1. 實驗日期:

2020/5/28

1. 實驗名稱:

Riverbed Modeler: TCP: Transmission Control Protocol

1. 實驗相關技術資訊:

* Explain the differences between fast retransmit (快速重傳)and fast recovery (快速恢復)in TCP.

**FAST RETRANSMIT**

接收端:

為某區段遺失，代表後續接收的區段順序錯誤，因此不一使用延遲確認(Delayed ACK)而是立即確認(immediate ACK)。目的是要讓傳素端知道接收端的是亂序區段以及預期接收的序列號。

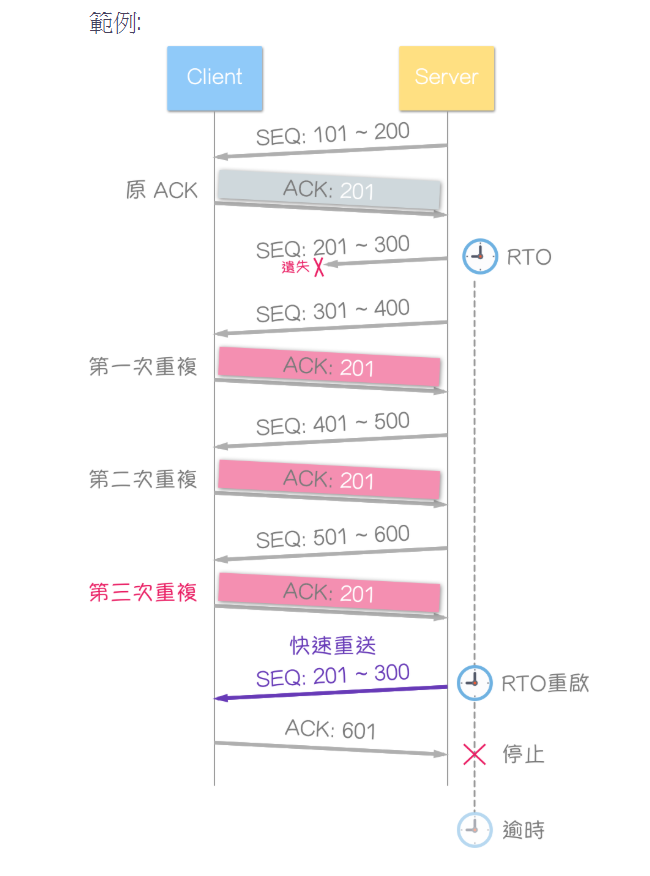
傳送端:

傳送端一開始並不清楚 「重複的確認」的原因 (這可能由多種網路問題)，

若收到 1 ~ 2 個重複確認，傳送端 會假設 是網路對區段的重新排序 或 複製 所引起，

而收到 3 個 或 更多的 重複確認，則明顯地指示 區段已遺失，

於是 TCP 立即重送 「似乎是」遺失的區段，而非等到計時器逾時



**FAST RECOVERY**

當快速重送送出「似乎是」遺失的區段後，TCP 進入 快速恢復 演算法階段，直到收到 非重複 的 確認 (ACK) 區段。不執行慢啟動(SLOW START)的原因是: 重複的確認區段，除了是區段遺失的跡象，往往也代表 該區段 已離開網路，因此，我們得知它已不再消耗網路資源。

快速重送/快速恢復 演算法 由以下幾點實現:

1. 傳送端 接收到 1 ~ 2 個 重複確認 (duplicate ACK) 時:

若 接收端 接收視窗 允許，可以 接續發送 先前未發送過的區段。

總 飛行大小 (FilghtSize) 應保持:

FilghtSize < = cwnd + 2 \* SMSS

且 在傳輸這些新區段時，不得更改 壅塞視窗 (cwnd)，

其實就是剛提及的 有限傳輸演算法 (Limited Transmit Algorithm)。

此外，使用 SACK [RFC 2018] 的 傳送端，

不得發送新數據，除非傳入的 重複確認 包含新的SACK信息。

2. 接收到 3 個 重複確認 (duplicate ACK) 時:

如上方 「區段遺失」所述，慢啟動門檻 (ssthresh) 不應超出:

ssthresh = max(FlightSize / 2, 2 \* SMSS)

若有使用 有限傳輸演算法 (Limited Transmit Algorithm)，

藉其傳輸的新數據，不得加入此計算中。

3. 傳送端 重送 尚未被確認的資料 時:

代表至少 3 個區段 (Segment) 已離開網路 且 被接收端緩衝，

TCP 進入 快速恢復 演算法階段，

直到收到 非重複 的 確認 (ACK) 區段。

此時，需將 壅塞視窗 設為:

cwnd = ssthresh + 3 \* SMSS

這也是 快速恢復 與 慢啟動 最大的差異之處!

