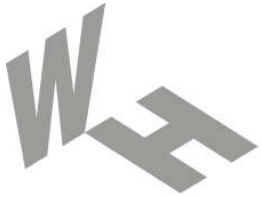


Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

Hypertext Transfer Protocol (HTTP)

Prof. Dr. (TU NN)
Norbert Pohlmann

Institut für Internet-Sicherheit – if(is)
Westfälische Hochschule, Gelsenkirchen
http://www.internet-sicherheit.de

if(is) internet-sicherheit.

Inhalt

- Ziele und Einordnung
- Das World Wide Web
- HTTP - Hypertext Transfer Protocol
  - HTTP - Methoden / Operationen
  - HTTP - Nachrichten
  - HTTP - Verbindungen
- Transport Layer Security (TLS) oder SSL (hier nur Idee)
- HTTP - Caching
- HTTP - Server-Cluster
- Zusammenfassung

Inhalt

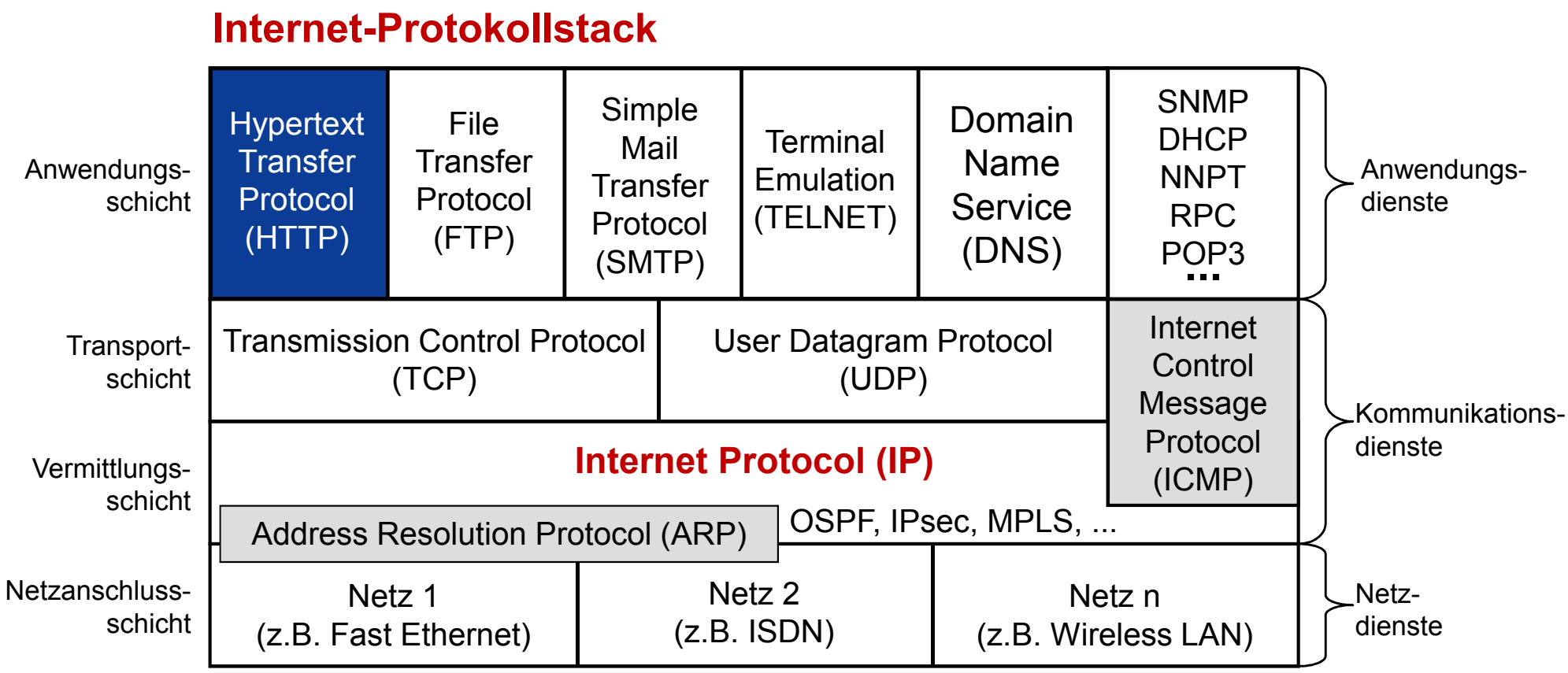
- **Ziele und Einordnung**

- Das World Wide Web
- HTTP - Hypertext Transfer Protocol
  - HTTP - Methoden / Operationen
  - HTTP - Nachrichten
  - HTTP - Verbindungen
- Transport Layer Security (TLS) oder SSL (hier nur Idee)
- HTTP - Caching
- HTTP - Server-Cluster
- Zusammenfassung

Hypertext Transfer Protocol (HTTP)
→ Ziele

- Gutes Verständnis für eines der wichtigsten Anwendungsprotokolle in der TCP/IP-Kommunikationsarchitektur
- Erlangen der Kenntnisse über die Aufgaben, Prinzipien und Mechanismen des HTTP-Protokolls
- Gewinnen von praktischen Erfahrungen über das HTTP-Protokoll mit Hilfe von Protokollanalysen und Statistiken (IAS)

Die Anwendungsebene
→ Hypertext Transfer Protocol (HTTP) - Einordnung



Inhalt

- Ziele und Einordnung
- **Das World Wide Web**
- HTTP - Hypertext Transfer Protocol
  - HTTP - Methoden / Operationen
  - HTTP - Nachrichten
  - HTTP - Verbindungen
- Transport Layer Security (TLS) oder SSL (hier nur Idee)
- HTTP - Caching
- HTTP - Server-Cluster
- Zusammenfassung

Das World Wide Web
→ Einleitung (1/5)

- Das **World Wide Web (WWW)** ist ein **riesiges verteiltes System**, das aus Millionen von Clients und Servern besteht, die auf verknüpfte Dokumente zugreifen.
- Die Server verwalten die Dokumente, während die Clients den Benutzern eine einfache Schnittstelle für die Darstellung und den Zugriff auf diese Dokumente bereitstellen.
- Seine enorme Beliebtheit ist auf die Tatsache zurückzuführen, dass es eine mehrfarbige grafische Benutzeroberfläche hat, die für Anfänger leicht zu bedienen ist, und dass es eine unglaubliche Fülle von Informationen über jedes erdenkliche Thema bietet.
- Das Web (WWW) begann 1989 am CERN, dem europäischen Zentrum für Atomphysik.
- Im Jahre 1994 unterzeichneten CERN und das MIT eine Vereinbarung über die Gründung des ***World Wide Web Consortium*** (***W3C***), einer Organisation zur Weiterentwicklung des Webs, zur Standardisierung von Protokollen und zur Förderung der Interoperabilität zwischen Webseiten.
- Die Homepage des Konsortiums befindet sich unter http://www.w3.org.

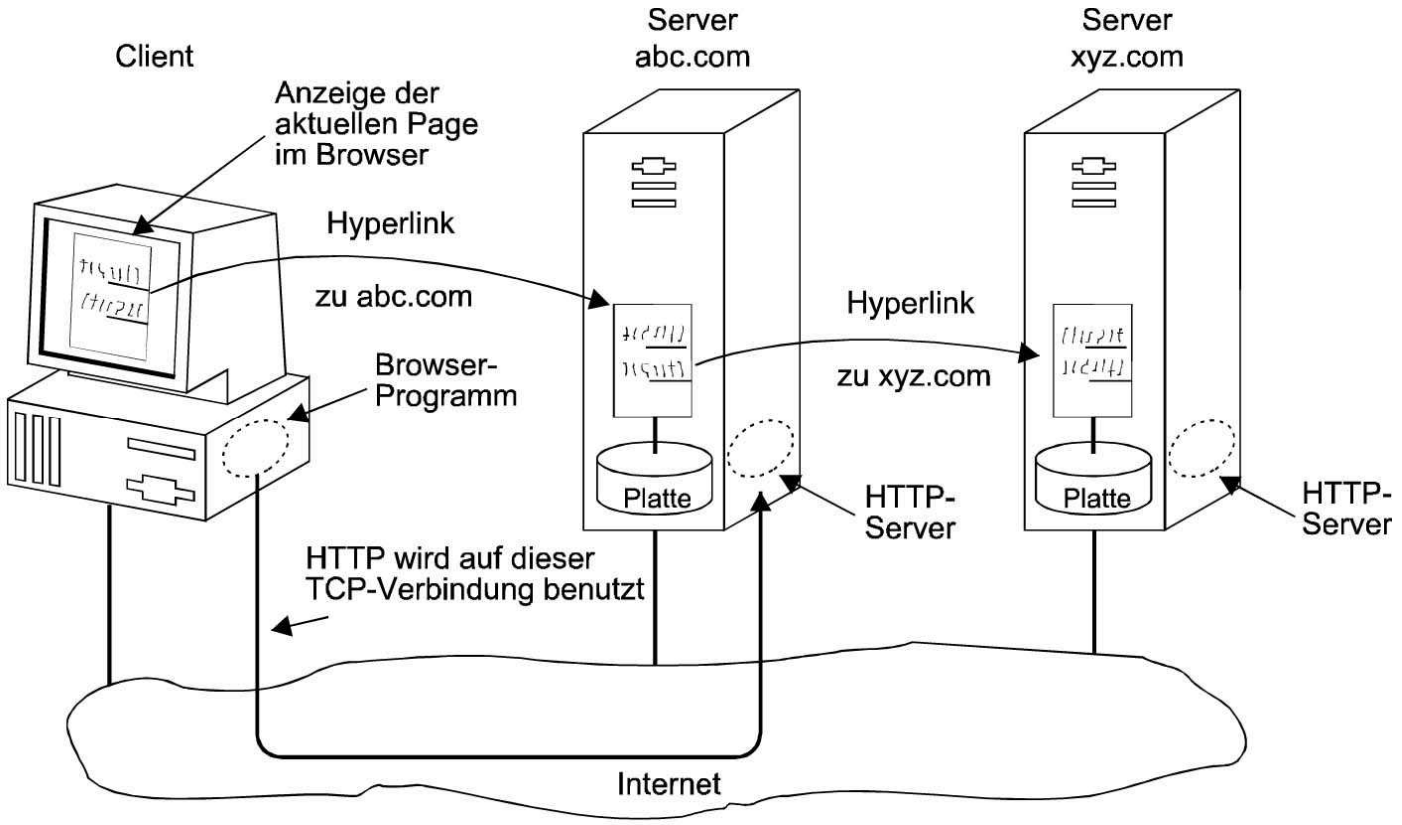
Das World Wide Web
→ Einleitung (2/5)

- Aus der Sicht des Benutzers besteht das Web aus einer riesigen weltweiten Sammlung von Dokumenten, die Webseiten genannt werden.
- Jede Seite kann Links (Zeiger) zu anderen Webseiten enthalten, die sich irgendwo auf der Welt befinden.
- Die Benutzer können einen Link, z.B. durch Anklicken folgen, um zu der betreffenden Seite zu gelangen.
- Dieser Prozess kann unendlich oft wiederholt werden.
- Das Konzept, dass eine Seite auf eine andere zeigt, wird als **Hypertext** bezeichnet.
- Webseiten werden in einem Programm namens **Browser** angezeigt.
- Mozilla Firefox und Internet Explorer sind hier die gängigen Lösungen.
- Der Browser holt die angeforderte Seite, interpretiert den Text und die enthaltenen Formatierungsbefehle und zeigt die Seite richtig formatiert am Bildschirm.

Das World Wide Web
→ Einleitung (3/5)

- Textstellen, die Links zu anderen Seiten bilden, heißen **Hyperlinks**.
- Hyperlinks sind am Bildschirm durch Unterstreichen, eine bestimmte Farbe oder beides hervorgehoben.
- Um einem Link zu folgen, setzt der Benutzer den Mauszeiger auf den markierten Bereich, wodurch sich der Cursor verändert, und klickt darauf.
- Die neue Seite kann sich auf dem gleichen Rechner wie die erste befinden oder auf irgendeinem anderen Rechner auf der Welt.
- Der Benutzer kann dies nicht erkennen.
- Seiten werden ohne weiteres Zutun des Benutzers abgerufen.
- Kehrt der Benutzer wieder zur Hauptseite zurück, werden alle Links, die er während der Sitzung durchlaufen hat, anders hervorgehoben (z.B. in einer anderen Farbe), um sie von den Links, die noch nicht verfolgt wurden, zu unterscheiden.

Das World Wide Web
→ Einleitung (4/5)



- Klickt ein Benutzer auf eine Textzeile, die mit einer Seite auf dem **Server abc.com** verknüpft ist, folgt der Browser dem Hyperlink, indem er eine Nachricht an den abc.com Server sendet und die Seite anfordert.
- Kommt die Seite an, wird sie angezeigt.
- Enthält diese Seite einen Hyperlink auf eine Seite auf dem **xyz.com-Server**, sendet der Browser eine Anforderung an diesen Rechner für diese Seite und so weiter.

Das World Wide Web
→ Einleitung (5/5)

- Im Wesentlichen ist ein Browser ein Programm, das eine Webseite abrufen und anzeigen, sowie Mausklicks auf Elemente auf der angezeigten Seite interpretieren kann.
- Wird ein Element ausgewählt, folgt der Browser dem Hyperlink und ruft die ausgewählte Seite ab.
- Daher benötigen eingebettete Hyperlinks eine Möglichkeit, eine andere Seite im Web zu benennen.
- Seiten werden mit Hilfe von **URLs (Uniform Resource Locators)** benannt.

Das World Wide Web
→ Uniform Resouce Locator (URL)

Protokoll	Domain-Name			/Pfad auf dem Server	
Protokoll	3rd Level	2nd Level	1st Level	/Pfad	/Datei
http://	**www.**	**mcs24.**	**de**	**/manual**	**/index.htm**

Protokoll	Das Übertragungsprotokoll gibt an, welcher Internet-Dienst gestartet werden soll. Wenn das Protokoll nicht angegeben ist, wird vom Browser automatisch http:// ergänzt. Folgende Dienste können eingegeben werden: **http://** steht für Hypertext Transfer Protocol https:// steht für Secure HTTP ftp:// steht für File Transfer Protocol ... Kann ein Browser ein bestimmtes Protokoll nicht darstellen, startet er automatisch das zugehörige Client-Programm
3rd Level D.	**www** verweist als Subdomain eindeutig auf einen Host mit WWW-Inhalten
2nd Level D.	**mcs24** ist ebenfalls ein Subdomain und weist auf den Betreiber des Server hin.
1st Level D.	**de** verweist auf das Land, in dem die 1st. Level Domäne vergeben wurde.
Pfad	Der Pfad gibt das **Verzeichnis auf dem Server** an, in welchem sich die Datei befindet. Pfadangaben werden mit „/“ voneinander getrennt.
Datei	Der Dateiname gibt an, welche Datei im angegebenen Verzeichnis aufgerufen werden soll. Der Deiteiname **index.htm** kennzeichnet die Homepage bzw. Startseite eines Verzeichnis. Wenn in der URL ein Dateiname fehlt, wird automatisch eine Datei mit dem Namen index.htm oder index.html gesucht.

Das World Wide Web
→ Dokumenttypen

- Der Dokumenttyp wird häufig in Form eines MIME-Typs ausgedrückt.
- **MIME** steht für **Multipurpose Internet Mail Extension** und wurde ursprünglich entwickelt, um Informationen über den Inhalt eines Nachrichtenrumpfes bereitzustellen, der als Teil von E-Mails versendet wurde.
- MIME wird aber auch für Web-Seiten genutzt.
- MIME unterscheidet zwischen übergeordneten Typen und untergeordneten Typen.

Das World Wide Web
→ Dokumenttypen (Auswahl)

Typ	Untergeordneter Typ	Beschreibung
Text	Plain	Nicht formatierter Text
	HTML	Text mit HTML-Befehlen
	XML	Text mit XML-Befehlen
Image	GIF	Bild im GIF-Format
	JPEG	Bild im JPEG-Format
Audio	Basic	Audio, 8-Bit-PCM, mit 8000 Hz gesamplet
	Tone	Ein bestimmter hörbarer Ton
Video	MPEG	Film im MPEG-Format
	Pointer	Darstellung eines Zeigegeräts für Präsentationen
Application	Octet-Stream	Eine nicht interpretierte Byte-Folge
	Postscript	Druckbares Dokument in Postscript
	PDF	Druckbares Dokument in PDF
Multipart	Mixed	Unabhängige Teile in vorgegebener Reihenfolge
	Parallel	Die Teile müssen gleichzeitig angezeigt werden

Inhalt

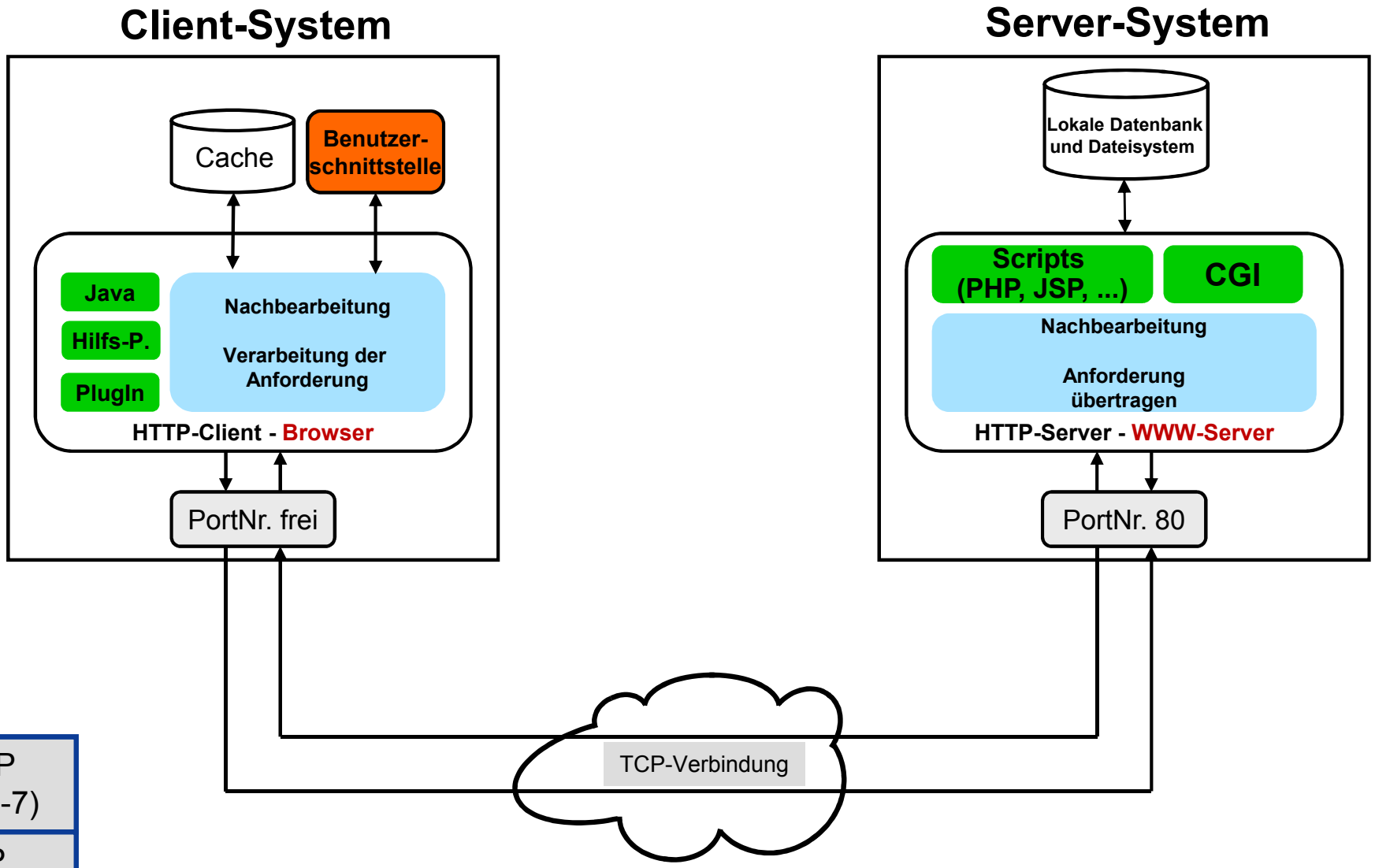
- Ziele und Einordnung
- Das World Wide Web
- **HTTP - Hypertext Transfer Protocol**
  - HTTP - Methoden / Operationen
  - HTTP - Nachrichten
  - HTTP - Verbindungen
- Transport Layer Security (TLS) oder SSL (hier nur Idee)
- HTTP - Caching
- HTTP - Server-Cluster
- Zusammenfassung

Hypertext Transfer Protocol (HTTP)
→ Standards und Literatur

- **RFCs**
  - RFC 1945 - HTTP 1.0
  - RFC 2616 - HTTP 1.1

- **Literatur**
  - Gourley, D; Tolly, B: **„HTTP - The Definitive Guide“**; O´Reilly, 2002; USA

Hypertext Transfer Protocol (HTTP)
→ Client-Server Beziehung

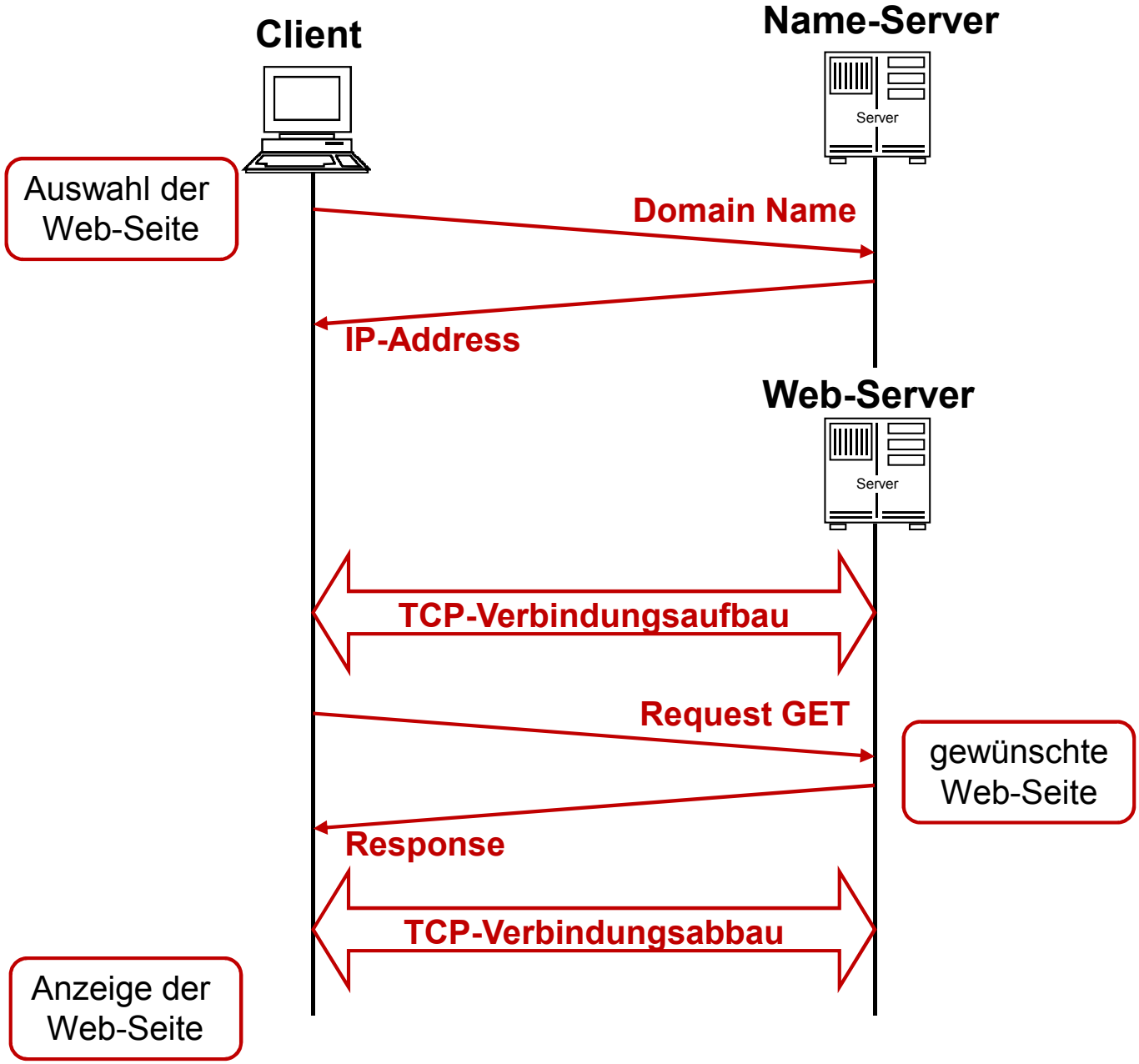


HTTP (OSI 5-7)
TCP
IP

Hypertext Transfer Protocol (HTTP)
→ Kommunikationsablauf (1/2)

- Der Benutzer aktiviert einen Hyperlink in einer Web-Seite, der Browser bestimmt daraufhin die dazugehörige URL Information.
- Der Browser fragt den Domain Name Service (DNS), um die entsprechende IP-Adresse zu ermitteln.
- Nachdem der Browser die IP-Adresse hat, baut er zu Port 80 eine TCP-Verbindung zum entspr. WWW-Server auf.
- Der Browser schickt die Seitenanforderung an den WWW-Server, der darauf die gewünschte Seite dem Browser über die etablierte TCP-Verbindung zuschickt.
- Die TCP-Verbindung wird wieder abgebaut.
- Der Browser bringt die Seite zur Anzeige auf dem Bildschirm des Benutzers.
- **Hinweis:**
  Die meisten Browser zeigen die Schritte, die durchgeführt werden, in einer Statuszeile am unteren Bildschirm an.

