Design Challenge Report

Group 6 — Cyrille Vanhulle、吳昭霆、陳泓弦

專題介紹

此實驗中會有八台裝置，有二台裝置分別作為 source 節點和 destination 節點，其餘六台裝置則為 relay 節點，目標是在一任意網路拓樸中，將封包從 source 端傳送至 destination 端。

Source 端會持續傳送含有 sequence number 為 0x00 的封包，當封包成功到達 destination 端，便傳送含有編號 sequence number 為 0x01 的封包，基本的封包格式如圖(一)，依此類推直至 source 端收到含有 0x13 的 Ack 封包結束。Source 端的程式部份是由助教提供，另外要注意的是，當 sequence number 為 0x0E 的封包成功到達 destination 端時，助教會隨機關掉某一台 relay 節點 (仍確保至少會有一條從 source 端到 destination 端的路由)，我們在此實驗中必須完成 relay 節點和 destination 端的程式。

Byte-0	Byte-1	Byte-2	Byte-3	Byte-4
Header	Header	Sequence	isACK	Payload
0x05	0x30	0x00~0x13	0x00 or 0x01	*user-defined*

傳輸協定設計

這次的 Design challenge，我們有想到 1)建立 routing table 2)直接用 flooding，經過評估後，因為題目要求總傳輸時間愈快愈好，因此最後我們選擇

用 flooding 作為傳輸方式。我們會先簡單介紹我們建立 routing table 的想法，再介紹 flooding 的做法，最後是我們的實驗結果及心得。

一、建立 routing table

我們不希望產生 broadcast storm 的問題，因此一開始的想法是透過建立 routing table 的方式，以 unicast 的方式傳送資料，整個過程會分為二階段，建立路由階段和資料傳輸階段，詳細的做法如下，首先我們在 Payload 欄位裡放入 3 個 flag - Type，Source Address 和 Relay Address。Type 我們定義 3 類，分別是 Request，Reply 和 Data，當 relay node 收到封包時，會先檢查 Payload 裡的 Type 欄位，並執行相對應行動，詳細如下。

I. Data

若已知前往 destination 端的路由時，即會進入資料傳輸階段，開始單傳實驗中要求的封包到 source 端，若尚未知道路由，則會產生一 Request 封包，並將 source 端地址放入 Payload 廣播出去。

II. Request

若未知目的端路由，會把自己的 address 加入 Payload 裡的 relay address 欄位，並繼續廣播 Request 封包給 neighbors 詢問路由，若已知則回傳 Reply 封包給詢問者。

III. Reply

當節點收到 Reply 封包時，把資訊填入自己維護的 routing table

當 Request 封包經過 relay node 的時候，會將自己的 address 加在 Payload

裡的 Relay address 直至封包到達 destination 端，然後由 destination 端選擇較短的路徑(hop 數較少的路徑)，unicast 一份 Reply 封包給，當 relay 節點收到時，則會，如此建立出 route。若要處理傳輸到一半關掉 relay node 的情況，勢必要加入額外的確認封包來更新 routing table，因此我們先測試在没有關掉 relay node 的情況下，建立 routing table 的 performance，整個過程大約費時 10 多秒完成。

這方法有個問題是，當路徑太長時 Payload 會變得太大，此外，為了處理傳輸到一半將某一個 relay 節點關掉，則必須要更新原本維護的 routing table，那麼勢必要加入額外的控制封包導致總耗時更久。後來我們的考量是說，因為此次實驗僅使用八個裝置，規模不大，為了追求更短的傳輸時間，我們改使用 flooding 方式進行傳輸。

