529698163 2020-04-20 23:31:31.365301628 +0300 EEST m=+280.001399597}
2020/04/20 23:31:33 [INFO] collected - id: object2000, temp: 0.231830, time: 2020-04-20 23:31:33.365188814 +0300 EEST m=+282.001286800
2020/04/20 23:31:33 [INFO] data successfully sent &{object2000 bundle0 0.2318388419376769 2020-04-20 23:31:33.365188814 +0300 EEST m=+282.001286800}
2020/04/20 23:31:35 [INFO] collected - id: object2000, temp: 0.627835, time: 2020-04-20 23:31:35.365312537 +0300 EEST m=+284.001410503
2020/04/20 23:31:35 [INFO] data successfully sent &{object2000 bundle0 0.6278346850000227 2020-04-20 23:31:35.365312537 +0300 EEST m=+284.001410503}
2020/04/20 23:31:37 [INFO] collected - id: object2000, temp: 0.498394, time: 2020-04-20 23:31:37.365190733 +0300 EEST m=+286.001288694
2020/04/20 23:31:37 [INFO] data successfully sent &{object2000 bundle0 0.4983943812759756 2020-04-20 23:31:37.365190733 +0300 EEST m=+286.001288694}
2020/04/20 23:31:39 [INFO] collected - id: object2000, temp: 0.889836, time: 2020-04-20 23:31:39.365217488 +0300 EEST m=+288.001315462
2020/04/20 23:31:39 [INFO] data successfully sent &{object2000 bundle0 0.88983688926036683 2020-04-20 23:31:39.365217488 +0300 EEST m=+288.001315462}
2020/04/20 23:31:41 [INFO] collected - id: object2000, temp: 0.825194, time: 2020-04-20 23:31:41.365215033 +0300 EEST m=+290.001313806
2020/04/20 23:31:41 [INFO] data successfully sent &{object2000 bundle0 0.82519395979489504 2020-04-20 23:31:41.365215033 +0300 EEST m=+290.001313806}
2020/04/20 23:31:43 [INFO] collected - id: object2000, temp: 0.392216, time: 2020-04-20 23:31:43.36518173 +0300 EEST m=+292.001279702
2020/04/20 23:31:43 [INFO] data successfully sent &{object2000 bundle0 0.3922161831540248 2020-04-20 23:31:43.36518173 +0300 EEST m=+292.001279702}
2020/04/20 23:31:45 [INFO] collected - id: object2000, temp: 0.589383, time: 2020-04-20 23:31:45.365176706 +0300 EEST m=+294.001274750
2020/04/20 23:31:45 [INFO] data successfully sent &{object2000 bundle0 0.5893838864007992 2020-04-20 23:31:45.365176706 +0300 EEST m=+294.001274750}
2020/04/20 23:31:47 [INFO] collected - id: object2000, temp: 0.929612, time: 2020-04-20 23:31:47.365188445 +0300 EEST m=+296.001286430
2020/04/20 23:31:47 [INFO] data successfully sent &{object2000 bundle0 0.9296116354490302 2020-04-20 23:31:47.365188445 +0300 EEST m=+296.001286430}
2020/04/20 23:31:49 [INFO] collected - id: object2000, temp: 0.572087, time: 2020-04-20 23:31:49.365178723 +0300 EEST m=+298.001276701
2020/04/20 23:31:49 [INFO] data successfully sent &{object2000 bundle0 0.5720868801443084 2020-04-20 23:31:49.365178723 +0300 EEST m=+298.001276701}
2020/04/20 23:31:51 [INFO] collected - id: object2000, temp: 0.508576, time: 2020-04-20 23:31:51.365180227 +0300 EEST m=+300.001278191
2020/04/20 23:31:51 [INFO] data successfully sent &{object2000 bundle0 0.5085763451434821 2020-04-20 23:31:51.365180227 +0300 EEST m=+300.001278191}
^C2020/04/20 23:31:51 [WARN] caught closing signal
2020/04/20 23:31:51 [WARN] shutting down generator
2020/04/20 23:31:51 [WARN] shutting down sender
```

16

Результати роботи системи під час обривів зв’язку

Повідомлення сервісу про отримання даних object2000 від object2001

На даному етапі обриви зв’язку можуть з’явитися з різних причин, якщо це космічний контекст, то один із об’єктів супутників може вийти в тінь, таким чином втративши зв’язок з Землею.

