HTTP Protokoll

1 Ablauf einer HTTP - Anfrage/Antwort Sequenz

Basierend auf: Sascha Kerskens Apache 2

Das TCP/IP-Anwendungsprotokoll HTTP (Hypertext Transfer Protocol) bildet die Basis der Kommunikation zwischen Webserver und Browser.

Die aktuelle Version ist HTTP/2, ihre Spezifikation steht in RFC 7540. Verbreitet ist jedoch weiterhin die Version HTTP/1.1 auf Basis von RFC 2616. Im Unterschied zur aktuellen Version HTTP/2 handelt es sich bei der Version HTTP/1.1 um ein klartextbasiertes Protokoll mit wenigen Client-Befehlen und definierten Server-Antworten.

HTTP ist ein typisches Request/Response-Verfahren (Anfrage und Antwort). Der Ablauf einer solchen HTTP-Anfrage und -Antwort ist immer derselbe:

1. Der Browser ermittelt aus dem URL den Hostnamen des angesprochenen Servers und stellt eine **TCP-Verbindung** zu diesem her. Standardmässig wird der **TCP-Server-Port 80** gewählt, ansonsten die Portnummer, die in dem URL steht. Angenommen, der angeforderte URL ist

   `http://www.mynet.de/seiten/info.html`

   Dann ist der Server `www.mynet.de` Der Browser ermittelt über DNS die IP-Adresse dieses Hosts und stellt eine Verbindung zu dessen Port 80 her.

2. Nachdem die Verbindung hergestellt ist, übermittelt der Browser eine Anfrage an den Server.
   Die erste Zeile enthält den **HTTP-Befehl**, den **Pfad** des angesprochenen Dokuments und die **Protokollversion**.
   In weiteren Zeilen folgen **optionale Header-Felder** nach dem Schema `Feldname: Wert`, z.B. Browser-Version, Hostname des anfordernden Rechners oder URL des Dokuments, in dem sich der angeklickte Hyperlink befand (den so genannten Referer).
   Bestimmte HTTP-Anfragen enthalten nach einer Leerzeile auch noch einen **Body**, in dem zusätzliche Daten geschickt werden.
   Der einfachste HTTP-Befehl (und der einzige in der ursprünglichen Version HTTP/0.9!) ist GET. Er bedeutet ganz einfach, dass ein bestimmtes Dokument geliefert werden soll.
   Gemäss dem Beispiel aus Punkt 1 wird hier der folgende GET-Befehl betrachtet:

   `GET /seiten/info.html HTTP/1.1`

   Hinweis: Die zugehörigen Header-Optionen werden weiter unten genauer erläutert.

3. Der Server nimmt die GET-Anfrage entgegen und muss je nach Dateityp des angeforderten Dokuments unterschiedlich verfahren: Wenn es sich um ein statisches Dokument handelt (was für dieses Beispiel der Einfachheit halber angenommen werden soll), kann er es einfach ausliefern. Handelt es sich dagegen um die Adresse eines serverseitigen Skripts, muss er zunächst das zuständige Pro-

gramm aufrufen, dessen Ausgabe entgegennehmen und diese genau wie ein statisches Dokument an den Client liefern.

Bevor der Server das Dokument (resp. den Inhalt) sendet, schickt er eine Statusmeldung und einen HTTP-Header mit diversen Angaben, gefolgt von einer Leerzeile. Die Statusmeldung in der ersten Zeile der ausgegebenen Daten besteht aus der HTTP-Version, einer standardisierten Statuscodenummer und deren Textauflösung. Die beiden häufigsten Statusmeldungen dürften 200 OK (Dokument ist vorhanden und wird mitgeliefert) und 404 Not Found (Dokument existiert nicht) sein. In diesem Beispiel soll gelten, dass das Dokument gefunden wurde. Die erste Zeile der Antwort lautet also:

```
HTTP/1.1 200 OK
```

In der nächsten Zeile beginnt der HTTP-Header. Die Mindestanforderung für den Header einer HTTP-Antwort ist die Angabe des **MIME-Types** des gelieferten Dokuments. Im Fall eines HTML-Dokuments sieht die entsprechende Header-Zeile so aus:

```
Content-Type: text/html
```

Weitere häufig verwendete Header-Felder enthalten beispielsweise das Änderungs- und das Verfallsdatum des Dokuments oder die Versionsangabe des Servers oder Cookies. Sie werden weiter unten ausführlich beschrieben. Der Body der HTTP-Antwort, im vorliegenden Beispiel also das eigentliche Dokument, muss durch eine Leerzeile vom Header getrennt werden.

