COMP 431 Internet Protocols & Services

Spring 2018  
Kevin Jeffay

Worksheet 5, September 24

1. After graduating UNC you take a job for a company developing a new operating system for the next generation of tablet computers. You are assigned to a team that is designing a web browser for the OS. A key design decision is whether or not your browser should use:

* Non-persistent HTTP connections,
* Persistent, non-pipelined HTTP connections,
* Persistent, pipelined HTTP connections, or
* Parallel, non-persistent, HTTP connections.

1. The design decision quickly comes down to the issue of how the tablets are likely to be used in practice. If possible, describe the usage conditions under which each HTTP paradigm will give superior performance than the other schemes. (Note: Although we’re dealing with tablet computers here, you need NOT consider issues of mobility, power consumption, resource constraints, *etc*.)

**Persistence only works if your content is all on the same server.**

Non-persistent HTTP connections will be better when:

* You need 2 RTTs, one for the handshake with TCP and 1 for the request/response. They allow multiple objects to be returned with only 1 TCP handshake. In the worst case, a persistent connection will devolve into a non-persistent connection, so it can never be worse.

Persistent, non-pipelined HTTP connections will be better when:

* All the data is on the same server.

Persistent, pipelined HTTP connections will be better when:

* Allows you to send another request before the previous request has been received. If t is the transmission time for a response, then k request/response exchanges can be made in as little as 2RTT + (k – 1) x t. The first RTT is to setup the connection. The second RTT is for the first request to go all the way to the server and the response to come back. At worst, it’s equal to a persistent connection (when there’s only 1 object to send).

Parallel, non-persistent, HTTP connections will be better when:

- Parallelism is limited by browsers. In theory, if they let you have k parallel connections, then up to k objects can be retrieved every k RTT. If you’re beyond k, persistent will be better, but if not, parallel will be better. If you get stuff on different servers, parallel will always be better than both persistent and pipeline. If it’s on the same server, and the number of objects is greater than k, pipeline will win.

*b*) Given that most computers have a single network interface that can only transmit or receive bytes of data from one connection at a time (*i.e.*, there is no true parallel transmission or reception), why is the use of parallel HTTP connections ever a good idea?

In practice, parallel is truly sequential because 1 thing can’t send multiple things at once. Parallel is always a win because it looks like pipelining but works on multiple servers. In practice, parallel gets 2RTT + (k – 1) x t retrieval time, which is the same as pipelining. If you have a large number of objects, then pipelining will be worse.

The good thing about parallel is that if you’re going to multiple servers, it guarantees that performance if you’re retrieving less than k objects.