Data Distribution Service資料傳輸

Data Distribution Service(DDS)簡介

DDS (Data Distribution Service)是一種用於分散式系統的標準化軟體架構，它是一種高效的資料傳輸協定。它是基於發布/訂閱模型的，透過在不同的應用程式之間傳遞資料，可以實現分散式系統間的資料共享。

DDS透過一個中央的資料管理器來控制資料的傳輸。在這個架構下，資料的發布者和訂閱者可以透過相同的介面來互相溝通。資料發布者可以發布資料到中央資料管理器，然後訂閱者可以通過訂閱該資料，從中央資料管理器中獲取該資料。

DDS架構具有以下優點：

1. 高效：DDS可以實現高效的資料傳輸，它可以在不同的應用程式之間傳遞大量的資料，而且傳輸速度非常快。
2. 可靠性高：DDS可以保證資料的可靠性，即使在傳輸過程中出現問題，也可以保證資料的準確性。
3. 易於開發：DDS提供了一個簡單的介面，使開發人員可以方便地進行應用程式的開發。
4. 可擴展性高：DDS可以擴展到大規模的系統中，而且可以實現多種不同的資料傳輸模型。

總之，DDS是一種高效、可靠、易於開發和可擴展的資料傳輸架構，它可以實現分散式系統之間的資料共享，是現代分布式系統中不可缺少的一部分。

DDS Proxy設計原則

本專案整合 RTI Data Distribution Service(以下簡稱 RTI DDS)來完成資料交換傳送工作。

考慮一個影像交換應用的例子，在未整合 RTI DDS 之前，各式影像傳輸的資料會透過標準的協定來進行交換與傳輸。由於發送端與接收端會跨越 Internet，因此發送端與接收端會有 IP Routing 的問題，故可以透過 RTI DDS 來完成路由的工作。

為了提高 RTI DDS 的多用性，功能性與彈性，設計了一個 RTI DDS Proxy 的架構，使用 Socket 的方式與原本的應用程式溝通並透過 RTI DDS 與遠方的 RTI DDS Proxy 與應用程式交換資料。

主程式流程

本專案主要作為 DDS Proxy 幫忙代收轉傳的工作，因為程式流程可以透過**幫忙 Publish 資料**的程式流程與**幫忙 Subscribe 資料**這兩個部份來說明。

Publish 資料

本章節將介紹 DDS Proxy 如何作為 DDS Publisher ，將應用程式透過 IP/Port 發送給 DDS Proxy 的資料 Publish 出去。

確認設定檔內容後就可以執行 DDS Proxy，執行後根據設定檔內容，主程式會產生一個 Proxy Worker thread，這個 Worker 程式中稱為 UDP DDS worker。此時主程式就會進入到一個無窮迴圈中並且進行列印 keep-alive 訊息讓執行者確認程式運作狀態。

UDP DDS worker thread 則是開始進行初始化動作，該動作包含有：

- 初始化 DDS: 根據設定建立 DDS Publisher, Subscriber, Data Reader, Data Writer,與設定相對應的 QoS 參數。
- 初始化 Socket: 根據設定建立 Worker 監聽與發送資料的 socket，並設定相關的 socket 參數

初始化完成後，為了提昇效能（但是會增加 CPU 使用率），DDS Proxy 程式會將 Publish 功能的下述兩個部份分開執行

1. 接收應用程式來的資料與
2. 透過 DDS Publisher 把收到資料 Publish 出去

故 UDP DDS worker 會在產生一個 UDP Socket Worker Thread 來處理前述**接收應用程式來的資料的部份**。 Proxy Worker thread 產生 UDP Socket Worker Thread 後，就會進入一個無窮迴圈中，開始檢查資料 Buffer 內是否有資料，如果有就進行 Publish 的動作。**注意，產生 UDP Socket Worker Thread 並非必須**，如果沒有獨立使用 Socket Worker Thread，那原本的無窮迴圈會變成檢查 Socket 是否有來自應用程式的資料，如果有有就進行 Publish 的動作。

UDP Socket Worker Thread 的部份則是單純去檢查監聽的 Socket 是否有收到資料，如果有就去尋找空的資料 buffer，並且把資料放入 Buffer 之中。

上述流程只是一個 DDS Proxy worker 於 Publish 端進行的行為，如果選擇了執行兩個或是三個 Proxy Workers，則以上的動作都會在重複進行兩次或是三次。

Subscribe 資料

同樣的執行 DDS Proxy 後，根據設定檔內容，主程式一樣也是產生一個 Proxy Worker thread 並開始進行初始化動作，初始化動作如Publish 資料一樣的運作方式。