Важливо зауважити, що в такій ситуації хоча *object2000* фактично втратив зв’язок з керуючим сервісом, все одно завдяки конфігурації та маршрутизації DTN можливо підтримувати зв’язок з керуючим сервісом через *object2001,* котрий у даному випадку окрім виконання обчислювальних операцій, стане об’єктом-медіатором, через який *object2000* та керуючий сервіс зможуть підтримувати зв’язок.

17

Результати роботи системи після налагодження зв’язку

Через проміжок часу *object2000* може стати знову доступним, тобто вийти з тіні, якщо у космічному контексті. Для цього користувач має активувати керуючий сервіс для цього об’єкта виконуючи команду “http PATCH :5000/object/object2000/on”. В такому разі, *object2000* перестає використовувати *object2001* у якості медіатора та повертає одно-канальний зв’язок з керуючим сервісом.

На рисунку керуючий сервіс продовжує отримувати дані *object2000* від bundle0, тобто від *object2000.*

Підводячи підсумки реалізації системи управління віддаленим об’єктом у мережі DTN, необхідно зазначити, що недоліки попередньої системи були виправлені.

18

Перспективи досліджень

Напрямком подальших досліджень може бути використання лімітів на кількість повідомлень та з’єднань.

Зазвичай ресурси віддалених систем обмежені, тож у випадку аварії повідомлення повинні бути обмежені.

Таким чином, можна розглянути алгоритми знаходження найефективнішого або об’єкта медіатора, що має велику кількість ресурсів.

19

Висновки

В ході виконання атестаційної роботи розроблено модель віддаленого об’єкту та керуючий сервіс на основі стандарту побудови протоколу мережі DTN. При проведенні розробки докладно розглянуті алгоритми передачі даних в мережі толерантної до переривань, був обраний найоптимальніший для системи, а саме flooding.

Результати тестування підтверджують коректність роботи системи. Статистичні та конфігураційні дані були успішно синхронізовані, а обрив зв’язку між віддаленим об’єктом та керуючим сервісом не мав наслідків.

Реалізація даної системи повністю підтверджує сучасність розглянутих методів, алгоритмів та інструментів побудови СРЧ. Використання таких методів повністю може використовуватися при розробці різних проектів, мета яких протестувати конфігурації об’єктів та чи матиме наслідки система у випадку обривів зв’язку. Поточна система є легко-конфігурованою, кодова база моделі об’єкту одна, тобто для створення нового об’єкту для тестування потрібно тільки змінити конфігураційний файл. Також, кожний із модулів підтримує алгоритми масштабованості, тобто система може мати нескінчену кількість об’єктів та керуючих сервісів.

Наукова новизна роботи полягає у реалізації способу вирішення проблеми втрати зв’язку між об’єктами в системах реального часу. Такі проблеми виникають все частіше і частіше. Одним із прикладів використання системи може бути пожежа поблизу дата-центру з важливими об’єктами та їх даними або втрата зв’язку із супутником та важливими науковими даними. В такому разі, система, що була реалізована в роботі, матиме можливість зберігти та передати важливу інформацію, таким чином зберігти витрачені ресурси. Простий інтерфейс та гнучка конфігурація виконаної роботи дозволить розробникам та науковцям підтвердити або спростувати свої розробки шляхом налаштування поточної реалізації.

Практична цінність роботи полягає в можливості використання поточної системи як сторонньої бібліотеки, адже кожний компонент та модуль системи не залежить один від одного, або має інтерфейс для контакту.

20

ДОДАТОК Б

Програмна реалізація керуючого сервісу

Головний файл (власний код, файл main.go)