Hypertext Transfer Protocol (HTTP)
→ Kommunikationsablauf (2/2)

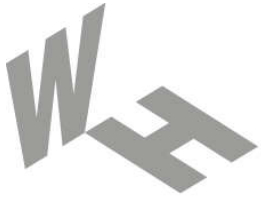


Hypertext Transfer Protocol (HTTP)
→ Kommunikation (1/2)

- Die gesamte Kommunikation im Web zwischen Client und Server basiert auf dem Hypertext Transfer Protocol (HTTP).
- HTTP ist ein einfaches Client-Server-Protokoll.
- HTTP ist ein zustandsloses Protokoll, d.h. es wird auf der Serverseite nicht gespeichert, welche Anfragen dem aktuellen Request vorangegangen sind.
- Serverseitig kann immer von demselben Zustand ausgegangen werden.
- Diese Eigenschaft ist insbesondere für den E-Commerce-Sektor ein Problem, das mit Hilfe von Cookies zu lösen versucht wird.
- Cookies sind jedoch nicht Bestandteil des HTTP-Standards.
- HTTP basiert auf TCP und nutzt den Port 80.

Hypertext Transfer Protocol (HTTP)
→ Kommunikation (2/2)

- Immer wenn ein Client eine Anforderung an einen Server absetzt, richtet er eine TCP-Verbindung zu dem Server ein und sendet die Anforderungsnachricht über die Verbindung.
- Dieselbe Verbindung wird verwendet, um die Antwort zu empfangen.
- Durch Verwendung von TCP als zugrunde liegendes Protokoll muss sich HTTP nicht um verlorene Anforderungen oder Antworten kümmern.
- Läuft etwas schief, wenn z.B. die Verbindung unterbrochen wird oder ein Timeout auftritt, wird ein Fehler gemeldet.
- Im Allgemeinen wird jedoch kein Versuch unternommen, eine Wiederherstellung von diesem Fehler vorzunehmen.



Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

HTTP - Methoden / Operationen

Prof. Dr. (TU NN)
Norbert Pohlmann

Institut für Internet-Sicherheit – if(is)
Westfälische Hochschule, Gelsenkirchen
http://www.internet-sicherheit.de

Hypertext Transfer Protocol (HTTP)
→ HTTP-Methoden - Überblick

- HTTP wurde als allgemeines Client-Server-Protokoll entwickelt, um Dokumente in beide Richtungen übertragen zu können.
- Ein Client kann jede Methode oder Operation, die auf dem Server ausgeführt werden soll, anfordern, indem er eine Anforderungsnachricht mit der gewünschten Operation an den Server sendet.
- Liste der gebräuchlichsten Anforderungsnachrichten:

Operationen	Beschreibung
Head	Anforderung, den Header eines Dokuments zurückzugeben
Get	Anforderung, ein Dokument an den Client zurückzugeben
Put	Anforderung, ein Dokument zu speichern
Post	Bereitstellung von Daten, die einem Dokument hinzugefügt werden sollen
Delete	Anforderung, ein Dokument zu löschen
Trace	Anforderung, eine Anfrage aus "Trace"-Gründen sofort zurückzusenden
Connect	reserviert für Proxy Server
Options	Anforderung, Eigenschaften des Servers und Dokumente abzufragen

Hypertext Transfer Protocol (HTTP)
→ HTTP-Methoden: „head"-Operation

- HTTP geht davon aus, dass jedem Dokument Metadaten zugeordnet sind, die in einem separaten Header gespeichert werden, der zusammen mit einer Anforderung oder Antwort gesendet wird.
- Die Operation „head" wird an den Server gesendet, wenn ein Client nicht das eigentliche Dokument benötigt, sondern nur die ihm zugeordneten Metadaten.
- Z.B. wird mit Hilfe der „head"-Operation die Zeit zurückgegeben, wann das betreffende Dokument geändert wurde.
- Diese Operation kann genutzt werden, um die Gültigkeit des Dokumentes zu überprüfen, wenn es sich im Cache des Client befindet (siehe PM Beispiel 1).
- Außerdem kann sie genutzt werden, um zu prüfen, ob es ein bestimmtes Dokument gibt, ohne das Dokument wirklich übertragen zu müssen.

Hypertext Transfer Protocol (HTTP)
→ HTTP-Methoden: „get"-Operation

- Die wichtigste und am häufigsten verwendete Operation ist „get"
- Diese Operation wird verwendet, um ein Dokument vom Server zu laden und es an den anfordernden Client zurückzugeben.
- Es kann spezifiziert werden, dass ein Dokument nur dann zurückgegeben werden soll, wenn es nach einer bestimmten Zeit verändert wurde.
- Darüber hinaus erlaubt HTTP, dass Dokumenten „Tags" zugeordnet werden, die als Zeichenstrings dargestellt werden, und lädt ein Dokument nur dann, wenn es bestimmten „Tags" entspricht.

Hypertext Transfer Protocol (HTTP)
→ HTTP-Methoden: „put"-Operation

- Die „put"-Operation ist das Gegenteil der „get"-Operation.
- Ein Client kann einen Server auffordern, ein Dokument unter einem bestimmten Namen zu speichern (der zusammen mit der Anforderung übergeben wird).
- Natürlich führt ein Server im Allgemeinen nicht blindlings „put"-Operationen aus, sondern akzeptiert nur Anforderungen von autorisierten Clients.

Hypertext Transfer Protocol (HTTP)
→ HTTP-Methoden: „post"-Operation

- Die „post"-Operation ist vergleichbar mit dem Speichern eines Dokumentes, außer dass ein Client anfordert, dass einem Dokument oder einer Dokumentenmenge Daten hinzugefügt werden.
- Ein typisches Beispiel ist das Posten eines Artikels in einer Newsgroup.
- Das unterscheidende Merkmal im Vergleich zu einer „put"-Operation ist, dass eine „post"-Operation angibt, welcher Dokumentengruppe ein Artikel „hinzugefügt" werden soll.
- Der Titel wird zusammen mit der Anforderung übergeben.
- Im Gegensatz dazu führt eine „put"-Operation ein Dokument sowie den Namen, unter dem der Server es speichern soll, mit sich.
- Außerdem werden die Inhalte von Eingabefeldern in Web-Seiten oft mit der „post"-Operation an die Server-Applikation gesendet (siehe PM Beispiel 5).

Hypertext Transfer Protocol (HTTP)
→ HTTP-Methoden: „delete"-Operation

- Die „delete"-Operation wird verwendet, um einen Server aufzufordern, das in der Nachricht angegebene Dokument zu entfernen.
- Ob das Löschen wirklich stattfindet, ist von den verschiedenen Sicherheitsmaßnahmen abhängig.

Hypertext Transfer Protocol (1.1)
→ HTTP-Methoden: „trace"-Operation

- Die „trace"-Operation dient dem Debuggen.
- Sie weist den Server an, die Anforderung zurückzusenden.
- Diese Operation ist nützlich, wenn Anforderungen nicht korrekt verarbeitet werden, und der Client wissen möchte, welche Anforderung der Server tatsächlich erhalten hat und wieviel Proxies (Gateways) es auf der Strecke zwischen Client und Server gibt (mit der Hilfe des Headers „Max-Forward).
- Jeder Proxy oder jeder Gateway hat die Möglichkeit, den Orginal-Header zu verändern.
- Mit der „trace"-Operation kann festgestellt werden, wie der Header letztendlich beim Server ankommt.

Hypertext Transfer Protocol (1.1)
→ HTTP-Methoden: „connect"-Operation

- Die „connect"-Operation ist reserviert für Proxy Server.
- Sie ermöglicht es einem Proxy dynamisch von einem Proxy zu einem Tunnel zu wechseln (Beispielsweise SSL Tunnel)

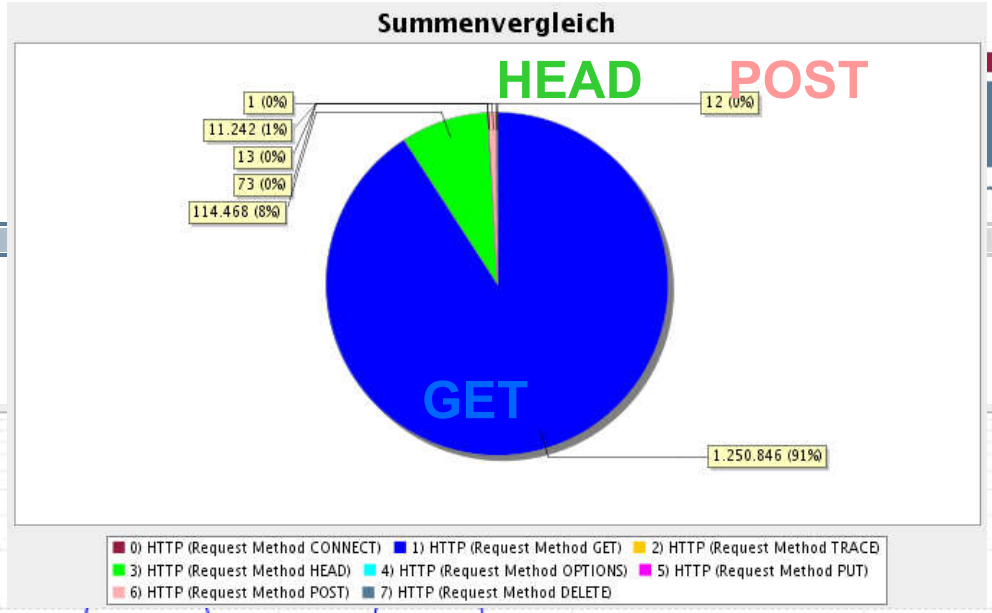
Hypertext Transfer Protocol (1.1)
→ HTTP-Methoden: „options"-Operation

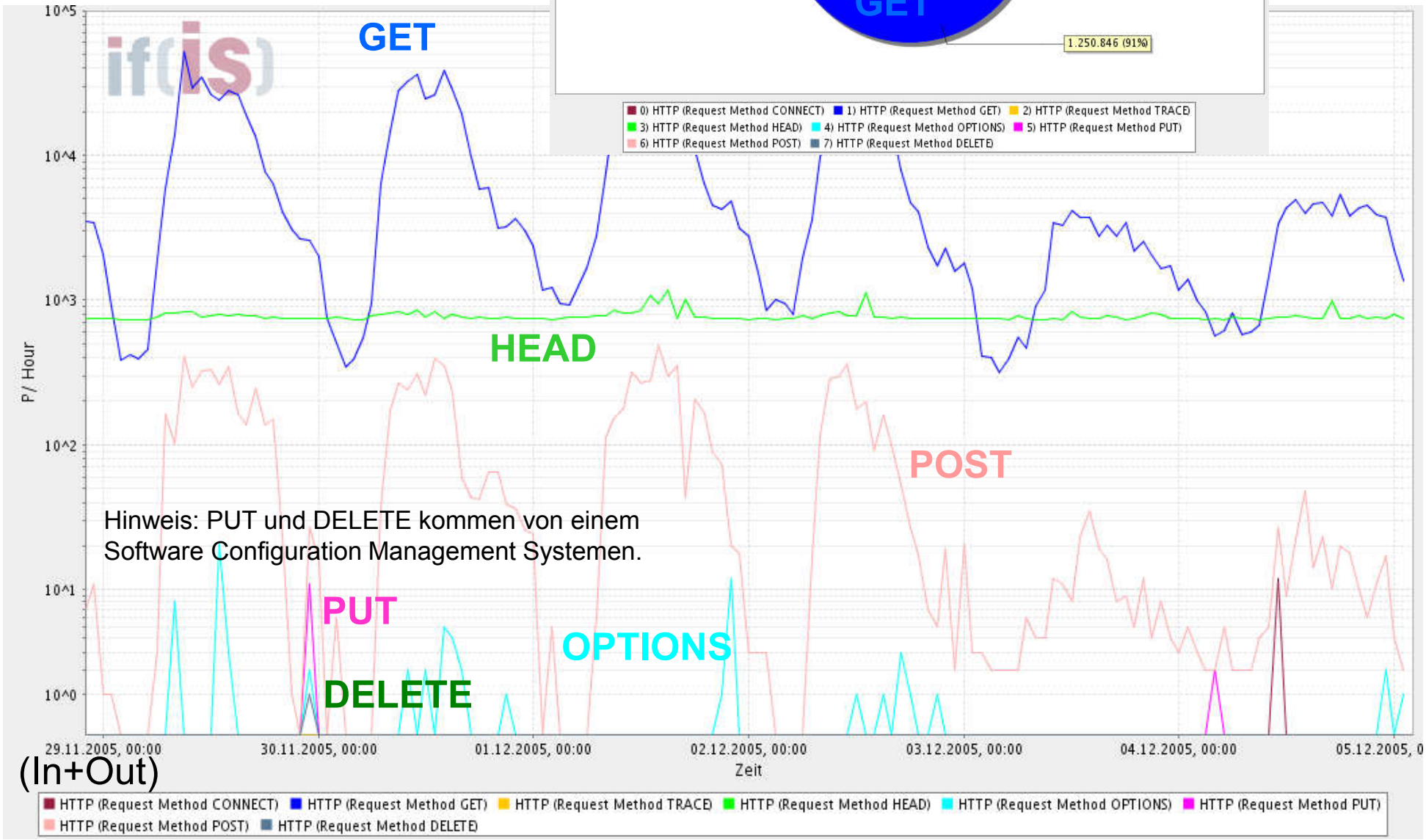
- Die „options"-Operation stellt für den Client eine Möglichkeit bereit, den Server über seine Eigenschaften oder die einer ausgewählten Datei abzufragen.
- Z.B. hat der Client die Möglichkeit mit dieser Operation abzufragen, welche Anfrage-Typen der Server unterstützt.
- Die „options"-Operation dient somit dazu, die Kompatibilität von Client und Server zu überprüfen.

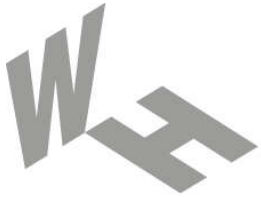
Hypertext Transfer Protocol (HTTP)
→ HTTP-Methoden - Zusammenfassung

- Die am häufigsten verwendeten Anfrage-Typen sind „get“, „post“ und „head“.
- Die Anfragen „put“ und „delete“ werden heutzutage noch nicht breit eingesetzt.
- Es ist aber zu erwarten, das sie zukünftig von großer Bedeutung sein werden.
- Mit HTTP kann mehr gemacht werden als zurzeit getan wird!

IAS: FB Informatik
→ HTTP-Methoden







Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

HTTP - Nachrichten

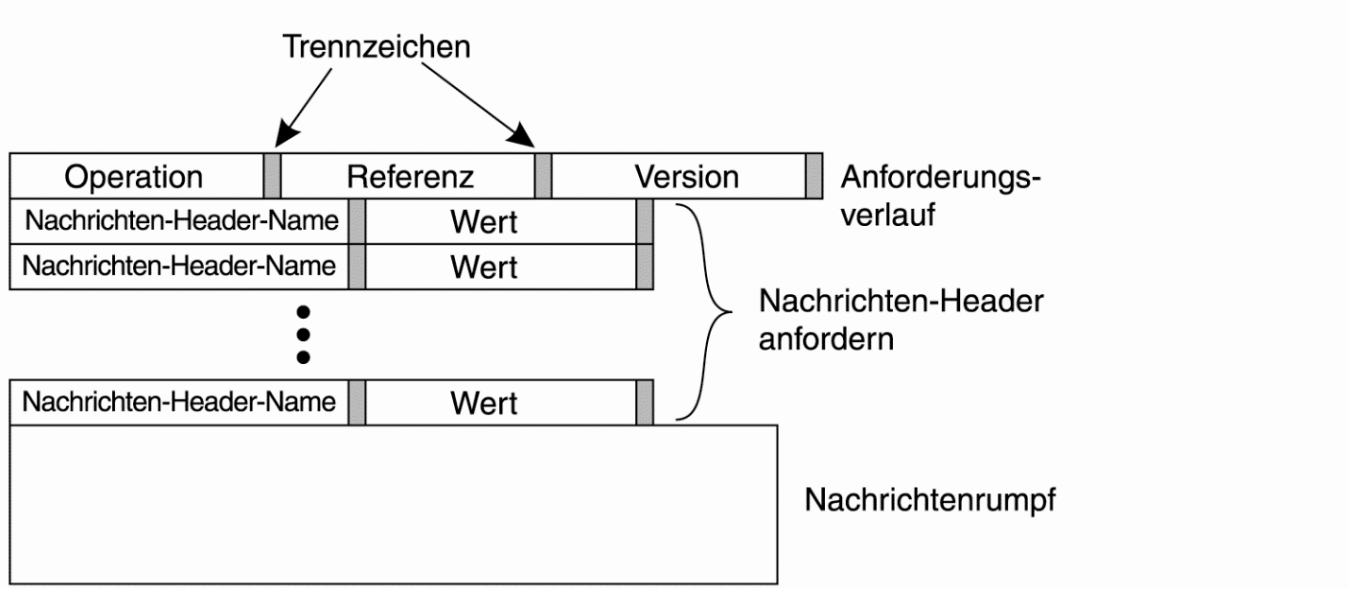
Prof. Dr. (TU NN)
Norbert Pohlmann

Institut für Internet-Sicherheit – if(is)
Westfälische Hochschule, Gelsenkirchen
http://www.internet-sicherheit.de

if(is) internet-sicherheit.

Hypertext Transfer Protocol (HTTP)
→ HTTP-Nachrichten - Aufbau der Anforderung (1/2)

- Die gesamte Kommunikation zwischen Client und einem Server findet über Nachrichten statt.
- HTTP kennt nur **Anforderungs- und Antwortnachrichten**.
- Eine Anforderungsnachricht besteht aus drei Teilen:

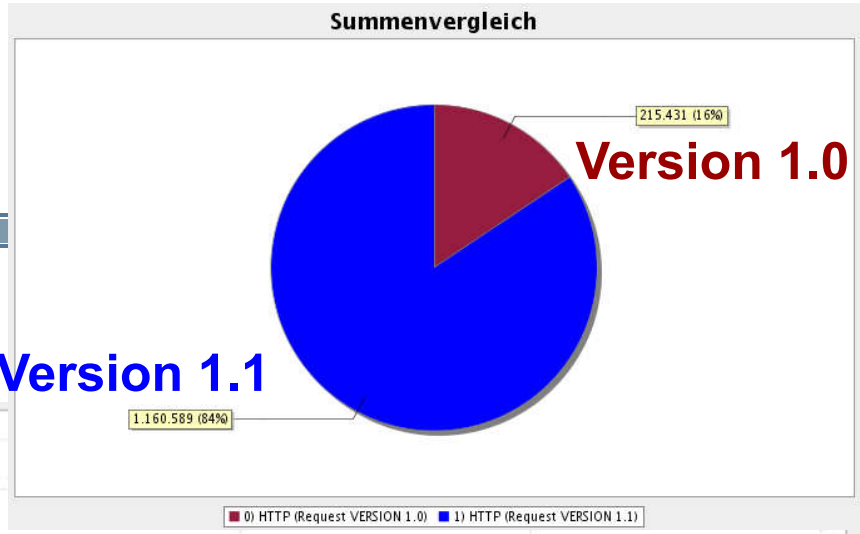


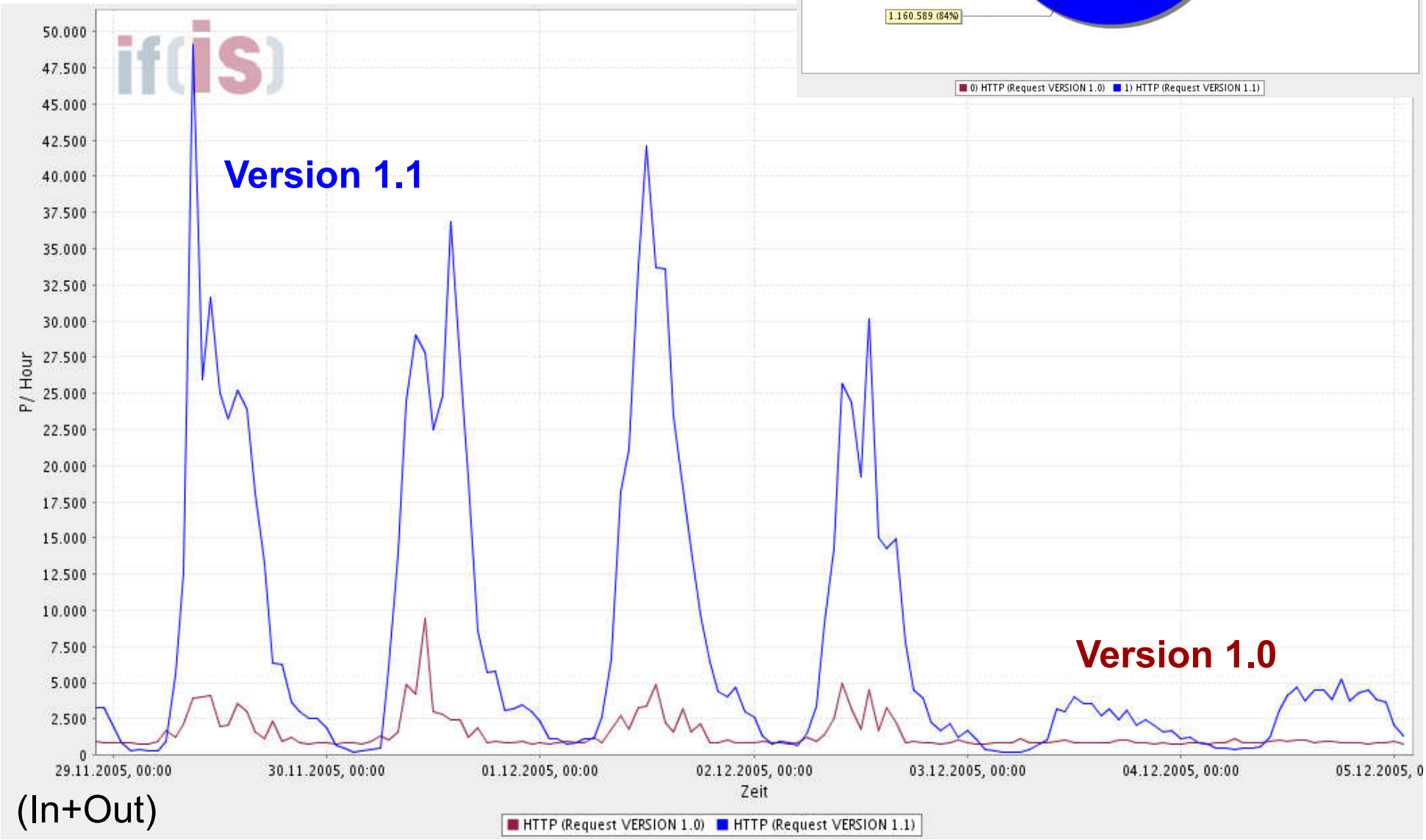
- **Anforderungsverlauf / Anforderungszeile**:
- **Nachrichten-Header**: Zusatzinformationen, die zwischen Client und Server ausgetauscht werden
- **Nachrichtenrumpf**: das eigentliche Dokument (z.B. PUT u. POST) in einem definierten Format

Hypertext Transfer Protocol (HTTP)
→ HTTP-Nachrichten - Aufbau der Anforderung (1/2)

- Die Anforderungszeile (Anforderungsverlauf) ist zwingend erforderlich und identifiziert die Operation, die der Client vom Server anfordert, zusammen mit einer Referenz auf das Dokument, das dieser Anforderung zugeordnet ist.
- Ein weiteres Feld wird verwendet, um die HTTP-Version zu identifizieren, die der Client erwartet.
- Anforderungszeile (Anforderungsverlauf)
  - Operation oder Methode: z.B. **GET**
  - Referenz (URL): z.B. **skripte.informatik.fh.gelsenkirchen.de**
  - Version: z.B **HTTP 1.1**

IAS: FB Informatik
→ HTTP-Methoden

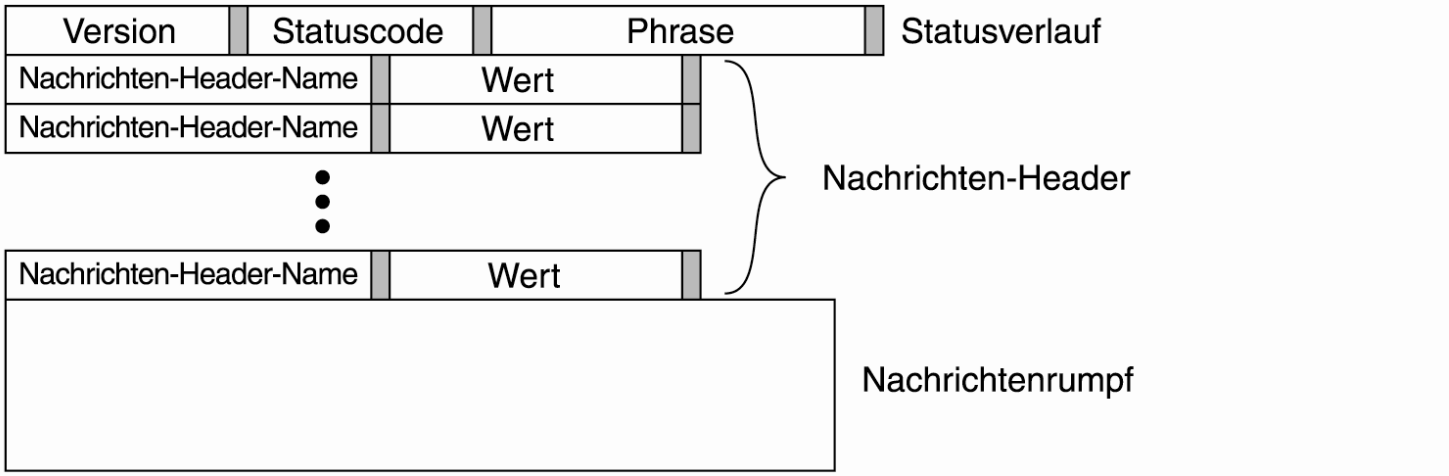




(In+Out)

Hypertext Transfer Protocol (HTTP)
→ HTTP-Nachrichten : Aufbau der Antwortnachricht (1/2)

- Eine Antwortnachricht beginnt mit einer Statuszeile, die eine Versionsnummer sowie einen dreistelligen Statuscode enthält.



