Computer Networks Exercise 2: DNS (2/2) & HTTP & c-Speed (1/2)

Spring Semester 2018

Answer Sheet

Q0: What is the difference between "naming" and "addressing"? Which problems does DNS solve? Why should we always use trusted DNS servers?

- Naming does not necessarily follow a strict set of rules, while the format of an address is well-defined and can easily be interpreted by a machine.

- It is much easier for users to remember domain names than IP addresses. The same service can have different addresses, which means a service can easily be relocated or accessed using different addresses (load-balancing).

- Imagine you want to go to eth-webshop.ch to buy a present. If you’re using an untrusted DNS server, it can redirect you to an attacker’s IP address instead of ETH’s. The attacker will then imitate ETH, and will serve you a "fake" ETH Webshop page, which will record and steal any information you enter (e.g. credit card information).

Q1: DNS uses BGP Anycast to distribute the traffic among DNS servers close to the destination according to some metric (e.g. lowest RTT or minimal congestion). Every name resolution request is sent using UDP to the nearest DNS server and in case of a packet loss, the UDP request is resent. Imagine a protocol which creates a TCP + TLS connection to the nearest DNS server provided by BGP Anycast and sends name resolution requests using TCP + TLS packets. What are the benefits and problems of this protocol?

- The TCP connection can time out. Either packets are sent regularly to keep the connection alive or the connection is reestablished after an inactivity period.

- The protocol requires the DNS server to keep TCP and TLS state for each client (limits scalability which is an essential property of DNS).
• BGP Anycast might modify the route by changing the destination server due to network congestion, which would break the TCP + TLS connection and require reestablishing it.

• The protocol would be more secure than DNS, since an attacker would not be able to inject a fake response.

Q2.1: The static "hosts.txt" approach is no longer in use. Explain why by giving at least 3 reasons with motivation.

• Scalability. It is not possible to incrementally update an non-indexed file. Updates would entail copying over the entire file again. A load-balanced hierarchical structure to issue updates is needed.

• Scalability. The file at the current size of the Internet would unnecessarily take too much space because users only visit a fraction of hostnames, which would be especially problematic in low-space machines such as in the Internet of Things.

• Management. Who is assigned which hostname becomes a management issue at a world-wide scale. For example, governments want to control their respective own top level domains.

• Consistency/availability. It is desired to have a stronger guarantee of propagation than "hoping that everyone will adopt my change by downloading an enormous file".

Q2.2: Given the DNS (caching) architecture explained during lecture, what happens from a user perspective when a hostname DNS record removal is issued? And what if the IP address of a hostname is changed?

DNS servers (and the OS/browser*) between the client and the authoritative DNS server cache records. Before the TTL is expired, in the first case users will still be directed to the original IP when typing in the hostname. In the second case, they will still be redirected to the old IP address. Once the TTL is expired, the records are flushed: in the first case users will no longer be sent to the original IP and will receive a notification that the hostname could not be resolved. In the second case, the users will be redirected to the new IP.

*http://kb.mozillazine.org/Network.dnsCacheExpiration (retrieved March 6th 2018)

Q2.3: Would you employ DNS in the Internet of Things (IoT)? Name an advantage and a disadvantage.

Advantage: it allows the host to be identified by hostname instead of by static IP, as such the destination of its communication can change its IP without
forcing a re-flash of the IoT devices that communicate with it. Disadvantage: too limited power, as such performing DNS queries might be too draining.

Q2.4: Would you use recursive or iterative DNS querying in IoT?

Recursive querying, since IoT devices have only limited computational resources.

Q2.5: How can you employ DNS for censorship? What are its limitations?

By blocking or spoofing DNS requests, which allows an attacker to return nothing or another website (DNS spoofing or poisoning). DNS is only a service that resolves hostnames; by only corrupting DNS queries, it is still possible to directly communicate when the user knows the IP address beforehand.

