Cameron Moberg, Evan Gauer, Eli Charleville

Dr. Chetan Jaiswal

CS 470: Networks

April 22 2018

Project 2: HTTP Proxy

1 Description

A web proxy delivers content on behalf of remote web servers. This web proxy attempts to follow RFC 2616 as closely as possible.

Upon running **ProxyServer**, a **ServerSocket** is created on the specified port and then the application listens waiting for a request to be made. Upon receiving a request, it immediately accepts it and spins up a new thread to handle the request then quickly goes back to listening, so as to handle multiple clients if needed.

Once the thread is started it parses the request's destination URL and immediately checks if the requested URL has been cached, and if it has been, checks if it is still *fresh*. If the webpage has gone stale, or if it requires validation according to it's Cache-Control HTTP header, the proxy then sends a request to the requested webpage checking to see if anything has changed. At this point two things can happen, either the server responds with a HTTP/1.1 200 OK message code, or a HTTP/1.1 304 Not-Modified is sent back. Upon a 200 OK response we re-cache the webpage with the fresh copy and update how long it is fresh for. A response of 304 Not-Modified means we serve the cached copy, and update how long its fresh for again.

If the webpage has not been cached, then the proxy opens a connection with the requested server and retrieves the webpage content. Upon retrieval it determines if the webpage is able to be cached depending on the response code and Cache-Controls HTTP/1.1 Header. According to RFC 2616 a web response can only be cached if it is response code 200, 203, 300, 301 or 410. In the specification 206 is also fine to cache, but it is too complicated to implement.

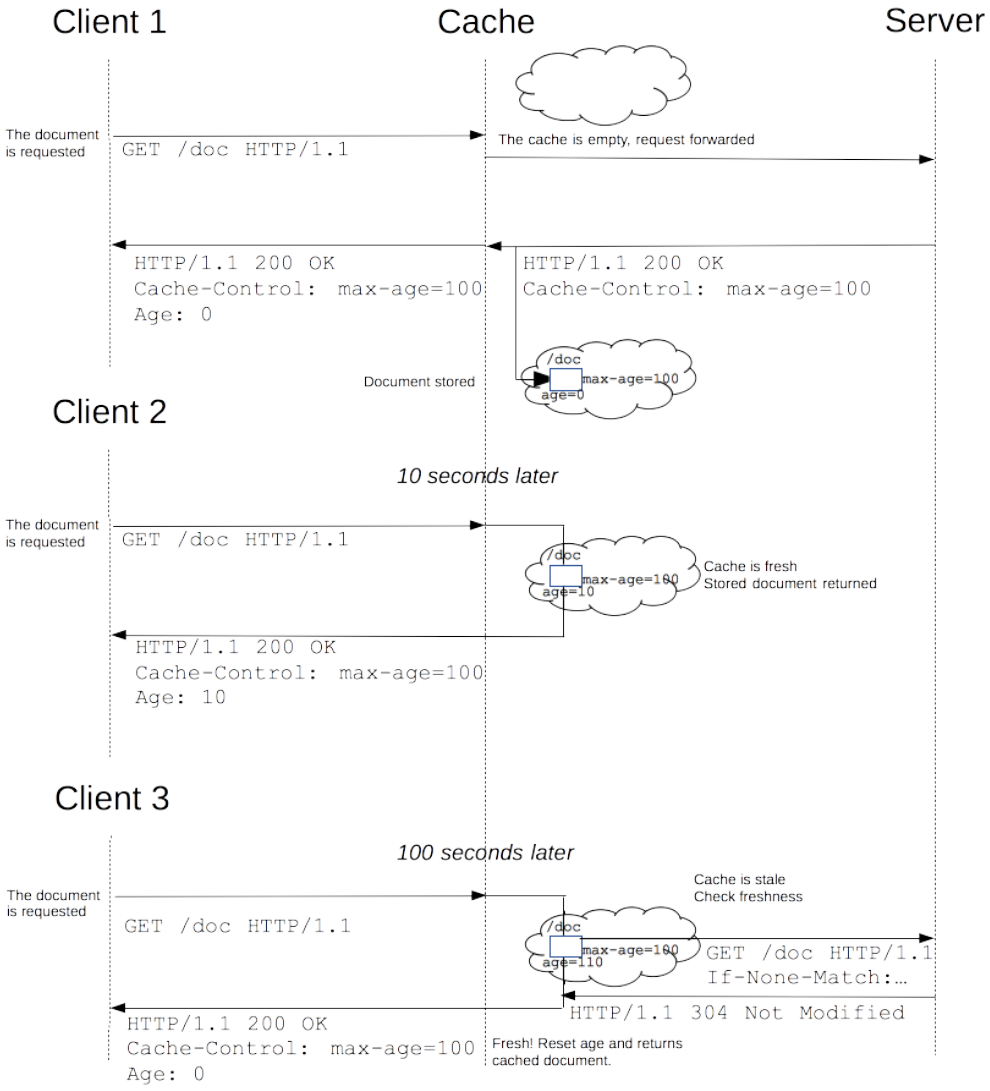
Once the response check is complete, we enter the *conditionalCache* method, where the HTTP Header is broken down to accurately cache for the correct time. If it contains *no-store* or *private*, we don't cache the response. If it contains *must-revalidate* or *proxy-revalidate* we set a flag in the **CachedSite** class that tells the proxy to revalidate everytime the cached site is requested. Lastly, if the Cache-Control header contains *public* we cache it as normal. After we determine if the response is indeed cacheable the program calculates how long to cache it for, if it is not explicitly given in the *max-age* portion of the header field. In the cache we store the max-age in milliseconds, the response's ETag field, when we last cached it, a mustRevalidate flag, and the response contents.

