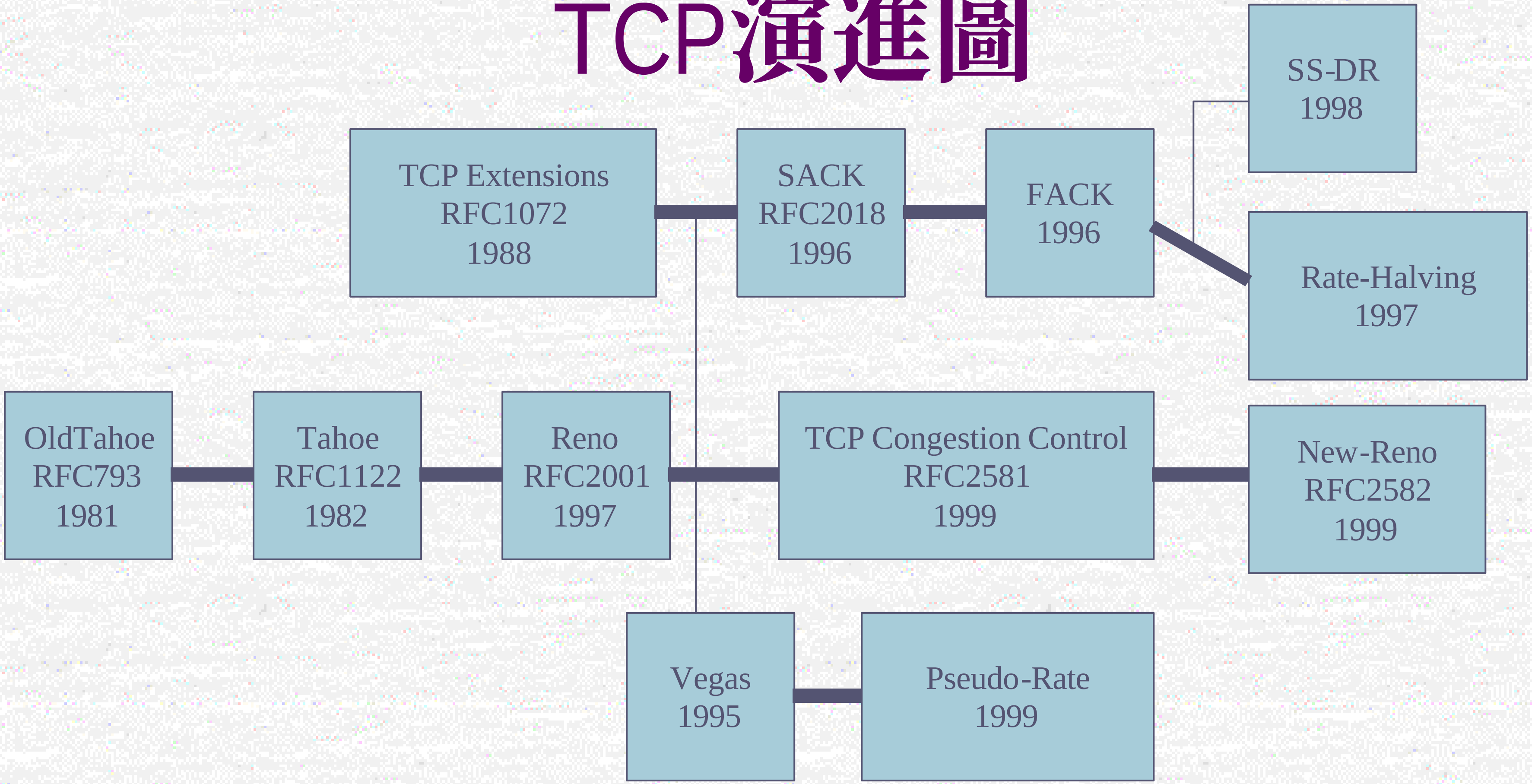
TCP congestion control

yhmiu

Outline

- Congestion control algorithms
- Purpose of RFC2581
- Purpose of RFC2582

TCP演進圖


SS-DR 1998
TCP Extensions RFC1072 1988
SACK RFC2018 1996
FACK 1996
Rate-Halving 1997
OldTahoe RFC793 1981
Tahoe RFC1122 1982
Reno RFC2001 1997
TCP Congestion Control RFC2581 1999
New-Reno RFC2582 1999
Vegas 1995
Pseudo-Rate 1999

TCP congestion control (Tahoe)

單位：byte

```
Initial:   cwnd = 1*segsize
           threshold = 64 KB
Loop:      if ( ACK received in time and cwnd <= ssthresh)
                cwnd += 1*segsize ;
           else if ( ACK received in time and cwnd > ssthresh)
                cwnd += segsize*segsize/cwnd + segsize/8 ;
           else if ( packet time out)
             {
                 ssthresh = cwnd/2 ;
                 cwnd = 1*segsize ;
                 time out 值加倍 ;
             }
```