4. Der Client nimmt die HTTP-Antwort des Servers entgegen. Er interpretiert zunächst die empfangenen Header-Daten und anschliessend das eigentliche Dokument. Bei einem HTML-Dokument hat dies meist zur Folge, dass er **zahlreiche weitere HTTP-Anfragen** stellen muss, um die in das Dokument eingebetteten Bilder und Multimedia-Dateien anzufordern.
   Formal sind sowohl die HTTP-Anfrage als auch die Server-Antwort RFC-822 kompatible Nachrichten. Dieses Format wurde ursprünglich für E-Mails entwickelt. Es besteht aus beliebig vielen Header-Zeilen mit dem Format Headername; Wert; der Body (Nachrichteninhalt) wird durch eine Leerzeile von den Headern getrennt.
   Das RFC-822-Format wurde später durch MIME erweitert, um E-Mails in anderen Formaten als reinem Text und mit Dateianhängen versenden zu können. Eine MIME-Nachricht benötigt das zusätzliche Header-Feld Content-Type, in dem der Datentyp des Body-Inhalts angegeben wird.

2 Die HTTP-Client-Anfrage

Neben dem bereits erwähnten GET gibt es noch einige andere **HTTP-Befehle** oder Client-Methoden. Die meisten von ihnen wurden mit HTTP/1.0 eingeführt; einige wenige dagegen erst mit HTTP/1.1.
In der nachstehenden Tabelle sehen Sie zunächst eine Übersicht über alle HTTP-Methoden. Anschliessend werden einige von ihnen genauer erläutert und durch Beispiele verdeutlicht.

Tabelle Übersicht über die HTTP-Methoden:

http-Methode	Seit Version	Bedeutung
GET	0.9	Dokument anfordern
HEAD	1.0	Nur Header anfordern
POST	1.0	Dokument anfordern und umfangreiche Formulardaten senden
PUT	1.0	Dokument auf dem Server ablegen
DELETE	1.0	Dokument vom Server löschen
LINK	1.0	Verknüpfung erstellen
UNLINK	1.0	Verknüpfung löschen
TRACE	1.1	Proxies im Header zeigen
CONNECT	1.1	Proxy-Zugriff auf gesicherte Server
OPTIONS	1.1	Liste von Optionen anfordern

Die »Vorabversion« HTTP/0.9 unterstützte einzig und allein GET-Anfragen ohne zusätzliche Header.

Die wichtigste Neuerung von HTTP/1.1 gegenüber der Version 1.0 waren nicht die drei zusätzlichen Anfragemethoden (die nicht häufig verwendet werden). Viel bedeutender ist die neu geschaffene Möglichkeit, die HTTP-Verbindung zu einem bestimmten Server nach Abschluss einer Dokumentanforderung offen (persistent) zu halten. Diese Option ist nützlich, wenn ein HTML-Dokument zahlreiche Bilder enthält, die vom selben Server stammen und ebenfalls per HTTP-Anfrage angefordert werden müssen. Wenn jeweils die Zeit für den Verbindungsaufbau wegfällt, wird die Seite schneller geladen.

In Version1.1 wurde mit dem Header-Parameter `Host:` auch die Möglichkeit geschaffen, mehrere Sites (Domains) einer IP-Adresse zuzuordnen (virtuelle Hosts) und damit IP-Adressen zu sparen.

Das Syntaxschema jeder HTTP-Anfrage sieht folgendermassen aus:

```
Methode Dateipfad HTTP/Version [Headername: Wert ..][Leerzeile, Body]
```

Die eckigen Klammern bedeuten, dass die jeweiligen Teile optional sind. Falls die Methode einen Body besitzt, muss er gemäss RFC 822 auf jeden Fall durch eine Leerzeile vom Header getrennt werden. Erforderlich ist er auf jeden Fall bei POST und PUT.