Finally the proxy returns the webpage contents to the user.

2 Thoughts

One of the main concerns we debated on was how to implement the cache. We had to choose between caching to disk in a file, using serializable, only caching to RAM, etc. We ended up deciding to purely store things in a POJO since it cut down on the complexities we would encounter with anything else. For example, if we wrote everything to disk, we would have no way of cleaning up those cached files efficiently, and we just weren't happy with that solution. Unfortunately our decision does not come without drawbacks. We store our webpage content in an array of bytes, which sounds great, until we realized that Java itself has a limitation on the size of the array: up to around two million elements. We decided on a size of 2^{24} elements, or around 16MiB, so extremely large websites will not fit on disk.

This is a flowchart of how our proxy handles the logic.



3 Contributions

- Cameron Moberg
  - Class Design
  - Caching Logic
- Evan Gauer
  - RFC 2616 Maintainer (Made sure we followed the rules)

  - Class Design

- HaCkErMaN eLi (Eli Charleville)
  - Multi-threaded Designer/Concurrency Expert
  - Class Design

```java
public class CachedSite
{
    private Instant cachedTime;

    private boolean mustRevalidate;
    private String etag;
    private long maxAge;

    private byte[] content;

    public CachedSite(byte[] content, Instant instant, String etag, boolean mustRevalidate, long maxAge)
    {
        this.content = content;
        this.cachedTime = instant;
        this.etag = etag;
        this.mustRevalidate = mustRevalidate;
        this.maxAge = maxAge;
    }

    public boolean isFresh()
    {
        // Never fresh if must revalidate, or if maxAge was never set.
        return !mustRevalidate && (cachedTime.plus(maxAge, ChronoUnit.SECONDS).isAfter(Instant.now()) || maxAge == -1);
    }

    public String getEtag()
    {
        return etag;
    }

    public void setEtag(String etag)
    {
        this.etag = etag;
    }

    public Instant getCachedTime()
    {
        return cachedTime;
    }

    public byte[] getContent()
    {
        return content;
    }
}
```

```
public class ProxyServer
{
    private static Map<String, CachedSite> siteMap = new ConcurrentHashMap<>();
    private static int PORT_NUM = 6969;

    public static void main(String[] args) throws IOException
    {
        ServerSocket serverSocket = null;
        boolean listening = true;

        try
        {
            serverSocket = new ServerSocket(PORT_NUM);
            System.out.println("Started proxy on port " + PORT_NUM);
        } catch (IOException e)
        {
            e.printStackTrace();
            System.exit(-1);
        }

        while (listening)
        {
            new ProxyThread(serverSocket.accept()).start();
        }
        serverSocket.close();
    }

    static Map<String, CachedSite> getSiteMap()
    {
        return siteMap;
    }
}
```

```
public class ProxyThread extends Thread
{
    private static final int BUFFER_SIZE = 2 << 24;
    private static final int DEFAULT_CACHE_TIME = 86400;

    private SimpleDateFormat httpTime = new SimpleDateFormat("EEE, dd MMM yyyy HH:mm:ss z", Locale.US);

    private Socket socket;
    //Since each thread is a unique HTTP connection, we have instance variables.
    private HttpURLConnection urlConnection;
    private String urlName;

    public ProxyThread(Socket socket)
    {
        super("ProxyThread");
        this.socket = socket;
    }

    /**
     * HTTP Time format for If-Modified-Since format is like
     * Wed, 21 Oct 2015 07:28:00 GMT
     *
     * @param instant The Instant you want to convert to HTTP time
     * @return returns String in If-Modified-Since format