- **Statusverlauf:**
- **Nachrichten-Header:** Zusatzinformationen, die zwischen Client und Server ausgetauscht werden
- **Nachrichtenrumpf:** das eigentliche Dokument (z.B. GET) in einem definierten Format

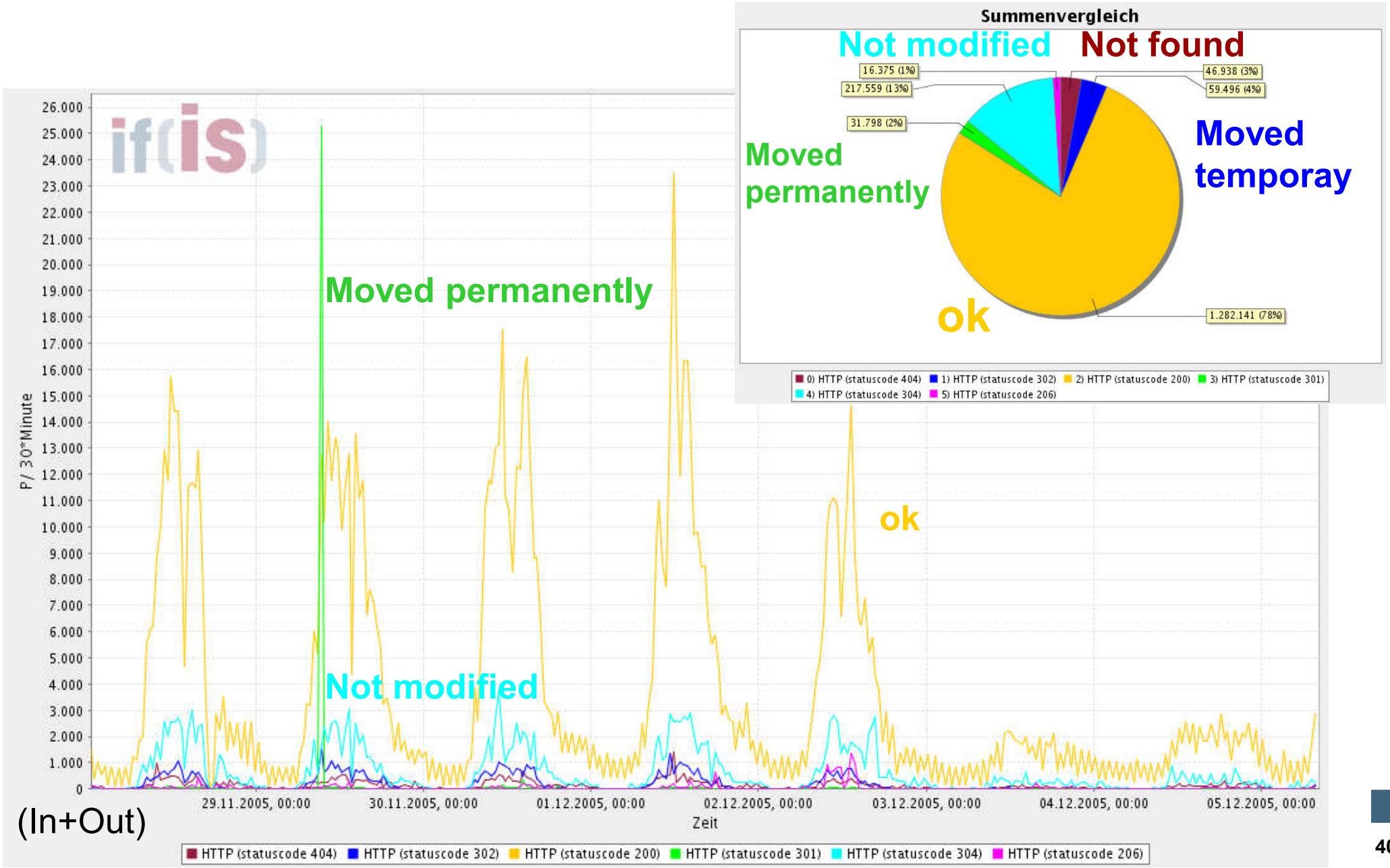
Hypertext Transfer Protocol (HTTP)
→ HTTP-Nachrichten : Aufbau der Antwortnachricht (2/2)

- Der Statusverlauf besteht aus:
  - Version: z.B **HTTP 1.1**
  - Statuscode: z.B. **200** *(Erfolgreiche Anforderung)*
  - Phrase: z.B. **OK** oder NOT OK *(Beschreibung des Statuscodes)*

- Die Bedeutung der Statuscodes:

Code	Bedeutung	Beispiel
1xx	Information	100 = Server stimmt zu, die Anforderung des Client zu bearbeiten
2xx	Erfolg	200 = ok; 204 = kein Inhalt vorhanden
3xx	Umleitung	301 = Seite verzogen; 304 = Seite im Cache noch gültig
4xx	Client-Fehler	403 = verbotene Seite; 404 = Seite nicht gefunden; 405 = Opera. nicht erlaubt
5xx	Server-Fehler	500 = interner Server-Fehler; 503 = versuch es später noch einmal

IAS: FB Informatik
→ HTTP-Statuscode



(In+Out)

Hypertext Transfer Protocol (HTTP)
→ HTTP-Nachrichten - Überblick

- Eine Anforderungs- oder Antwortnachricht kann zusätzlich Header enthalten.
- Hat ein Client z.B. eine post-Operation für ein schreibgeschütztes Dokument angefordert, reagiert der Server mit einer Nachricht 405 (Method Not Allowed), zusammen mit einem *Allow*-Nachrichten-Header, der die erlaubte Operation angibt (z.B. head und get).
- Als weiteres Beispiel könnte ein Client nur an einem Dokument interessiert sein, wenn seit einer bestimmten Zeit T dieses nicht verändert wurde.
- In diesem Fall wird die get-Anforderung des Clients durch ein *If-Modified-Sience*-Nachrichten-Header ergänzt, der den Wert T spezifiziert.

Hypertext Transfer Protocol (HTTP)
→ HTTP-Nachrichten : Header (1/2)

Header	Quelle	Inhalt
Accept	Client	Der Typ der Dokumente, die der Client verarbeiten kann
Accept-Charset	Client	Die für den Client erlaubten Zeichensätze
Accept-Encoding	Client	Die für den Client erlaubten Dokumentenkodierung
Accept-Language	Client	Die natürliche Sprache, die der Client verarbeiten kann
Accept-Ranges	Server	Der Server akzeptiert Byte-Bereichsanfragen
Authorization	Client	Liste der Berechtigungen des Clients
Cookie	Client	Sendet ein zuvor gesendetes Cookie an den Server zurück
Connection	Beide	Erlaubt Client und Server Info über den gewünschten Verbindungszustand
Content-Encoding	Server	Wie der Inhalt des Dokumentes kodiert ist (z.B. gzip)
Content-Language	Server	Natürliche Sprache des Dokumentes
Content-Length	Server	Seitenlänge in Byte
Content-Type	Server	MIME-Type des Dokumentes
Date	Beide	Datum und Zeit der gesendeten Nachricht
ETtag	Server	Die dem zurückgegebenen Dokument zugeordneten Tags
Expires	Server	Die Zeit, wie lange die Anwort gültig bleibt
From	Client	E-Mail-Adresse des Clients

Hypertext Transfer Protocol (HTTP)
→ HTTP-Nachrichten : Header (2/2)

Header	Quelle	Inhalt
Host	Client	DNS-Name des Web-Servers
If-Mach	Client	Die Tags, die das Dokument haben sollte
If-None-Match	Client	Die Tags, die das Dokument nicht haben sollte
If-Modified-Since	Client	Weist den Server an, ein Dokument nur dann zurückzugeben, wenn es seit der angegebenen Zeit verändert wurde.
If-Unmodified-Since	Client	Weist den Server an, ein Dokument nur dann zurückzugeben, wenn es seit der angegebenen Zeit nicht verändert wurde.
LastModified	Server	Die Zeit, wann das zurückgegebene Dokument zuletzt verändert wurde
Location	Server	Eine Dokumentenreferenz, an die der Client seine Anforderung umleiten soll
Max-Forward	Client	Bestimmt die max. Anzahl von Hops die zwischen Client und Server zulässig sind
Referer	Client	Verweist auf das vom Client zuletzt angeforderte Dokument
Server	Server	Informationen über den Server
Set-Cookie	Server	Der Server möchte, dass der Client ein Cookie speichert
Upgrade	Beide	Das Applikationsprotokoll, zu dem der Sender wechseln will
User-Agent	Client	Informationen über den Browser und dessen Plattform
Warning	Beide	Informationen über den Status der Daten in der Nachricht
WWW-Authenticate	Server	Sicherheitsanforderung, auf die der Client antworten soll

Hypertext Transfer Protocol (HTTP)
→ HTTP-Nachrichten - Beispiele (1/3)

- Nachrichten-Header können zusammen mit einer Anforderung oder Antwort gesendet werden.
- Es gibt verschiedene Header, die der Client einem Server senden kann, um zu erklären, welche Antwort er entgegennehmen kann.
- Z.B. könnte ein Client in der Lage sein, Antworten entgegenzunehmen, die mit dem Komprimierungswerkzeug „gzip“ komprimiert wurden.
- In diesem Fall sendet der Client einen *Accept-Encoding*-Nachrichten-Header zusammen mit seiner Anforderung, mit dem Inhalt „Accept-Encoding:gzip“.
- Analog dazu kann ein *Accept*-Nachrichten-Header verwendet werden, um etwa zu spezifizieren, dass nur HTML-Webseiten zurückgegeben werden können.
- Die *Location*-Nachricht-Header wird verwendet, um den Client auf ein anderes Dokument umzuleiten.

Hypertext Transfer Protocol (HTTP)
→ HTTP-Nachrichten - Beispiele (2/3)

- Der *Upgrade*-Nachrichten-Header wird verwendet, um zu einem anderen Protokoll zu wechseln.
  - Z.B. könnte Client und Server http/1.1 verwenden, um zunächst eine generische Möglichkeit zu besitzen, eine Verbindung einzurichten.
  - Der Server kann unmittelbar antworten, indem er dem Client mitteilt, dass er die Kommunikation mit einer sicheren Version von HTTP fortsetzen will, wie z.B. HTTPS.
  - In diesem Fall sendet der Server einen *Upgrade*-Header mit dem Inhalt „Upgrade: HTTPS“.
- Der *Host*-Header gibt dem Namen des Servers an. Er stammt aus dem URL. Der *Host*-Header ist zwingend erforderlich.
  - Er wird verwendet, weil einige IP-Adressen verschiedene DNS-Namen unterstützen können und der Server eine Möglichkeit haben muss, anzugeben, an welchen Host die Anforderung übergeben werden soll.

Hypertext Transfer Protocol (HTTP)
→ HTTP-Nachrichten - Beispiele (3/3)

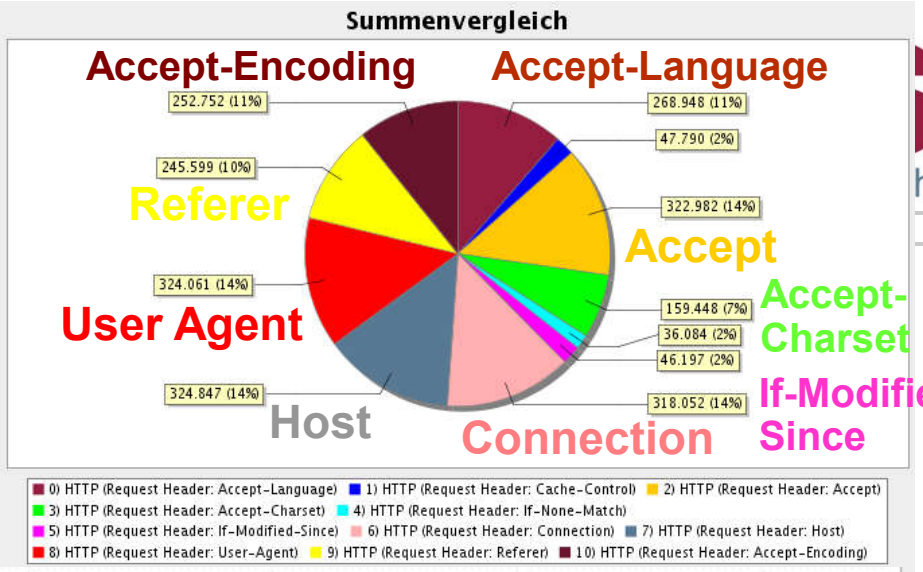
- Der *User-Agent*-Nachrichten-Header erlaubt dem Client, den Server über seinen Browser, das Betriebssystem und andere Eigenschaften zu informieren.

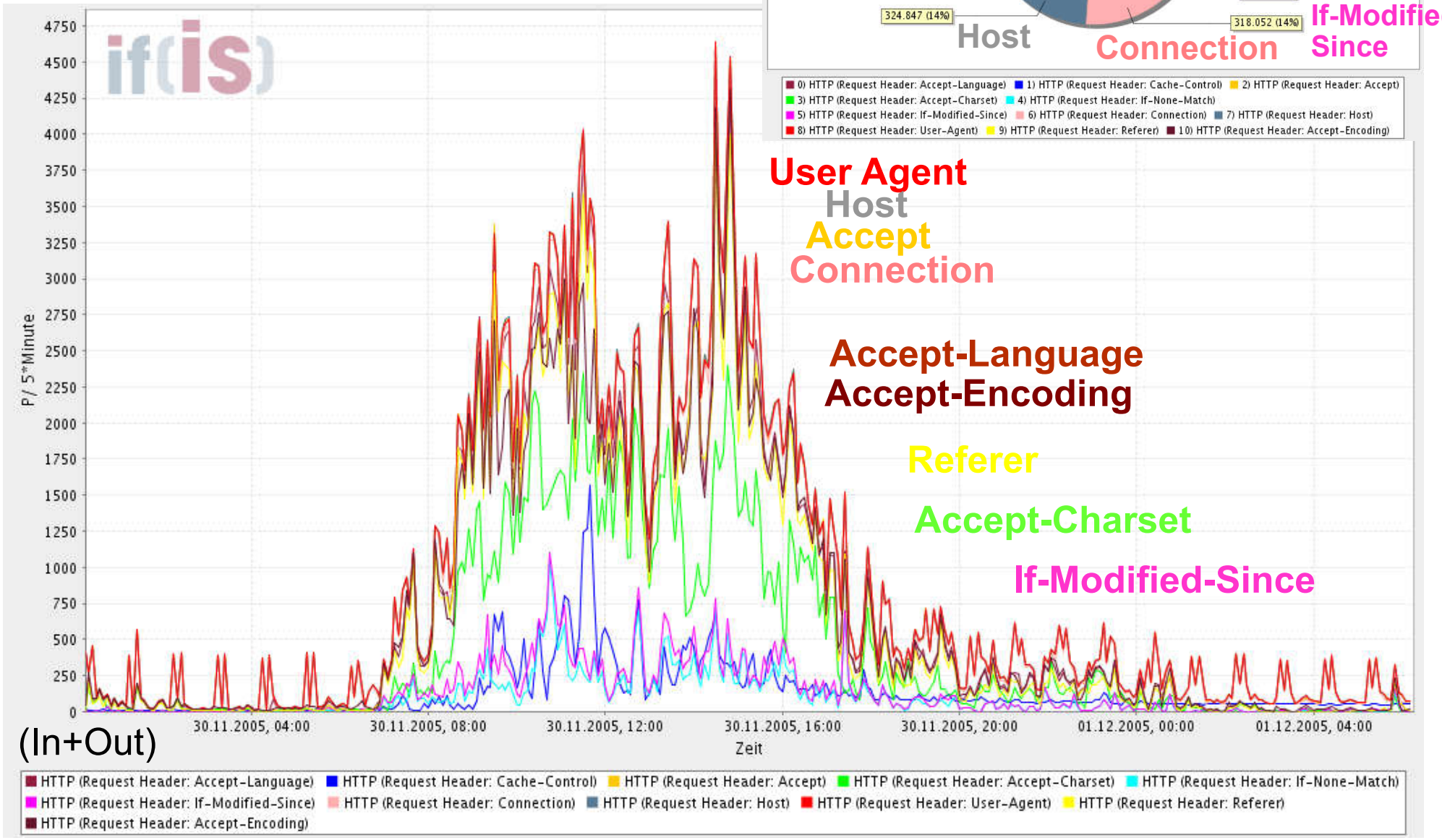
- Der *WWW-Authentication*-Nachrichten-Header gibt das vom Server geforderte Authentifizierungsschema und den Authentifizierungsbereich an.
  - Es wird zusammen mit dem Statuscode 401 (Unauthorized) übermittelt.
  - Ein oft benutztes Schema ist Basic, bei dem ein Benutzername und ein Passwort gefordert werden.

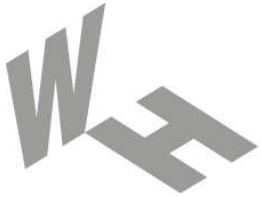
- **usw. → siehe Protokollmitschnitte der Beispiele!**

IAS: FB Informatik → HTTP-Header

Hinweis: Es kommen alle Header vor, hier nur größer als 2 %.







Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

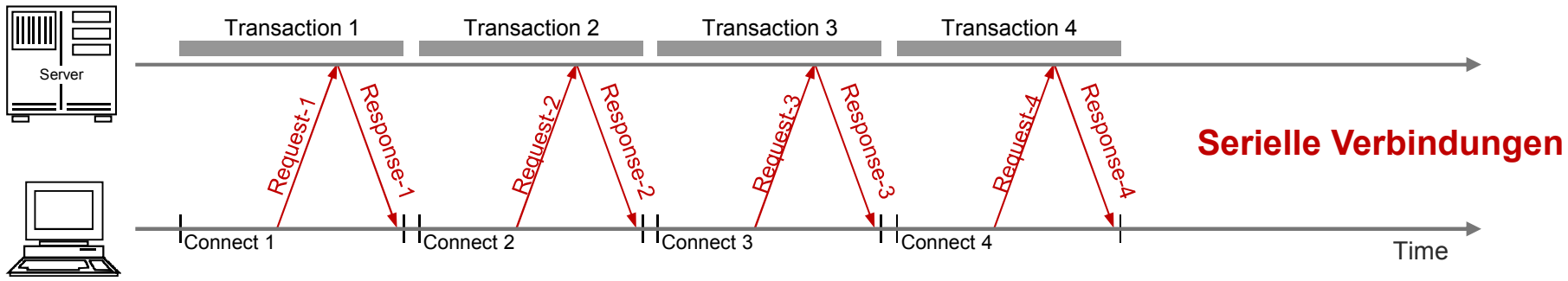
HTTP - Verbindungen

Prof. Dr. (TU NN)
Norbert Pohlmann

Institut für Internet-Sicherheit – if(is)
Westfälische Hochschule, Gelsenkirchen
http://www.internet-sicherheit.de

Hypertext Transfer Protocol (HTTP)
→ HTTP-Verbindungen - Übersicht (1/2)

- Eines der Probleme bei den ersten Versionen von HTTP war die ineffiziente Verwendung von TCP-Verbindungen.
- Jedes Webdokument setzt sich aus mehreren verschiedenen Dateien vom selben Server zusammen.
- Um ein Dokument korrekt anzuzeigen, ist es erforderlich, dass auch diese Dateien an den Client übertragen werden.
- Jede dieser Dateien ist im Prinzip ein weiteres Dokument, für das der Client eine separate Anforderung (z.B. GET) an den Server, auf dem sie abgelegt sind, absetzen kann.
- Wenn z.B. 12 Dateien mit 12 HTTP-Verbindungen hintereinander angefordert werden, kann die Übertragung sehr lange dauern.

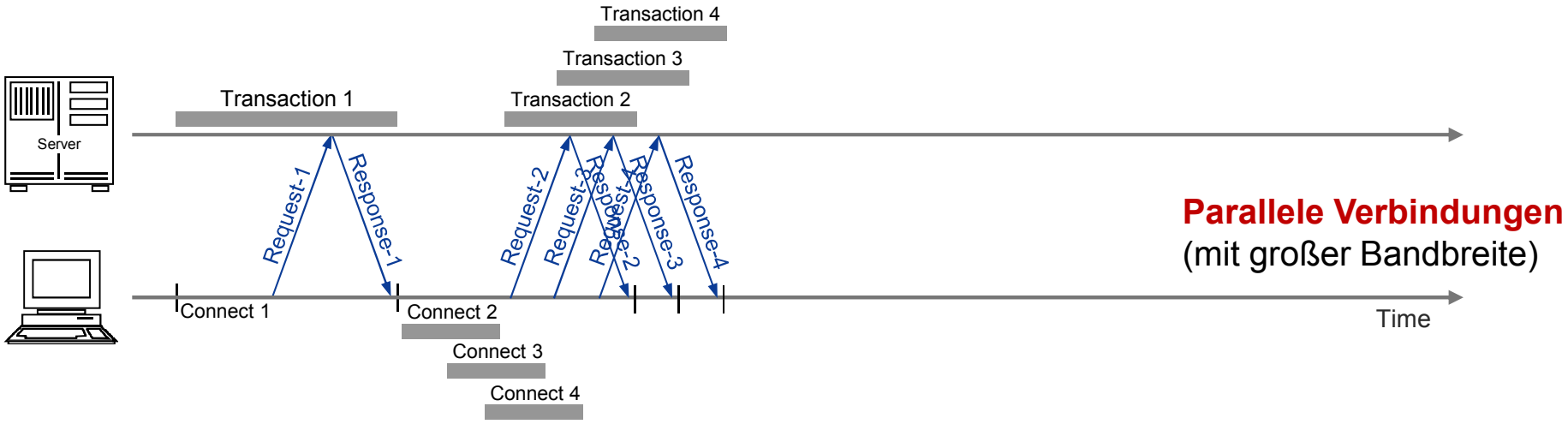


Hypertext Transfer Protocol (HTTP)
→ HTTP-Verbindungen - Übersicht (2/2)

- Da Web-Seiten oft viele Dateien beinhalten, kann die Übertragung aller zugehörigen Elemente (Dateien) sehr lange dauern.
- Es gibt einige Techniken, die genutzt werden können, um Übertragung zu optimieren:
  - Parallele Verbindungen
  - Persistente Verbindungen
  - Pipelined Verbindungen
  - Kombinationen

Hypertext Transfer Protocol (HTTP)
→ Parallele HTTP-Verbindungen

- HTTP ermöglicht den Cients verschiedene TCP-Verbindungen parallel aufzubauen.
- Damit ist die gesamte Übertragungszeit signifikant kürzer.

