# Network Programming

**Dr. Mohamed** R. Ghetas

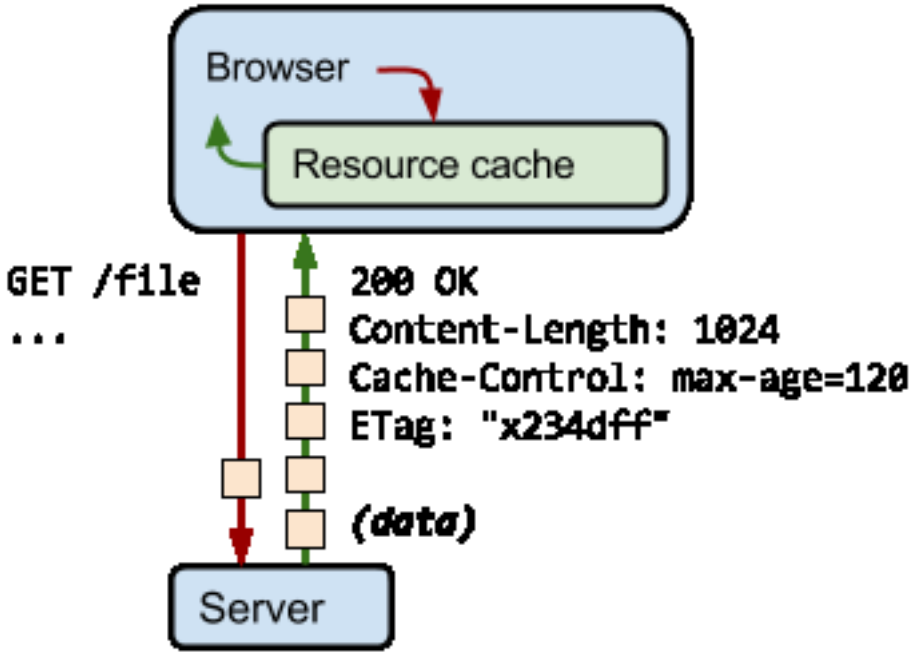
# HTTP Caching

- Fetching something over the network can be both slow and expensive.
  - Large responses require many roundtrips between the client and server
  - High possible delays
  - High costs
- As a result, the ability to cache and reuse previously fetched resources is a critical aspect of optimizing for performance.
- The **Cache-Control** general-header field is used in HTTP to specify directives (instructions) for caching mechanisms (in a caching system) in both, requests and responses. Caching directives are unidirectional, meaning that a given directive in a request is not implying that the same directive is to be given in the response.
- Web caching system by using Proxy servers will be discussed Later.

# Basic mechanisms for controlling caches (1)

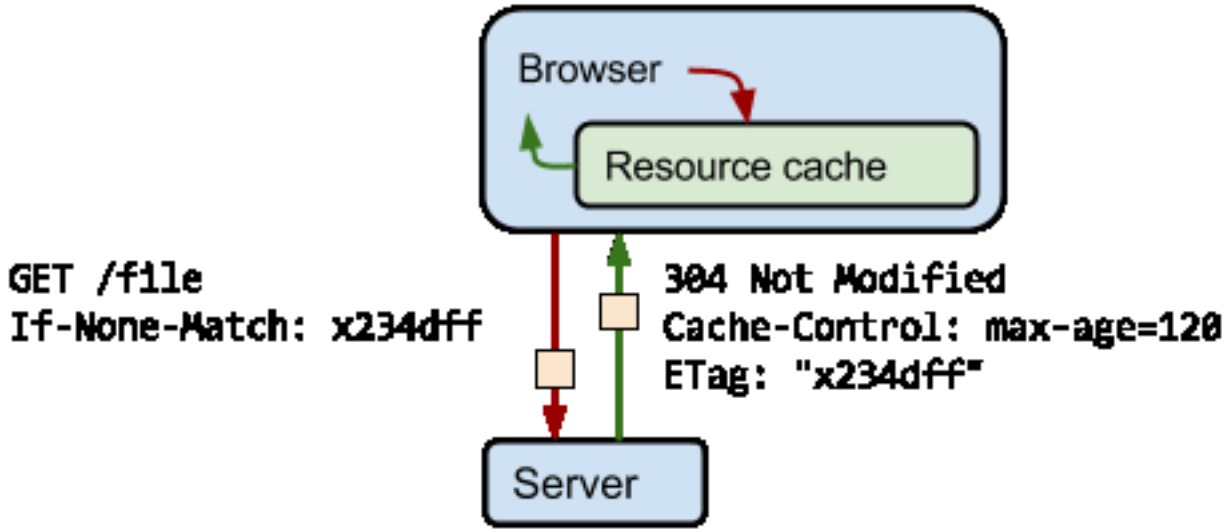
HTTP defines three basic mechanisms for controlling caches:Freshness, Validation and Invalidation.

