資料中心網路中有效率的資料快取機制

廖文華
大同大學資訊經營系
whliao@ttu.edu.tw

蒯思齊 李威德
大同大學資訊經營系
SsuChiKuai@gmail.com

摘要

隨著科技的進步，網路的使用量不斷上升，社群網路或線上影音網站等需要取得大量特定資料的應用逐漸受到重視。內容中心網路(CCN)是一種新的網路通訊模式，設計來取代當前的網際網路。CCN 中的節點有高速快取(Cache)的作用，當資料封包經過節點，節點會把資料封包的內容快取下來，未來有其他使用者發送相同的興趣封包時，節點可以直接將快取的內容交給使用者。在這樣的環境下快取位置的選擇是相當重要的問題，選擇的不好可能會造成緩存的替換速度過快、查詢的效率不佳等問題。本篇論文針對 CCN 環境中的資料緩存方式提出 CW 演算法，用來增加資料查詢的效率以及避免不必要的快取花費。最後我們提出實驗證明我們的方法確實能夠有效的增加查詢效率與降低快取的成本。

關鍵詞：內容中心網路、資料快取、資料查詢

一、簡介

隨著網路技術越來越進步，加上各種科技產品的出現，用戶能夠更輕鬆、更快速地將自己所創建的內容分享出去，如訊息、影像、圖片或文件等，迅速地擴展人與人之間的溝通或經驗交流。這些使用者所產生出的內容不斷增加，當傳輸檔案時，使用者並不需要知道內容在那裡或如何得到，這改變了傳統使用網際網路(Internet)以 IP 為基礎的方式。

在資料快取部分，當今可擴展性和高效率的內容分發需求不斷增長，以此為動機，發展了資料名稱對象(Named Data Objects, NDOs)作為基礎的網路架構，此架構被稱為資訊中心網路(Information Centric Networking, ICN)。ICN 的目標是提供內容分發和流通性上更適合現今所使用的網路架構，與當今的網際網路模式相比，目前是以主機名稱為中心的通訊模式，例如網頁伺服器、個人電腦、筆記型電腦、手機和其他設備等; ICN 架構則充分利用網路中快取方式作儲存並透過複製來進行多方通訊，使發送者及接收者完成互動。

而在路由部分，網路上產生大量的訊息內容或資料。這些資料本身有其意義，同時可以被描述。而目前在路由的部分已經發展出不同於以往的方法，即基於網路內容名稱作為查詢和路由的一種網路模式，像是資料導向網路結構 (Data-Oriented Network Architecture, DONA)、發布/訂閱網際網路路由模式(Publish/Subscribe Internet Routing Paradigm, PSIRP)、資訊網路(Network of Information, NetInf)、和內容中心網路(Content Centric Networking, CCN)等。

CCN 是一種新的網路通訊模式，主要設計來取代當前的網際網路。與目前的 TCP/IP 模式相比，CCN 具有以下特點接收端中心通訊模式、分層式的內容命名方式以及高速快取和轉發體系架構。接收端中心通訊模式指的是客戶端要取得資訊透過發送一個興趣封包(Interest Packet)給離接收端最近的節點，若最近節點沒有所要的內容時，就會將興趣封包轉發(Forward)給其他的節點來取得資料。分層式的內容命名方式指的是 CCN 環境底下並不以特定的主機端紀錄位址(Address)，而是對內容對象本身做命名，內容的分層式命名類似於 URLs，接收端所發送的興趣封包會以內容名字的前置(Prefix)作為轉發決策階段時的匹配。高速快取和轉發體系架構指的是每一個 CCN 中的節點都有高速快取的作用，當資料封包經過節點，節點會把資料封包的內容快取下來，當未來有其他使用者發送相同的興趣封包時，節點可以直接將快取的內容交給使用者。

CCN 預期能夠解決現今網路的一些問題，如流動性、安全性和多路徑等。CCN 中的節點都有高速快取(Cache)的作用，當資料封包經過節點，節點會把資料封包的內容快取下來，當未來有其他使用者發送相同的興趣封包時，節點可以直接將快取的內容交給使用者。有效地利用 CCN 節點的特性，優化環境中資料的轉發行為，可以讓使用者在最短時間內取得資料。在這樣的環境下快取位置的選擇是相當重要的問題，存放的位置選擇的不好可能會造成緩存的替換速度過快、查詢的效率不佳以及更大的成本等問題。

