Problem 1

An HTTP server can set a cookie in a browser through the Set-Cookie header. Assuming the following HTTP response was returned for request to `https://www.example.com/hello.html`

```
HTTP/1.0 200 OK
Set-Cookie: A=a; path=/us, Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure; HttpOnly
Set-Cookie: B=b; domain=example.com; path=/, Expires=Wed, 13 Jan 2001 22:23:01 GMT; Secure;
    HttpOnly
Set-Cookie: C=c; domain=example.com; path=/us, Expires=Wed, 13 Jan 2021 22:23:01 GMT; Secure;
    HttpOnly
Set-Cookie: D=d; domain=test.example.com; path=/, Expires=Wed, 13 Jan 2022 22:23:01 GMT
Set-Cookie: D=d; domain=hello.www.example.com; path=/, Expires=Wed, 13 Jan 2022 22:23:01 GMT
```

A cookie will be sent on all subsequent connections (until it expires) to servers for which `Domain` is a suffix. If the `Domain` attribute is omitted, most browsers will send the cookie only to the host that set it. The cookie name-value pair is sent by the browser unaccompanied by any other attributes such as `Domain`.

A server is allowed to set the `Domain` attribute of a Set-Cookie header, provided that (1) the cookie's domain is the same as, or a parent of, the origin domain and (2) the cookie's domain is not a TLD, a public suffix, or a parent of a public suffix. As an example, `x.y.z.com` can set a cookie domain to itself or parents: `x.y.z.com`, `y.z.com`, `z.com`. But not `com`, which is a public suffix. Examples of public suffixes: `com`, `edu`, `uk`, `co.uk`, `blogspot.com`, `compute.amazonaws.com` (see full list at `https://publicsuffix.org/`).

If the `Set-Cookie` header specifies the `Secure` attribute for a cookie, that cookie will only be sent from the browser to the server over the `https:` scheme, never over the `http:` scheme.

Fill in blank fields in the subsequent HTTP requests (assume it is over HTTPS)?

1.
```
GET /us/faq.html HTTP/1.0
Host: example.com
Cookie: C=c
```

2.
```
GET /us/faq.html HTTP/1.0
Host: www.example.com
Cookie: A=a; C=c
```

3.
```
GET /us/cart.html HTTP/1.0
Host: test.example.com
Cookie: C=c
```

4.
```
GET /us/about.html HTTP/1.0
Host: hello.www.example.com
Cookie: C=c; D=d
```

5.
```
GET us/cart.html HTTP/1.0
Host: www2.example.com
Cookie: C=c
```

ASSUMPTIONS: www.example.com can set cookies for parent (example.com, as stated) and for subdomains (as specified in Wikipedia).
NOTES: B=b cookie is expired, so it's not sent. www.example.com cannot set cookies for it's neighbors so the first D=d cookies is invalid.

Problem 2

For each of the questions below, describe answer in terms of low-level packet sequences, drops, or network-level packet reordering.

1. A specific case where HTTP/1.1 wins in performance compared to HTTP/1.0
2. A specific case where HTTP/2 wins in performance compared to HTTP/1.1
3. A specific case where QUIC wins in performance compared to HTTP/2

1. A client requests an HTML file from a web server, which references 10 images. With HTTP/1.0, the client must first request the HTML file, then the 10 images that it contains one at a time, with each request on a separate connection. This incurs the cost of establishing a new connection each time. HTTP/1.1 makes this more efficient in two ways: by maintaining a persistent connection with the client and allowing the client to pipeline requests. The client will request the HTML, parse it, realize it needs 10 images, and request all 10 images back-to-back over the same initial connection. The server will respond with back to back responses containing each image. This avoids us having to wait an request/response round trip for each image and new connection, as HTTP/1.0 would have us do.
2. HTTP/2 wins, for example, when making/receiving requests with a lot of associated headers. HTTP/2 will compress headers into a binary format to make network transfer more efficient. In addition to this, HTTP/2 also has a server-push feature; e.g. in the previous example, the server could preemptively send the HTML file and its 10 associated images to the client, without waiting for the client to parse the HTML and request the images first.
3. QUIC wins over HTTP/2 in two ways: no head-of-line blocking because QUIC is built on top of UDP, and it's not always necessary to wait an RTT to establish a connection in QUIC like it is in TCP (which HTTP depends on). Instead, each side of the connection will store some state about the handshake, and in the best case the server will remember you and reconnections incur no overhead cost. QUIC wins by at least 1 RTT over HTTP in any case where you've visited the server recently.