- **Freshness:**allows a response to be used without re-checking it on the origin server. It can be controlled by both the server and the client. For example, the Expires response header gives a date when the document becomes stale (outdated), and the **Cache-Control: max-age** directive tells the cache how many seconds the response is fresh for. In the shown figure, the server to instructs the client to cache a file ,of size $1024$ byte, for up to $120$ seconds.



# Basic mechanisms for controlling caches (2)

- **Validation:** can be used to check whether a cached response is still good after it becomes stale. For example, if the response has a **Last-Modified** header, a cache can make a conditional request using the **If-Modified-Since** header to see if it has changed. The ETag (entity tag) mechanism also allows for both strong and weak validation.
  - **ETag** header is used to communicate a validation token.
  - Validation token (hash/some fingerprint for a file) is typically generated at the server (see figure), so the client can send it on the next request for the same resource by using **If-None-Match** header. If it is the same, the download can be omitted and the server returns a **304 Not Modified** response.



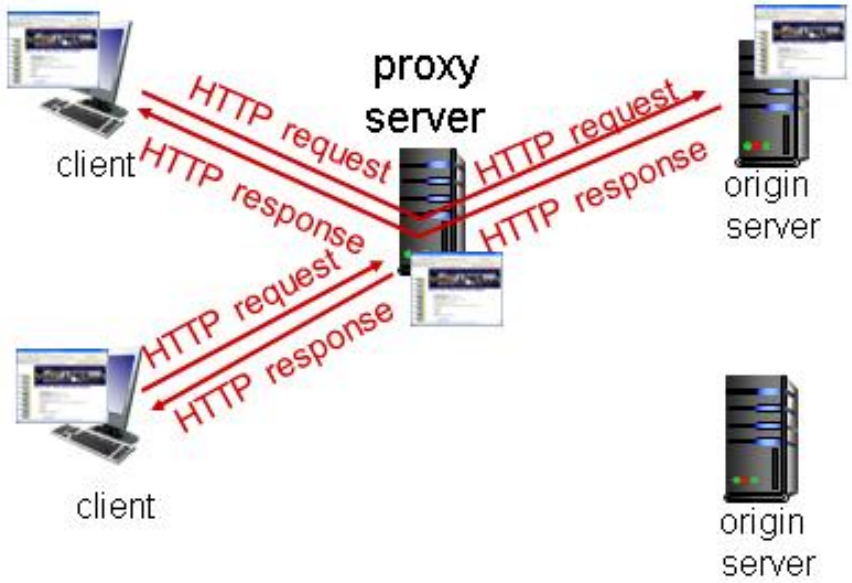
# Basic mechanisms for controlling caches (3)

- **Invalidation:** is usually a side effect of another request that passes through the cache. For example, if a URL associated with a cached response subsequently gets a POST, PUT or DELETE request, the cached response will be invalidated (typically GET and HEADER methods are most common with cache).