二、Flooding

我們的做法如下，將 relay 節和 destination 節點分開來處理。

Destination node：

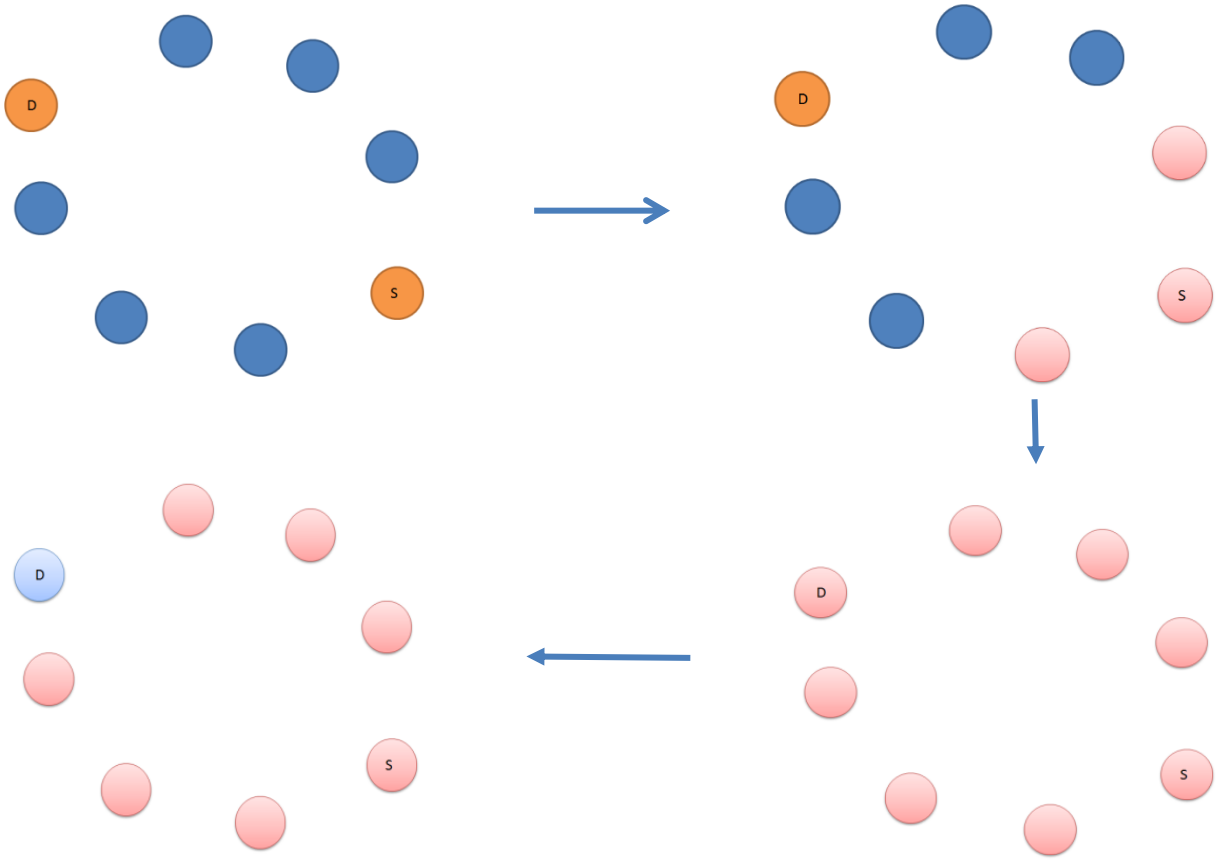
本身有一變數 seq 紀錄當前收到的最新 sequence，預設是 0。當收到的封包 isACK= 0 且 Sequence = seq 的時候，複製此封包並將 isACK 設為 1，廣播出去，自身的 seq +1。