Q3. URLs

Q3.1: Which of the following URL statements are true, and which are false?

1. The directory path must map to the underlying file system. (true / false) false
2. Port 80 and 443 are the respective default ports for HTTP and HTTPS. (true / false) true
3. An IP address is a valid hostname. (true / false) true
4. The port numbers enable multiple services to be simultaneously active on the same host. (true / false) true

Q3.2: Why are there standard port numbers?

Two answers are possible depending on interpretation of the question: (a) because of convention, it is not necessary for a user to define the port number every time: the default port associated with the protocol (i.e. 80 for HTTP) has become entrenched standard for Web access, or (b) because when a connection is established, the initiator must define which port it wants to connect to and do. The service must already be listening to receive the packets.

Q4. Critical path analysis

Within a dependency graph, the critical path decides the minimum amount of time it takes to fully load all dependencies. The next couple of questions are in relation to the dependency graph shown in Fig. 1.

Q4.1: With $A = 4$, $B = 19$: what is the critical path?

The critical path is the longest path. There are three paths: $\max\{3 + 5 + 4 + 10 + 10, 3 + 19 + 10 + 10, 3 + 10\} = \max\{32, 42, 13\} = 42$. So path FIN-N1-N-
N5-N6 is the critical path.

Q4.2: With $A = 5$, $B \sim \text{Poisson}(7)$: what is the probability FIN-N1-N3-N5-N6 is strictly the critical path?

Let's call $L_1 = \text{length}(FIN-N1-N3-N5-N6)$, $L_2 = \text{length}(FIN-N1-N2-N4-N5-N6)$ and $L_3 = \text{length}(FIN-N1-N6)$. $L_1$ and $L_2$ are always longer than $L_3$ because edges cannot be negative and $L_1$ and $L_2$ are always strictly more than 13 long.

We want to know:

$$P(L_1 > L_2) = P(3 + B + 10 + 10 > 3 + 5 + A + 10 + 10)$$
$$= P(B > 5 + A)$$
$$= P(B - A > 5)$$
$$= P(B - 5 > 5)$$
$$= P(B > 10)$$

$$= 1 - P(B \leq k = 10) = 1 - e^{-\lambda} \sum_{i=0}^{\text{floor}(k)} \frac{\lambda^i}{i!}$$

$$= 1 - e^{-7} \sum_{i=0}^{10} \frac{7^i}{i!}$$

$$= 1 - 0.90148$$
$$= 0.09852$$

Q5: Which are the main differences between a GET and a POST request? When should one use GET instead of POST and why?
The main difference between GET and POST methods is defined in terms of form data encoding. With GET, the query parameters are encoded in the URI. On the other hand, with POST the parameters are embedded in the body of the request. This encoding difference has a couple of consequences:

- GET requests can be cached, POST requests can’t (as the POST is not idempotent, see below).
- GET requests remain in the browser history, POST requests don’t (as their form data is not stored).
- GET requests can be bookmarked, POST requests can’t (as their form data is not stored).
- GET requests have length restrictions on its parameters (maximum URL length is enforced), POST requests haven’t.

The official recommendations say that "GET" should be used only in case of idempotent processing, meaning that the user request is a pure data retrieval that will not cause any change in the WEB application state. Furthermore, GET requests should never be used when dealing with sensitive data. This is due to the fact that servers, proxies, and user agents can log the request URI in some place where it might be visible to third parties. In other words, **NEVER** submit an authentication form with the GET method!

**Q6:** What does it mean that HTTP is a stateless protocol? How can we force it to keep a state?

HTTP is stateless in the sense that each request is handled independently with respect to the others. Some applications, anyway, need to maintain a state. This is possible through the use of the so-called **Cookies**. Cookies are small pieces of data, stored in the client browser in the form of key-value pairs. Cookies can also provide authentication: authentication cookies, in fact, are the most common method used by web servers in order to know whether a user is logged in or not, and with which account they are logged in.