Problem 3

Suppose within your Web browser you click on a link to obtain a Web page. The IP address for the associated URL is cached in your local host, so a DNS look-up is not needed. Suppose that the Web page associated with the link is a small HTML file, consisting only of references to 100 very small objects on the same server. Let RTT_0 denote the RTT between the local host and the server containing the object. How much time elapses (in terms of RTT_0) from when you click on the link until your host receives all of the objects, if you are using:

1. HTTP/1.0 without parallel TCP connections?
2. HTTP/1.0 with parallel TCP connections?
3. HTTP/1.1 without parallel connections, but with pipelining?
4. HTTP/2
5. QUIC

Ignore any processing, transmission, or queuing delays in your calculation.

1. 2 RTT (TCP connection handshake and fetch HTML) + 2*100 RTTs (1 connection handshake + 1 fetch)
2. 1 RTT (TCP connection handshake) + 1 RTT (HTML file) + 2 RTTs
   Happening in parallel: we're spawning 100 parallel TCP connections, each of which has to establish a connection (1 RTT) and request a single object (1 RTT) for a total of 2 RTTs (assuming perfect parallelism)
3. 1 RTT (TCP connection handshake) + 1 RTT (HTML file) + 1 RTT (request other objects)
   Note: we are ignoring the amount of time it takes to send the data over the network, this costs an extra:
   $$\frac{\text{total number of bytes requested}}{\text{transmission rate}}$$
4. Assuming server push: 1 RTT (TCP connection) + 1 RTT (request)
   We request the HTML page and the server will push all associated objects with the request, so we don't incur any more RTTs.
5. 1 or 2 RTTs
   If the server "remembers" you, QUIC will not incur an extra RTT to establish a connection to the server. Otherwise, a handshake will be performed and it will cost an RTT. In either case, we have to request the HTML file from the server, which costs another RTT, but QUIC supports server-push so we'll receive all 100 objects used by the HTML file in the ideal case.

Problem 4

1. What is the difference between `From:` in the mail message and `Mail From:` in SMTP?
2. How in SMTP one marks the end of a message body?
3. How in HTTP one marks the end of a message body?

1. The `Mail From:` line is used internally by SMTP to deliver the message. The `From:` line is delivered in the message itself, and is seen by the recipient. In an analogy to the postal service, `Mail From:` is the address written on the envelope that the postal service uses to deliver the letter, and `From:` is written on the letter itself, so the recipient knows where the letter came from. Note that this means that it's not a strict rule that these fields match; you could write anything you like in the `From:` field.
2. In SMTP, the message body is terminated with `<CR><LF>.<CR><LF>`
3. In HTTP, the message body is not terminated with a character sequence. Instead, in HTTP/1.0, closing the connection is interpreted as the end of file. In HTTP/1.1, a special Content-Length header could tell you how many bytes of information to expect. Note: the message headers and message body are always separated by one blank line.

Problem 5

Given the source of received email message below, give a brief analysis (may not be precise) of how the original message traveled?

You should include:

1. How many times (approximately) the message changed hands using SMTP protocol
2. How many countries the message traveled. You may want to use "whois" command line tool and `https://www.maxmind.com/en/geoip-demo`, `https://www.iplocation.net/`, and similar geo-ip online databases.

If possible, give your interpretation of why each move happened.

```
Delivered-To: someemail@gmail.com
Received: by 10.157.5.104 with SMTP id 95csp541615otw;
       Tue, 5 Apr 2016 20:09:51 -0700 (PDT)
X-Received: by 10.67.1.65 with SMTP id be1mr67403234pad.46.1459912191121;
       Tue, 05 Apr 2016 20:09:51 -0700 (PDT)
Return-Path: <boost-bounces@lists.boost.org>
Received: from tx0-csb1.smtp.ucla.edu (tx0-csb1.smtp.ucla.edu. [2607:f010:3fe:102:0:ff:fe01:ae])
       by mx.google.com with ESMTPS id 23si1315046pft.23.2016.04.05.20.09.50
       for <someemail@gmail.com>
       (version=TLS1_2 cipher=ECDHE-RSA-AES128-GCM-SHA256 bits=128/128);
       Tue, 05 Apr 2016 20:09:51 -0700 (PDT)
Received-SPF: neutral (google.com: 2607:f010:3fe:102:0:ff:fe01:ae is neither permitted
             nor denied by best guess record for domain of boost-bounces@lists.boost.org)
             client-ip=2607:f010:3fe:102:0:ff:fe01:ae;
Authentication-Results: mx.google.com;
      spf=neutral (google.com: 2607:f010:3fe:102:0:ff:fe01:ae is neither permitted
      nor denied by best guess record for domain of boost-bounces@lists.boost.org)
      smtp.mailfrom=boost-bounces@lists.boost.org;
      dmarc=fail (p=NONE dis=NONE) header.from=gmail.com
```