3 Die Methode GET

Die wichtigste HTTP-Methode überhaupt ist GET. Sie dient dazu, den Webserver zur Auslieferung eines Dokuments oder der Ausgabe eines serverseitigen Programms aufzufordern.

Eine GET-Anfrage besteht nur aus dem **Befehl** selbst und einigen **optionalen Header-Feldern**, besitzt aber **keinen Body**. Wenn ein Browser beispielsweise das Dokument *info.html* im Webserver-Verzeichnis *seiten* anfordern möchte, sieht die Anfrage etwa so aus:

```
GET /seiten/info.html HTTP/1.1

Accept: */*
Accept-Language: de, en-US
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1)
Host: www.mynet.de
Connection: Keep-Alive
```

Befehlszeile

Header

Die Headerdaten bedeuten Folgendes:

- Accept: * / *.
  Der Browser akzeptiert Dokumente mit beliebigem MIME-Type (* / * bedeutet: beliebiger Typ/beliebiger Untertyp).
- Accept-Language: de. en–US.
  Der Browser bevorzugt Dokumente in den Sprachen Deutsch und US-Englisch (in dieser Reihenfolge).
- Accept-Encoding: gzip. deflate.
  Neben unkomprimierten Daten unterstützt der Browser auch Dateien, die mit GZIP beziehungsweise ZIP (deflate) komprimiert sind.
- User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1).
  Versionsangabe des Browsers. Dies dient dem Erstellen der bekannten Browser-Statistiken und ermöglicht es Webentwicklern, an verschiedene Browser angepasste Versionen von Dokumenten anzubieten. In diesem Beispiel handelt es sich um den Internet Explorer 9.0 unter Windows 7.
- **Host**: www.mynet.de.
  Zur Unterscheidung virtueller Hosts (mehrere Sites mit gleicher IP-Adresse). Da die IP-Datagramme nur die IP-Adresse, aber nicht den Hostnamen enthalten, weiss der Server nur durch dieses Header-Feld, welche konkrete Website gemeint ist.
- **Connection**: Keep-Alive.
  Die HTTP-Verbindung soll nach der Beantwortung der Anfrage offen gehalten werden (nur HTTP/1.1). Der Server kann allerdings nicht dazu gezwungen werden, die Verbindung beizubehalten; es handelt sich nur um eine Art Empfehlung.

Wenn die Anfrage erfolgreich beantwortet wird, könnte die Antwort des Servers so aussehen:

```
HTTP/1.1 200 OK                                                    Antwort

Date: Thu, 26 Apr 2011 13:50:25 GMT
Server: Apache/2.4.1
Last-Modified: Fri. 21 Nov 2010 01:45:53 GMT
ETag: "2e986a-b20-fe93242e"                                        Header
Content-Length: 1936
Connection: close
Content-Type: text/html

-- Leerzeile --
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//Transitional//EN">    Body
<html> <head> […]
```

Die Headerdaten bedeuten Folgendes:

- Date: Thu, 26 Apr 2011 13:50:25 GMT.
  Gibt Datum und Uhrzeit auf dem Server inklusive Zeitzone an.
- Server: Apache/2.4.1.
  Servertyp und Version des Servers (aus Sicherheitsgründen meist ohne Version).
- Last-Modified: Fri, 21 Nov 2010 01:45:53 GMT.
  Datum und Uhrzeit der letzten Änderung der Seite. Diese Angabe ermöglicht es dem Browser beispielsweise, zu entscheiden, ob die Kopie der Seite in seinem Cache noch aktuell ist.
- ETag: "2c986a-b20-fc93242e".
  Das ETag (Entity-Tag) wird vom Server erzeugt, wenn er entsprechend konfiguriert ist. Es bestätigt die Identität eines Dokuments über den URL beziehungsweise den Pfad hinaus. Dies ist nützlich für Caching- und Proxy-Zwecke.
- Content-Length: 1936.
  Länge des Bodys in Bytes. Der Browser kann anhand dieser Angabe überprüfen, ob die Lieferung vollständig ist.
- Connection: close.
  Der Server schliesst die Verbindung nach Abschluss dieser Anfrage.
- Content-Type: text /html.
  Der MIME-Type des Dokuments ist HTML, Haupttyp Text, Untertyp HTML.