本篇論文將針對資料快取的方式進行討論。本論文內容如下，第二章為文獻探討，第三節將制定論文中的系統架構。第四章提出 CCN 的資料快取方式。第五章進行實驗模擬與討論，第六章為結論。

二、文獻探討

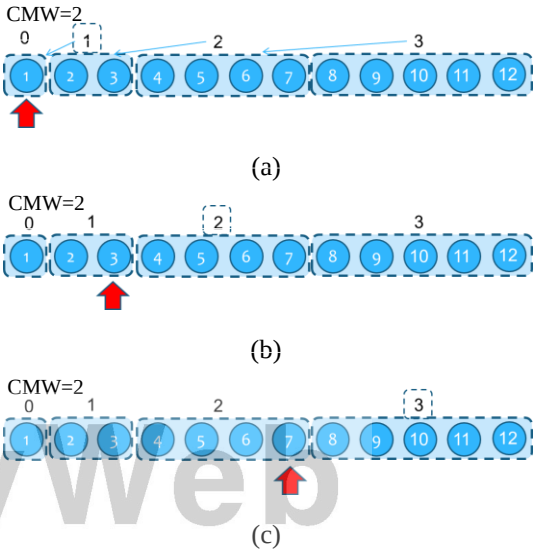
CCN 是一種新的網路通訊模式，被設計來取代當前的網際網路，與目前的 TCP/IP 模式相比，CCN 具有接收端中心通訊模式、分層式的內容命名方式以及高速快取和轉發體系架構幾項特點。

接收端中心通訊模式為接收端要取得資訊透過發送一個興趣封包，首先會傳送到離接收端最近的節點，若最近節點沒有所要的內容時，就會產生轉發行為，將興趣封包轉發給其他的節點來取得資料，在 CCN 環境下至多有一個資料封包(Data Packet)回覆給所發送興趣封包的接收端。分層式的內容命名方式代表 CCN 環境底下並不以特定的主機端紀錄位址，而是對內容對象本身做命名，內容的分層式命名類似於 URLs，接收端所發送的興趣封包會以內容名字的前置作為轉發決策階段時的匹配。高速快取和轉發體系架構則是每一個 CCN 中的節點都有高速快取(Cache)的作用，也就是說當有轉發的行為發生時，資料封包所經過的節點都會把資料封包的內容快取下來，當未來有其他使用者發送相同的興趣封包時，就能夠更快速地讓使用者接收到。

在過去的研究中，CCN 已有許多問題被探討，如文獻[2]中，當資料封包在回傳給使用者時，所有經過的節點都會將資料快取下來，當使用者接收到資料後，這些快取的節點發送通知給所連結的鄰居，告知鄰居節點自己所快取的資料，當有相同資料被請求時，能夠更快的將資料還傳給使用者。文獻[5]提出了當節點收到需求封包將資料回傳時，若是原始路徑中有節點損壞無法傳輸時避過損壞節點找到新的路徑，將資料回傳給需求者的方法。文獻[4]中，當路徑上流量堵塞時以螞蟻演算法找出其他流量較低的路徑，幫助使用者得到更好的服務品質。

在 CCN 環境中，每個節點都有高速快取的能力，透過每個節點的快取，能夠讓使用者更快取得所要求的資料。但為了避免資料快取在不必要的位置，付出更多成本或資料替換速度過快，快取位置的選擇是相當重要的問題。文獻[6][7][8]提出新的快取機制，改善原先若資料封包經過的節點都要將資料快取的方式，避免在 CCN 環境中，同時有許多點快取相同一筆資料的情形。文獻[6]中，作者提出資料高速快取位置及搜索策略(chunk caching location and searching scheme, CLS)，有快取過資料的節點都會記錄該資料的資料名稱、上一跳之交付者、交付目標以及目前與伺服器的距離(hop)。當使用者向 Server 發送請求，Server 接收到請求後將資料回傳給所連結的 CCN 節點，此時指定節點將資料快取，再把資料回傳給使用者。當有相同請求發送時，資料回傳後會將資料快取到下一個節點中，原先快取資料的節點則會記錄將資料傳送到哪個點上。因此，利用此模式，確保同一條路徑上只會有一筆相同的資料。但是當這樣的方式運用到樹狀結構時，會產生繞路的問題。因所有 CCN 節點都有容量限制，文獻[3]提出當有新資料需要快取但節點容量不足時，利用演算法選擇較不熱門的資料作替換。