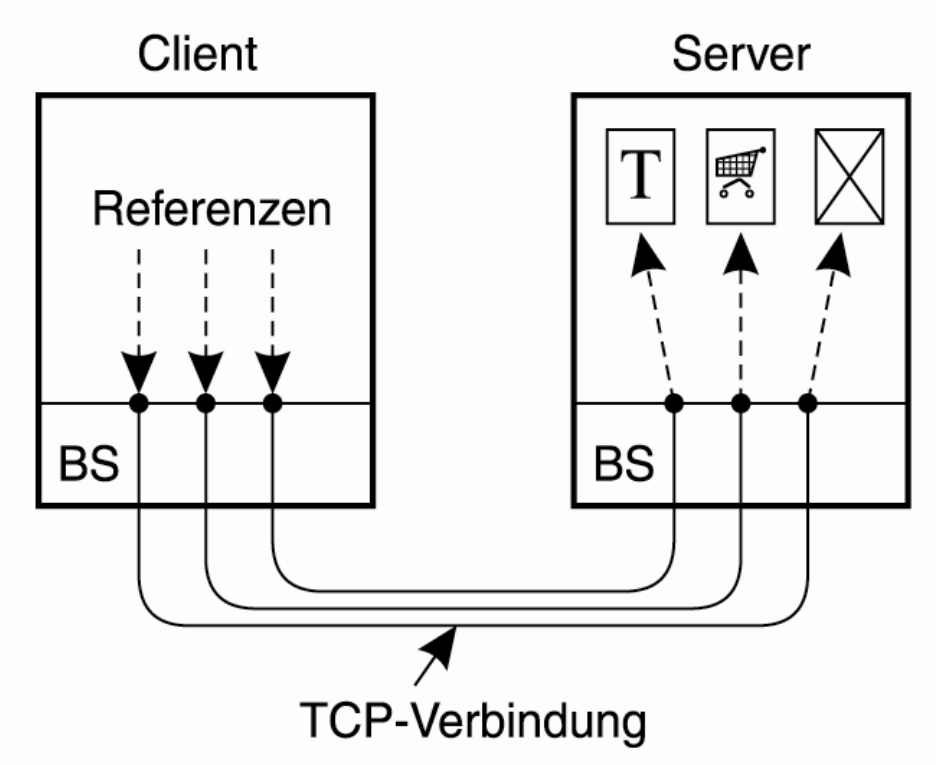


- Parallele Verbindungen müssen ***nicht*** immer schneller sein!
- Ist die ***Bandbreite***, die zur Verfügung steht, sehr klein, wirkt sich der Vorteil nur sehr gering aus.
- Auch die **Anzahl der offenen Verbindungen**, kann auf der Server-Seite zu **Performance-Problemen** führen (100 gleichzeitige Benutzer haben 100 TCP-Verbindungen auf bedeutet für den Server 10.000 Verbindungen!).

Hypertext Transfer Protocol (HTTP)
→ Persistente HTTP-Verbindungen (1/3)

- In HTTP Version 1.0 und älter wurde für jede Anforderung an einen Server eine separate Verbindung eingerichtet.

- Hatte der Server geantwortet, wurde die Verbindung wieder abgebaut.
- Solche Verbindungen werden auch als **nicht persistent** bezeichnet (parallele Verbindungen).
- Ein großer Nachteil von nicht persistenten Verbindungen ist, dass es relativ kostspielig ist, eine TCP-Verbindung einzurichten.

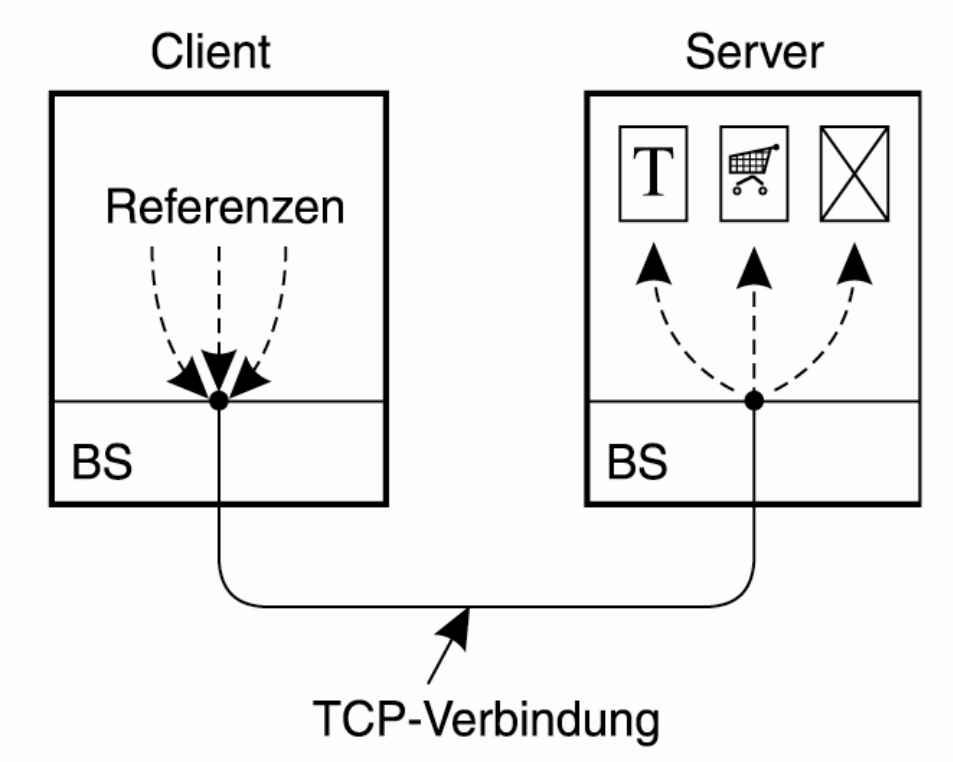


- Demzufolge kann es eine **wesentliche Zeit dauern**, ein gesamtes Dokument mit all seinen Elementen an einen Client zu übertragen.

Hypertext Transfer Protocol (HTTP)
→ Persistente HTTP-Verbindungen (2/3)

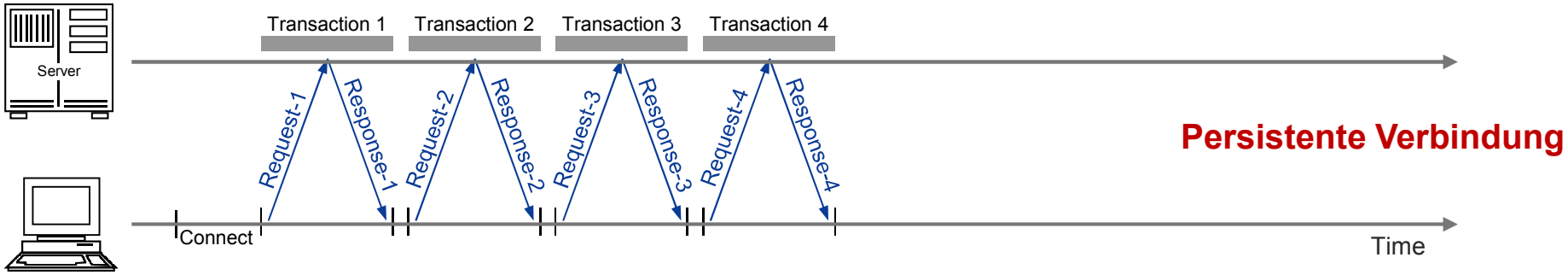
- Ein besserer Ansatz, der in HTTP Version 1.1 angewendet wird, ist die Verwendung einer **persistenten Verbindung**, die genutzt werden kann, um mehrere Anforderungen abzusetzen (und ihre jeweiligen Antworten zu empfangen), ohne dass eine separate Verbindung pro (Anforderungs-, Antwort-) Paar benötigt wird.

- Bei persistenten Verbindungen werden die Auf- und Abbauzeiten der parallelen Verbindungen gespart.
- Dieser Ansatz ist besonders günstig bei geringen Bandbreiten und bei der Verwendung von **„Slow Start“**.

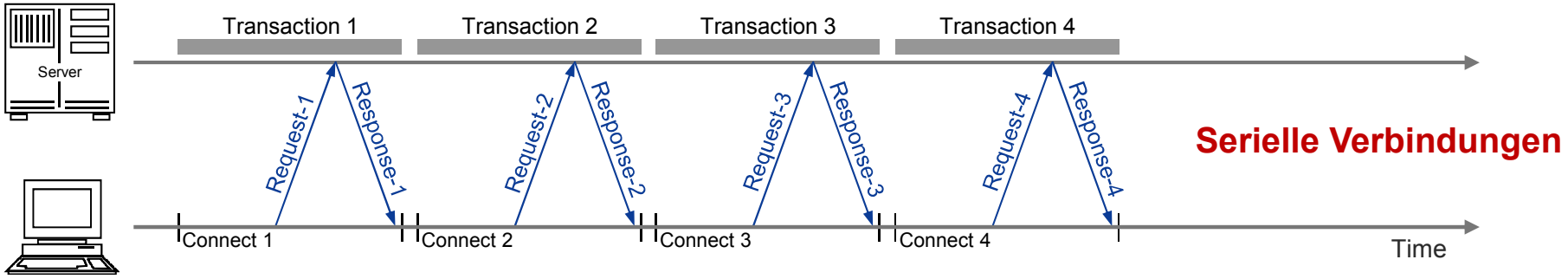


Hypertext Transfer Protocol (HTTP)
→ Persistente HTTP-Verbindungen (3/3)

- Persistente Verbindungen bleiben offen für alle Transaktionen, bis entweder der Client oder der Server die Verbindung schließt.

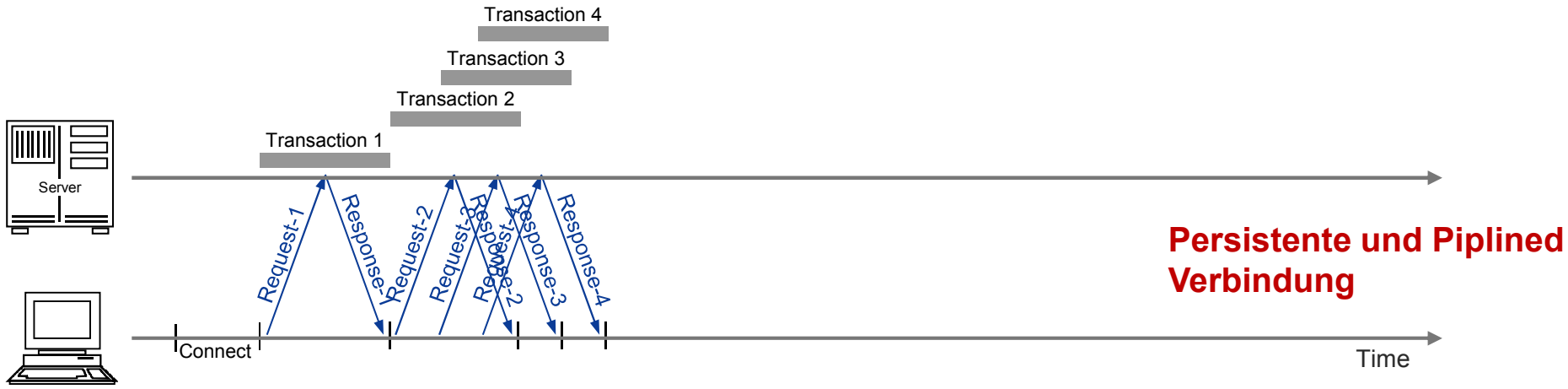


- Zum Vergleich eine serielle Verbindung:

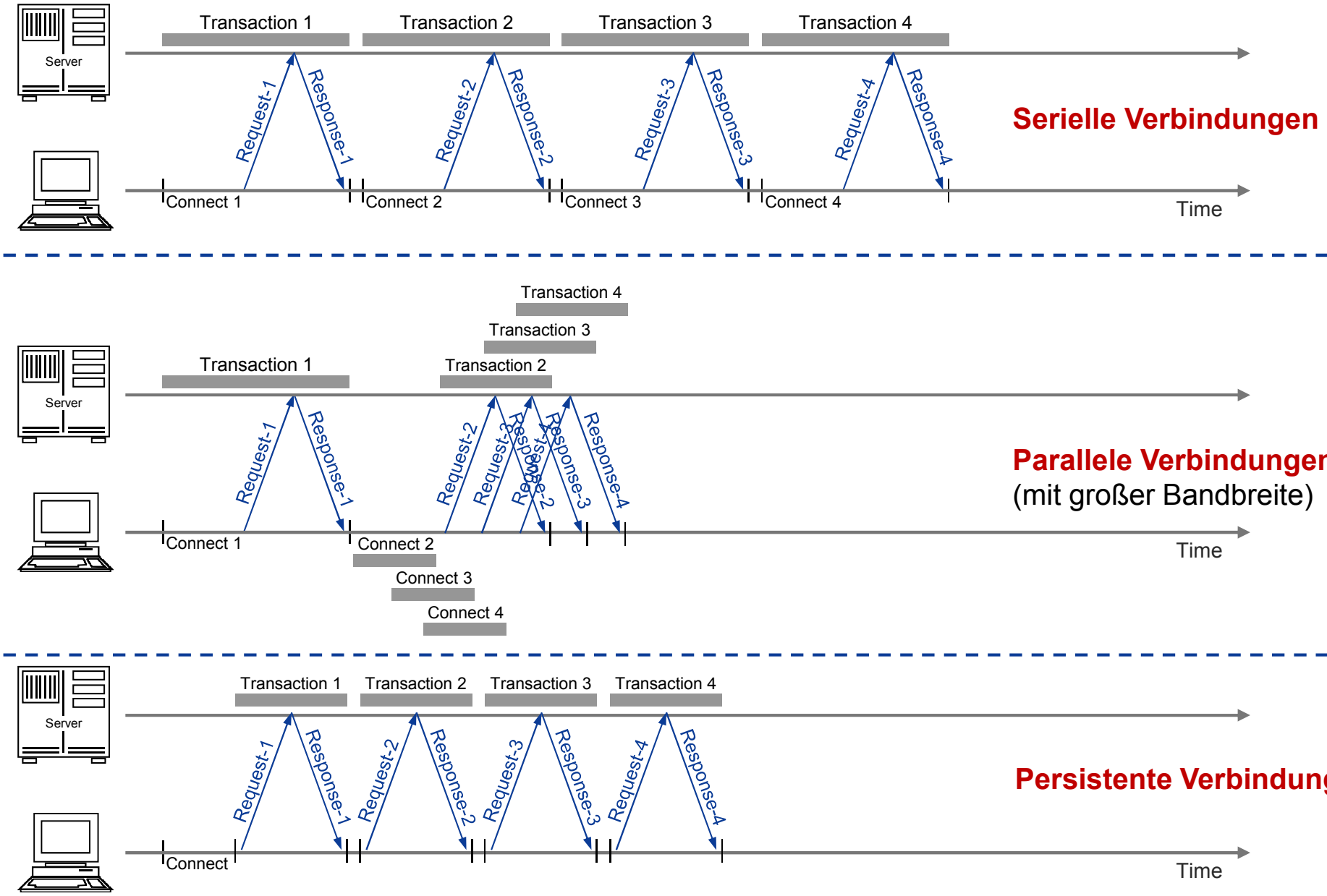


Hypertext Transfer Protocol (HTTP)
→ Pipelining HTTP-Verbindungen

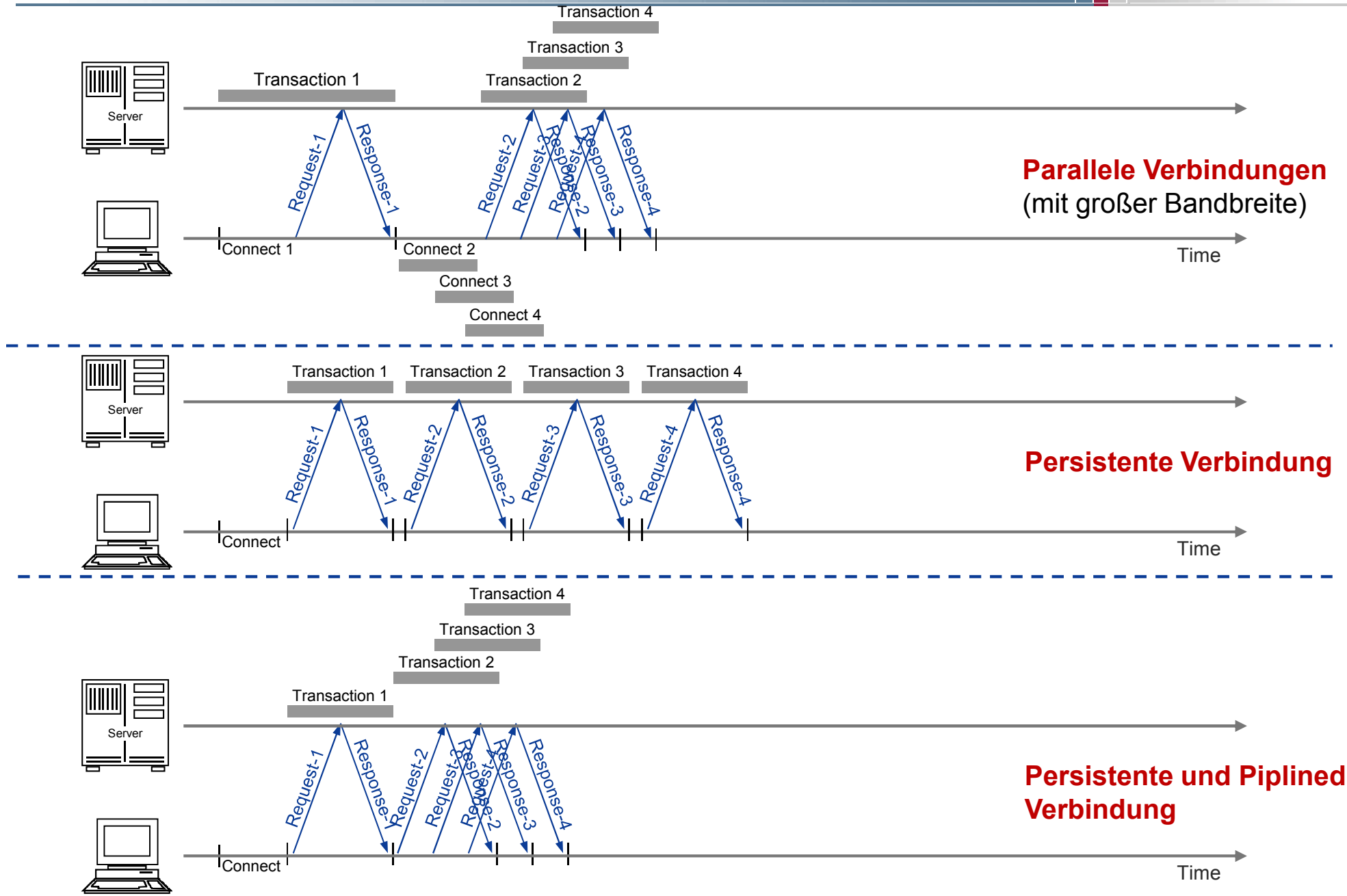
- Um die Leistung weiter zu verbessern, kann ein Client mehrere Anforderungen nacheinander absetzen, ohne auf die Antwort auf die erste Anforderung zu warten.
- Diese Methode wird als **Pipelining** bezeichnet.



Hypertext Transfer Protocol (HTTP)
→ HTTP-Verbindungen : Vergleiche (1/2)



Hypertext Transfer Protocol (HTTP)
→ HTTP-Verbindungen : Vergleiche (2/2)



Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

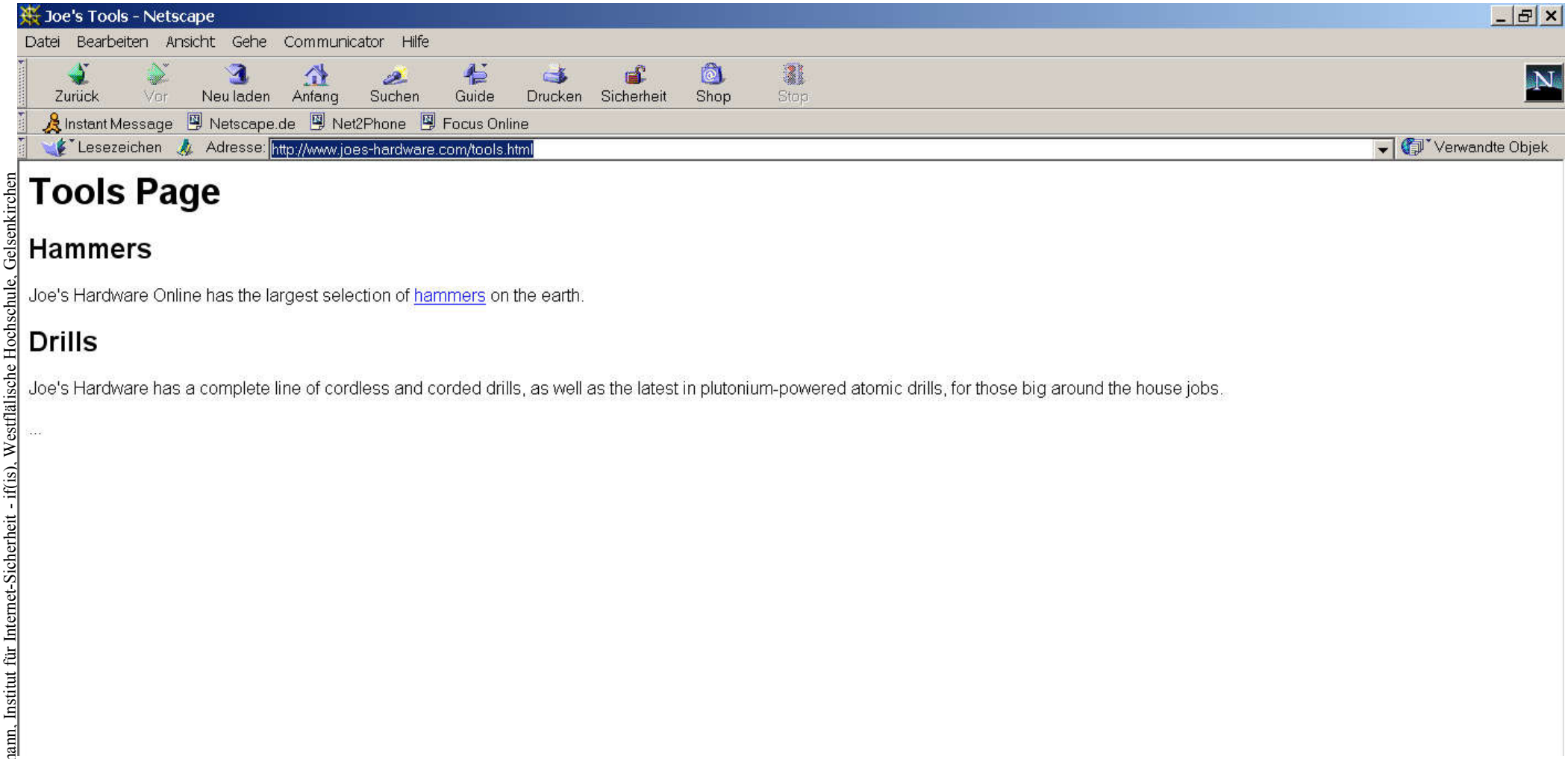
HTTP - Protokollmitschnitte

Prof. Dr. (TU NN)
Norbert Pohlmann

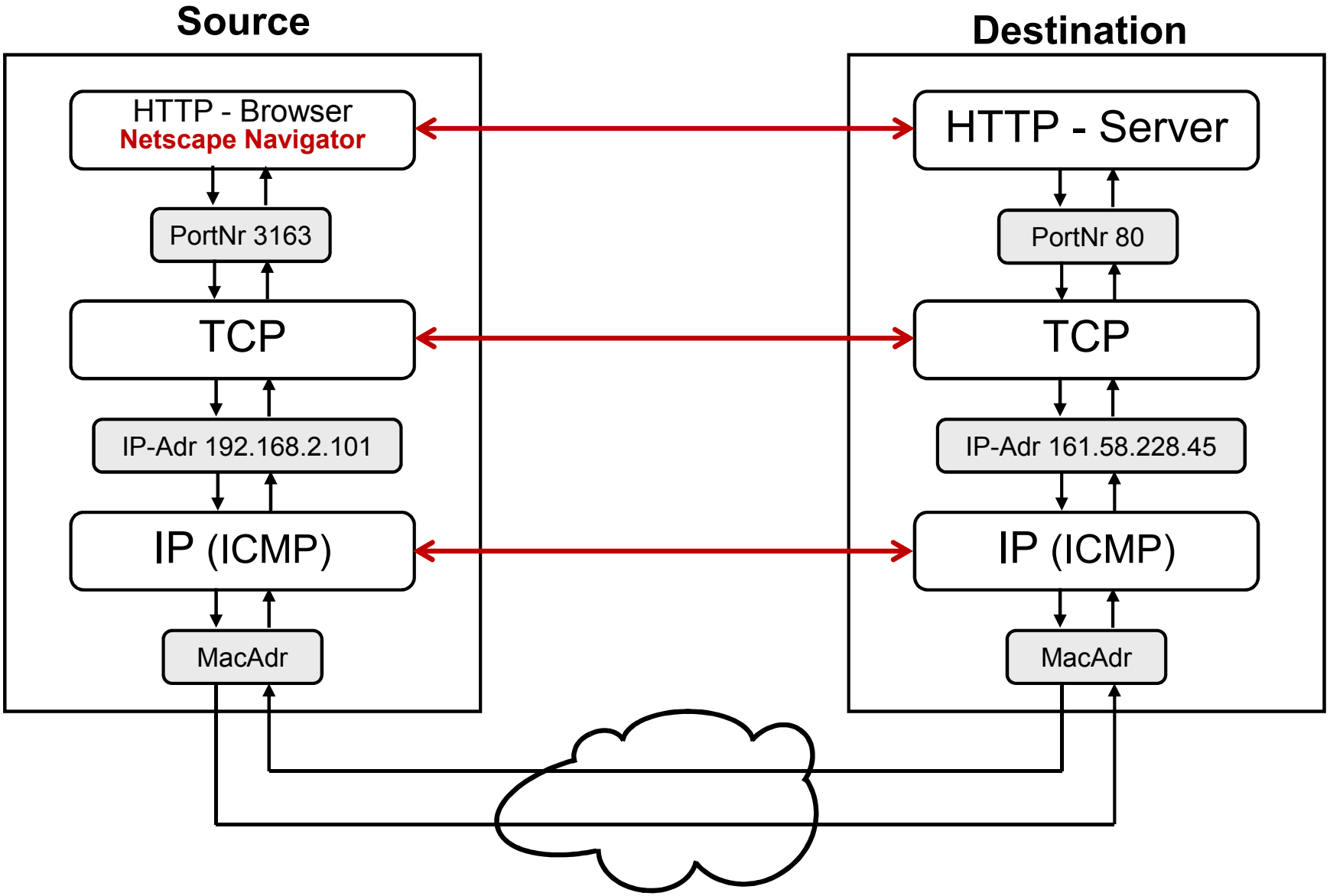
Institut für Internet-Sicherheit – if(is)
Westfälische Hochschule, Gelsenkirchen
http://www.internet-sicherheit.de

if(is) internet-sicherheit.