Problem 5 continued on next page...

```
Received: from wowbagger.crest.iu.edu (wowbagger.crest.iu.edu [129.79.39.203])
  by mx-csb1.smtp.ucla.edu (8.14.4/8.14.4/Debian-4) with ESMTP id u3639edr022349
  for <alexander.afanasyev@ucla.edu>; Tue, 5 Apr 2016 20:09:42 -0700
Received: by wowbagger.crest.iu.edu (Postfix, from userid 495)
  id 3B85C160AE1; Tue, 5 Apr 2016 23:09:39 -0400 (EDT)
X-Spam-Checker-Version: SpamAssassin 3.3.1 (2010-03-16) on
  wowbagger.crest.iu.edu
X-Spam-Level:
X-Spam-Status: No, score=-2.6 required=5.0 tests=BAYES_00,DKIM_ADSP_CUSTOM_MED,
  FREEMAIL_FROM,RCVD_IN_DNSWL_LOW autolearn=unavailable version=3.3.1
Received: from wowbagger.crest.iu.edu (localhost [127.0.0.1])
  by wowbagger.crest.iu.edu (Postfix) with ESMTP id F08E015FE36;
  Tue, 5 Apr 2016 23:09:37 -0400 (EDT)
X-Original-To: boost@lists.boost.org
Delivered-To: boost@lists.boost.org
Received: by wowbagger.crest.iu.edu (Postfix, from userid 495)
  id 938BE15FE36; Tue, 5 Apr 2016 23:09:36 -0400 (EDT)
Received: from plane.gmane.org (plane.gmane.org [80.91.229.3])
  by wowbagger.crest.iu.edu (Postfix) with ESMTP id 1381215FD3D
  for <boost@lists.boost.org>; Tue, 5 Apr 2016 23:09:36 -0400 (EDT)
Received: from list by plane.gmane.org with local (Exim 4.69)
  (envelope-from <gcp-boost@m.gmane.org>) id 1andqX-0005Zr-Uv
  for boost@lists.boost.org; Wed, 06 Apr 2016 05:09:34 +0200
Received: from hullpq2102w-lp140-02-1088711752.dsl.bell.ca ([64.228.108.72])
  by main.gmane.org with esmtp (Gmexim 0.1 (Debian))
  id 1AlnuQ-0007hv-00
  for <boost@lists.boost.org>; Wed, 06 Apr 2016 05:09:33 +0200
Received: from philippeb8 by hullpq2102w-lp140-02-1088711752.dsl.bell.ca with
  local (Gmexim 0.1 (Debian)) id 1AlnuQ-0007hv-00
  for <boost@lists.boost.org>; Wed, 06 Apr 2016 05:09:33 +0200
X-Injected-Via-Gmane: http://gmane.org/
To: boost@lists.boost.org
From: Phil Bouchard <philippeb8@gmail.com>
Date: Tue, 5 Apr 2016 23:09:29 -0400
Lines: 20
Message-ID: <ne1ul8$6vo$1@ger.gmane.org>
References: <ndl0nh$l5q$1@ger.gmane.org> <ndn0eg$gu2$1@ger.gmane.org>
  <ndsbk2$s6b$1@ger.gmane.org> <ndscmk$avi$1@ger.gmane.org>
  <ndsdue$s1n$1@ger.gmane.org>
  <A0F04769-49D8-4CD6-BEEE-FF321E5E1275@ptd.net>
Mime-Version: 1.0
X-Complaints-To: usenet@ger.gmane.org
X-Gmane-NNTP-Posting-Host: hullpq2102w-lp140-02-1088711752.dsl.bell.ca
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:38.0) Gecko/20100101
  Thunderbird/38.5.1
In-Reply-To: <A0F04769-49D8-4CD6-BEEE-FF321E5E1275@ptd.net>
Subject: Re: [boost] [Root Pointer] Seeking a Review Manager
X-BeenThere: boost@lists.boost.org
X-Mailman-Version: 2.1.15
Precedence: list
Reply-To: boost@lists.boost.org
List-Id: Boost developers' mailing list <boost.lists.boost.org>
List-Unsubscribe: <http://lists.boost.org/mailman/options.cgi/boost>,
  <mailto:boost-request@lists.boost.org?subject=unsubscribe>
List-Archive: <http://lists.boost.org/boost/>
List-Post: <mailto:boost@lists.boost.org>
List-Help: <mailto:boost-request@lists.boost.org?subject=help>
List-Subscribe: <http://lists.boost.org/mailman/listinfo.cgi/boost>,
  <mailto:boost-request@lists.boost.org?subject=subscribe>
```