文獻[1]中，融合資料存取的熱門度，作者提出一個漸進式的資料快取方式 WAVE。該作法將一筆資料分割成很多個 chunk，當使用者向 Server 要求資料時，Server 將把這筆資料由該路徑所要求的次數記錄下來，並且將該資訊夾帶在資料封包中並且回傳資料。路徑中的 CCN 節點根據該筆資料經過的次數決定要快取多少個 chunk 的資料在自己的記憶體內。被請求次數越多的資料將被快取越多部份在節點的記憶體中，同時也會越靠近使用者端。如圖一所示，當使用者向伺服器要求資料時，節點會根據上一步傳來的次數資料 $t=1$ 以及 CMW 值作為快取的依據，以第一次而言，其值為 $CMW^{t-1}=1$，代表快取的資料就是第一個 chunk，並且會將次數設定為 $t-1$ 轉發。當第

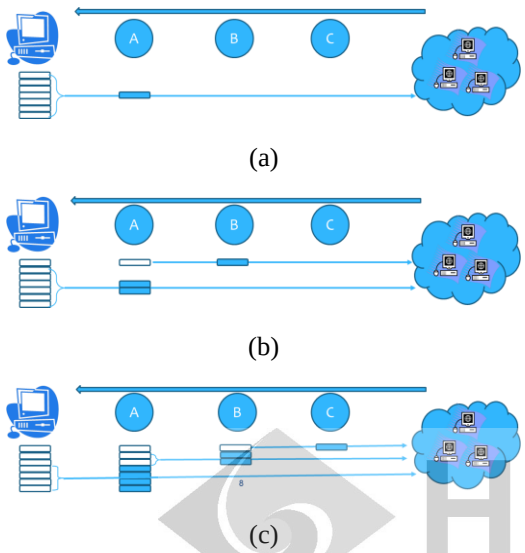


圖一：節點回傳資料機制。(a)使用者第一次請求資料時，cache 第一個 chunk；(b)使用者第二次請求資料，cache 第一至第三個 chunk；(c) 使用者第二次請求資料，cache 第一至第七個 chunk。

二次提出請求時，節點收到的 $t=2$ 計算出的值是 $CMW^{2-1}=2$ ，而暫存下的資料除了第一次快取的第一個 chunk 之外，需要另外再補上第 2、3 個。同理類推，當第三次請求時，節點將快取編號 1 至 7 的節點。這個方法的好處是能夠根據資料的熱門程度決定快取的數量以及位置，加速資料的取得同時避免掉多餘不必要的快取花費。但是該作法假設的路由環境太多單純，只考慮傳輸節點全部位於同一條路徑的狀況。當用在實際上的網路環境中時，該作法會出現資料遺漏的狀況。

三、系統架構

當網路環境中的使用者向 Server 要求資料時，Server 將一筆資料分割成 k 個 chunk，同時把這筆資料由該路徑所要求的次數記錄下來，並且將該資訊夾帶在資料封包中並且回傳資料。基本流程如圖二所示。圖二(a)中，當使用者第一次向伺服器要求資料時，伺服器會將所有的 chunk 都回傳給需求者同時夾帶次數資訊，而節點 A 收到資料時，可以得知這筆資料第一次被請求，因此會將資料中的第一個 chunk 快取，以便當再次有相同的請求發生時可以減輕伺服器的負擔。圖二(b)中，第二次提出請求時，節點 A 補快取第 2、3 個 chunk，而節點 B 處理的步驟如同節點 A 第一次請求時的步驟。而在圖二(c)中，我們可以發現在請求次數變多之後，路徑上的節點會將整份資料都快取下來，而在越接近需求點的位置快取的資料量越多。