其中 3 \* SMSS 的 3，

就是指 已離開網路 且被接收端緩衝的 3 個區段。

4. 若 傳送端 仍持續接收到 重複確認 (3 個以上):

TCP 繼續保持 快速恢復 階段，

且對每個 重複確認 (duplicate ACK)，以 1 \* SMSS 增加 壅塞視窗 (cwnd) 大小。

如此地擴充 壅塞視窗 (cwnd)，

是為了反映出 那些已離開網路的額外區段，

已不再消耗網路資源。

5. 當接收端 接收視窗 允許:

當先前未傳送的資料有效，

並且 接收端 接收視窗 (rwnd)，允許 傳送端 目前的 壅塞視窗 (cwnd)，

TCP 傳送端 應發送 1 \* SMSS 的未傳送資料。

6. 當「先前未被確認的資料」之 ACK 抵達:

TCP 必須將 壅塞視窗 (cwnd)，設為 慢啟動門檻 (ssthresh)，

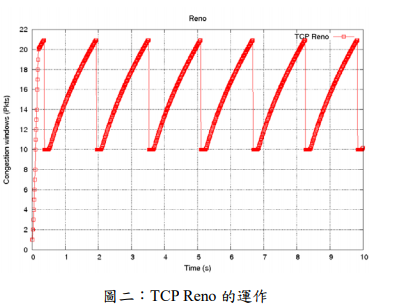
這稱為 (視窗的) — — deflating，即步驟 2 所述:

ssthresh = max (FlightSize / 2, 2 \* SMSS)

* Explain the differences between TCP Reno and TCP Vegas.

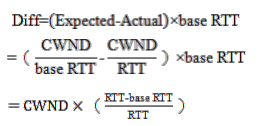
**TCP Reno**

基本上 TCP Reno 在慢啟動階段、壅塞避免階段和快速重傳機制這三階段的運作方式和 TCP Tahoe 的運作方式是一樣的。但是 TCP Reno 增加了一個新的機制為快速復原。

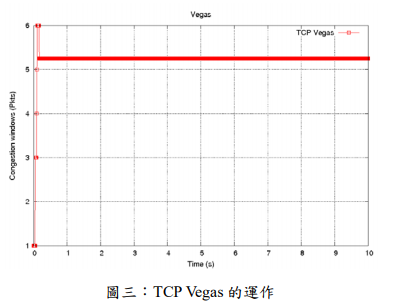
TCP Reno 在收到收到三次的 Duplicate ACK後除了重傳遺失封包並將 ssthresh值設成偵測到有封包遺失時，當下的視窗大小的一半。完成快速重傳機制後，會直接進入快速復原階段，直到接收端收到遺失封包的 ACK，才離開快速復原階段並直接進入壅塞避免階段，不再重新進入 Slow Start 階段。同樣藉由圖二來觀察 TCP Reno 的運作，在進入快速重傳機制之後 TCP Reno 會重送遺失的封包與從新設定 ssthresh 值，與 TCP Tahoe 不一樣的是 TCP Reno 在結束快速重傳機制後，就緊接的進入快速復原，從圖二中我們可以發現，壅塞視窗只下降到當下在快速重傳機制所設定的新 ssthresh 值，接下來就直接進入壅塞避免階段。

**TCP Vegas**

TCP Vegas 是一種依據 RTT 來調整壅塞視窗的大小與偵測網路壅塞狀況的 TCP 版本。TCP Vegas 藉由觀察每一次量測的 RTT 時間，藉由比較實際傳輸率和預期傳輸率並計算出兩者的差值 Diff，如公式（1），接著再搭配慢啟動階段、壅塞避免階段與快速重傳這三個機制來調整壅塞視窗、提高吞吐量與減少封包的遺失。TCP Vegas 的壅塞控制機制如下：

（1）

base RTT 為在整個傳輸過程中最小的 RTT 時間，該值通常是建立連線後第一個被發送的封包的 RTT 時間；RTT 為目前量測到的 RTT 時間，RTT 時間為封包從發送端發送出去與收到接收端發送此封包的 ACK 之間的時間，其中也包含了佇列延遲時間與其他開銷的時間；CWND 為壅塞視窗大小，即為我們發送的封包數；Expected 為期望的傳輸率；

Actual 為實際的傳輸率；Diff 為實際傳輸率和預期傳輸率兩者的差值，該值也表示目前網路佇列的長度與可用頻寬的依據。從圖三可以看到 TCP Vegas 的運作情形，從連線一開始在慢啟動階段就不斷增加壅塞視窗大小，接著結束緩啟動階段並進入壅塞避免階段。從圖中可以看出 TCP Vegas 的機制並沒有發生像 TCP Reno 的周期性網路壅塞的現象。並且 TCP Vegas 採用測量每一回的 RTT 的方式調整壅塞視窗大小，比起 TCP Reno 使用偵測封包遺失作為調整依據更能快速反應實際網路的狀況。

雖然 TCP Vegas 可以避免發生 TCP Reno 與 TCP Tahoe 的缺點，但將 TCP Vegas 應用在高 BDP 網路上仍會碰到一些問題： 第一，因為 RTT 值過大而造成壅塞視窗更新速度過慢導致無法利用到整個網路頻寬。第二，因為 RTT 值過大導致提早結束慢啟動階段，使得必須花費大量時間在壅塞避免階段才能達到對網路頻寬的完全利用。第三，壅塞視窗成長速度仍不足夠應付高 BDP 網路。

1. 參考資料:

<https://notfalse.net/28/tcp-congestion-control>

<https://people.dyu.edu.tw/paper/9607390_c.pdf>

<http://oplab.im.ntu.edu.tw/csimweb/system/application/views/files/IMP/20110191>