Hypertext Transfer Protocol (HTTP)
→ Protokollmitschnitt - Beispiel 1 (1/4)



Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 1 (2/4)



Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 1 (3/4)

```
No. Time       Source          Destination     Protocol Info
  1 0.000000   192.168.2.101   192.168.2.1     DNS      Standard query A www.joes-hardware.com
  2 0.234871   192.168.2.1     192.168.2.101   DNS      Standard query response CNAME joes-hardware.com A 161.58.228.45
  3 0.442260   192.168.2.101   161.58.228.45   TCP      3163 > http [SYN] Seq=343924507 Ack=0 Win=64240 Len=0
  4 0.591326   161.58.228.45   192.168.2.101   TCP      http > 3163 [SYN, ACK] Seq=1145840618 Ack=343924508 Win=16872 Len=0
  5 0.591416   192.168.2.101   161.58.228.45   TCP      3163 > http [ACK] Seq=343924508 Ack=1145840619 Win=64676 Len=0
  6 0.592711   192.168.2.101   161.58.228.45   HTTP     GET /tools.html HTTP/1.0
  7 0.770411   161.58.228.45   192.168.2.101   HTTP     HTTP/1.1 304 Not Modified
  8 0.873553   192.168.2.101   161.58.228.45   TCP      3163 > http [ACK] Seq=343924859 Ack=1145840835 Win=64460 Len=0

. . .

. . .
 82 19.679595  192.168.2.101   161.58.228.45   TCP      3163 > http [FIN, ACK] Seq=343924859 Ack=1145840835 Win=64460 Len=0
 94 19.866455  161.58.228.45   192.168.2.101   TCP      http > 3163 [ACK] Seq=1145840835 Ack=343924860 Win=16872 Len=0
 95 19.867356  161.58.228.45   192.168.2.101   TCP      http > 3163 [FIN, ACK] Seq=1145840835 Ack=343924860 Win=16872 Len=0
 96 19.867389  192.168.2.101   161.58.228.45   TCP      3163 > http [ACK] Seq=343924860 Ack=1145840836 Win=64460 Len=0
```

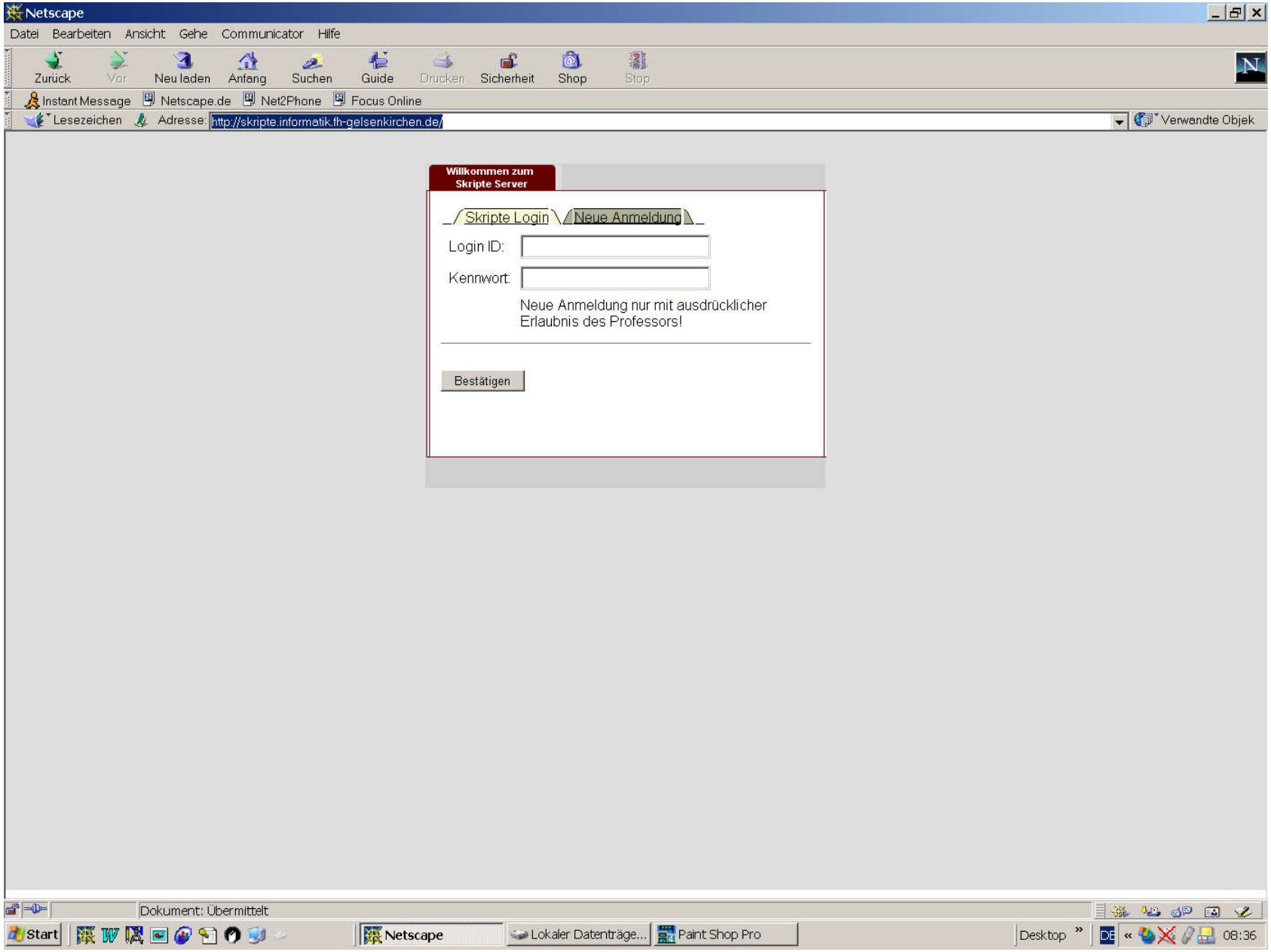
```
No. Time       Source          Destination     Protocol Info
  6 0.592711   192.168.2.101   161.58.228.45   HTTP     GET /tools.html HTTP/1.0
GET /tools.html HTTP/1.0
If-Modified-Since: Fri, 12 Jul 2002 07:50:17 GMT; length=433
Connection: Keep-Alive
User-Agent: Mozilla/4.78 [de] (Windows NT 5.0; U)
Host: www.joes-hardware.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: de
Accept-Charset: iso-8859-1,*,utf-8
```

```
No. Time       Source          Destination     Protocol Info
  7 0.770411   161.58.228.45   192.168.2.101   HTTP     HTTP/1.1 304 Not Modified
HTTP/1.1 304 Not Modified
Date: Mon, 11 Aug 2003 16:56:21 GMT
Server: Apache/1.3.27 OpenSSL/0.9.6i (Unix) FrontPage/5.0.2.2510
Connection: Keep-Alive
Keep-Alive: timeout=5, max=20
ETag: "5fa0f6-1b1-3d2e8a39"
```

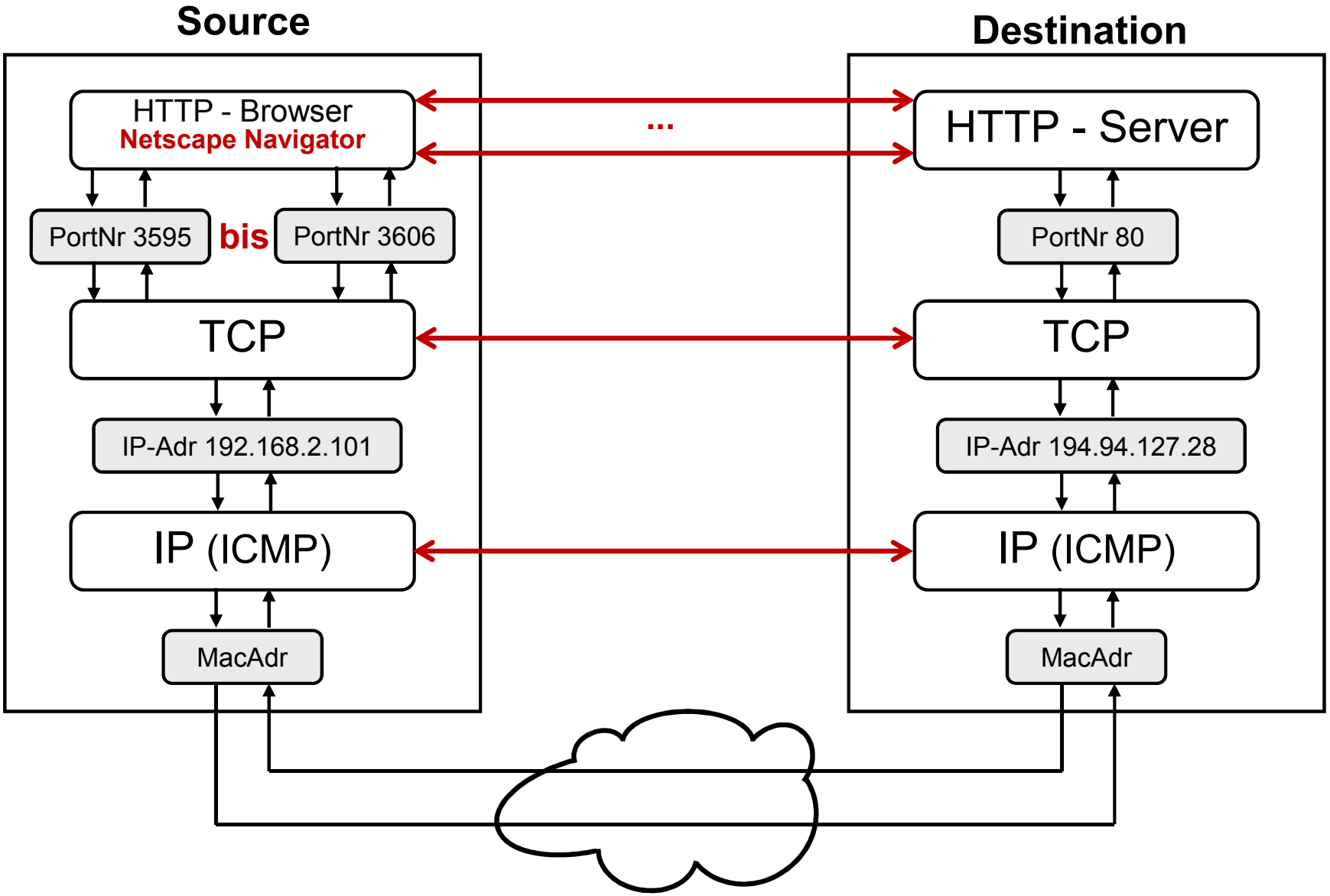
Hypertext Transfer Protocol (HTTP)
→ Protokollmitschnitt - Beispiel 1 (4/4)

- Sehr einfaches Beispiel!
- Da die statische Seite schon im Browser-Cache vorhanden ist und mit *If-Modified-Since* im Anforderungs-Header überprüft wurde, ob eine Veränderung stattgefunden hat, wurde das Dokument ***nicht*** neu geladen (Angezeigt durch *Not-Modified* im Antword-Header).
- Da der Server mit „Connection : Keep-Alive“ geantwortet hat wurde die TCP-Verbindung erst noch aufrecht erhalten (Abbau erst No. 82-96).

Hypertext Transfer Protocol (HTTP)
→ Protokollmitschnitt - Beispiel 2 (1/17)



Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2a (2/17)



Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2a (3/17)

	No.	Time	Source	Destination	Protocol	Info
DNS	1	0.000000	192.168.2.101	192.168.2.1	DNS	Standard query A skripte.informatik.fh-gelsenkirchen.de
	2	0.106976	192.168.2.1	192.168.2.101	DNS	Standard query response A 194.94.127.28
	3	0.109244	192.168.2.101	194.94.127.28	TCP	3595 > http [SYN] Seq=59733602 Ack=0 Win=64240 Len=0
	4	0.188949	194.94.127.28	192.168.2.101	TCP	http > 3595 [SYN, ACK] Seq=3748658550 Ack=59733603 Win=15466 Len=0
	5	0.189043	192.168.2.101	194.94.127.28	TCP	3595 > http [ACK] Seq=59733603 Ack=3748658551 Win=64676 Len=0
	6	0.189621	192.168.2.101	194.94.127.28	HTTP	GET / HTTP/1.0
	7	0.284787	194.94.127.28	192.168.2.101	TCP	http > 3595 [ACK] Seq=3748658551 Ack=59733899 Win=15466 Len=0
	8	0.306023	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
	9	0.306836	194.94.127.28	192.168.2.101	TCP	http > 3595 [FIN, ACK] Seq=3748659059 Ack=59733899 Win=15466 Len=0
	10	0.306912	192.168.2.101	194.94.127.28	TCP	3595 > http [ACK] Seq=59733899 Ack=3748659060 Win=64168 Len=0
alt	11	0.321231	192.168.2.101	213.229.30.35	TCP	3591 > http [FIN, ACK] Seq=3909756 Ack=3504729356 Win=63418 Len=0
	12	0.323180	192.168.2.101	194.94.127.28	TCP	3596 > http [SYN] Seq=59844526 Ack=0 Win=64240 Len=0
alt	13	0.325602	192.168.2.101	213.229.30.35	TCP	3592 > http [FIN, ACK] Seq=4101197 Ack=3506544717 Win=64676 Len=0
	14	0.327121	192.168.2.101	194.94.127.28	TCP	3597 > http [SYN] Seq=59905962 Ack=0 Win=64240 Len=0
	15	0.332567	192.168.2.101	194.94.127.28	TCP	3595 > http [FIN, ACK] Seq=59733899 Ack=3748659060 Win=64168 Len=0
alt	16	0.396615	213.229.30.35	192.168.2.101	TCP	http > 3591 [RST] Seq=3504729356 Ack=0 Win=0 Len=0
	17	0.402676	194.94.127.28	192.168.2.101	TCP	http > 3596 [SYN, ACK] Seq=3743468517 Ack=59844527 Win=15466 Len=0
	18	0.402739	192.168.2.101	194.94.127.28	TCP	3596 > http [ACK] Seq=59844527 Ack=3743468518 Win=64676 Len=0
	19	0.403120	192.168.2.101	194.94.127.28	HTTP	GET /handler.php HTTP/1.0
alt	20	0.404872	213.229.30.35	192.168.2.101	TCP	http > 3592 [RST] Seq=3506544717 Ack=0 Win=0 Len=0
	21	0.412144	194.94.127.28	192.168.2.101	TCP	http > 3597 [SYN, ACK] Seq=3751015321 Ack=59905963 Win=15466 Len=0
	22	0.412216	192.168.2.101	194.94.127.28	TCP	3597 > http [ACK] Seq=59905963 Ack=3751015322 Win=64676 Len=0
	23	0.412597	192.168.2.101	194.94.127.28	HTTP	GET /blank.html HTTP/1.0
	24	0.417166	194.94.127.28	192.168.2.101	TCP	http > 3595 [ACK] Seq=3748659060 Ack=59733900 Win=15466 Len=0
	25	0.503448	194.94.127.28	192.168.2.101	TCP	http > 3596 [ACK] Seq=3743468518 Ack=59844834 Win=15466 Len=0
	26	0.527299	194.94.127.28	192.168.2.101	TCP	http > 3597 [ACK] Seq=3751015322 Ack=59906269 Win=15466 Len=0
	27	0.531833	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
	28	0.533623	194.94.127.28	192.168.2.101	TCP	http > 3597 [FIN, ACK] Seq=3751015533 Ack=59906269 Win=15466 Len=0
	29	0.533720	192.168.2.101	194.94.127.28	TCP	3597 > http [ACK] Seq=59906269 Ack=3751015534 Win=64465 Len=0
	30	0.535879	192.168.2.101	194.94.127.28	TCP	3597 > http [FIN, ACK] Seq=59906269 Ack=3751015534 Win=64465 Len=0
	31	0.604937	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
	32	0.613467	192.168.2.101	194.94.127.28	TCP	3598 > http [SYN] Seq=60016455 Ack=0 Win=64240 Len=0
	33	0.620241	194.94.127.28	192.168.2.101	HTTP	Continuation
	34	0.620338	192.168.2.101	194.94.127.28	TCP	3596 > http [ACK] Seq=59844834 Ack=3743471330 Win=64676 Len=0
	35	0.621691	194.94.127.28	192.168.2.101	TCP	http > 3597 [ACK] Seq=3751015534 Ack=59906270 Win=15466 Len=0
	36	0.691073	194.94.127.28	192.168.2.101	TCP	http > 3598 [SYN, ACK] Seq=3752931348 Ack=60016456 Win=15466 Len=0
	37	0.691170	192.168.2.101	194.94.127.28	TCP	3598 > http [ACK] Seq=60016456 Ack=3752931349 Win=64676 Len=0
	38	0.691572	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/theme.css HTTP/1.0
	39	0.717854	194.94.127.28	192.168.2.101	HTTP	Continuation
	40	0.732698	194.94.127.28	192.168.2.101	HTTP	Continuation
	41	0.732757	192.168.2.101	194.94.127.28	TCP	3596 > http [ACK] Seq=59844834 Ack=3743474142 Win=64676 Len=0

Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2a (4/17)

	No.	Time	Source	Destination	Protocol	Info
	42	0.737917	194.94.127.28	192.168.2.101	HTTP	Continuation
	43	0.737964	192.168.2.101	194.94.127.28	TCP	3596 > http [ACK] Seq=59844834 Ack=3743474548 Win=64271 Len=0
	44	0.792748	194.94.127.28	192.168.2.101	TCP	http > 3598 [ACK] Seq=3752931349 Ack=60016832 Win=15466 Len=0
	45	0.829872	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
	46	0.845181	194.94.127.28	192.168.2.101	HTTP	Continuation
	47	0.845266	192.168.2.101	194.94.127.28	TCP	3598 > http [ACK] Seq=60016832 Ack=3752934161 Win=64676 Len=0
	48	0.940566	194.94.127.28	192.168.2.101	HTTP	Continuation
	49	0.944882	194.94.127.28	192.168.2.101	HTTP	Continuation
	50	0.944962	192.168.2.101	194.94.127.28	TCP	3598 > http [ACK] Seq=60016832 Ack=3752935896 Win=64676 Len=0
	51	0.954676	192.168.2.101	194.94.127.28	HTTP	GET /miolo/common.js HTTP/1.0
	52	1.091729	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
	53	1.104811	194.94.127.28	192.168.2.101	HTTP	Continuation
	54	1.104891	192.168.2.101	194.94.127.28	TCP	3598 > http [ACK] Seq=60017195 Ack=3752938708 Win=64676 Len=0
	55	1.120080	194.94.127.28	192.168.2.101	HTTP	Continuation
	56	1.135379	194.94.127.28	192.168.2.101	HTTP	Continuation
	57	1.135459	192.168.2.101	194.94.127.28	TCP	3598 > http [ACK] Seq=60017195 Ack=3752941520 Win=64676 Len=0
	58	1.201518	194.94.127.28	192.168.2.101	HTTP	Continuation
	59	1.216360	194.94.127.28	192.168.2.101	HTTP	Continuation
	60	1.216419	192.168.2.101	194.94.127.28	TCP	3598 > http [ACK] Seq=60017195 Ack=3752944332 Win=64676 Len=0
	61	1.232104	194.94.127.28	192.168.2.101	HTTP	Continuation
	62	1.246968	194.94.127.28	192.168.2.101	HTTP	Continuation
	63	1.247034	192.168.2.101	194.94.127.28	TCP	3598 > http [ACK] Seq=60017195 Ack=3752947144 Win=64676 Len=0
	64	1.262257	194.94.127.28	192.168.2.101	HTTP	Continuation
	65	1.274369	194.94.127.28	192.168.2.101	HTTP	Continuation
	66	1.274435	192.168.2.101	194.94.127.28	TCP	3598 > http [ACK] Seq=60017195 Ack=3752949643 Win=64676 Len=0
	67	1.473458	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//pix.gif HTTP/1.0
	68	1.476424	192.168.2.101	194.94.127.28	TCP	3599 > http [SYN] Seq=60281792 Ack=0 Win=64240 Len=0
alt	69	1.477652	192.168.2.101	213.229.30.35	TCP	3594 > http [FIN, ACK] Seq=4302246 Ack=3490550879 Win=63330 Len=0
	70	1.479026	192.168.2.101	194.94.127.28	TCP	3600 > http [SYN] Seq=60332435 Ack=0 Win=64240 Len=0
	71	1.592481	194.94.127.28	192.168.2.101	TCP	http > 3599 [SYN, ACK] Seq=3748696092 Ack=60281793 Win=15466 Len=0
	72	1.592574	192.168.2.101	194.94.127.28	TCP	3599 > http [ACK] Seq=60281793 Ack=3748696093 Win=64676 Len=0
alt	73	1.592585	213.229.30.35	192.168.2.101	TCP	http > 3594 [RST] Seq=3490550879 Ack=0 Win=0 Len=0
	74	1.592628	194.94.127.28	192.168.2.101	TCP	http > 3600 [SYN, ACK] Seq=3749680594 Ack=60332436 Win=15466 Len=0
	75	1.592652	192.168.2.101	194.94.127.28	TCP	3600 > http [ACK] Seq=60332436 Ack=3749680595 Win=64676 Len=0
	76	1.592660	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
	77	1.593961	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//cl.gif HTTP/1.0
	78	1.594783	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//cr.gif HTTP/1.0
	79	1.602817	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_left_spacer.gif HTTP/1.0
	80	1.700268	194.94.127.28	192.168.2.101	TCP	http > 3599 [ACK] Seq=3748696093 Ack=60282230 Win=15466 Len=0
	81	1.711123	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
	82	1.716635	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_left_back.gif HTTP/1.0

Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2a (5/17)