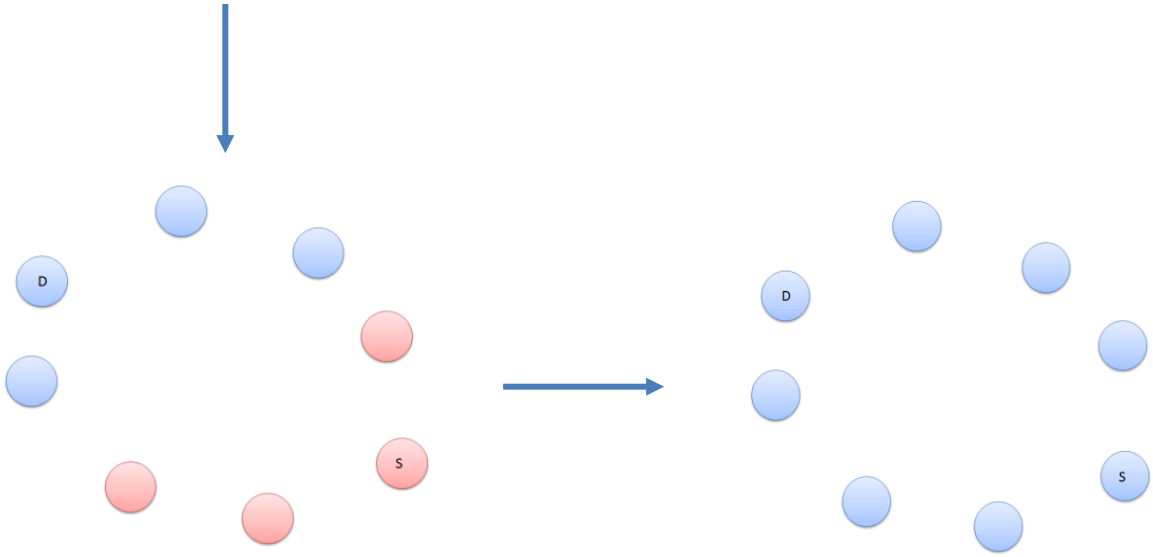
Relay node：

我們設立變數 status，初始值為 0，當有收到封包的時候把 status 改成 1，並且在 status = 1 的時候才會執行傳送封包動作。

節點本身有一個紀錄 sequence 值的 no 變數，預設為 0，只有在收到兩種情況，才會將封包記錄下來並且廣播出去。這兩種情況分別是：1) isACK = 0 且 sequence > no 2) isACK = 1 且 sequence >= no。 原因是，假設 source 端傳送 Sequence = 10 的封包, 但是有一 relay 節點的 no = 5(可能收訊不好漏掉之後的封包)，他就知道 source 跟 destination 已經進展到後面了，就更新 no 值並且加入他們。如此一來，relay 節點就不會理會 Sequence 比自己低的封包，避免網路中有太多不必要的封包在進行廣播。

以下是 flooding 想法的示意圖，將 destination、source 預設為橘色，relay 預設為藍色，顏色改變時表示正在傳輸，且同顏色代表傳播同 sequence 的封包。





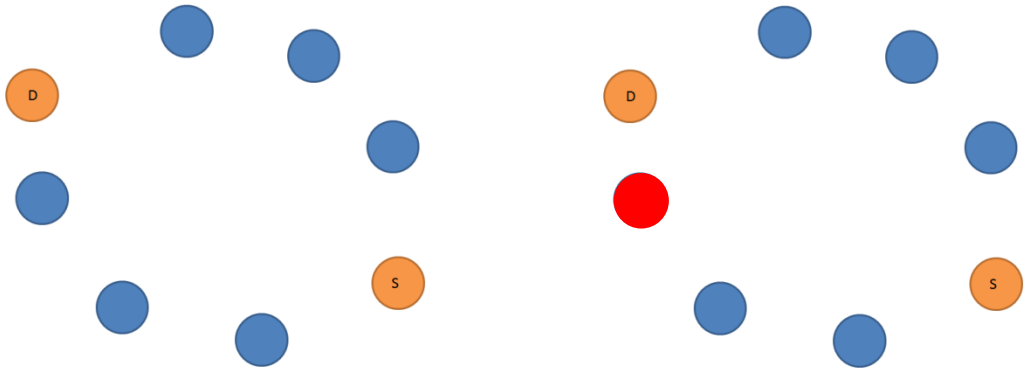
實驗結果

Demo 的結果如表：

	開始時間	結束時間	耗時
第一次	58.48.834	59.00.108	00.11.274
第二次	03.37.782	03.42.407	00.05.625

第一次實驗耗時 11 秒，第二次實驗我們在中途關掉其中一個節點，耗時僅不到 6 秒，造成兩次的差異最主要是中途少了一個節點，導致網路中廣播的封包減少了，大幅減少封包彼此碰撞而需要重傳的時間。

兩次的分布如下圖所示：



第一次　　第二次

雖然在 flooding 方法中用廣播的方式進行傳輸，有可能會使網路陷入廣播風暴，但從結果來看在小規模網路時，若目標是要求較快速的傳輸時間，flooding 是個可行的做法，傳輸時間甚至遠小於建 routing table 方法所需的 10 秒。雖然我們沒做在 routing table 方法中途拿掉一個節點的總傳輸時間，但可以合理推測，在中途拿掉節點勢必要傳輸更多的控制封包來更新 routing table，因此時間會遠大於 10 秒鐘。

專題心得

這次的專題讓我們把課堂所學到的理論方法實作出來，實際接觸硬體裝置的程式，讓我們了解如何透過 SDK 和裝置互動，實際硬體如何呼叫函式來接收、傳送封包，如何去取出封包的 Payload 並做出相對應的行為等，實在是受益良多。另外因為有位組員是法國人，我們是用英文進行團隊溝通，一開始其實討論不太順暢，透過不斷的溝通交流，慢慢的都能聽懂彼此想法，是一次很不錯的體驗。