圖二：使用者查詢與資料快取機制。(a)使用者第一次請求資料；(b)使用者第二次請求資料；(c) 使用者第二次請求資料。

(一) 資料快取模式

參照論文[1]提出的觀點，路徑中的 CCN 節點根據該筆資料經過的次數決定要快取多少個 chunk 的資料在自己的記憶體內。被請求次數越多的資料將被快取越多份，同時也會越靠近使用者端。這個方法的好處是能夠根據資料的熱門程度決定快取的數量以及位置，加速資料的取得同時避免掉多餘不必要的快取花費。伺服器在收到查詢時會記錄一個次數資料 r，並且將次數資料與所查詢的資料一起回傳。路徑上的所有節點會計算一個標記值，計算方式會根據該節點的資料來源是不是伺服器有所不同，如下列方程式所示。

$$cw=\begin{cases} r-1 & from\ server \\ cw-1 & else \end{cases} \tag{1}$$

這邊設定一個基本值(chunk marking window, CMW)標記為 x，當而每一個節點在資料第 cw 次被詢問時，節點將儲存的快取集合 CD 如下列方程式所示。

$$CD=\sum_{i=0}^{cw} x^i \tag{2}$$

(二) 資料回傳模式

當伺服器收到使用者的查詢要求時，會將要求的次數記錄下來，同時將請求次數與資料一併傳出。環境中的 CCN 節點會根據收到的次數資料判斷要將編號多少的 chunk 快取下來。在這個前提下，若同樣的檔案已經多次被請求，伺服器可以推論出該傳輸路徑上一定會有其他的節點已經將部分的 chunk 快取下來，如此一來伺服器只要回傳尚未被快取過的 chunk 即可，不足的部分由路徑上的 CCN 節點補足。如此一來可以減少網路環境中伺服器的負擔，同時增加使用者取得檔案的速度。我們設定節點收到的資料的請求次數為 r，由伺服器回傳的檔案集合 SD 如下列方程式所示。

$$SD=\sum_{i=x^{r-1}}^{k} Chunk(i) \tag{3}$$

由於目前的次數是 r，因此伺服器可以假設第 x^{r-1} 筆 chunk 之前的資料已經被路徑上的 CCN 節點所快取了，因此並不需要再傳一次。我們設定節點收到的資料的請求次數為 cw，需要由節點負責提供的資料 PD 如下列方程式所示。

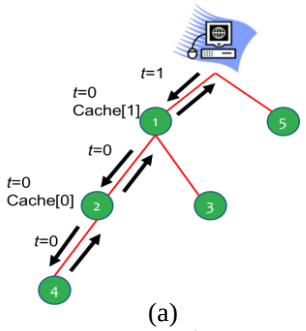
$$PD=\sum_{i=x^{cw-1}}^{x^{cw}} chunk(i) \tag{4}$$

當 CNN 節點傳輸資料時，除了轉發由上一步所取得的資料之外，也需要將資料補上。節點收到標示值為 cw，因此可以推論第 x^{cw-1}

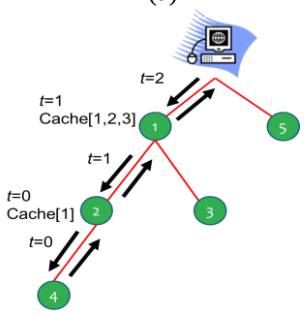
筆 chunk 之前的資料後續的路徑上會有其他節點快取下來，而上一步的節點已經假設第 x^{cw} 筆 chunk 之前的資料會由路徑提供，因此可以由此推知需要補上的 chunk 的總數。我們的目標是提出一個符合下列限制的資料快取與回傳方法。我們將環境設定為由 n 個節點與 m 個伺服器所形成的環境 $G(V,E)$，V 是的 n 與 m 的集合，E 是 n 與 m 或是 n 與 n 之間有建立網路連線之路徑的集合。查詢次數越多的資料，伺服器的負擔越小。到達使用者之 chunk 數量必須等於原本的數量 k。節點的快取量要最小化。