No.	Time	Source	Destination	Protocol	Info
83	1.725478	194.94.127.28	192.168.2.101	TCP	http > 3600 [ACK] Seq=3749680595 Ack=60332873 Win=15466 Len=0
84	1.730925	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
85	1.736199	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_right_spacer.gif HTTP/1.0
86	1.762852	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
87	1.767756	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_right_front.gif HTTP/1.0
88	1.821365	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
89	1.826573	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_left_front.gif HTTP/1.0
90	1.847457	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
91	1.852785	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_right_back.gif HTTP/1.0
92	1.874441	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
93	1.934314	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
94	1.958154	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
95	1.964727	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_front.gif HTTP/1.0
96	1.965670	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_back.gif HTTP/1.0
97	1.972991	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_front.gif HTTP/1.0
98	2.071056	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
99	2.071604	192.168.2.101	194.94.127.28	TCP	3598 > http [FIN, ACK] Seq=60018981 Ack=3752951095 Win=64676 Len=0
100	2.094639	192.168.2.101	194.94.127.28	TCP	3601 > http [SYN] Seq=60523436 Ack=0 Win=64240 Len=0
101	2.097163	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
102	2.097644	192.168.2.101	194.94.127.28	TCP	3599 > http [FIN, ACK] Seq=60283574 Ack=3748697622 Win=64676 Len=0
103	2.099126	192.168.2.101	194.94.127.28	TCP	3602 > http [SYN] Seq=60580663 Ack=0 Win=64240 Len=0
104	2.123731	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
105	2.124225	192.168.2.101	194.94.127.28	TCP	3600 > http [FIN, ACK] Seq=60334221 Ack=3749682076 Win=64676 Len=0
106	2.125652	192.168.2.101	194.94.127.28	TCP	3603 > http [SYN] Seq=60615109 Ack=0 Win=64240 Len=0
107	2.150174	194.94.127.28	192.168.2.101	TCP	http > 3598 [ACK] Seq=3752951095 Ack=60018982 Win=15466 Len=0
108	2.151077	194.94.127.28	192.168.2.101	TCP	http > 3598 [FIN, ACK] Seq=3752951095 Ack=60018982 Win=15466 Len=0
109	2.151100	192.168.2.101	194.94.127.28	TCP	3598 > http [ACK] Seq=60018982 Ack=3752951096 Win=64676 Len=0
110	2.174093	194.94.127.28	192.168.2.101	TCP	http > 3601 [SYN, ACK] Seq=3743687500 Ack=60523437 Win=15466 Len=0
111	2.174180	192.168.2.101	194.94.127.28	TCP	3601 > http [ACK] Seq=60523437 Ack=3743687501 Win=64676 Len=0
112	2.174616	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_front.gif HTTP/1.0
113	2.178550	194.94.127.28	192.168.2.101	TCP	http > 3599 [ACK] Seq=3748697622 Ack=60283575 Win=15466 Len=0
114	2.179432	194.94.127.28	192.168.2.101	TCP	http > 3599 [FIN, ACK] Seq=3748697622 Ack=60283575 Win=15466 Len=0
115	2.179461	192.168.2.101	194.94.127.28	TCP	3599 > http [ACK] Seq=60283575 Ack=3748697623 Win=64676 Len=0
116	2.182155	194.94.127.28	192.168.2.101	TCP	http > 3602 [SYN, ACK] Seq=3741374749 Ack=60580664 Win=15466 Len=0
117	2.182187	192.168.2.101	194.94.127.28	TCP	3602 > http [ACK] Seq=60580664 Ack=3741374750 Win=64676 Len=0
118	2.182581	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_back.gif HTTP/1.0
119	2.203283	194.94.127.28	192.168.2.101	TCP	http > 3600 [ACK] Seq=3749682076 Ack=60334222 Win=15466 Len=0
120	2.204282	194.94.127.28	192.168.2.101	TCP	http > 3600 [FIN, ACK] Seq=3749682076 Ack=60334222 Win=15466 Len=0
121	2.204329	192.168.2.101	194.94.127.28	TCP	3600 > http [ACK] Seq=60334222 Ack=3749682077 Win=64676 Len=0
122	2.207736	194.94.127.28	192.168.2.101	TCP	http > 3603 [SYN, ACK] Seq=3740480867 Ack=60615110 Win=15466 Len=0
123	2.207829	192.168.2.101	194.94.127.28	TCP	3603 > http [ACK] Seq=60615110 Ack=3740480868 Win=64676 Len=0
124	2.278909	194.94.127.28	192.168.2.101	TCP	http > 3601 [ACK] Seq=3743687501 Ack=60523887 Win=15466 Len=0
125	2.284358	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
126	2.314017	194.94.127.28	192.168.2.101	TCP	http > 3602 [ACK] Seq=3741374750 Ack=60581113 Win=15466 Len=0
127	2.319891	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
128	2.473195	192.168.2.101	194.94.127.28	TCP	3602 > http [ACK] Seq=60581113 Ack=3741375133 Win=64293 Len=0
129	2.473254	192.168.2.101	194.94.127.28	TCP	3601 > http [ACK] Seq=60523887 Ack=3743687880 Win=64297 Len=0
130	3.169870	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_back.gif HTTP/1.0
131	3.170893	192.168.2.101	194.94.127.28	TCP	3601 > http [FIN, ACK] Seq=60523887 Ack=3743687880 Win=64297 Len=0

Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2a (6/17)

No.	Time	Source	Destination	Protocol	Info
132	3.172912	192.168.2.101	194.94.127.28	TCP	3604 > http [SYN] Seq=60923169 Ack=0 Win=64240 Len=0
133	3.173324	192.168.2.101	194.94.127.28	TCP	3602 > http [FIN, ACK] Seq=60581113 Ack=3741375133 Win=64293 Len=0
134	3.175168	192.168.2.101	194.94.127.28	TCP	3605 > http [SYN] Seq=61003783 Ack=0 Win=64240 Len=0
135	3.268289	194.94.127.28	192.168.2.101	TCP	http > 3603 [ACK] Seq=3740480868 Ack=60615559 Win=15466 Len=0
136	3.273745	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
137	3.274200	192.168.2.101	194.94.127.28	TCP	3603 > http [FIN, ACK] Seq=60615559 Ack=3740481251 Win=64293 Len=0
138	3.274621	194.94.127.28	192.168.2.101	TCP	http > 3601 [ACK] Seq=3743687880 Ack=60523888 Win=15466 Len=0
139	3.275812	194.94.127.28	192.168.2.101	TCP	http > 3601 [FIN, ACK] Seq=3743687880 Ack=60523888 Win=15466 Len=0
140	3.275868	192.168.2.101	194.94.127.28	TCP	3601 > http [ACK] Seq=60523888 Ack=3743687881 Win=64297 Len=0
141	3.276452	192.168.2.101	194.94.127.28	TCP	3606 > http [SYN] Seq=61094067 Ack=0 Win=64240 Len=0
142	3.278482	194.94.127.28	192.168.2.101	TCP	http > 3604 [SYN, ACK] Seq=3742834044 Ack=60923170 Win=15466 Len=0
143	3.278524	192.168.2.101	194.94.127.28	TCP	3604 > http [ACK] Seq=60923170 Ack=3742834045 Win=64676 Len=0
144	3.278896	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_front.gif HTTP/1.0
145	3.280385	194.94.127.28	192.168.2.101	TCP	http > 3602 [ACK] Seq=3741375133 Ack=60581114 Win=15466 Len=0
146	3.281327	194.94.127.28	192.168.2.101	TCP	http > 3602 [FIN, ACK] Seq=3741375133 Ack=60581114 Win=15466 Len=0
147	3.281357	192.168.2.101	194.94.127.28	TCP	3602 > http [ACK] Seq=60581114 Ack=3741375134 Win=64293 Len=0
148	3.284466	194.94.127.28	192.168.2.101	TCP	http > 3605 [SYN, ACK] Seq=3740781940 Ack=61003784 Win=15466 Len=0
149	3.284510	192.168.2.101	194.94.127.28	TCP	3605 > http [ACK] Seq=61003784 Ack=3740781941 Win=64676 Len=0
150	3.284857	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_back.gif HTTP/1.0
151	3.353315	194.94.127.28	192.168.2.101	TCP	http > 3603 [ACK] Seq=3740481251 Ack=60615560 Win=15466 Len=0
152	3.354233	194.94.127.28	192.168.2.101	TCP	http > 3603 [FIN, ACK] Seq=3740481251 Ack=60615560 Win=15466 Len=0
153	3.354264	192.168.2.101	194.94.127.28	TCP	3603 > http [ACK] Seq=60615560 Ack=3740481252 Win=64293 Len=0
154	3.361859	194.94.127.28	192.168.2.101	TCP	http > 3606 [SYN, ACK] Seq=3740935862 Ack=61094068 Win=15466 Len=0
155	3.361918	192.168.2.101	194.94.127.28	TCP	3606 > http [ACK] Seq=61094068 Ack=3740935863 Win=64676 Len=0
156	3.362301	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//sl.gif HTTP/1.0
157	3.394227	194.94.127.28	192.168.2.101	TCP	http > 3604 [ACK] Seq=3742834045 Ack=60923620 Win=15466 Len=0
158	3.399694	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
159	3.405016	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//sr.gif HTTP/1.0
160	3.429339	194.94.127.28	192.168.2.101	TCP	http > 3605 [ACK] Seq=3740781941 Ack=61004233 Win=15466 Len=0
161	3.434794	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
162	3.459313	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//sl.gif HTTP/1.0
163	3.464906	194.94.127.28	192.168.2.101	TCP	http > 3606 [ACK] Seq=3740935863 Ack=61094505 Win=15466 Len=0
164	3.479798	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
165	3.486040	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//sr.gif HTTP/1.0
166	3.504914	192.168.2.101	194.94.127.28	TCP	3596 > http [FIN, ACK] Seq=59844834 Ack=3743474548 Win=64271 Len=0
167	3.507682	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
168	3.564831	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
169	3.595302	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
170	3.595364	194.94.127.28	192.168.2.101	TCP	http > 3596 [ACK] Seq=3743474548 Ack=59844835 Win=15466 Len=0
171	3.616456	192.168.2.101	194.94.127.28	TCP	3605 > http [FIN, ACK] Seq=61004670 Ack=3740782663 Win=63954 Len=0
172	3.616741	192.168.2.101	194.94.127.28	TCP	3606 > http [FIN, ACK] Seq=61094942 Ack=3740936542 Win=63997 Len=0
173	3.674904	192.168.2.101	194.94.127.28	TCP	3604 > http [ACK] Seq=60924057 Ack=3742834763 Win=63958 Len=0
174	3.693432	194.94.127.28	192.168.2.101	TCP	http > 3605 [ACK] Seq=3740782663 Ack=61004671 Win=15466 Len=0
175	3.694331	194.94.127.28	192.168.2.101	TCP	http > 3605 [FIN, ACK] Seq=3740782663 Ack=61004671 Win=15466 Len=0
176	3.694354	192.168.2.101	194.94.127.28	TCP	3605 > http [ACK] Seq=61004671 Ack=3740782664 Win=63954 Len=0
177	3.697933	194.94.127.28	192.168.2.101	TCP	http > 3606 [ACK] Seq=3740936542 Ack=61094943 Win=15466 Len=0
178	3.698835	194.94.127.28	192.168.2.101	TCP	http > 3606 [FIN, ACK] Seq=3740936542 Ack=61094943 Win=15466 Len=0
179	3.698853	192.168.2.101	194.94.127.28	TCP	3606 > http [ACK] Seq=61094943 Ack=3740936543 Win=63997 Len=0
180	8.101942	195.145.119.188	192.168.2.101	NTP	NTP

Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2a (7/17)

Protocol Hierarchy Statistics

Protocol	% Packets	Packets	Bytes	End Packets	End Bytes
⊟ Frame	100.00%	180	54241	0	0
⊟ Ethernet	100.00%	180	54241	0	0
⊟ Internet Protocol	100.00%	180	54241	0	0
⊟ User Datagram Protocol	1.67%	3	344	0	0
Domain Name Service	1.11%	2	254	2	254
Network Time Protocol	0.56%	1	90	1	90
⊟ Transmission Control Protocol	98.33%	177	53897	109	6462
Hypertext Transfer Protocol	37.78%	68	47435	68	47435

- Summe der Bytes = 54.241
- Summe der TCP-Verbindungen = 12 (Port 3595 bis 3606)
  - einige mit Keep-Alive (mehrere GET über eine Verbindung)
  - insgesamt 26 GET-Anforderungen
  - 2 Dateien 2 mal, 2 Dateien 4 mal insgesamt 18 unterschiedliche Dateien
- Zeit: 3.59 s

Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2a (8/17) - Port 3595

```
No. Time       Source           Destination    Protocol Info
    3 0.109244  192.168.2.101    194.94.127.28  TCP      3595 > http [SYN] Seq=59733602 Ack=0 Win=64240 Len=0
    4 0.188949  194.94.127.28    192.168.2.101  TCP      http > 3595 [SYN, ACK] Seq=3748658550 Ack=59733603 Win=15466 Len=0
    5 0.189043  192.168.2.101    194.94.127.28  TCP      3595 > http [ACK] Seq=59733603 Ack=3748658551 Win=64676 Len=0
    6 0.189621  192.168.2.101    194.94.127.28  HTTP     GET / HTTP/1.0
    7 0.284787  194.94.127.28    192.168.2.101  TCP      http > 3595 [ACK] Seq=3748658551 Ack=59733899 Win=15466 Len=0
    8 0.306023  194.94.127.28    192.168.2.101  HTTP     HTTP/1.1 200 OK
    9 0.306836  194.94.127.28    192.168.2.101  TCP      http > 3595 [FIN, ACK] Seq=3748659059 Ack=59733899 Win=15466 Len=0
   10 0.306912  192.168.2.101    194.94.127.28  TCP      3595 > http [ACK] Seq=59733899 Ack=3748659060 Win=64168 Len=0
   15 0.332567  192.168.2.101    194.94.127.28  TCP      3595 > http [FIN, ACK] Seq=59733899 Ack=3748659060 Win=64168 Len=0
   24 0.417166  194.94.127.28    192.168.2.101  TCP      http > 3595 [ACK] Seq=3748659060 Ack=59733900 Win=15466 Len=0
```

```
No. Time       Source           Destination    Protocol Info
  6 0.189621   192.168.2.101    194.94.127.28  HTTP     GET / HTTP/1.0
GET / HTTP/1.0
```

Connetion:

```
Connection: Keep-Alive
User-Agent: Mozilla/4.78 [de] (Windows NT 5.0; U)
Host: skripte.informatik.fh-gelsenkirchen.de
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: de
Accept-Charset: iso-8859-1,*,utf-8
```

```
No. Time       Source           Destination    Protocol Info
  6 0.189621   192.168.2.101    194.94.127.28  HTTP     GET / HTTP/1.0
HTTP/1.1 200 OK
Date: Wed, 06 Aug 2003 16:45:46 GMT
Server: Apache/1.3.22 (Unix) PHP/4.3.2
X-Powered-By: PHP/4.3.2
```

Connection

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

<html>
<frameset rows="*,0" border=0 frameBorder=no frameSpacing=0>
  <frame name="content" src="handler.php" frameBorder=no frameSpacing=0
         marginHeight=0 marginWidth=0 noresize>
  <frame name="util"    src="blank.html" frameBorder=no frameSpacing=0
         marginHeight=0 marginWidth=0 noresize scrolling=no>
</frameset>
</html>
```

Siehe Nr. 19 GET Port 3596

Siehe Nr. 23 GET Port 3597

Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2a (9/17) - Port 3596

```
No. Time       Source           Destination      Protocol Info
 12 0.323180   192.168.2.101    194.94.127.28    TCP      3596 > http [SYN] Seq=59844526 Ack=0 Win=64240 Len=0
 17 0.402676   194.94.127.28    192.168.2.101    TCP      http > 3596 [SYN, ACK] Seq=3743468517 Ack=59844527 Win=15466 Len=0
 18 0.402739   192.168.2.101    194.94.127.28    TCP      3596 > http [ACK] Seq=59844527 Ack=3743468518 Win=64676 Len=0
 19 0.403120   192.168.2.101    194.94.127.28    HTTP     GET /handler.php HTTP/1.0
 25 0.503448   194.94.127.28    192.168.2.101    TCP      http > 3596 [ACK] Seq=3743468518 Ack=59844834 Win=15466 Len=0
 31 0.604937   194.94.127.28    192.168.2.101    HTTP     HTTP/1.1 200 OK
 33 0.620241   194.94.127.28    192.168.2.101    HTTP     Continuation
 34 0.620338   192.168.2.101    194.94.127.28    TCP      3596 > http [ACK] Seq=59844834 Ack=3743471330 Win=64676 Len=0
 39 0.717854   194.94.127.28    192.168.2.101    HTTP     Continuation
 40 0.732698   194.94.127.28    192.168.2.101    HTTP     Continuation
 41 0.732757   192.168.2.101    194.94.127.28    TCP      3596 > http [ACK] Seq=59844834 Ack=3743474142 Win=64676 Len=0
 42 0.737917   194.94.127.28    192.168.2.101    HTTP     Continuation
 43 0.737964   192.168.2.101    194.94.127.28    TCP      3596 > http [ACK] Seq=59844834 Ack=3743474548 Win=64271 Len=0
166 3.504914   192.168.2.101    194.94.127.28    TCP      3596 > http [FIN, ACK] Seq=59844834 Ack=3743474548 Win=64271 Len=0
170 3.595364   194.94.127.28    192.168.2.101    TCP      http > 3596 [ACK] Seq=3743474548 Ack=59844835 Win=15466 Len=0
```

```
No. Time       Source           Destination      Protocol Info
 19 0.403120   192.168.2.101    194.94.127.28    HTTP     GET /handler.php HTTP/1.0
GET /handler.php HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.78 [de] (Windows NT 5.0; U)
Host: skripte.informatik.fh-gelsenkirchen.de
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: de
Accept-Charset: iso-8859-1,*,utf-8
No. Time       Source           Destination      Protocol Info
 31 0.604937   194.94.127.28    192.168.2.101    HTTP     HTTP/1.1 200 OK
HTTP/1.1 200 OK
Date: Wed, 06 Aug 2003 16:45:46 GMT
Server: Apache/1.3.22 (Unix) PHP/4.3.2
X-Powered-By: PHP/4.3.2
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: PHPSESSID=23636f77d582b55bab00a75ede09ecb3; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Connection: close
Content-Type: text/html
```

Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2a (10/17) - Port 3596

```
No. Time       Source          Destination     Protocol Info
 31 0.604937   194.94.127.28   192.168.2.101   HTTP     HTTP/1.1 200 OK
<html>
<head>
<!-- START OF STYLE SHEETS -->
<link rel="stylesheet" href="/miolo/themes/slash/theme.css">
<!-- END OF STYLE SHEETS -->
<!-- START OF SCRIPTS -->
<script language="JavaScript" src="/miolo/common.js"></script>
<!-- END OF SCRIPTS -->
<!-- START OF META INFORATION -->
<meta name="Generator" content="MIOLO Version 0.1; http://miolo.codigolivre.org.br">
<!-- END OF META INFORMATION -->
<!-- START OF THEME HEADER -->
<meta name="Theme-Author" content="MIOLO Slash Theme">
<!-- END OF THEME HEADER -->
<title></title>
</head>
<body class="themeBody">
<table width="100%" cols="3" cellspacing="8" cellpadding="0" border="0">
  <tr>
    <td colspan="3">
     </td>
  </tr>
  <tr>
    <td width="15%" valign="top">
      </td>
    <td width="70%" height="100%" valign="top">
<table class="themeContent" cellpadding="0" cellspacing="0" width="100%">
  <tr>
    <td> <table width="150" border="0" cellpadding="0" cellspacing="0">
 <tr valign="top" bgcolor="#660000">
 <td bgcolor="#dddddd"><img src="/miolo/themes/slash//pix.gif" width="3" height="3"
alt=""></td>
 <td><img src="/miolo/themes/slash//cl.gif" width="7" height="10" alt=""></td>
 <td><font size="1" color="#ffffff"><B><div class="formTitle"><center>Willkommen zum
<br>Skripte Server</center></div>
</b></font></td>
 <td align="right"><img src="/miolo/themes/slash//cr.gif" width="7" height="10"
...
```

Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2a (11/17) - Port 3604/3605

doppelte Übertragung von „gif“-Dateien : ein Beispiel

No.	Time	Source	Destination	Protocol	Info
132	3.172912	192.168.2.101	194.94.127.28	TCP	**3604** > http [SYN] Seq=60923169 Ack=0 Win=64240 Len=0
142	3.278482	194.94.127.28	192.168.2.101	TCP	http > 3604 [SYN, ACK] Seq=3742834044 Ack=60923170 Win=15466 Len=0
143	3.278524	192.168.2.101	194.94.127.28	TCP	3604 > http [ACK] Seq=60923170 Ack=3742834045 Win=64676 Len=0
144	3.278896	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_front.gif HTTP/1.0
157	3.394227	194.94.127.28	192.168.2.101	TCP	http > 3604 [ACK] Seq=3742834045 Ack=60923620 Win=15466 Len=0
158	3.399694	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
159	3.405016	192.168.2.101	194.94.127.28	**HTTP**	**GET /miolo/themes/slash//sr.gif HTTP/1.0**
167	3.507682	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
173	3.674904	192.168.2.101	194.94.127.28	TCP	3604 > http [ACK] Seq=60924057 Ack=3742834763 Win=63958 Len=0

No.	Time	Source	Destination	Protocol	Info
141	3.276452	192.168.2.101	194.94.127.28	TCP	**3606** > http [SYN] Seq=61094067 Ack=0 Win=64240 Len=0
154	3.361859	194.94.127.28	192.168.2.101	TCP	http > 3606 [SYN, ACK] Seq=3740935862 Ack=61094068 Win=15466 Len=0
155	3.361918	192.168.2.101	194.94.127.28	TCP	3606 > http [ACK] Seq=61094068 Ack=3740935863 Win=64676 Len=0
156	3.362301	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//sl.gif HTTP/1.0
163	3.464906	194.94.127.28	192.168.2.101	TCP	http > 3606 [ACK] Seq=3740935863 Ack=61094505 Win=15466 Len=0
164	3.479798	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
165	3.486040	192.168.2.101	194.94.127.28	**HTTP**	**GET /miolo/themes/slash//sr.gif HTTP/1.0**
169	3.595302	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
172	3.616741	192.168.2.101	194.94.127.28	TCP	3606 > http [FIN, ACK] Seq=61094942 Ack=3740936542 Win=63997 Len=0
177	3.697933	194.94.127.28	192.168.2.101	TCP	http > 3606 [ACK] Seq=3740936542 Ack=61094943 Win=15466 Len=0
178	3.698835	194.94.127.28	192.168.2.101	TCP	http > 3606 [FIN, ACK] Seq=3740936542 Ack=61094943 Win=15466 Len=0
179	3.698853	192.168.2.101	194.94.127.28	TCP	3606 > http [ACK] Seq=61094943 Ack=3740936543 Win=63997 Len=0

- Scheinbar synchronisiert die HTTP-Applikation nicht die Anforderungen an Dateien!

HTTP - Protokollmittschnitt (1/)
→ Protokollmitschnitt - Beispiel 2a (11/17) - Port 3606

```
No. Time       Source           Destination      Protocol Info
134 3.175168   192.168.2.101    194.94.127.28    TCP      3605 > http [SYN] Seq=61003783 Ack=0 Win=64240 Len=0
148 3.284466   194.94.127.28    192.168.2.101    TCP      http > 3605 [SYN, ACK] Seq=3740781940 Ack=61003784 Win=15466 Len=0
149 3.284510   192.168.2.101    194.94.127.28    TCP      3605 > http [ACK] Seq=61003784 Ack=3740781941 Win=64676 Len=0
150 3.284857   192.168.2.101    194.94.127.28    HTTP     GET /miolo/themes/slash/tab_center_back.gif HTTP/1.0
160 3.429339   194.94.127.28    192.168.2.101    TCP      http > 3605 [ACK] Seq=3740781941 Ack=61004233 Win=15466 Len=0
161 3.434794   194.94.127.28    192.168.2.101    HTTP     HTTP/1.1 200 OK
162 3.459313   192.168.2.101    194.94.127.28    HTTP     GET /miolo/themes/slash//sl.gif HTTP/1.0
168 3.564831   194.94.127.28    192.168.2.101    HTTP     HTTP/1.1 200 OK
171 3.616456   192.168.2.101    194.94.127.28    TCP      3605 > http [FIN, ACK] Seq=61004670 Ack=3740782663 Win=63954 Len=0
174 3.693432   194.94.127.28    192.168.2.101    TCP      http > 3605 [ACK] Seq=3740782663 Ack=61004671 Win=15466 Len=0
175 3.694331   194.94.127.28    192.168.2.101    TCP      http > 3605 [FIN, ACK] Seq=3740782663 Ack=61004671 Win=15466 Len=0
176 3.694354   192.168.2.101    194.94.127.28    TCP      3605 > http [ACK] Seq=61004671 Ack=3740782664 Win=63954 Len=0
```

```
No. Time       Source           Destination      Protocol Info
150 3.284857   192.168.2.101    194.94.127.28    HTTP     GET /miolo/themes/slash/tab_center_back.gif HTTP/1.0
GET /miolo/themes/slash/tab_center_back.gif HTTP/1.0
Referer: http://skripte.informatik.fh-gelsenkirchen.de/handler.php
Connection: Keep-Alive
User-Agent: Mozilla/4.78 [de] (Windows NT 5.0; U)
Host: skripte.informatik.fh-gelsenkirchen.de
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png
Accept-Encoding: gzip
Accept-Language: de
Accept-Charset: iso-8859-1,*,utf-8
Cookie: PHPSESSID=23636f77d582b55bab00a75ede09ecb3
```