初始化完成後，主程式已經初始化一個 DDS Subscriber，這個 DDS Subscriber 將使用 DDS 官方 Library 提供之相關函數針對所設定的 Subscribe Topic 進行監聽 DDS 資料的動作。當發現有資料時，`on_data_available`函數會被呼叫來取得 Publisher 端發送的資料。收到資料後，我們就把資料透過初始階段的 `Socket` 快速的將資料發送給應用程式端。

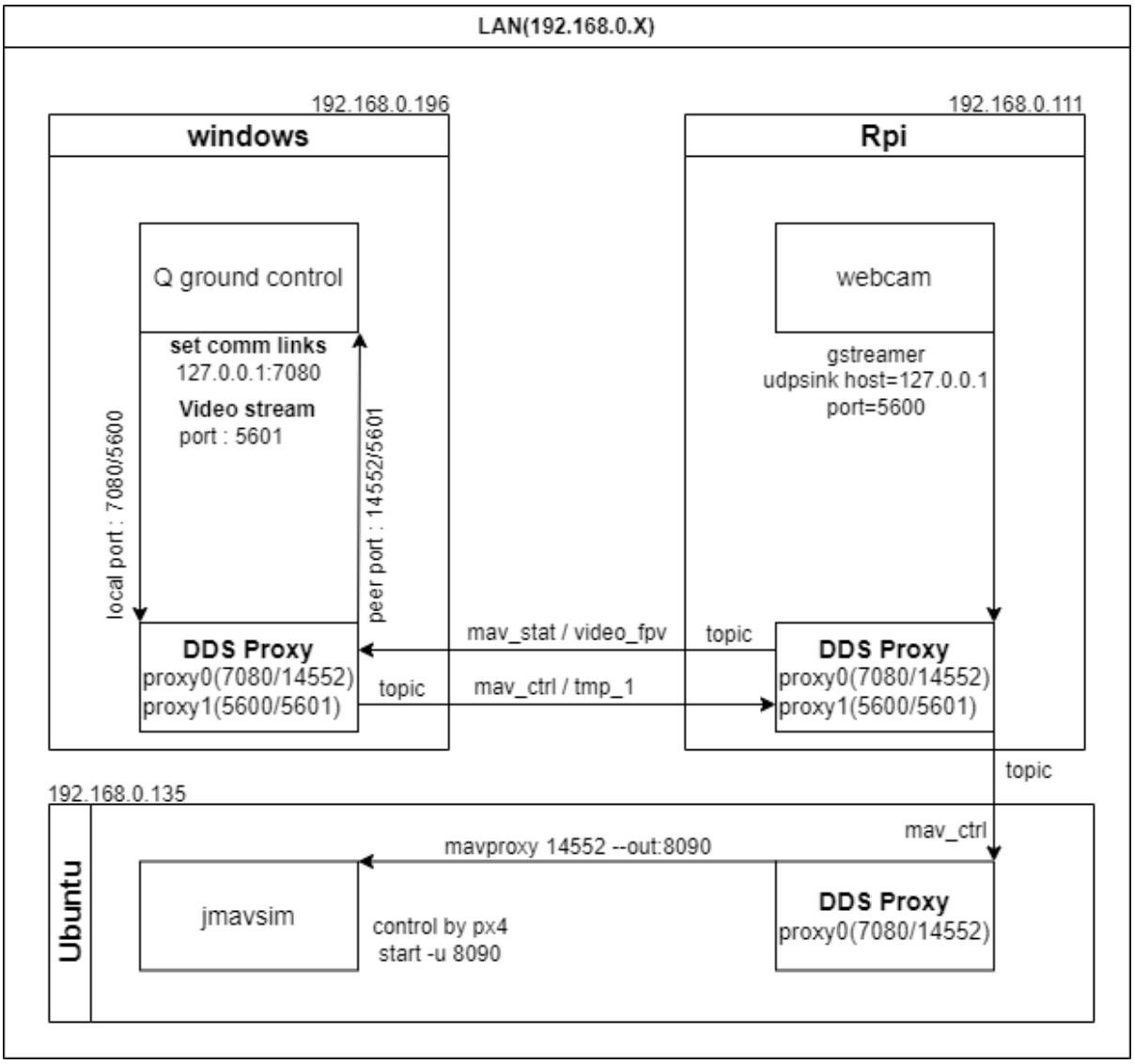
系統架構

本次使用三種設備進行資料傳輸，以RTI DDS v6.1.1版本傳輸

Windows	Ubuntu	Rpi
visual studio 2019	oracle VM	pi 4B
	ubuntu 20.0.4	
安裝rti_connext.exe，將rtipkg安裝到launcher中，並匯入license	安裝rti_connext.run，將x64的rtipkg安裝到launcher中，並匯入license	安裝rti_connext.run，將arm的rtipkg安裝到launcher中，並匯入license
將rti_dds設為NDDSHOME環境變數	export NDDSHOME=rti_connext路徑	export NDDSHOME=rti_connext路徑
建置DDSProxy專案	./make build	./make build -r 8
執行專案	於obj中找到Proxy資料夾執行DDSProxy	於obj中找到Proxy資料夾執行DDSProxy