四、啟發式演算法

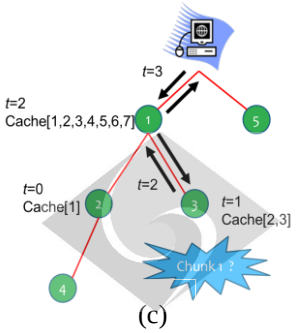
參照論文[1]，當使用者提出需求之後，Server 將一筆資料分割成 k 個 chunk，同時把這筆資料由該路徑所要求的次數記錄下來，並且將該資訊夾帶在資料封包中並且回傳資料。



(a)



(b)



(c)

圖三：使用者要求資料遺漏示意圖。(a)第一位使用者位於節點 4 請求資料；(b)第二位使用者位於節點 4 請求資料；(c)第三位使用者位於節點 3 請求資料，同時發生了遺漏編號 1 的 chunk。

位於路徑上的節點收到這個資訊的時候會先判斷需求次數是否為 0，若為 0 則不做任何處理而直接將資料代傳傳出。而若需求次數不為零，節點將將需求次數減 1，同時依據上述之資料快取方式將資料儲存於自己的記憶體之中之後再將資料送出。所有節點將一一執行這個步驟直到資料到達提出需求的使用者為止。

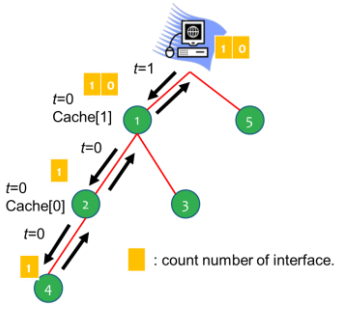
但是真實的網路環境中，路由狀況非常複雜。這樣的做法只考慮所有的節點位於同一路徑上的狀況，並未考慮當兩條以上的路徑同時由某個節點通往伺服器時會發生的狀況。如圖三所示，路徑第一位與第二位的使用者經由 4、2、1 像伺服器請求資料，第一次時伺服器將需求次數標示為 1，並且將封包傳出，來到節點 1 時由於需求次數不為 0，因此 1 節點根據上述之快取模式將 chunk 快取至編號 1，在將需求次數減 1 後送出。而節點 2 與 4 則因需求次數為 0 因此不做快取直接將資料轉送。當使用者第二次送出第二次需求時的步驟同理類推，此時節點 1 中已快取了編號 1~3 的 chunk。而當第三位使用者由節點 3 送出請求時，對於伺服器而言這是第三次的請求，因此將需求編號標為 3 並且推測編號 1~3 的資料於該路徑上已有節點將其快取下來，因此只將編號 4 以後之 chunk 送出，此時節點 1 收到後將需求次數減一後，推測編號 1 之 chunk 在路徑上已經有其他節點快取，因此將編號 2 與 3 之 chunk 補上，快取了編號 4~7 的 chunk。但到達節點 3 的時候，由於對於節點 3 這是第一次需求，因此並沒有標號 1 的 chunk，因此使用者發生了資料短缺的狀況。

表一：CW 快取演算法。

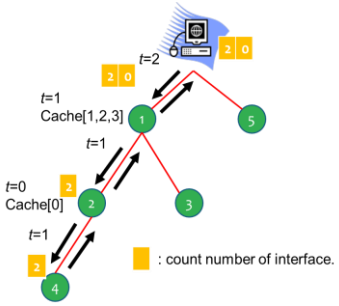
Algorithm 1: CW router cache algorithm	
Input:	x: chunk marking window (CMW) base. cw: chunk marking window (CMW) state. t: total number of data chunks. *Input_Chunkset:* Chunk set received from last hop router. *Input_Check(k):* a set of request receive times of interface k.
Output:	*S(TD,cw)*: output set of data chunk and chunk state. *CD*: cached data set by router.
1.	**if** $cw>0$ **then**
2.	Marked chunk $mc=\sum_{j=0}^{cw} x^j$
3.	$CD=\sum_{i=0}^{mc} Chunk(i)$ $TD = Input\_Chunkset +$
4.	$\sum_{i=x^{Input\_Check(k)}}^{x^{cw}} Chunk(i)$
5.	$cw = Input\_Check(k)-1$
6.	**else**
7.	$CD=\phi$
8.	$TD = Input\_Chunkset$
9.	cw = 0
10.	**end if**