```
No. Time       Source           Destination      Protocol Info
161 3.434794   194.94.127.28    192.168.2.101    HTTP     HTTP/1.1 200 OK
HTTP/1.1 200 OK
Date: Wed, 06 Aug 2003 16:45:49 GMT
Server: Apache/1.3.22 (Unix) PHP/4.3.2
Last-Modified: Fri, 15 Feb 2002 12:27:09 GMT
ETag: "1974d2-59-3c6cfe9d"
Accept-Ranges: bytes
Content-Length: 89
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: image/gif
```

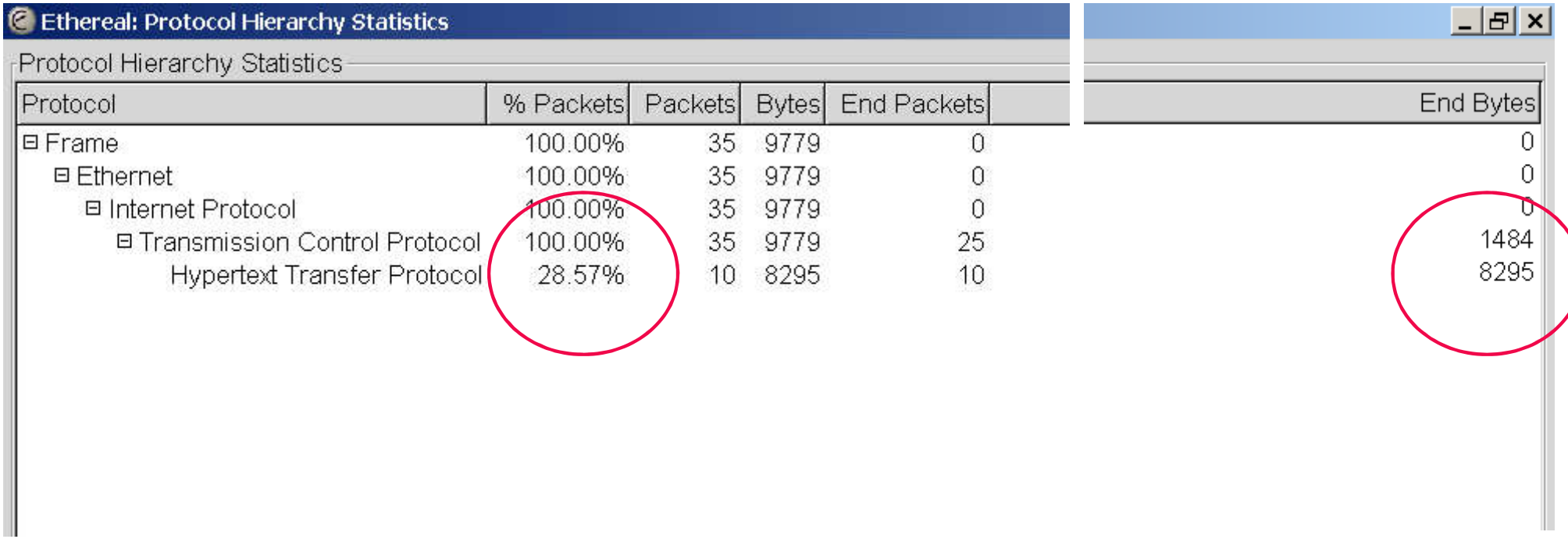
- Hier erlaubt der Server, dass die Verbindung offen bleibt!
- Deshalb wird auch eine zweite GET-Anforderung über den gleichen Port versendet.

Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2b (12/17)

2. Anforderung der selben Seite, die meisten Dateien werden aus dem Cache entnommen!

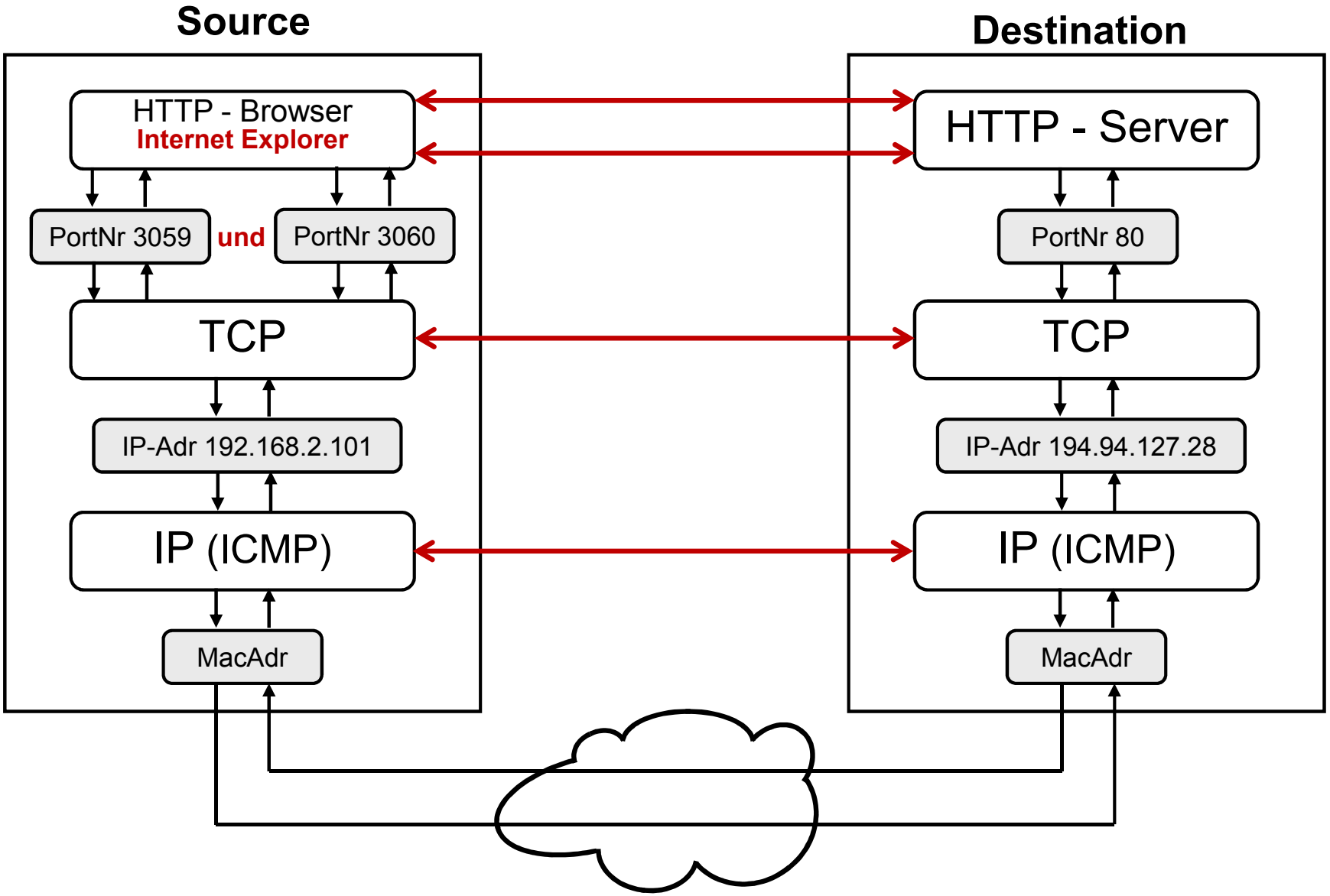
No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.2.101	194.94.127.28	TCP	3372 > http [SYN] Seq=1694389153 Ack=0 Win=64240 Len=0
2	0.078868	194.94.127.28	192.168.2.101	TCP	http > 3372 [SYN, ACK] Seq=817768044 Ack=1694389154 Win=15466 Len=0
3	0.078957	192.168.2.101	194.94.127.28	TCP	3372 > http [ACK] Seq=1694389154 Ack=817768045 Win=64676 Len=0
4	0.079400	192.168.2.101	194.94.127.28	**HTTP**	**GET / HTTP/1.0**
5	0.175604	194.94.127.28	192.168.2.101	TCP	http > 3372 [ACK] Seq=817768045 Ack=1694389502 Win=15466 Len=0
6	0.184685	194.94.127.28	192.168.2.101	**HTTP**	**HTTP/1.1 200 OK**
7	0.185520	194.94.127.28	192.168.2.101	TCP	http > 3372 [FIN, ACK] Seq=817768553 Ack=1694389502 Win=15466 Len=0
8	0.185606	192.168.2.101	194.94.127.28	TCP	3372 > http [ACK] Seq=1694389502 Ack=817768554 Win=64168 Len=0
9	0.236280	192.168.2.101	194.94.127.28	TCP	3373 > http [SYN] Seq=1694490162 Ack=0 Win=64240 Len=0
10	0.240639	192.168.2.101	194.94.127.28	TCP	3374 > http [SYN] Seq=1694538783 Ack=0 Win=64240 Len=0
11	0.253794	192.168.2.101	194.94.127.28	TCP	3372 > http [FIN, ACK] Seq=1694389502 Ack=817768554 Win=64168 Len=0
12	0.313738	194.94.127.28	192.168.2.101	TCP	http > 3373 [SYN, ACK] Seq=818289113 Ack=1694490163 Win=15466 Len=0
13	0.313835	192.168.2.101	194.94.127.28	TCP	3373 > http [ACK] Seq=1694490163 Ack=818289114 Win=64676 Len=0
14	0.314272	192.168.2.101	194.94.127.28	**HTTP**	**GET /handler.php HTTP/1.0**
15	0.318695	194.94.127.28	192.168.2.101	TCP	http > 3374 [SYN, ACK] Seq=822686246 Ack=1694538784 Win=15466 Len=0
16	0.318774	192.168.2.101	194.94.127.28	TCP	3374 > http [ACK] Seq=1694538784 Ack=822686247 Win=64676 Len=0
17	0.319154	192.168.2.101	194.94.127.28	**HTTP**	**GET /blank.html HTTP/1.0**
18	0.330062	194.94.127.28	192.168.2.101	TCP	http > 3372 [ACK] Seq=817768554 Ack=1694389503 Win=15466 Len=0
19	0.414075	194.94.127.28	192.168.2.101	TCP	http > 3373 [ACK] Seq=818289114 Ack=1694490522 Win=15466 Len=0
20	0.441071	194.94.127.28	192.168.2.101	TCP	http > 3374 [ACK] Seq=822686247 Ack=1694539142 Win=15466 Len=0
21	0.445596	194.94.127.28	192.168.2.101	**HTTP**	**HTTP/1.1 200 OK**
22	0.446938	194.94.127.28	192.168.2.101	TCP	http > 3374 [FIN, ACK] Seq=822686458 Ack=1694539142 Win=15466 Len=0
23	0.447029	192.168.2.101	194.94.127.28	TCP	3374 > http [ACK] Seq=1694539142 Ack=822686459 Win=64465 Len=0
24	0.450143	192.168.2.101	194.94.127.28	TCP	3374 > http [FIN, ACK] Seq=1694539142 Ack=822686459 Win=64465 Len=0
25	0.508785	194.94.127.28	192.168.2.101	**HTTP**	**HTTP/1.1 200 OK**
26	0.524103	194.94.127.28	192.168.2.101	**HTTP**	**Continuation**
27	0.524186	192.168.2.101	194.94.127.28	TCP	3373 > http [ACK] Seq=1694490522 Ack=818291926 Win=64676 Len=0
28	0.525785	194.94.127.28	192.168.2.101	TCP	http > 3374 [ACK] Seq=822686459 Ack=1694539143 Win=15466 Len=0
29	0.617223	194.94.127.28	192.168.2.101	**HTTP**	**Continuation**
30	0.632978	194.94.127.28	192.168.2.101	**HTTP**	**Continuation**
31	0.633062	192.168.2.101	194.94.127.28	TCP	3373 > http [ACK] Seq=1694490522 Ack=818294738 Win=64676 Len=0
32	0.636820	194.94.127.28	192.168.2.101	**HTTP**	**Continuation**
33	0.636869	192.168.2.101	194.94.127.28	TCP	3373 > http [ACK] Seq=1694490522 Ack=818295086 Win=64329 Len=0
34	1.589574	192.168.2.101	194.94.127.28	TCP	3373 > http [FIN, ACK] Seq=1694490522 Ack=818295086 Win=64329 Len=0
35	1.664994	194.94.127.28	192.168.2.101	TCP	http > 3373 [ACK] Seq=818295086 Ack=1694490523 Win=15466 Len=0

Hypertext Transfer Protocol (HTTP 1.0)
→ Protokollmitschnitt - Beispiel 2b (13/17) - Statistik



- Summe der Bytes = 9.779 (das sind nur ca. 18 % ohne Cache!)
- Summe der TCP-Verbindungen = 3 (Port 3372 bis 3374)
- Zeit: 1.66 s
- Da mit PHP gearbeitet wird, wurde die 1. Seite (Beispiel 2a) nicht gespeichert!

Hypertext Transfer Protocol (HTTP 1.1)
→ Protokollmitschnitt - Beispiel 2c (14/17)



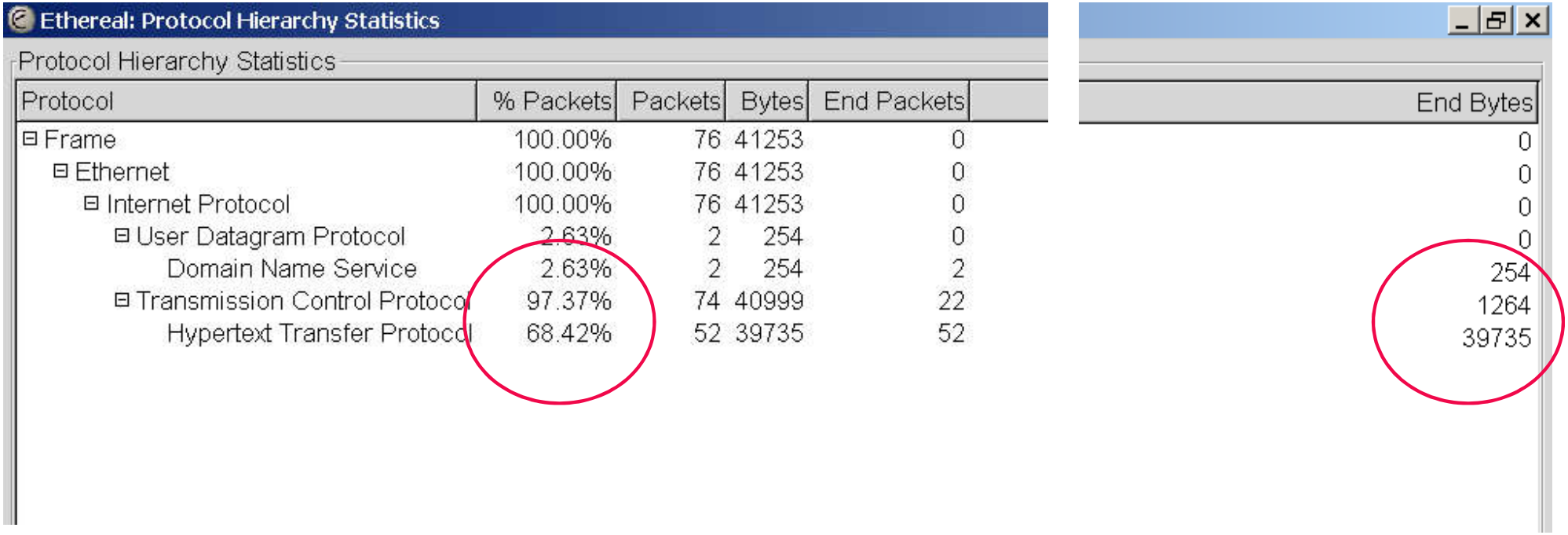
Hypertext Transfer Protocol (HTTP 1.1)
→ Protokollmitschnitt - Beispiel 2c (15/17)

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.2.101	192.168.2.1	DNS	Standard query A skripte.informatik.fh-gelsenkirchen.de
2	0.092605	192.168.2.1	192.168.2.101	DNS	Standard query response A 194.94.127.28
3	0.094621	192.168.2.101	194.94.127.28	TCP	3059 > http [SYN] Seq=464346021 Ack=0 Win=64240 Len=0
4	0.172324	194.94.127.28	192.168.2.101	TCP	http > 3059 [SYN, ACK] Seq=1051269194 Ack=464346022 Win=15466 Len=0
5	0.172417	192.168.2.101	194.94.127.28	TCP	3059 > http [ACK] Seq=464346022 Ack=1051269195 Win=64676 Len=0
6	0.172687	192.168.2.101	194.94.127.28	HTTP	GET / HTTP/1.1
7	0.271751	194.94.127.28	192.168.2.101	TCP	http > 3059 [ACK] Seq=1051269195 Ack=464346379 Win=15466 Len=0
8	0.296149	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
9	0.303674	192.168.2.101	194.94.127.28	HTTP	GET /handler.php HTTP/1.1
10	0.315209	192.168.2.101	194.94.127.28	TCP	3060 > http [SYN] Seq=464429656 Ack=0 Win=64240 Len=0
11	0.405858	194.94.127.28	192.168.2.101	TCP	http > 3060 [SYN, ACK] Seq=1043718677 Ack=464429657 Win=15466 Len=0
12	0.405953	192.168.2.101	194.94.127.28	TCP	3060 > http [ACK] Seq=464429657 Ack=1043718678 Win=64676 Len=0
13	0.406257	192.168.2.101	194.94.127.28	HTTP	GET /blank.html HTTP/1.1
14	0.416216	194.94.127.28	192.168.2.101	TCP	http > 3059 [ACK] Seq=1051269781 Ack=464346803 Win=15466 Len=0
15	0.506624	194.94.127.28	192.168.2.101	TCP	http > 3060 [ACK] Seq=1043718678 Ack=464430080 Win=15466 Len=0
16	0.541791	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
17	0.640486	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
18	0.646557	192.168.2.101	194.94.127.28	TCP	3060 > http [ACK] Seq=464430080 Ack=1043718967 Win=64387 Len=0
19	0.647045	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/theme.css HTTP/1.1
20	0.655787	194.94.127.28	192.168.2.101	HTTP	Continuation
21	0.655872	192.168.2.101	194.94.127.28	TCP	3059 > http [ACK] Seq=464346803 Ack=1051272593 Win=64676 Len=0
22	0.671073	194.94.127.28	192.168.2.101	HTTP	Continuation
23	0.767362	194.94.127.28	192.168.2.101	HTTP	Continuation
24	0.767449	192.168.2.101	194.94.127.28	TCP	3059 > http [ACK] Seq=464346803 Ack=1051275405 Win=64676 Len=0
25	0.772621	194.94.127.28	192.168.2.101	HTTP	Continuation
26	0.775927	192.168.2.101	194.94.127.28	HTTP	GET /miolo/common.js HTTP/1.1
27	0.788509	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
28	0.803372	194.94.127.28	192.168.2.101	HTTP	Continuation
29	0.803442	192.168.2.101	194.94.127.28	TCP	3060 > http [ACK] Seq=464430455 Ack=1043721779 Win=64676 Len=0
30	0.818641	194.94.127.28	192.168.2.101	HTTP	Continuation
31	0.887326	194.94.127.28	192.168.2.101	HTTP	Continuation
32	0.887403	192.168.2.101	194.94.127.28	TCP	3060 > http [ACK] Seq=464430455 Ack=1043723513 Win=64676 Len=0
33	0.905935	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
34	0.920800	194.94.127.28	192.168.2.101	HTTP	Continuation
35	0.920874	192.168.2.101	194.94.127.28	TCP	3059 > http [ACK] Seq=464347165 Ack=1051278707 Win=64676 Len=0
36	0.947561	194.94.127.28	192.168.2.101	HTTP	Continuation
37	0.950936	194.94.127.28	192.168.2.101	HTTP	Continuation

Hypertext Transfer Protocol (HTTP 1.1)
→ Protokollmitschnitt - Beispiel 2c (16/17)

No.	Time	Source	Destination	Protocol	Info
38	0.951002	192.168.2.101	194.94.127.28	TCP	3059 > http [ACK] Seq=464347165 Ack=1051281519 Win=64676 Len=0
39	0.966226	194.94.127.28	192.168.2.101	HTTP	Continuation
40	0.981528	194.94.127.28	192.168.2.101	HTTP	Continuation
41	0.981610	192.168.2.101	194.94.127.28	TCP	3059 > http [ACK] Seq=464347165 Ack=1051284331 Win=64676 Len=0
42	1.016168	194.94.127.28	192.168.2.101	HTTP	Continuation
43	1.031459	194.94.127.28	192.168.2.101	HTTP	Continuation
44	1.031506	192.168.2.101	194.94.127.28	TCP	3059 > http [ACK] Seq=464347165 Ack=1051287143 Win=64676 Len=0
45	1.046754	194.94.127.28	192.168.2.101	HTTP	Continuation
46	1.058858	194.94.127.28	192.168.2.101	HTTP	Continuation
47	1.058923	192.168.2.101	194.94.127.28	TCP	3059 > http [ACK] Seq=464347165 Ack=1051289642 Win=64676 Len=0
48	1.076263	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//pix.gif HTTP/1.1
49	1.076576	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//cl.gif HTTP/1.1
50	1.186009	194.94.127.28	192.168.2.101	TCP	http > 3060 [ACK] Seq=1043723513 Ack=464430829 Win=15466 Len=0
51	1.195056	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
52	1.195736	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//cr.gif HTTP/1.1
53	1.201365	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
54	1.259278	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_left_spacer.gif HTTP/1.1
55	1.298101	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
56	1.298815	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_left_front.gif HTTP/1.1
57	1.370528	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
58	1.379717	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_right_front.gif HTTP/1.1
59	1.398440	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
60	1.403179	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_left_back.gif HTTP/1.1
61	1.481665	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
62	1.485155	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_right_back.gif HTTP/1.1
63	1.507769	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
64	1.512579	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_right_spacer.gif HTTP/1.1
65	1.587401	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
66	1.592343	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//sl.gif HTTP/1.1
67	1.613495	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
68	1.634290	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_front.gif HTTP/1.1
69	1.692674	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
70	1.693603	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash/tab_center_back.gif HTTP/1.1
71	1.735878	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
72	1.739453	192.168.2.101	194.94.127.28	HTTP	GET /miolo/themes/slash//sr.gif HTTP/1.1
73	1.795267	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
74	1.840270	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
75	1.949099	192.168.2.101	194.94.127.28	TCP	3060 > http [ACK] Seq=464433114 Ack=1043726059 Win=63612 Len=0
76	1.949149	192.168.2.101	194.94.127.28	TCP	3059 > http [ACK] Seq=464349450 Ack=1051291856 Win=63955 Len=0

Hypertext Transfer Protocol (HTTP 1.1)
→ Protokollmitschnitt - Beispiel 2c (17/17)



- Summe der Bytes = 41.253 (das sind ca. **76 % von HTTP 1.0**)
- Summe der TCP-Verbindungen = 2 (Port 3059 und 3060)
  - insgesamt 18 GET-Anforderungen
  - insgesamt 18 unterschiedliche Dateien
- Zeit: 1.94 s
- Ist effektiver als Beispiel 2a (HTTP 1.0)

Hypertext Transfer Protocol (HTTP 1.1)
→ Protokollmitschnitt - Beispiel 3 (1/2)

Login per „post“-Operator über Port 3200

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.2.101	192.168.2.1	DNS	Standard query A skripte.informatik.fh-gelsenkirchen.de
2	0.000872	192.168.2.1	192.168.2.101	DNS	Standard query response A 194.94.127.28
3	0.004279	192.168.2.101	194.94.127.28	TCP	3200 > http [SYN] Seq=3572450486 Ack=0 Win=64240 Len=0
4	0.083644	194.94.127.28	192.168.2.101	TCP	http > 3200 [SYN, ACK] Seq=1283757424 Ack=3572450487 Win=15466 Len=0
5	0.083739	192.168.2.101	194.94.127.28	TCP	3200 > http [ACK] Seq=3572450487 Ack=1283757425 Win=64676 Len=0
6	0.084001	192.168.2.101	194.94.127.28	HTTP	POST /handler.php?module=common&action=login&item= HTTP/1.1
7	0.084043	192.168.2.101	194.94.127.28	HTTP	Continuation
8	0.195218	194.94.127.28	192.168.2.101	TCP	http > 3200 [ACK] Seq=1283757425 Ack=3572451104 Win=15466 Len=0
9	0.239763	194.94.127.28	192.168.2.101	TCP	http > 3200 [ACK] Seq=1283757425 Ack=3572451596 Win=15466 Len=0
10	0.624699	194.94.127.28	192.168.2.101	HTTP	HTTP/1.1 200 OK
11	0.640004	194.94.127.28	192.168.2.101	HTTP	Continuation
12	0.640095	192.168.2.101	194.94.127.28	TCP	3200 > http [ACK] Seq=3572451596 Ack=1283760237 Win=64676 Len=0
13	0.732675	194.94.127.28	192.168.2.101	HTTP	Continuation
14	0.748422	194.94.127.28	192.168.2.101	HTTP	Continuation
15	0.748511	192.168.2.101	194.94.127.28	TCP	3200 > http [ACK] Seq=3572451596 Ack=1283763049 Win=64676 Len=0
16	0.758236	194.94.127.28	192.168.2.101	HTTP	Continuation
17	0.944406	192.168.2.101	194.94.127.28	TCP	3200 > http [ACK] Seq=3572451596 Ack=1283763899 Win=63826 Len=0

Hypertext Transfer Protocol (HTTP 1.1)
→ Protokollmitschnitt - Beispiel 3 (2/2)