     */
    private String toHttpTimeFormat(Instant instant)
    {
        httpTime.setTimeZone(TimeZone.getTimeZone("GMT"));
        return httpTime.format(new Date(instant.toEpochMilli()));
    }

    private Instant fromHttpTimeFormat(String date)
    {
        try
        {
            return httpTime.parse(date).toInstant();
        } catch (ParseException e)
        {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * Checks if a given URL (in String format) has been cached already.
     *
     * @param url The URL to check, in String format.
     * @return Whether or not has been cached yet.
     */
```

```
private boolean isCached(String url)
{
    return ProxyServer.getSiteMap().containsKey(url);
}

private byte[] getStreamOutput(InputStream in) throws IOException
{
    ByteArrayOutputStream buffer = new ByteArrayOutputStream();

    int nRead;
    byte[] data = new byte[BUFFER_SIZE];

    while ((nRead = in.read(data, 0, data.length)) != -1)
    {
        buffer.write(data, 0, nRead);
    }
    return buffer.toByteArray();
}

/**
 * According to RFC2616
 * A response received with a status code of 200, 203, 206, 300, 301 or 410 MAY
 * be stored by a cache and used in reply to a subsequent request,
 * subject to the expiration mechanism, unless a cache-control directive
 * prohibits caching.
 *
 * @param urlResponseCode The Response code
 * @return Whether the URL is cacheable
 */
private boolean returnCodeOk(int urlResponseCode)
{
    switch (urlResponseCode)
    {
        case HttpURLConnection.HTTP_OK:
        case HttpURLConnection.HTTP_NOT_AUTHORITATIVE:
        case HttpURLConnection.HTTP_MULT_CHOICE:
        case HttpURLConnection.HTTP_MOVED_PERM:
        case HttpURLConnection.HTTP_GONE:
            return true;
        default:
            return false;
    }
}

private void conditionalCache(HttpURLConnection huc, byte[] websiteContent)
{
    boolean revalidate = false;
    long maxAge = -1;
```

```
    String cacheControl = huc.getHeaderField("Cache-Control");

    if (cacheControl != null)
    {
        String[] cacheControlArr = cacheControl.split(", ");

        for (String option : cacheControlArr)
        {
            if (option.contains("max-age"))
            {
                // the max-age substring is 7 characters long
                maxAge = Long.valueOf(option.substring(8));
            }

            switch (option)
            {
                // This breaks out of conditionalCache method, and does not conditionalCache anything.
                case "no-store":
                case "private":
                    return;
                case "must-revalidate":
                case "proxy-revalidate":
                    revalidate = true;
                case "public":
                default:
                    continue;
            }
        }
    }

    if (maxAge == -1)
    {
        maxAge = findExpireTime(huc.getHeaderField("Date"), huc.getHeaderField("Expires"),
                huc.getHeaderField("Last-Modified"));
    }

    String etag = huc.getHeaderField("ETag");

    CachedSite cachedSite = new CachedSite(websiteContent, Instant.now(), etag, revalidate, maxAge);
    ProxyServer.getSiteMap().put(urlName, cachedSite);
}

/**
 * Finds length of time before conditionalCache of website goes stale
 *
 * @param date         The HTTP Header "Date"
 * @param expires      The HTTP Header "Expires"
 * @param lastModified The HTTP Header "Last-Modified"
 * @return Number of seconds until expiration.
```

```
 */
private int findExpireTime(String date, String expires, String lastModified)
{
    Instant dateInstant, expiresInstant, lastModInstant = null;

    dateInstant = date == null ? Instant.now() : fromHttpTimeFormat(date);
