Network Programming: Ch. 22: Advanced UDP Sockets

Li-Hsing Yen
NYCU
Ver. 1.0.0

Advanced UDP Sockets

- Receiving Flags, Destination IP Address, and Interface Index
- Datagram Truncation
- When to Use UDP instead of TCP
- Adding Reliability to a UDP Application
- Binding Interface Addresses
- Concurrent UDP Servers

新函數recvfrom_flags

- 第八章介紹的recvfrom函數接收UDP datagram時無法得到下列資訊
  - Destination IP address
  - 收到datagram的介面index
  - datagram是否為broadcast或multicast等
- 使用第14章介紹的recvmsg函數可以得到上述資訊，但使用上太複雜
- 作者因此定義recvfrom_flags函數取代recvfrom，可得到上述資訊

recvfrom_flags的定義

```
#include <unp.h>
ssize_t
recvfrom_flags (int fd, void *ptr, size_t nbytes, int *flagsp,
        SA *sa, socklen_t *salenptr, struct in_pktinfo *pktp);
```

同 recvfrom

新參數

同 recvfrom

新定義

return: number of bytes read or written if OK, -1 on error

*flagsp即是呼叫recvmsg的msg_flag傳回值(位於struct msghdr中)。由此值可得知datagram是否為multicast, broadcast等。

```
struct in_pktinfo {
   struct in_addr  ipi_addr;      目地端位址
   int             ipi_ifindex;   接收介面
};
```

recvfrom_flags的實作

- recvfrom_flags事實上是間接呼叫recvmsg得到所需資訊
  - *flagsp即是呼叫recvmsg時，參數struct msghdr中的msg_flag傳回值。
  - struct in_pktinfo是由struct msghdr中的ancillary data中抓出IP_RECVDSTADDR與IP_RECVIF兩個data objects得到的
- 課本程式碼很複雜原因是要處理不同作業系統版本間的差異

Datagram Truncation

- 當應用程式所準備的buffer不夠存放要讀取的UDP datagram時，在msghdr結構中傳回的msg_flag會設定MSG_TRUNC
- 不同系統對此事件有不同的處理方式
  - 有些會通知應用程式，有些不會；有些會丟棄超出部份的資料，有些可在下次讀取
- 最簡單方式：永遠準備夠大的buffer

When to Use UDP Instead of TCP?

- UDP的優點(與TCP相較)
  - 支援broadcasting與multicasting
  - No connection setup or teardown
  - lower minimum transaction time

TCP提供的特性

- 應用程式不見得需要用到下列TCP所能提供的特性
  - Positive acknowledgement, retransmission of lost packets, duplicated detection, and sequencing of packets reordered by the network
  - Windowed flow control
  - Slow start and congestion avoidance

建議

- UDP must be used for broadcast or multicast applications
  - 因為TCP不支援
- UDP can be used for simple request-reply applications but error detection must then be built into the application
  - 資料量少不須flow control和congestion avoidance
- UDP should not be used for bulk data transfer
  - 改用TCP才能符合需求

Adding Reliability to a UDP Application

- We are going to use UDP for a request-reply application
- We must add two features to our client
  - Timeout and retransmission to handle datagrams that are discarded
  - Sequence numbers so that the client can verify that a reply is for the appropriate request

UDP應用程式要處理的細節

- Adding sequence numbers is simple
- Handling timeout and retransmission
  - Send a request and wait N seconds
  - If no reply was received, retransmit and wait another N seconds
  - After this has happened some number of times, give up
- This is a linear retransmit timer

Linear Retransmit Timer的問題

- Datagram在Internet中來回一次所需時間不定(WAN與LAN所需時間差異甚大)
- 所以Timeout時間應考慮真正測得的RTT與RTT在一段時間內的變化
- 我們為每一個送出的packet計算它的retransmission timeout (RTO) 值
- 可採用Jacobson的方法(下頁)

Jacobson 計算 RTO 值的方法

變數定義

RTO: retransmission timeout
measuredRTT: the actual round-trip time for a packet
srtt: the smoothed RTT estimator
rttvar: the smoothed mean deviation estimator

計算方法

$$delta = measuredRTT - srtt$$
$$srtt = srtt + g \times delta$$
$$rttvar = rttvar + h(|delta| - rttvar)$$
$$RTO = srtt + 4 \times rttvar$$

g = 1/8 is the gain applied to the RTT estimator

h = 1/4 is the gain applied to the mean deviation estimator

RTO 值計算方法範例

原先

srtt: 2.80
rttvar: 0.60

$\downarrow$ *measuredRTT*: 3.2

delta: 0.40
srtt: $2.80 + 1/8 \times 0.40 = 2.85$
rttvar $= 0.60 + ¼ \times (0.40 - 0.60) = 0.55$
$\text{RTO} = 2.85 + 4 \times 0.55 = 5.05$

delta = *measuredRTT* - *srtt*
srtt = *srtt* + *g* $\times$ *delta*
rttvar = *rttvar* + *h*(|*delta*| – *rttvar*)
RTO = *srtt* + 4 $\times$ *rttvar*

When RTO Expires

- An *exponential backoff* must be used for the next RTO (doubling the RTO)
- Actually three possible scenarios are
  - The request is lost, or
  - The reply is lost, or
  - The RTO is too small
- 如果我們重送request後再收到reply，無法分辨是屬於上列何種情形，要如何計算RTT值?

可能是第一或第二個reply

Karn’s Algorihm

- Applies whenever a reply is received for a request that was retransmitted
- 決定此種狀況下RTT與RTO的計算
- 如果RTT有在算，不要用它來更新*srtt*及*rttvar*
- 如某個重送的request的reply在time out前收到，下個packet仍延用目前的RTO值
- 只有當我們收到某個未重送過的request的reply時，才會更新*srtt*與*rttvar*並重算RTO值

RFC 1323

- 比Karn’s algorithm更簡單的做法
- 對每一個發出的request貼上送出當時的時間戳記(timestamp)，server回應時要將收到的request的時間戳記附上送回
- 對每一個收到的reply我們都可以正確計算RTT值
- Server與client的時間不必同步

22.6 Binding Interface Addresses

- IP_RECVDSTADDR socket option 可讓我們得知收到的 UDP datagram 的目的位址
- 本節介紹另一方法，用第17章介紹的get_ifi_info函數抓出所有介面的IPv4位址(含aliases)，每一個位址bind到一個socket，並為每個位址fork一個child process
- 每個child process收到的datagram都是指定送給該child所屬的某個特定IP位址的

22.7 Concurrent UDP Servers

- Most UDP servers are iterative
  - 依序處理收到的UDP datagrams
- A concurrent UDP server
  - 可以平行處理來自不同client的請求
- concurrent TCP server很容易實作是因為每一個TCP connection都有唯一的socket pair
  - 每一個connection可交由一個子程序負責

Two Different Types of UDP Servers

1. A simple UDP sever that reads a client request, sends a reply, and is then finished
   - 不必設計成concurrent UDP server
2. A UDP server that exchanges multiple datagrams with the client (e.g., TFTP)
   - server要如何分辨某個client送出的第一筆request與後續的datagrams呢?

$\Rightarrow$ 後續的datagrams使用不同的port

A Concurrent UDP Server