```
package main

import (
	"log"
	"management/pkg/maindb"
	"management/pkg/statsdb"
	"management/service"
	"management/transport"
	"sync"
)

func main() {
	statsStorage := statsdb.Init()
	defer statsStorage.Close()

	mainStorage, err := maindb.Init()
	if err != nil {
		log.Fatal(err.Error())
	}

	wg := sync.WaitGroup{}
	wg.Add(1)
	go service.Run(mainStorage, statsStorage, &wg)
	wg.Add(1)
	go transport.Start(mainStorage, statsStorage, &wg)

	wg.Wait()
}
```

Ініціалізація статистичної БД (власний код, файл statsdb/conn.go)

```
package statsdb

import (
	"fmt"
	influx "github.com/influxdata/influxdb/client/v2"
	"log"
	"management/model"
)
```

```
func Init() model.StatsDataStore {
	DBHost := "localhost"
	DBPort := "8086"

	db, err := influx.NewHTTPClient(influx.HTTPConfig{Addr: fmt.Sprintf("http://%s:%s",
DBHost, DBPort)})
	if err != nil {
		log.Fatalf("[ERROR] stats db init failed - (%s)", err.Error())
		return nil
	}

	//_, _, err = db.Ping(3 * time.Second)
	//if err != nil {
	//	log.Fatalf("[ERROR] stats db ping failed - (%s)", err.Error())
	//}

	return &model.StatsDB{RW: db}
}
```

Ініціалізація основної БД (власний код, файл maindb/conn.go)

```
package maindb

import (
	"database/sql"
	"fmt"
	_ "github.com/go-sql-driver/mysql"
	"github.com/kelseyhightower/envconfig"
	"log"
	"management/model"
)

type config struct {
	User     string `split_words:"true" required:"true"`
	Password string `split_words:"true" required:"true"`
	Host     string `split_words:"true" required:"true"`
	Port    string `split_words:"true" required:"true"`
	Name     string `split_words:"true" required:"true"`
}

func Init() (model.MainDataStore, error) {
	var c config
	if err := envconfig.Process("MYSQL", &c); err != nil {
		return nil, err
	}

	var connString = fmt.Sprintf("%s:%s@tcp(%s:%s)/%s?parseTime=true", c.User,
c.Password, c.Host, c.Port, c.Name)

	db, err := sql.Open("mysql", connString)
	if err != nil {
		log.Fatalln("[error] db.Open: ", err, " exiting now ...")
		db.Close()
		return nil, err
	}
```

```
        err = db.Ping()
        if err != nil {
                log.Fatalln("[error] db.Ping: ", err, " exiting now ...")
                db.Close()
                return nil, err
        }

        return &model.DB{RW: db}, nil
}
```

Транспортний компонент (власний код, файл transport.go)

```
package transport

import (
        "encoding/gob"
        "fmt"
        "github.com/kelseyhightower/envconfig"
        "log"
        "management/model"
        "net"
        "sync"
)

type config struct {
        ReceiveHost string `split_words:"true" required:"true"`
        ReceivePort string `split_words:"true" required:"true"`
        SendHost    string `split_words:"true" required:"true"`
        SendPort    string `split_words:"true" required:"true"`
}

type service struct {
        mainStore  model.MainDataStore
        statsStore model.StatsDataStore
}

func Start(mainStorage model.MainDataStore, statsStorage model.StatsDataStore, wg
*sync.WaitGroup) {
        defer wg.Done()

        var c config
        if err := envconfig.Process("TRANSPORT", &c); err != nil {
                log.Printf("[ERROR] transport config parse failed - (%s)", err.Error())
                return
        }

        wg.Add(1)
        c.Listen(mainStorage, statsStorage, wg)
}

func (c *config) Listen(mainStorage model.MainDataStore, statsStorage model.StatsDataStore,
wg *sync.WaitGroup) {
        defer wg.Done()
```

```
		service := service{mainStore: mainStorage, statsStore: statsStorage}
		addr := fmt.Sprintf("%s:%s", c.ReceiveHost, c.ReceivePort)
		l, err := net.Listen("tcp", addr)
		if err != nil {
			log.Printf("[ERROR] tcp listen - (%s)", err.Error())
			return
		}

		defer l.Close()
		log.Printf("[INFO] listening on: %s", addr)
		for {
			conn, err := l.Accept()
			if err != nil {
				log.Printf("[ERROR] tcp accept - (%s)", err.Error())
				return
			}

			go service.handleRequest(conn)
		}
	}

	func (s *service) handleRequest(conn net.Conn) {
		defer conn.Close()
		for {
			decoder := gob.NewDecoder(conn)

			var req model.Request
			err := decoder.Decode(&req)
			if err != nil {
				log.Printf("[ERROR] request decode failed - (%s)", err.Error())
				// TODO handle error
				return
			}

			object, err := s.mainStore.GetObjectByID(req.Data.ObjectID)
			if err != nil {
				log.Printf("[ERROR] getting object by id failed - (%s):", err.Error())
			}

			if object.MainActive {
				log.Printf("[INFO] received data from object - (%s), bundle - (%s)",
req.Data.ObjectID, req.Data.NetworkName)
				err = s.mainStore.AddObjectData(&req.Data)
				if err != nil {
					log.Printf("[ERROR] saving object data failed - (%s)",
err.Error())
				}
			} else if !object.MainActive && req.Data.NetworkName !=
object.NetworkName {
				log.Println("[WARN] DATA FROM UNAVAILABLE DEVICE")
				log.Printf("[INFO] received data from object - (%s), bundle - (%s)",
req.Data.ObjectID, req.Data.NetworkName)
				err = s.mainStore.AddObjectData(&req.Data)
				if err != nil {
					log.Printf("[ERROR] saving object data failed - (%s)",
err.Error())
```