    expiresInstant = expires == null ? null : fromHttpTimeFormat(expires);
    lastModInstant = lastModified == null ? null : fromHttpTimeFormat(lastModified);

    if (expiresInstant != null && dateInstant != null)
    {
        return (int) Duration.between(dateInstant, expiresInstant).getSeconds();
    }
    else if (expiresInstant == null && dateInstant != null && lastModInstant != null)
    {
        return (int) Duration.between(lastModInstant, dateInstant).getSeconds() / 10;
    }
    else
    {
        return DEFAULT_CACHE_TIME;
    }
}

/**
 * Connects to a given URL.
 *
 * @param huc HTTP Connection that user wants to extract data from
 * @return Returns InputStream of the webpage content.
 */
private InputStream getWebpageContent(HttpURLConnection huc) throws IOException
{
    // If it gets to this point and is cached, we must revalidate.
    if (isCached(urlName))
    {
        CachedSite cachedSite = ProxyServer.getSiteMap().get(urlName);
        Instant lastVisit = cachedSite.getCachedTime();

        huc.setRequestProperty("If-Modified-Since", toHttpTimeFormat(lastVisit));

        if (cachedSite.getEtag() != null)
        {
            huc.setRequestProperty("If-None-Match", cachedSite.getEtag());
        }
    }

    InputStream inputStream;
    switch (huc.getResponseCode())
    {
        case HttpURLConnection.HTTP_OK:
```

```
            inputStream = huc.getInputStream();
            break;
        case HttpURLConnection.HTTP_MOVED_PERM:
            //Manually redirect to new webpage
            URL redirURL = new URL(huc.getHeaderField("Location"));
            this.urlConnection = (HttpURLConnection) redirURL.openConnection();
            return getWebpageContent(urlConnection);
        case HttpURLConnection.HTTP_NOT_MODIFIED:
            inputStream = new ByteArrayInputStream(ProxyServer.getSiteMap().get(urlName).getContent());
            break;
        case HttpURLConnection.HTTP_BAD_REQUEST:
        case HttpURLConnection.HTTP_NOT_IMPLEMENTED:
        default:
            try
            {
                inputStream = huc.getInputStream();
            } catch (IOException e)
            {
                inputStream = huc.getErrorStream();
            }
            break;
    }
    return inputStream;
}

public void run()
{
    try
    {
        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        DataOutputStream out = new DataOutputStream(socket.getOutputStream());

        String inputLine;

        if ((inputLine = in.readLine()) != null)
        {
            // First line of request is GET http://bing.com HTTP/1.1
            urlName = inputLine.split(" ")[1];
        }

        // If is cached, and is fresh, just directly write the cached version to the user.
        if (isCached(urlName) && ProxyServer.getSiteMap().get(urlName).isFresh())
        {
            out.write(getStreamOutput(new ByteArrayInputStream(ProxyServer.getSiteMap().get(urlName).getContent())));
            out.close();
            in.close();
            return;
        }
```

```
            // Else it opens a connection to the url
            urlConnection = (HttpURLConnection) new URL(urlName).openConnection();
            byte[] webpageData = getStreamOutput(getWebpageContent(urlConnection));

            // Check response is even cacheable
            if (returnCodeOk(urlConnection.getResponseCode()))
            {
                conditionalCache(urlConnection, webpageData);
            }

            //Write to output (e.g. back to client)
            out.write(webpageData);
            out.close();
            in.close();
            if (socket != null)
            {
                socket.close();
            }
        } catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```