Der Browser kann über eine GET-Anfrage **auch Formulardaten** übermitteln, das heisst, die Eingabe in ein HTML-Formular wird zusammen mit der Anfrage gesendet.
In der Formulardefinition im HTML-Dokument kann zu diesem Zweck festgelegt werden, ob das Formular mittels GET oder POST übertragen wird.
Bei der Formulardatenübertragung mit GET werden die Formulardaten hinter einem «?» an den angeforderten URL angehängt.

Wenn ein HTML-Formular seine Daten per GET versenden soll, benötigt das zuständige <form>-Tag ein Attribut namens method mit dem Wert "GET". Zum Beispiel wird das folgende Formular mittels GET an das PHP-Skript test. php versandt:

```
<form action="test.php" methode="GET">

...

</form>
```

Die Befehlszeile der HTTP-Anfrage, die beim Absenden dieses Formulars generiert wird, sieht so aus:

```
GET /test.php?Formulardaten HTTP/1.1
```

Der Anhang hinter dem Fragezeichen wird als **Query-String** bezeichnet. Die einzelnen Formularfelder haben die Form Name=Wert und werden durch &-Zeichen oder Semikola (;) voneinander getrennt. Zusätzlich müssen die eigentlichen Formulardaten URL-codiert werden, weil zahlreiche Zeichen in URLs nicht zulässig sind: Leerzeichen werden durch + ersetzt; fast alle Zeichen, die keine Buchstaben oder Ziffern sind, werden durch ein %-Zeichen und ihren hexadezimalen ASCII- beziehungsweise ANSI-Code ersetzt. Betrachten Sie beispielsweise folgende Formularfelder:

```
user=Peter Müller

age=35

place=Köln
```

Angenommen, das Formular wird an das PHP-Skript guest.php im Server-Verzeichnis gb versendet. Dann sieht die eigentliche Befehlszeile der entsprechenden GET-Anfrage so aus:

```
GET /gb/guest.php?user=Peter+M%FCller&age=35&place=K%F6ln HTTP/1.1
```

4 Die Methode POST

POST ist die bevorzugte Methode zur Übermittlung von Formulardaten an den Server. Anders als bei GET werden die Daten nicht an den URL des angesprochenen Skripts angehängt, sondern im **Body** der HTTP-Anfrage übertragen. Dies hat wichtige Vorteile:

- Die Datenmenge kann beliebig gross sein. Bei GET ist sie auf 2000 Zeichen beschränkt, da dies die maximale Länge für URLs ist.
- Die Formulardaten müssen nicht zwangsläufig URL-codiert werden wie bei GET. Statt dessen kann wie bei einer HTTP-Antwort ein Content -Type-Header angegeben werden - dies erledigt der Browser automatisch, wenn die HTML-Formulardefinition ein Attribut namens enctype enthält, das den entsprechenden MIME-Type angibt, zum Beispiel
  ```
  <form action="test.php" method="POST" enctype="application/x www-form-urlencoded">  ….
  </form>
  ```
  Diese Angabe sorgt für die standardmässige URL-Codierung. Eine interessantere Variante ist der **MIME-Type multipart/form-data**. Dabei handelt es sich um eine MIME-Multipart-Nachricht, die einer E-Mail mit Attachments (multipart/mixed) ähnelt. Formulardaten in diesem Format können aus mehreren Teilen mit jeweils eigenem Content - Type-Header bestehen. Dies ermöglicht vor allem **Datei-Uploads** per HTML-Formular.
- Wenn ein serverseitiges Skript per POST-Anfrage angefordert wird, ist garantiert, dass es bei jedem Aufruf neu ausgeführt wird. Am separaten Versand der Daten im Body der HTTP-Anfrage bemerken der Browser und die eventuell beteiligten Proxy-Server, dass diese Daten auf jeden Fall an das eigentliche serverseitige Programm ausgeliefert werden müssen.
  GET-Anfragen hängen die Formulardaten dagegen wie erwähnt an den URL an, so dass es sich formal gar nicht um Datenversand handelt. Bei serverseitigen Anwendungen ist es aber oft wichtig, dass sie auch wirklich bei jedem Aufruf ausgeführt werden, weil sie meist auch etwas hinter den Kulissen erledigen - zum Beispiel Daten in einer Datenbank ergänzen.
  Aus diesem Grund wird bei einer GET-Anfrage, die ein serverseitiges Skript aufruft, oft eine durch Programmierung erzeugte Zufallszahl oder Datum und Uhrzeit als Pseudoparameter angefügt: Da sich auf diese Weise unterschiedliche URLs wie script. php?dummy=52784 oder script. php?dummy=2003_1L 30_15_45 ergeben, kommt kein Browser oder Proxy-Server auf die Idee, einen weiteren Aufruf des Skripts für ein bereits im Cache befindliches Dokument zu halten.