# Web caches (proxy server)

goal: satisfy client requests without origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



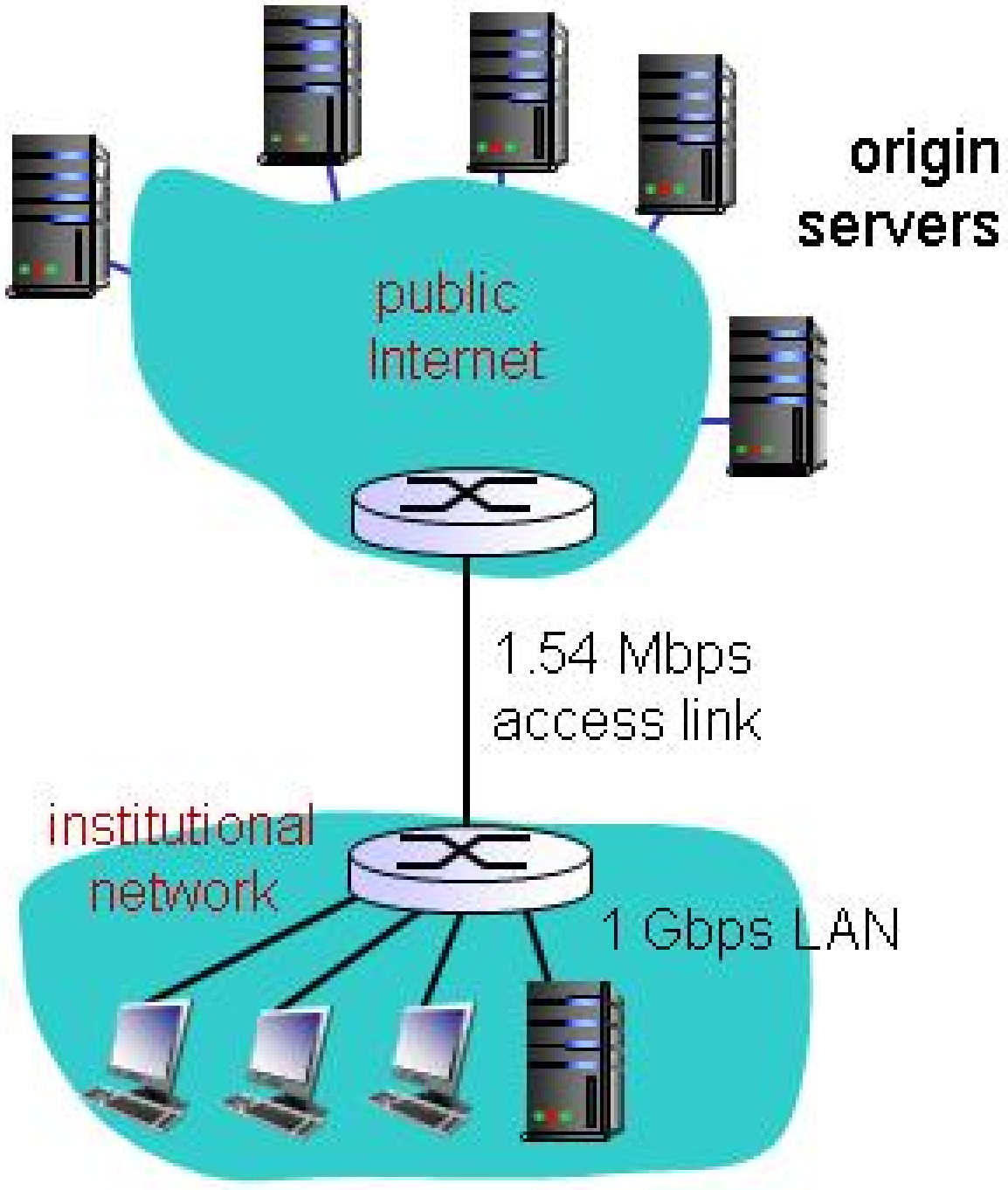
# More about Web caching

- cache acts as both client and server
  - server for original requesting client
  - client to origin server
- typically cache is installed by ISP (university, company, residential ISP)
- why Web caching?
  - reduce response time for client request
  - reduce traffic on an institutions access link
  - Internet dense with caches: enables poor content providers to effectively deliver content (also for P2P file sharing)