```
				}
			}
			resp := model.Response{
				ObjectID:   object.ID,
				MainActive: object.MainActive,
			}
			encoder := gob.NewEncoder(conn)
			err = encoder.Encode(resp)
			if err != nil {
				log.Printf("[ERROR] response encode failed - (%s)", err.Error())
			}
		}
	}
```

Програмна реалізація моделі віддаленого об’єкту

Головний файл (власний код, main.go)

```
package main

import (
	"object/service"
	"object/transport/receiver"
	"sync"
)

func main() {
	wg := sync.WaitGroup{}

	wg.Add(1)
	go service.Run(&wg)

	wg.Add(1)
	go receiver.Start(&wg)

	wg.Wait()
}
```

Об’єднуючий сервіс збирання, обчислення та відправки даних (власний код, файл service.go)

```
func Run(wg *sync.WaitGroup) {
	defer wg.Done()
	var c transport.Config
	var uidConfig receiver.UIDConfig

	if err := envconfig.Process("MAIN", &c); err != nil {
		log.Printf("[ERROR] parsing main bundle config failed - (%s)", err.Error())
```

```
		return
	}

	if err := envconfig.Process("UID", &uidConfig); err != nil {
		log.Printf("[ERROR] parsing object bundle config failed - (%s)", err.Error())
		return
	}
	done := make(chan struct{})
	errors := make(chan error)
	generatedData := make(chan *model.ObjectData)

	conn, err := sender.NewConn(&c)
	if err != nil {
		log.Println("[ERROR]:", err.Error())
		return
	}
	go trackClose(done)

	wg.Add(1)
	go data.Generate(uidConfig, generatedData, errors, done, wg)

	wg.Add(1)
	go data.Send(conn, generatedData, errors, done, wg)
}

func trackClose(done chan<- struct{}) {
	sigs := make(chan os.Signal, 1)
	signal.Notify(sigs, os.Interrupt, syscall.SIGTERM)
	for {
		s := <-sigs
		switch s {
		case os.Interrupt, syscall.SIGTERM:
			log.Println("[WARN] caught closing signal")
			close(done)
			return
		}
	}
}
```

Функція відправки даних (власний код, файл data.go)

```
func Send(conn net.Conn, data chan *model.ObjectData, errors chan<- error, done chan struct{},
wg *sync.WaitGroup) {
	defer wg.Done()
	for {
		select {
		case <-done:
			close(data)
			log.Println("[WARN] shutting down sender")
			return
		case d := <-data:
			req := model.Request{
				Type: "device data",
				Data: *d,
```

```
                }
                encoder := gob.NewEncoder(conn)
                err := encoder.Encode(req)
                if err != nil {
                        // TODO handle error
                        log.Printf("[ERROR] request send failed - (%s)", err.Error())
                        errors <- err
                }

                var resp model.Response
                decoder := gob.NewDecoder(conn)
                err = decoder.Decode(&resp)
                if err != nil {
                        log.Printf("[ERROR] response decode failed - (%s)", err.Error())
                        errors <- err
                }

                if !resp.MainActive {
                        type neighbour struct {
                                SendHost    string `split_words:"true" required:"true"`
                                SendPort    string `split_words:"true" required:"true"`
                                NetworkName string `split_words:"true" required:"true"`
                        }
                        var c neighbour
                        if err := envconfig.Process("NEIGHBOUR", &c); err != nil {
                                log.Printf("[ERROR] parsing neighbour bundle config
failed - (%s)", err.Error())
                                return
                        }

                        config := &transport.Config{
                                SendHost: c.SendHost,
                                SendPort: c.SendPort,
                        }

                        log.Println("[WARN] main bundle not available, sending to
neighbour ...")
                        neighbourConn, err := sender.NewConn(config)
                        if err != nil {
                                log.Printf("[ERROR] new connection to neighbour
bundle failed - (%s)", err.Error())
                                return
                        }