為了解決這樣的問題，我們提出了 CW 演算法。我們利用紀錄需求由哪一條路徑近來的方式來避免上述的問題。我們的演算法詳細的步驟如表一所示。

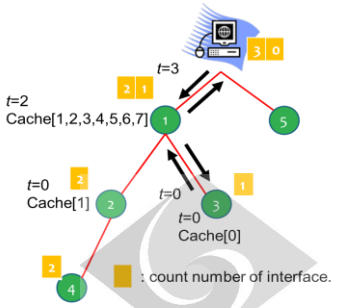
以下我們將提出一個範例說明我們的演算法。如圖四(a)所示，當第一位使用者由節點 4 請求資料時，節點將其記錄後將資料傳給伺服器。此時對於伺服器而言請求次數是第一次，因此將次數 1 連同資料一起傳出。當節點 1 收到這筆資料時，節點 1 會將伺服器回報的次數與自己計算的次數進行比對。由於此時節點紀錄的數據與伺服器夾帶的資料相同，因此不做額外處理，將編號 1 的 chunk 快取同時將次數減 1 並且連同資料往下傳送。編號 2 與編號 4 的節點由於收到資料時的次數是 0，因此不做處理直接將資料轉送出去。圖四(b)中，第



(a)



(b)



(c)

圖四：CW 演算法。(a)第一位使用者位於節點 4 請求資料；(b)第二位使用者位於節點 4 請求資料；(c)第三位使用者位於節點 3 請求資料。

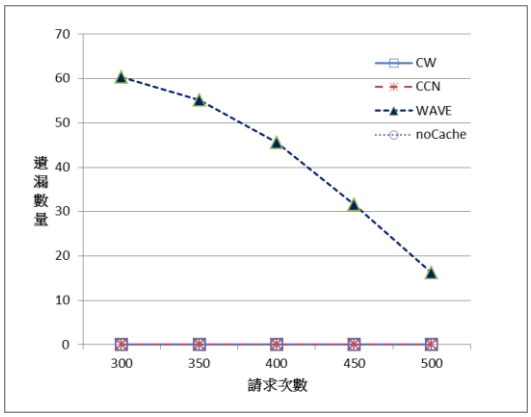
二位使用者同樣由節點 4 請求資料，伺服器此時判斷這是第二次要求了，因此伺服器可以假設第一次要求時已經有節點會將編號 1 的 chunk 做快取，在傳輸資料時只需要將編號 1 的 chunk 以外的部分傳出即可，因此伺服器會將次數標記為 2 同時連同編號 1 以外的其他 chunk 一起傳出。到節點 1 時，節點取得的次數是二，根據方程式(1)因此將編號 1、2、3 的 chunk 都快取下來後。同時節點可以利用次數資訊推算出伺服器傳過來的資料中並沒有編號 1 的 chunk，因此將次數減 1 後另外需要將標號 1 的 chunk 補上之後再連同伺服器傳過來的資料一起轉送。而節點二由於收到的時候次數是一，同理類推會將編號 1 的 chunk 快取，之後將次數減一並且轉送出去。圖四(c)中，第三位使用者在節點 3 請求資料，此時節點 1 會多出一筆由另一路徑提出請求的紀錄。等同於第一位與第二位使用者請求時的判斷，伺服器會將編號 1、2、3 以外的 chunk 連同次數 3 一起送出。而當資料來到節點 1 時，由於伺服器傳下來的次數為 3，但是節點 1 要轉發出去的路徑其實是第一次，因此節點 1 並不能夠直接由伺服器發送的次數來推估自己應該補上的 chunk 編號，而是改由伺服器送出的次數與轉發路徑的請求次數之間的差距來決定。因此，節點 1 將伺服器送來的次數忽略，改參照路徑的請求次數標記，本範例而言請求次數為 1，因此節點 1 需要將編號 1、2、3 的 chunk 都補上之後再將資料傳出。