# Some Network terminology Definitions

- Round-trip delay time (RTD) or round-trip time (RTT): is the length of time it takes for a signal to be sent plus the length of time it takes for an acknowledgment of that signal to be received, i.e. the time of delivering the request plus the time of receiving the response.
- Network utilization is the ratio of current network traffic to the maximum traffic that the port/Network can handle. High utilization ratio (close to $1$ or $100\%$) means congested network with long queues and high delays. Low utilization ratio (close to zero or $0\%$) means the inverse which is desired in Network design.
- A cache hit occurs when the requested data can be found in a cache, while a cache miss occurs when it cannot. Thus, cache hit rate is the rate of finding requested data in a cache. Note that, cache hit rate $= 1-$ cache miss rate.

# Caching example



# Caching example ... Contin.

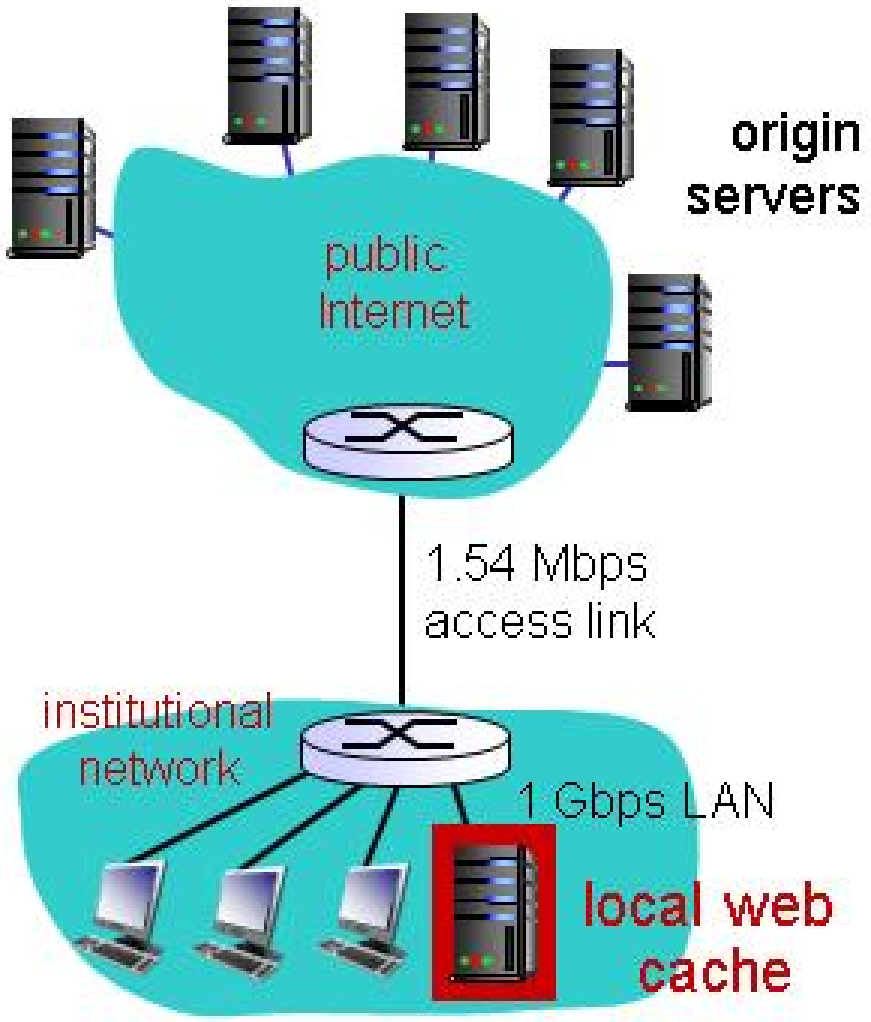
assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- Thus, avg data rate to browsers: 1.50 Mbps
- Round Trip Delay (RTD) from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- Utilization
- LAN utilization (ratio of the average data rate to Browsers to the available LAN data rate) : $\frac{1.5 \text{ Mbps}}{1000 \text{ Mbps}} = 0.0015 = 0.15\%$
- Access link utilization (ratio of the average data rate to Browsers to the available access link data rate): $\frac{1.5 \text{ Mbps}}{1.54 \text{ Mbps}} = 0.974 = 97.4\%$
- Higher utilization indicates higher delay, e.g. delay in the given access link would be greater than the delay in the given LAN
- total delay = Internet delay + access delay = $2$ sec + minutes
- Reduce access delay by increasing access link data rate, e.g. 154 Mbps, so access link utilization will be $0.00974 = 0.97\%$, and the total delay = $2$ sec + msec. (Cost: increased access link speed (not cheap))