Problem 5 continued on next page...

```
Content-Transfer-Encoding: 7bit
Content-Type: text/plain; charset="us-ascii"; Format="flowed"
Errors-To: boost-bounces@lists.boost.org
Sender: "Boost" <boost-bounces@lists.boost.org>
X-Probable-Spam: no
X-Spam-Hits: 0.202
X-Spam-Report: DKIM_ADSP_CUSTOM_MED,FREEMAIL_FORGED_FROMDOMAIN,FREEMAIL_FROM,
    HEADER_FROM_DIFFERENT_DOMAINS
X-Scanned-By: MIMEDefang 2.75 on 169.232.46.172

On 04/04/2016 04:27 AM, Rob Stewart wrote:
> On April 3, 2016 8:53:34 PM EDT, Phil Bouchard <philippeb8@gmail.com> wrote:
>> On 04/03/2016 08:32 PM, Edward Diener wrote:
>>>
>>> I still am not getting what root_ptr<someType> is. Is it a
>>> replacement for shared_ptr ?
>>
>> There is no performance loss in using root_ptr/node_ptr over shared_ptr
>> so therefore yes it is a replacement for shared_ptr/weak_ptr.
>
> That isn't quite an answer to his question. If I understand you correctly, the point of your
    library is to
provide a means to manage groups of related memory allocations using a root_pointer. Each related
     memory
allocation is, I presume, a node_pointer created from, or attached to, one root_pointer. Because
    node_ptrs are
grouped and owned by a root_ptr, they are (can be?) destroyed, as a group, when the corresponding
     root_ptr is
destroyed, regardless of cycles.

I adapted the text here:
http://philippeb8.github.io/root_ptr/root_ptr/intro.html#root_ptr.intro.root_pointer

And I added important notes and a pitfall to the tutorial:
http://philippeb8.github.io/root_ptr/root_ptr/tutorial.html

There is not much to add to the 'basic' section of the tutorial because
if you wrongly use make_root and make_node then you will get compile errors.



-----------------------------------------------
Unsubscribe & other changes: http://lists.boost.org/mailman/listinfo.cgi/boost
```

This message is received approximately (only a lower bound) 11 times and travels through 3 countries: Canada, Norway, and the US. In chronological order:

1. The sender's own computer does some virus/spam/other maintenance locally before sending the email. This is the first mail server to receive that message, seemingly run by an ISP/telephone company.
2. A mail server managed by Gmane; from Wikipedia: "allows users to access electronic mailing lists as if they were Usenet groups".
3. Another one of Gmane's mail servers, possibly on the same machine given the word "local".
4. A Postfix mail server run by Indiana State University, seemingly in charge of Boost related resources.
5. Another Postfix mail server run by Indiana State, not enough info to determine why this server exists.
6. Yet another Indiana State mail server, this time it seems that the original message was delivered to the mailing list address, so this server is in charge of sending this message to everyone on that list.
7. First the mailing list server checks for spam to avoid spamming everyone in the mailing list.
8. Now one of the particular emails on the mailing list is alexander.afanasyev@ucla.edu, so the mailing list server generates a message destined for that email. It's received by a ISU mail server.
9. UCLA emails are managed by Google, and Google's servers are doing some authentication to make sure that this message is originating from where it claims to come from.
10. An intermediary server that the message went through, seems to just forward the message.
11. Message is finally delivered.