五、實驗分析

本節中，我們將對提出的 CW 演算法進行評估。我們主要的比較對象是論文[1]中提出的 WAVE 演算法。實驗部分將針對網路效能與快取表現兩個方面進行比較。為了模擬我們的方法，我們在環境中放置了 5 個伺服器，環境中有 150 個 CCN 節點分別與這些伺服器經由樹狀結構進行連線。我們的實驗總共進行的次數為 300~500 次。

(一) 封包遺失量

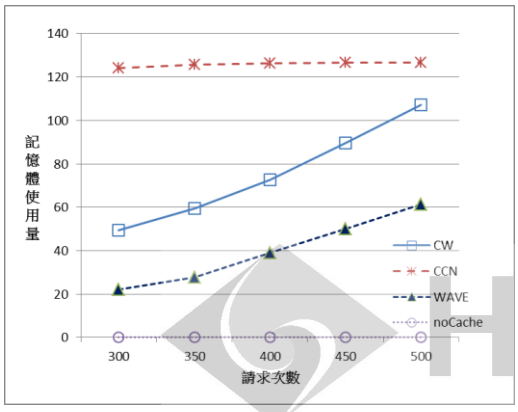
首先我們比較的是封包遺失的狀況。當傳統不使用 Cache 的方法由於當路徑上沒有找到資料時，需求就算送到伺服器，伺服器會將完整的檔案經由原路徑傳回使用者端，其中並不會發生遺漏。CCN 方法改良的傳統的方法，當產生需求時，路徑上的節點會將資料 Cache 一份下來，而下次當同樣的需求經由該節點要傳送至伺服器時，該節點會將會代替伺服器將資料傳給請求端以減少伺服器的負擔，在這樣的情況下，依然不會產生封包遺失的狀況。而



圖五：封包遺失狀況。

WAVE 的方法，將一筆資料分割成很多個 chunk，當使用者向 Server 要求資料時，Server 將把這筆資料由該路徑所要求的次數記錄下來，並且將該資訊夾帶在資料封包中並且回傳資料。但這個方法並沒有考慮到當樹狀的結構發生時，由於資料的請求來自不同的路徑，因此位於路徑頂端的節點會根據資料被請求的次數假設其路徑上的節點已經有 Cache 的資料，因此傳輸較少的 Chunk，讓路徑上的節點能將缺少的檔案補上，但是事實上路徑並沒有 Cache 任何資料在記憶體中，因此在使用者端會出現檔案缺少的情況。而我們提出的 CW 方法中，節點會記錄請求是由哪個介面所請求的，因此我們的方法能夠正確地傳輸完整的資料回到請求端而不會產生遺漏的狀況。當 Chun 遺失的狀況越少，代表需要重新向伺服器詢問並且取得資料的次數越少，這代表使用者將在更短的延遲內取得資料，同時代表網路整體的傳輸負擔越小。

(二) 記憶體使用量



圖六：記憶體使用量。

傳統不使用 Cache 的作法中，需求端需要資料的時候就向伺服器提出要求。伺服器收到需求之後就經由網路直接將資料回傳給需求端，這之中不會有 Cache 的動作而環境中的節點並不會消耗任何的儲存空間，但是如此一來網路負擔與伺服器的負擔會很重。而 CCN 的方法中，伺服器收到需求之後就會將資料沿著原路徑傳回，同時路徑上的節點會將該資料 Cache 下來，當第二次以後的請求出現，若是路徑上有經過有 Cache 資料的感測節點，該節點可以代替伺服器回傳資料，因此可以經由較短的路徑取得資料。但是由於當請求發生時，資料就會直接在路經中多一份 Cache，如此一來記憶體將會使用的很快，也很容易出現 Cache 下來的其實是指被請求過一次的資料，造成記憶體的浪費。在 WAVE 的方法中，會發生資料遺漏的狀況，因此會發生原本某些節點該 Cache 的資料事實上並沒有被 Cache 下來的狀況。而我們的 CW 演算法，考量到該筆資料被請求的次數來進行 Cache，被請求次數越多的會 Cache 越多部分在路徑上，同時不會因為資料遺漏而產生原本應該要 Cache 的資料沒有被 Cache 的狀況。