# Caching example: install local cache




*Cost:* web cache (cheap!)

# Caching example: install local cache ... Contin.

Calculating access link utilization, delay with cache:

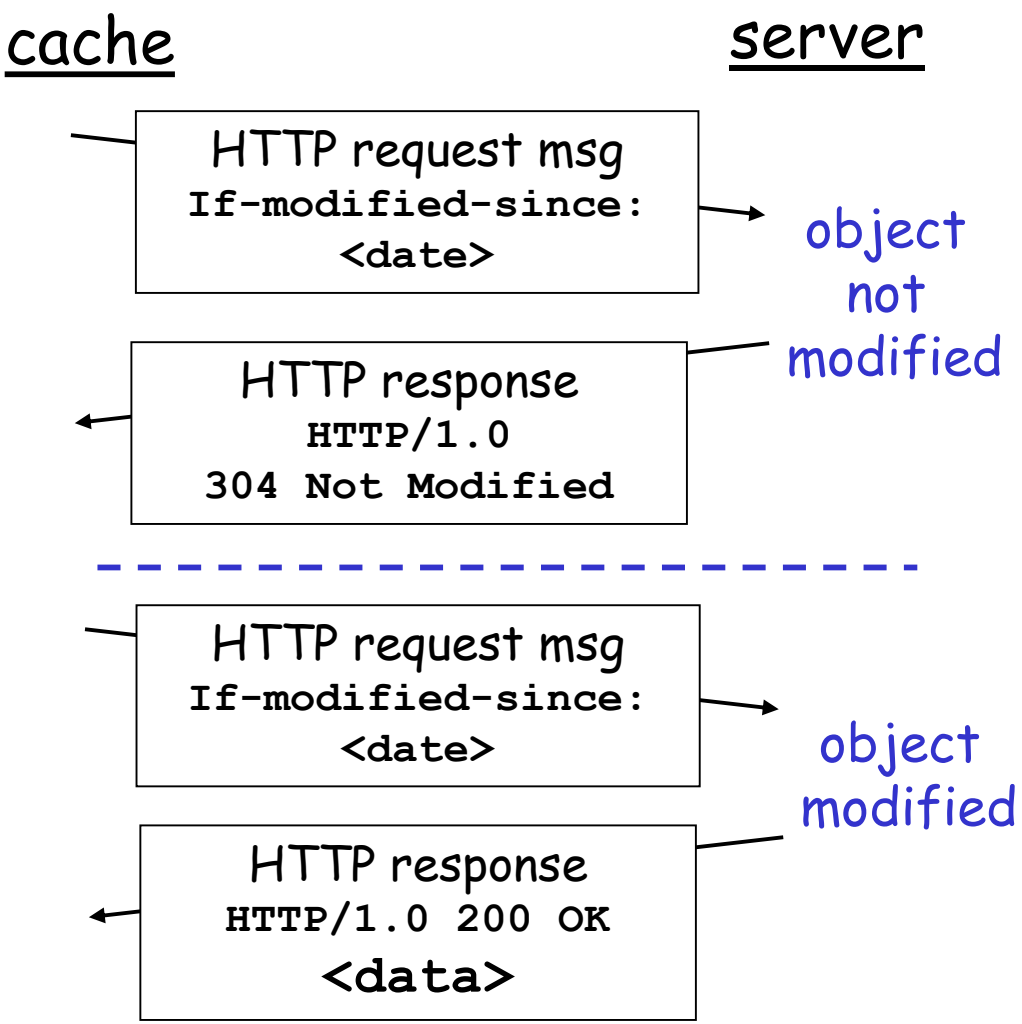
- suppose cache hit rate is 0.4
  - $40\%$ requests satisfied at cache, $60\%$ requests satisfied at origin
- access link utilization:
  - $60\%$ of requests use access link
- data rate to browsers over access link $= 0.6 * 1.50$ Mbps $= 0.9$ Mbps
  - utilization $= 0.9/1.54 = 58\%$
- total delay:
  - $= 0.6\times$ (delay from origin servers) $+0.4\times$ (delay when satisfied at cache)
  - $= 0.6(2sec) + 0.4(\ msecs) = \ 1.2$ secs
  - less than with 154 Mbps link (and cheaper too!)