Nehmen Sie als Beispiel an, die drei folgenden Formularfelder sollten mittels POST an das serverseitige Skript *bestell.php* im Verzeichnis *pizza* versandt werden:

```
groesse=m
belag=funghi
extra=Bitte schneiden!
```

Das zugehörige HTML-Formular ist folgendermassen definiert:

```
<form action="bestell.php" method="POST" enctype="application/x-www-form-urlencoded"> ....
</form>
```

Beim Absenden des Formulars wird aus diesen Angaben die folgende POST-Anfrage erzeugt (das enctype-Attribut sorgt auch hier für eine URL-Codierung der Formulardaten):

```
POST /pizza/bestell.php HTTP/1.1
```
Befehlszeile

```
Accept: */*
Accept-Language: de, en-US
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1)
Referer: http://www.mynet.de/pizza/bestell.html
Host: www.mynet.de
Content-Length: 47
Content-Type: application/x-www-form-urlencoded
```
Header

```
-- Leerzeile --
groesse=m&belag=funghi&extra=Bitte+schneiden%21
```
Body

Die meisten der diversen Header-Felder wurden bereits erläutert.
Bemerkenswert ist, dass in dieser Anfrage Content-Type und Content-Length vorkommen, die bei Anfragen wie GET oder HEAD nur in der Antwort vorkommen. Sie beschreiben wie bei einer HTTP-Antwort den (in diesem Fall aus der enctype-Angabe im <form>-Tag stammenden) Datentyp und die Länge des Body-Inhalts.
Der Header Referer gibt an, von welchem Dokument die Anfrage ausging, das heisst, auf welcher Seite der Hyperlink angeklickt oder das Formular abgeschickt wurde.

Wenn der Server eine POST-Anfrage erhält, reicht er die erhaltenen Formulardaten an das im URL angegebene Skript weiter. Der Server antwortet im Erfolgsfall wie bei einer GET-Anfrage mit einer HTTP-Antwort mit dem Statuscode 200 und einem Body, der ein Dokument enthält, das der Browser daraufhin anzeigt, zum Beispiel

```
HTTP/1.1 200 OK                                                    Antwort

Date: Thu, 26 Apr 2011 19:50:25 GMT
Server: Apache/2.4.1
Last-Modified: Fri. 21 Nov 2008 07:28:34 GMT
ETag: "394bcd-2c2-6d94b63a"                                        Header
Accept-Ranges: bytes
Connection: close
Content-Length: 2782 Content-Type: text/html

-- Leerzeile --
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//Transitional//EN">
<html>                                                             Body
<head>
[…]
```

5 Laborübung: Network Analyzer / Packet Sniffer

Zeitaufwand: ca. 60 Min.