                        req.Data.NetworkName = c.NetworkName
                        encoder := gob.NewEncoder(neighbourConn)
                        err = encoder.Encode(req)
                        if err != nil {
                                // TODO handle error
                                log.Printf("[ERROR] request send failed - (%s)",
err.Error())
                                errors <- err
                        }
                }

                log.Println("[INFO] data successfully sent", d)
        }
```

```
	}
}
```

Функція отримання даних від об’єкт-сусіда (власний код, receiver.go)

```
package receiver

import (
	"encoding/gob"
	"fmt"
	"log"
	"net"
	"object/model"
	"object/transport"
	"object/transport/sender"
	"sync"

	"github.com/kelseyhightower/envconfig"
)

type UIDConfig struct {
	ListenHost  string `split_words:"true" required:"true"`
	ListenPort  string `split_words:"true" required:"true"`
	Name        string `split_words:"true" required:"true"`
	NetworkName string `split_words:"true" required:"true"`
}

func Start(wg *sync.WaitGroup) {
	defer wg.Done()
	var c UIDConfig
	if err := envconfig.Process("UID", &c); err != nil {
		log.Printf("[ERROR] parsing object bundle config failed - (%s)", err.Error())
		return
	}

	var mainConfig transport.Config
	if err := envconfig.Process("MAIN", &mainConfig); err != nil {
		log.Printf("[ERROR] parsing main bundle config failed - (%s)", err.Error())
		return
	}
	log.Printf("[INFO] object name: %s, network name: %s", c.Name, c.NetworkName)

	wg.Add(1)
	c.Listen(wg, &mainConfig)
}

func (c *UIDConfig) Listen(wg *sync.WaitGroup, mainConfig *transport.Config) {
	defer wg.Done()

	addr := fmt.Sprintf("%s:%s", c.ListenHost, c.ListenPort)
	l, err := net.Listen("tcp", addr)
	if err != nil {
		log.Printf("[ERROR] tcp listen - (%s)", err.Error())
		return
```

```
		}

		defer l.Close()
		log.Printf("[INFO] listening on: %s", addr)
		for {
			conn, err := l.Accept()
			if err != nil {
				log.Printf("[ERROR] tcp accept - (%s)", err.Error())
				return
			}

			go handleRequest(conn, mainConfig)
		}
	}

	func handleRequest(conn net.Conn, mainConfig *transport.Config) {
		defer conn.Close()
		decoder := gob.NewDecoder(conn)
		var req model.Request
		err := decoder.Decode(&req)
		if err != nil {
			log.Printf("[ERROR] request decode failed - (%s)", err.Error())
			// TODO handle error
		}

		log.Printf("[INFO] received data from bundle - (%s); sending to main bundle", 
req.Data.ObjectID)
		mainConn, err := sender.NewConn(mainConfig)
		if err != nil {
			log.Printf("[ERROR] connection to main bundle failed - (%s)", err.Error())
			return
		}

		encoder := gob.NewEncoder(mainConn)
		err = encoder.Encode(req)
		if err != nil {
			// TODO handle error
			log.Printf("[ERROR] request send failed - (%s)", err.Error())
		}
		log.Printf("[INFO] successfully transfered data to main bundle")
	}
```

Моделі віддаленого об’єкту

```
	package model

	import "time"

	type Object struct {
		ID          string
		Name        string
		NetworkName string
```

```
	Status      bool
	CreatedAt   time.Time
}

type ObjectData struct {
	ObjectID    string
	NetworkName string
	Temperature float64
	CreatedAt   time.Time
}

type Request struct {
	Type string
	Data ObjectData
}

type Response struct {
	ObjectID   string
	Action     string
	Active     bool
	MainActive bool
}

type NetworkConfig struct {
	Name      string
	Neighbour string
	Main      string
}
```

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮІК. 46741Х.001 ПЗ	Пояснювальна записка	70 с.
		Графічні документи	
2		Слайд-презентація	20 слайдів
		Носії інформації	
3		Компакт-диск (CD)	1 шт.

Змін.	Арк	Номер докум.	Підп.	Дата	ГЮІК. 46741Х.001 ВД		
Розроб.		Євчук К.П.			Системи управління реального часу віддаленими об'єктами	Літ	Аркуш	Аркушів
Перевір.		Шкіль О.С.				У		
Н.контр.		Філіппенко І.В.			Відомість атестаційної роботи	ХНУРЕ Кафедра АПОТ		
Затв.		Чумаченко С.В.						