# Conditional GET

- Goal: don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request

  `If-modified-since: <date>`
- server: response contains no object if cached copy is up-to-date:

  `HTTP/1.0 304 Not Modified`



# HTTP request Cache-Control:directives (1)

| Directive | Meaning |
|---|---|
| no-cache | A cache can keep a cached copy of the document, but must always revalidate it before sending it back to the client. |
| no-store | Remove information promptly after forwarding. The cache should not store anything about the client request or server response. This option prevents the accidental storing of secure or sensitive information in the cache. |
| max-age = . . . seconds | Do not send responses older than seconds. The cache can send a cached document that has been retrieved within a certain number of seconds from the time it was sent by the origin server. |

# HTTP request Cache-Control:directives (2)

| | |
|---|---|
| max-stale [ = . . . seconds ] | The cache can send a cached document that is older than its expiration date. If seconds are given, it must not be expired by more than that time. |
| min-fresh = . . . seconds | Send data only if still fresh after the specified number of seconds. The cache can send a cached document only if there are at least a certain number of seconds between now and its expiration time. |
| only-if-cached | Do not retrieve new data. The cache can send a document only if it is in the cache, and should not contact the origin-server to see if a newer copy exists. This option is useful when network connectivity from the cache to origin-server is poor. |

# HTTP response Cache-Control:directives (1)

| Directive | Meaning |
|---|---|
| public | The document is cacheable by any cache. |
| private | The document is not cacheable by a shared cache. |
| no-cache | A cache can keep a cached copy of the document, but must always revalidate it before sending it back to the client. |
| no-store | Do not store the returning document. Remove information promptly after forwarding. |

# HTTP response Cache-Control:directives (2)

| | |
|---|---|
| no-transform | Do not convert the entity-body. Useful for applications that require that the message received is exactly what was sent by the server. |
| must-revalidate | The cache must verify the status of stale documents, i.e., the cache cannot blindly use a document that has expired. |
| proxy-revalidate | Client must revalidate data except for private client caches. Public caches must verify the status of stale documents. Like must-revalidate, excluding private caches. |
| max-age = . . . seconds | The document should be considered stale in the specified number of seconds from the time of retrieval. |
| s-maxage = . . . seconds | The same as max-age, except for public/shared caches. This directive is ignored by private caches. |

# Defining optimal Cache-Control policy

- Follow the decision tree below to determine guidelines for optimal caching policy for a particular resource, or a set of resources, that your application uses. Ideally, you should aim to cache as many responses as possible on the client for the longest possible period, and provide validation tokens for each response to enable efficient revalidation. This tree can be modified based on design requirements.