(三) 網路流量負擔

傳統不使用 Cache 的作法中，因為每次需求端出現需求都會直接向伺服器提出需求，因此網路中很容易因為伺服器會將完整的檔案經由網路傳給使用者而出現大量的流量負擔。在 CCN 的方法中，由於資料提出請求之後，需求端提出需求之後有可能還沒有到達伺服器就由路徑上的節點將完整的資料交給需求端，因此網路中的負擔最為輕微。而在 WAVE 演算法中，由於環境中的節點 Cache 時並不會將資料完整的 Cache 下來，而是將一部分的 Chunk 進行 Cache，因此雖然在記憶體的使用效率上會比較優秀，然而與 CCN 的方法比較起來，網路的負擔將較為嚴重。但是，因為 WAVE 的方法中會出現資料遺失的狀況，因此在實驗中，WAVE 演算法由於會出現資料遺失的狀況，因此會產生再次請求。也因此，在同樣利用 chunk 機制的 CW 演算法中，由於能夠正確的傳輸完整的檔案，因此網路負擔會

圖七：網路流量負擔。

比 WAVE 演算法要低。

六、結論

CCN 中的節點會把資料封包的內容快取下來，當未來有其他使用者發送相同的興趣封包時，節點可以直接將快取的內容交給使用者。在這樣的環境下快取位置的選擇是相當重要的問題，存放的位置選擇的不好可能會造成緩存的替換速度過快、查詢的效率不佳以及更大的成本等問題。過去部分做法中，大部分並沒有考慮資料的熱門度，越熱門的資料應該要距離使用者越近，如此才能夠降低使用者取得的延遲，同時降低網路的傳輸負擔。而部分有考慮到熱門度的做法，對於樹狀的環境並沒有加以討論。本篇論文針對 CCN 環境中的資料緩存方式提出 CW 演算法，用來增加資料查詢的效率以及避免不必要的快取花費，同時依據熱門度讓資料緩存的效率更佳，不致浪費節點儲存空間。最後我們提出實驗證明我們的方法確實能夠有效的增加查詢效率與降低快取的成本。

致謝

本研究承科技部與大同大學補助，計畫編號：NSC102-2221-E-036-012 及大同大學 B103-N05-037，特此致謝。

參考文獻

[1] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "WAVE: Popularity-based and collaborative in-network caching for content-oriented networks," *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2012.*

[2] R. Ishiyama, K. Tsukamoto, Y. Koizumi, and H. Ohsaki, "On the Effectiveness of Diffusive Content Caching in Content-Centric Networking," *IEICE Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT)*, 2012.

[3] S. J. Kang, S. W. Lee, and Y. B. Ko, "A Recent Popularity Based Dynamic Cache Management for Content Centric Networking," *IEEE International Conference on Ubiquitous and Future Networks (ICUFN)*, 2012.

[4] A. Z. Khan, S. Baqai, and F. R. Dogar, "QoS Aware Path Selection in Content Centric Networks," *IEEE International Conference on Communications (ICC)*, 2012.

[5] Y. Kim, J. An, and Y. H. Lee, "CCNFRR: Fast one-hop Re-Route in CCN," *IEEE International Conference on Communications (ICC)*, 2012.

[6] Y. Li, T. Lin, H. Tang, and P. Sun, "A Chunk Caching Location and Searching Scheme in Content Centric Networking," *IEEE International Conference on Communications (ICC)*, 2012.

[7] J. M. Wang and B. Bensaou, "Improving Content-centric Networks Performance with Progressive, diversity-load driven Caching," *IEEE International Conference on Communications in China (ICCC)*, 2012.

[8] J. M. Wang and B. Bensaou, "Progressive Caching in CCN," *IEEE Global Communications Conference (GLOBECOM)*, 2012.