Congestion control algorithms (RFC2001)

- Slow start
- Congestion avoidance
- Fast retransmit
- Fast recovery

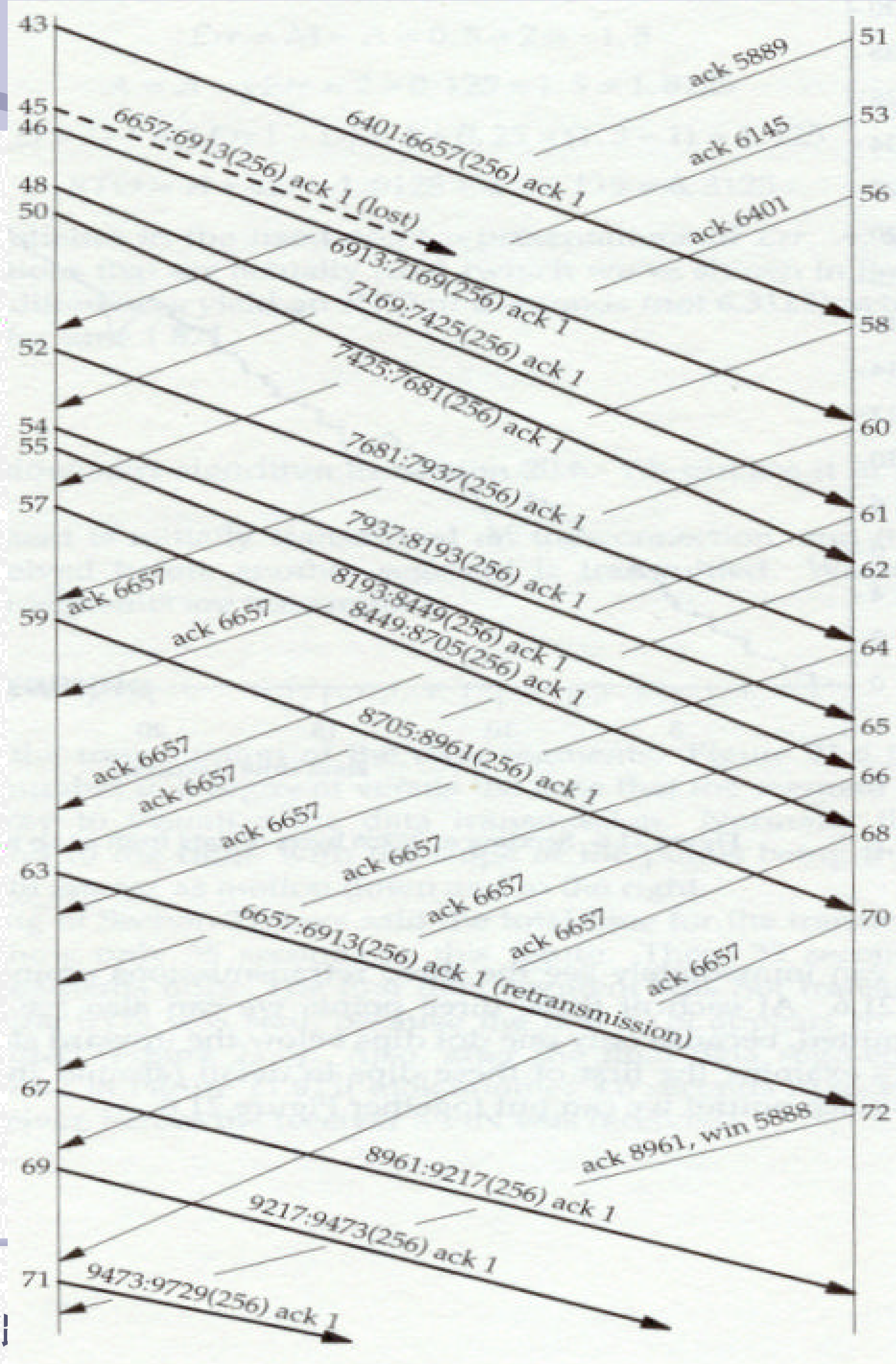
Fast retransmit and Fast recovery

- Using the number of duplicate acks receiving to decide lost or out of order
- After “Fast retransmit” algorithm sends the missing segment, “Fast recovery” algorithm governs the transmission of data until a non-duplicate ack arrives

The duplicate ack

- receiver SHOULD send an immediate duplicate ACK when an out-of-order segment arrives

Example of duplicate ack to detect lost



TCP congestion control(Reno)