Einführung

Der Network-Analyzer / Packet Sniffer ist eines der wichtigsten Tools eines Netzwerktechnikers und zwar nicht um in den Daten anderer rum zu schnüffeln, sondern um bei Problemen die Pakete und die Vorgänge im Netz bis in letzte Detail analysieren zu können. Der Analyzer kennt alle verschiedenen Header und Headerfelder der unterstützen Protokolle und stellt diese möglichst leicht lesbar dar. Die Interpretation diese Werte bleibt jedoch dem Menschen überlassen, das heisst zur Fehlerbehebung müssen sie selber die korrekten Werte kennen!

Durch einen speziellen Modus der Netzwerkkarte (Promiscous Mode) zeichnet der Network-Analyzer nicht nur die Datenübertragung des eigenen PCs auf, sondern sämtliche Pakete, die er auf dem Kabel sieht. Da in einem produktiven Netz oft sehr viele Daten übertragen werden, sind Filter ein wichtiges Feature von Analyzern. Damit definieren Sie, welche Daten für Sie interessant sind und aufgezeichnet werden (z.B. nur bestimmten Protokolle oder Adressen).

Ziele

- Sie verstehen, wozu ein Network-Analyzer dient und wie man ihn einsetzt.
- Sie können mit einem Network-Analyzer Daten aufzeichnen, die verschiedenen Header in den aufgezeichneten Daten erkennen und den OSI-Layers und Protokollen zuordnen.
- Sie können die Filter eines Network-Analyzers so konfigurieren, dass nur Pakete, die den Vorgaben entsprechen, aufgezeichnet werden, beispielsweise HTTP-Pakete.

Voraussetzungen

Sie kennen den Aufbau des OSI-Schichtmodells und können ein Protokollposter (Übersicht in der die Protokolle den Layers zugeordnet sind) lesen.

Werkzeuge / Vorbereitung

- Vernetzter Windows oder Linux PC mit installiertem Network-Analyzer (z.B. Wireshark + WinPcap).
- Protokollposter, um die Protokolle zuzuordnen.

Aufgabe

Lernen Sie die Bedienung eines Network-Analyzers kennen. Identifizieren Sie in einem aufgezeichneten Paket die Header der Protokolle in den verschiedenen Layers. Da Sie in einem produktiven Netz immer sehr viele Datenpakete aufzeichnen, ist es schwierig, die für Sie interessanten Pakete zu finden. Konfigurieren Sie zu die-

sem Zweck die Filter des Network-Analyzers. Führen Sie die im Abschnitt „Vorgehen“ beschriebenen Schritte durch und dokumentieren Sie diese mit Printscreens. Beantworten Sie anschliessend die Kontrollfragen.

Abgabe Dokumente

- Antworten zu den Kontrollfragen aus diesem Dokument
- Dokumentation der Network-Analyzer-Beispiele mit Printscreens

Abgabetermin

- Siehe BSCW

Vorgehen

Lesen Sie zuerst das Dokument HTTP-Protokoll als Einleitung zu dieser Laborübung durch. Starten Sie den Network-Analyzer. Starten Sie den Capture-Vorgang (Aufzeichnung). Nachdem Sie einige Pakete aufgezeichnet haben. Schauen Sie sich die Pakete an, die mit HTTP bezeichnet sind.

Konfigurieren Sie nun den Filter des Network-Analyzers so, dass Sie nur noch HTTP-Pakete, die von und zu Ihrem PC gehen, sehen. Dazu müssen Sie die Dokumentation/Help studieren.

Beschreiben Sie den Ablauf einer HTTP-Anfrage mit der entsprechenden Antwort an einen von ihnen gewählten Webserver, beispielsweise Wikipedia oder Google.

Rufen Sie eine Website bei der Sie sich einloggen müssen auf, beispielsweise BSCW.

Kontroll-Fragen

- Was für verschiedene Fenster sehen Sie in ihrem Netzwerk-Analyzer und was stellen diese dar?

- Wie überprüfen Sie ihren Filter?

- Welche Statuscodes liefert ihnen der Webserver?

- Welche Informationen gibt ihr Browser im Internet bekannt?
  Könnte dies ein Problem sein?

- Wie werden Logins und Passwörter übermittelt?
  Welche Konsequenzen ziehen Sie daraus?

- Ergeben sich Konsequenzen aus diesen Erkenntnissen?