DDS應用於無人機模擬測試

以windows系統操作無人機地面站(Q ground control)，Ubuntu模擬無人機飛行與飛控板控制，樹梅派傳輸fpv即時影像，系統在LAN中實現，使Q ground control與飛控板可以雙向溝通，系統架構圖如下。



1. Windows(飛控介面):
    - DDS傳輸mavlink控制命令&接收無人機狀態
    - DDS接收樹梅派fpv影像顯示於Q ground control
    - Q ground control連接DDS通道，設定飛行任務，發送mavlink控制命令
2. 樹梅派(可裝置於無人機上):
    - DDS接收mavlink控制命令&傳送無人機狀態
    - DDS傳輸fpv即時影像
    - 介接Q ground control與無人機飛控板

3. Ubuntu(模擬無人機與飛控板):
    - 以maxproxy建立DDS與飛控板的mavlink通道
    - 使用PX4與jmavsim模擬無人機飛行畫面與控制

DDS資料傳輸測試

可於內網LAN中測試，測試設備可以下列方式組合

```
1.ubuntu(pub/sub) - ubuntu(pub/sub)
2.ubuntu(pub/sub) - windows(pub/sub)
3.ubuntu(pub/sub) - Rpi(pub/sub)
4.Rpi(pub/sub)    - windows(pub/sub)
```

文字傳輸

1. windows

   執行專案，與proxy於背景代傳資料，使用python的socket向指定port傳輸資料，proxy會自動將資料pub到domain中；或以socket接收指定port的資料，當訂閱的topic有資料更新時，proxy會自動將資料sub至port中。

2. ubuntu

   需將VM的網路卡設置為橋接介面卡，使VM在網域中有獨立的ip，接著執行proxy於背景，另開新終端機使用nc指令傳送/監聽port資料。

   ```
   nc -> udp_pub -> udp_sub -> nc (listen)
   ```

3. Rpi

   將Rpi連限制相同網域中，執行proxy於背景，另開新終端機使用nc指令傳送/監聽port資料。

影像串流傳輸

1. windows

   執行專案，與proxy於背景代傳資料，另開新終端機執行下列指令。

   - pub gstreamer h.264 video

```
gst-launch-1.0.exe -v ksvideosrc do-stats=TRUE ! videoconvert !
x264enc speed-preset=superfast tune=zerolatency ! rtph264pay !
udpsink host=127.0.0.1 port=(指定的local port)
```

- sub gstreamer h.264 video

```
gst-launch-1.0.exe -v udpsrc port=(指定的peer port) ! application/x-rtp, payload=96 !
rtpjitterbuffer ! rtph264depay ! video/x-h264,stream-format=byte-stream,alignment=nal !
h264parse ! avdec_h264 ! videoconvert ! fpsdisplaysink
```

2. ubuntu

   執行proxy於背景，另開新終端機執行下列指令，VM中無法擷取實體攝影機，因此僅能接收。

   - sub gstreamer h.264 video

```
gst-launch-1.0 -v udpsrc port=(指定的peer port) ! application/x-rtp, payload=96 !
rtpjitterbuffer ! rtph264depay ! video/x-h264,stream-format=byte-stream,alignment=nal !
h264parse ! avdec_h264 ! videoconvert ! fpsdisplaysink
```

3. Rpi

   執行proxy於背景，另開新終端機執行下列指令。

   - pub gstreamer h.264 video

```
gst-launch-1.0 -vv -e v4l2src device=/dev/video0 !
video/x-raw, format=YUY2, width=1280, height=720 ! videoconvert !
x264enc tune=zerolatency bitrate=700 speed-preset=superfast !
rtph264pay mtu=100 config-interval=1 ! udpsink host=127.0.0.1 port=(指定的local port)
```

   - sub gstreamer h.264 video

```
gst-launch-1.0 -v udpsrc port=(指定的peer port) ! application/x-rtp, payload=96 !
rtpjitterbuffer ! rtph264depay ! video/x-h264,stream-format=byte-stream,alignment=nal !
h264parse ! avdec_h264 ! videoconvert ! fpsdisplaysink
```