單位：byte

```
Initial:    cwnd = 1*segsize
            ssthresh = 64 KB
Loop:       if ( ACK received in time and cwnd < = ssthresh)
                cwnd+= 1*segsize ;
             else if ( ACK received in time and cwnd> ssthresh)
                cwnd+= segsize * segsize / cwnd ;   /*  equation 2  */
             else if ( packet time out) {
                  ssthresh = cwnd / 2 ;
                  cwnd = 1*segsize ;
                  time out 值加倍                }
             while (3 duplicate ACK received)  {
                  ssthresh = max( cwnd / 2 , 2*segsize );
                  cwnd = ssthresh + 3 * segsize ;
                  if ( a duplicate ACK received )
                       cwnd += 1*segsize ;
                  else if (a non-duplicate ACK received)
                       { cwnd = ssthresh ;  break ; }
                  }
```

Purpose of RFC2581

- Let Reno become more general version
- 設定了某些變數允許的範圍，但不指定變數的值，以保持實作TCP時的彈性
- Additional considerations
  - Restarting idle connection to Slow Start
  - The delayed ACK algorithm can be used

New definition or suggestion of Reno implementation by RFC2581 (1)

- Initial cwnd絕不可大於2*segsize (bytes)
- Initial ssthresh的值可設任意大，反正發生congestion就一定會被降下來
- When cwnd=ssthresh,the sender can use either slow start or congestion avoidance
- During congestion avoidance,increment of cwnd is executed on incoming non-duplicate ACK

New definition or suggestion of Reno implementation by RFC2581 (2)

- cwnd given in equation 2 will fail to increase,if cwnd is vary large
  - Let (segsize*sgesize/cwnd) = 1byte
- Tahoe在equation 2的右式加上常數
  - actually lead to diminished performance
- During congestion avoidance,當被ack的byte數達到cwnd時,cwnd可一次增加一個segsize
- 以byte為cwnd的單位,equation 2較易計算

New definition or suggestion of Reno implementation by RFC2581 (3)

- When detect segment lost
  - ssthresh <= max (FS/2 , 2*segsize)
- When detect segment lost by timeout
  - Set cwnd = 1*segsize

Restarting idle connection

- When TCP has not received a segment for more than one retransmission timeout
  - To avoid TCP sending a cwnd-size line-rate burst into the network after an idle period.
  - Set IW=min(IW , cwnd) ; then go to slow start

The delayed ACK algorithm

- For at least every second segment generating an ack
- An ack must be generated within 500 ms of the arrival of the first unacknowledged packet
- MUST NOT generate more than one ACK for every incoming segment

Purpose of RFC2582

- 修改Reno版本的fast recovery來解決multiple-packet-loss的問題

example

- Case1 (single packet dropped from a window)
  - The ack for this packet will ack all of the packets transmitted before fast retransmit was entered
- Case2 (multiple packet dropped from a window)
  - The ack for the retransmitted packet will ack some but not all all of the packets transmitted before fast retransmit
- Case 2 called partial acknowledgement

Difference form Reno and NewReno under case 2

- Reno版本
  - When sender receives first partial ack, it transfers form fast recovery state to congest avoidance state
- NewReno版本
  - 其將partial ack當作packet遺失的指標，故其在multiple-packet-loss的情況下，每收到一個partial ack就可認定又有packet遺失了，一直到fast recover結束為止

NewReno

```
Initial : send_high= the initial send seqnum;
While (receive 3 duplicate ack) {
        if (duplicate ack cover no more than send_high)   break;
        else if (dupliacte ack cover more than send_high) {
              ssthresh = max(FS/2 , s*segsize);
              recover = highest seq num transmitted;
              cwnd = ssthresh + 3*segsize;
              if (receive a duplicate ack)
                  cwnd += 1*segsize;
              else if (receive a non-duplicate ack) {
                   if (this ack not ack up to recover) /* partial ack */ {
                        cwnd -= the amount of data acked;
                        reset restransmit timer;
                        if ( send a new segment allowed by window )
                            cwnd += 1*segsize;     }
                   else if (this ack ack up to recover)
                        break;
                   }
              else if (packet timeout)
                   send_high=highest seqnum transmitted ;   break;
     }
}
```

Problem

- Why to check send_high when entering Fast retransmit state
  - 避免已經fast retransmit後，又碰到retransmit timeout，而造成不必要的重傳
  - 當fast retransmit sate檢查duplicate ack不超過send_high時，皆將控制權交給retransmit timer去控制是否重傳

Problem

- Why to check recover when entering Fast retransmit state
  - 用來判斷是否發生multiple-packet-loss，而產生了partial ack

Reference

- RFC0793
- RFC1122
- RFC2001
- RFC2581
- RFC2582