```
No. Time       Source          Destination     Protocol Info
  6 0.084001   192.168.2.101   194.94.127.28   HTTP     POST /handler.php?module=common&action=login&item= HTTP/1.1

POST /handler.php?module=common&action=login&item= HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/msword,
application/vnd.ms-powerpoint, */*
Referer: http://skripte.informatik.fh-gelsenkirchen.de/handler.php
Accept-Language: de
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Q312461)
Host: skripte.informatik.fh-gelsenkirchen.de
Content-Length: 492
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: PHPSESSID=2e1d1ab402d97b8045d9b7821cf6ac2f

frm_uid=stud_nwm&frm_pwd=nwmnwm&frm_hidden=Neue+Anmeldung+nur+mit+ausdr%FCcklicher+Erlaubnis+des+Professors%21&enviar=Best%E4
tigen&frm_tentativas=&frm_login=&frm_nome=&frm_nachname=&frm_email=&frm_matrik=&frm_pr_code=&frm_=Das+Kennwort+wird+zu+Ihrer+
Email-
Adresse+geschickt.%3Cbr%3EDeshalb+m%FCssen+Sie+unbedingt+eine+g%FCltige+angeben%2C+sonst+ist+es+Ihnen+nicht+m%F6glich%2C+sich
+an+das+System+anzumelden+und+der+angegebene+Benutzername+verf%E4llt.&frm_currpage_=Skripte+Login&frm_submit_=1

No. Time       Source          Destination     Protocol Info
 10 0.624699   194.94.127.28   192.168.2.101   HTTP     HTTP/1.1 200 OK

HTTP/1.1 200 OK
Date: Fri, 08 Aug 2003 09:55:16 GMT
Server: Apache/1.3.22 (Unix) PHP/4.3.2
X-Powered-By: PHP/4.3.2
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

... neues Dokument ...
```

Problem: Passwort wird im ***Klartext*** übertragen

Hypertext Transfer Protocol 2.0 (HTTP/2) → Einführung

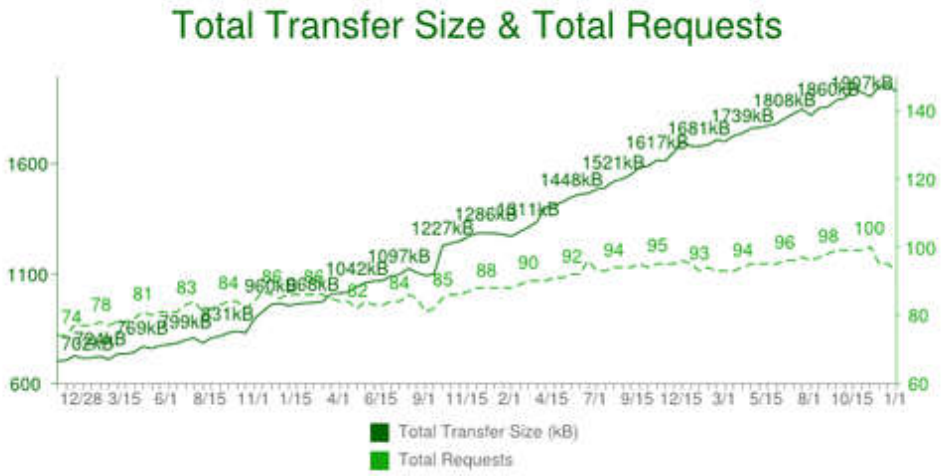
- seit Mai 2015 ist HTTP/2 in RFC 7540 spezifiziert
- Ausgangspunkt von HTTP/2 war SPDY
- Die existierende Syntax aus HTTP/1.x bleibt bestehen
- Verbessert die Client/Server-Kommunikation
- Latenzzeiten werden durch Headerkompression vermindert

Hypertext Transfer Protocol 2.0 (HTTP/2)
→ HTTP/2 Motivation

Overhead

- Weiterentwicklung des Internets
  - Früher statisch, heute dynamischer
  - Stichwörter: Web 2.0 und Ajax
  - Zu großer Overhead bei HTTP/1.1
    - Mit HTTP/1.1 nur ein ausstehender TCP-Request möglich
    - Parallele Requests haben viel Overhead
      → beeinträchtigen die Performanz
- HTTP/2 soll die Latenzzeiten senken, um bis zu 50%

Webseiten werden größer



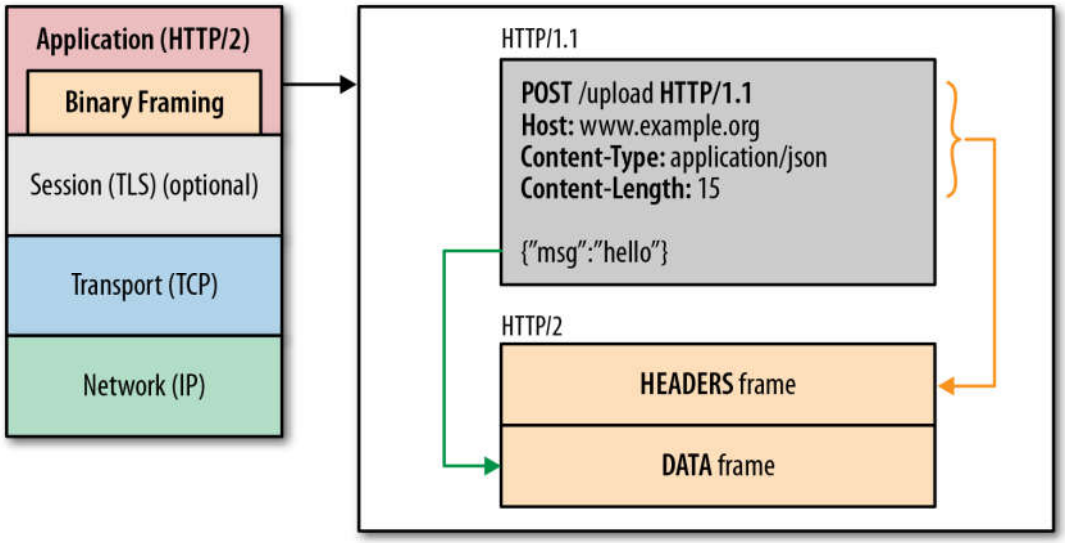
Über die Zeit steigt die Größe der Webseiten und damit auch die Anzahl an Requests
=> Anstieg der Pageload-Zeit

Hypertext Transfer Protocol 2.0 (HTTP/2)
→ Grundgerüst HTTP/2

- GET, POST, … bleiben erhalten (Infrastruktur des Netzwerks bleibt gleich)
- HTML-Markup wird benutzt (Webseitenaufbau kann beibehalten werden)
- PUSH ermöglicht es dem Server aktiv Antworten an Clients zu versenden
- Verschlüsselung ist nicht zwingend erforderlich, die meiste Browser bieten HTTP/2 jedoch nur verschlüsselt (z. B. Firefox und Chrome)

Hypertext Transfer Protocol 2.0 (HTTP/2) → Neue Funktionen - Basic Features(1/4)

- Binäres statt textuelles Protokoll
- Bei der Übertragung werden die Header in Binärdateien umgewandelt
  - Analyse von diesen Headern nur mit Hilfe von Wireshark oder ähnlichen Tools möglich
  - Fehler durch unterschiedliche Terminierungssequenzen, optionale Whitespaces werden vermieden



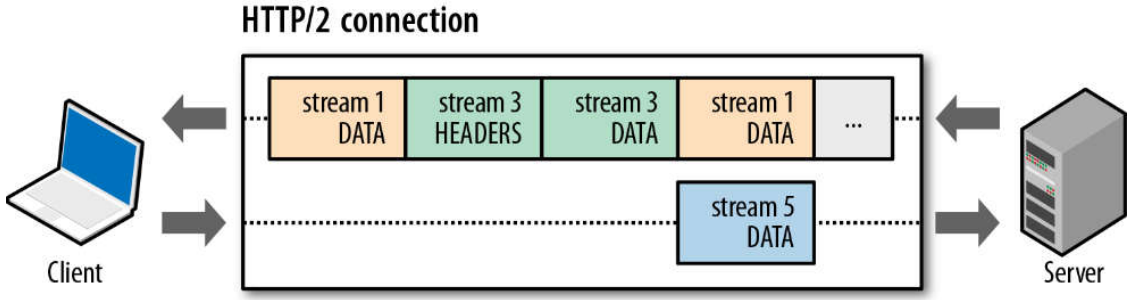
Die HTTP-Struktur wird in zwei Frames gepackt (Message) und diese werden zwischen Client und Server bidirektional verschickt

Bit	+0..7	+8..15	+16..23	+24..31
0	Length			Type
32	Flags			
40	R	Stream Identifier		
...	*Frame Payload*			

Der verwendete 9-Byte Binärframe

Hypertext Transfer Protocol 2.0 (HTTP/2) → Neue Funktionen - Basic Features(2/4)

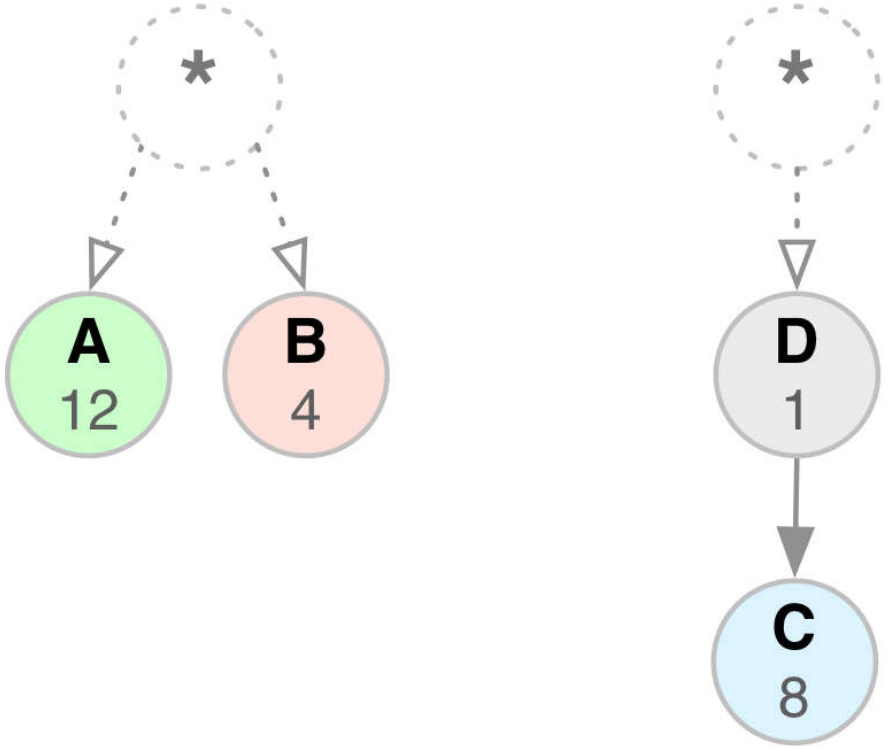
- Multiplexe Datenströme über **eine** TCP Verbindung
  - Persistente Pipelined Verbindung
  - Kein „Head-of-Line“-Blocking auf HTTP-Ebene mehr
  - Messages bestehen z.B. aus HEADERS und DATA
    - Optional: PRIORITY



Insgesamt sind drei Streams auf einer Leitung. Die Frames können überlappen und werden am Endpunkt wieder zusammengesetzt.

Hypertext Transfer Protocol 2.0 (HTTP/2) → Neue Funktionen - Basic Features(3/4)

- Priorisierung möglich
  - Bitte bearbeite erst D und dann C
- Gewichtung möglich
  - A hat ein Gewicht von 12, B von 4 => A sollte 2/3 der verfügbaren Ressourcen erhalten
- Der Server kann dies berücksichtigen, muss es aber nicht!



A,B,C,D sind Streams

Hypertext Transfer Protocol 2.0 (HTTP/2)
→ Neue Funktionen - Basic Features(4/4)

- **Kompression der Header**
  - HPACK reduziert die Header um ~85% für requests und responses
  - HPACK ist nicht- wie zlib bei SPDY, vom CRIME-Exploit betroffen

Hinweis: HPACK ist der Kompressionsalgorithmus den HTTP/2 nutzt

⇒ Reduzierung der Menge an Datenpaketen und die Menge an Bytes wird kleiner

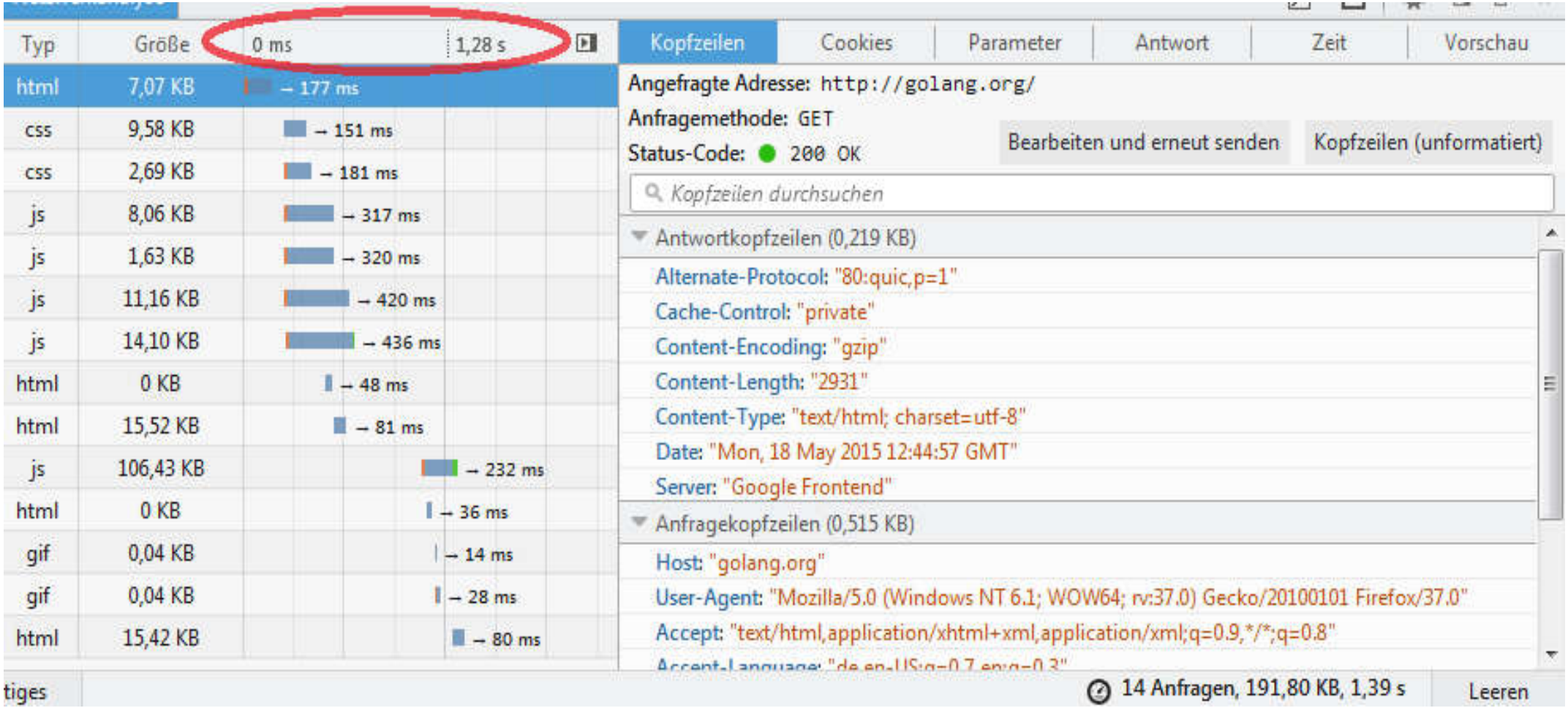
- **Server Push**
  - lässt den Server Daten in den Cache des Clients schreiben
  - Wenn der Client Ressource X anfordert und der Server „intelligent“ schließt, dass der Client dann auch Ressource Z benötigt, kann der Server diese schon in den Cache des Clients „pushen“

Hypertext Transfer Protocol 2.0 (HTTP/2) → Effizienz

- Bei HTTP/1.x enthalten 74% der aktiven TCP-Verbindungen nur eine HTTP-Operation
- HTTP/2 reduziert diese kostenintensiven Verbindungen auf 25%
  - Eine Menge Overhead wird gespart!
- Dies schont auch die Server/Client-Ressourcen
  - Operative Kosten werden zusätzlich reduziert!

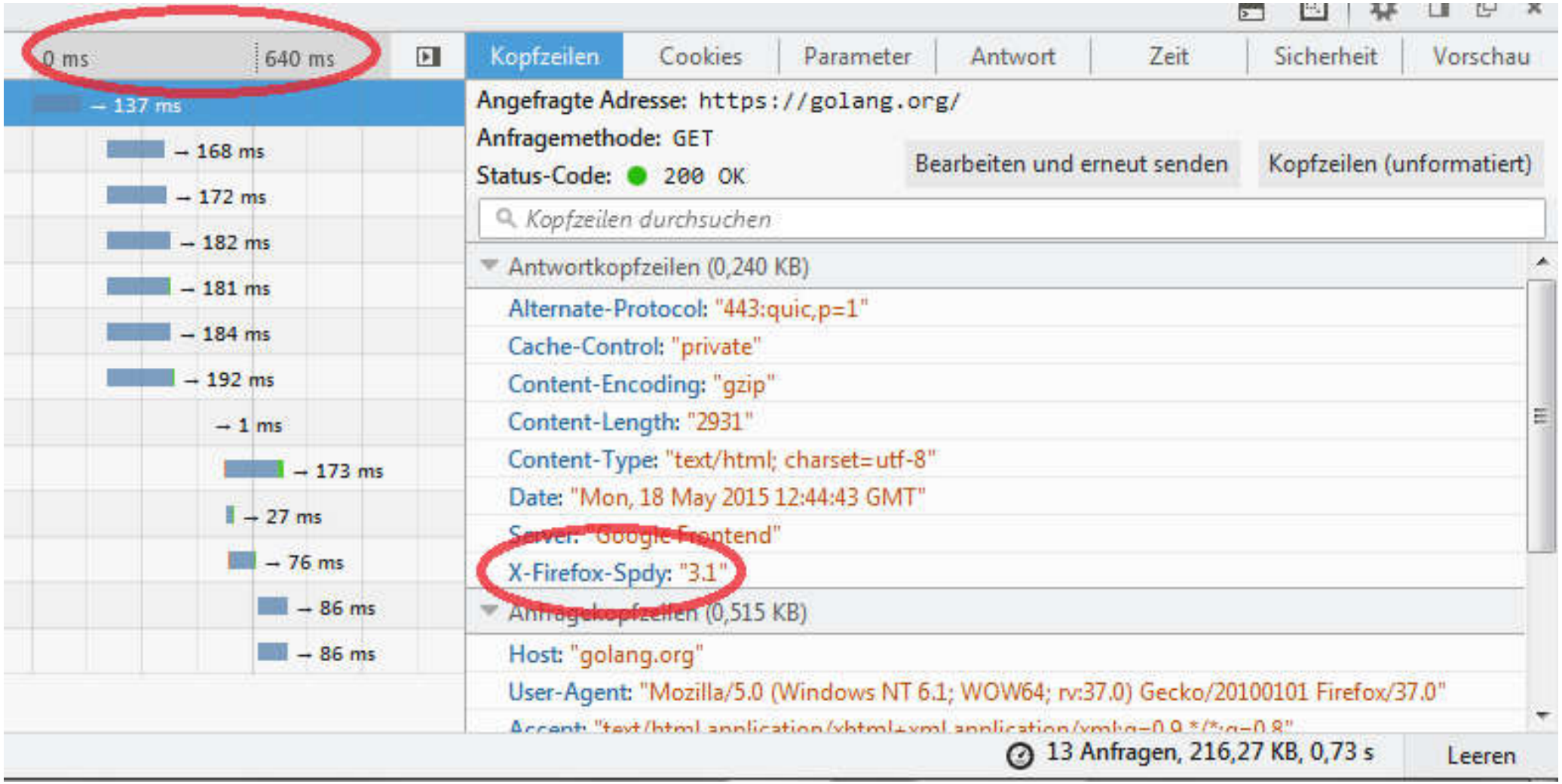
Hypertext Transfer Protocol 2.0 (HTTP/2) → Ein kleiner Test(1/2)

Testseite: http://www.golang.org



Hypertext Transfer Protocol 2.0 (HTTP/2) → Ein kleiner Test (2/2)

Testseite: https://www.golang.org



Hypertext Transfer Protocol 2.0 (HTTP/2)
→ Probleme

- Viele Proxies und Firewalls unterstützen HTTP/2 noch nicht
  - Viele Seiten die für HTTP/1.1 optimiert sind, merken anfangs nichts von diesem Performanceschub

- Durch das Entfernen von Performance-Bottlenecks entstehen an anderen Stellen Neue:
  - Head-Of-Line-Blocking nicht mehr bei HTTP, dafür jetzt bei TCP!

Hypertext Transfer Protocol 2.0 (HTTP/2)
→ Zusammenfassung und Ausblick

- Nur eine TCP-Connection pro Verbindung
- Request → Stream
  - Multiplexen
  - Priorisieren
- Binäres Protokoll
  - Priorisierung
  - „Flow Control“
  - Server Push
- Header-Kompression

Inhalt

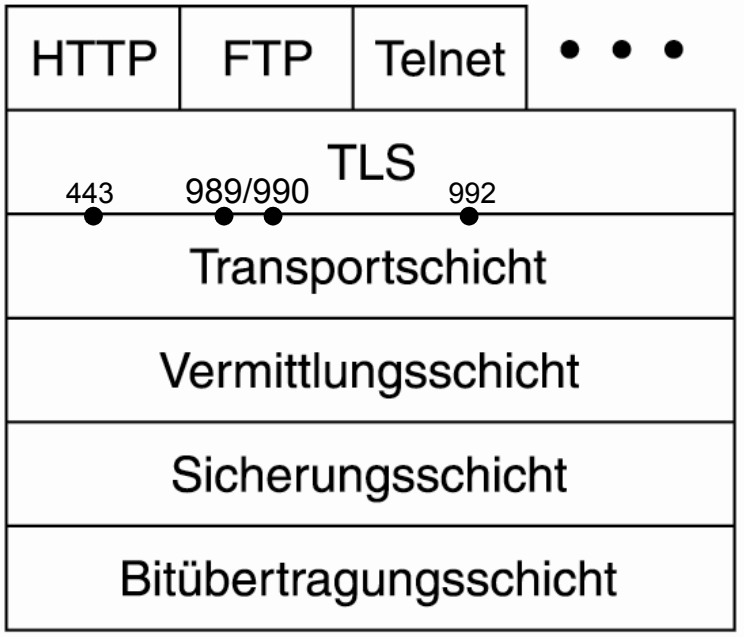
- Ziele und Einordnung
- Das World Wide Web
- HTTP - Hypertext Transfer Protocol
  - HTTP - Methoden / Operationen
  - HTTP - Nachrichten
  - HTTP - Verbindungen
- **Transport Layer Security (TLS) oder SSL**
- HTTP - Caching
- HTTP - Server-Cluster
- Zusammenfassung

Hypertext Transfer Protocol (HTTP)
→ Transport Layer Security (TLS) oder SSL

- Betrachtet man die offene Natur des Internets, ist die Nutzung eines Sicherheitsdienstes, der die Kommunikation von Client und Server gegen verschiedene Angriffe schützt, von wesentlicher Bedeutung.
- Die meisten Sicherheitsaspekte im Web haben mit der Einrichtung einer vertrauenswürdigen Verbindung zwischen Client und Server zu tun.
- Der vorherrschende Ansatz im Web ist die Verwendung von **SSL (Secure Socket Layer)**, ursprünglich von Netscape vorgeschlagen.
- Obwohl SSL niemals formal standardisiert wurde, wird es von den meisten Web-Clients und -Servern unterstützt.
- Vor einiger Zeit wurde eine Aktualisierung von SSL vorgenommen, zunächst in RFC 2246 abgelegt und heute als **TLS (Transport Layer Security)** bezeichnet.
- TLS ist ein applikationsunabhäniges Sicherheitsprotokoll, das logisch auf einem Transportprotokoll aufsetzt.

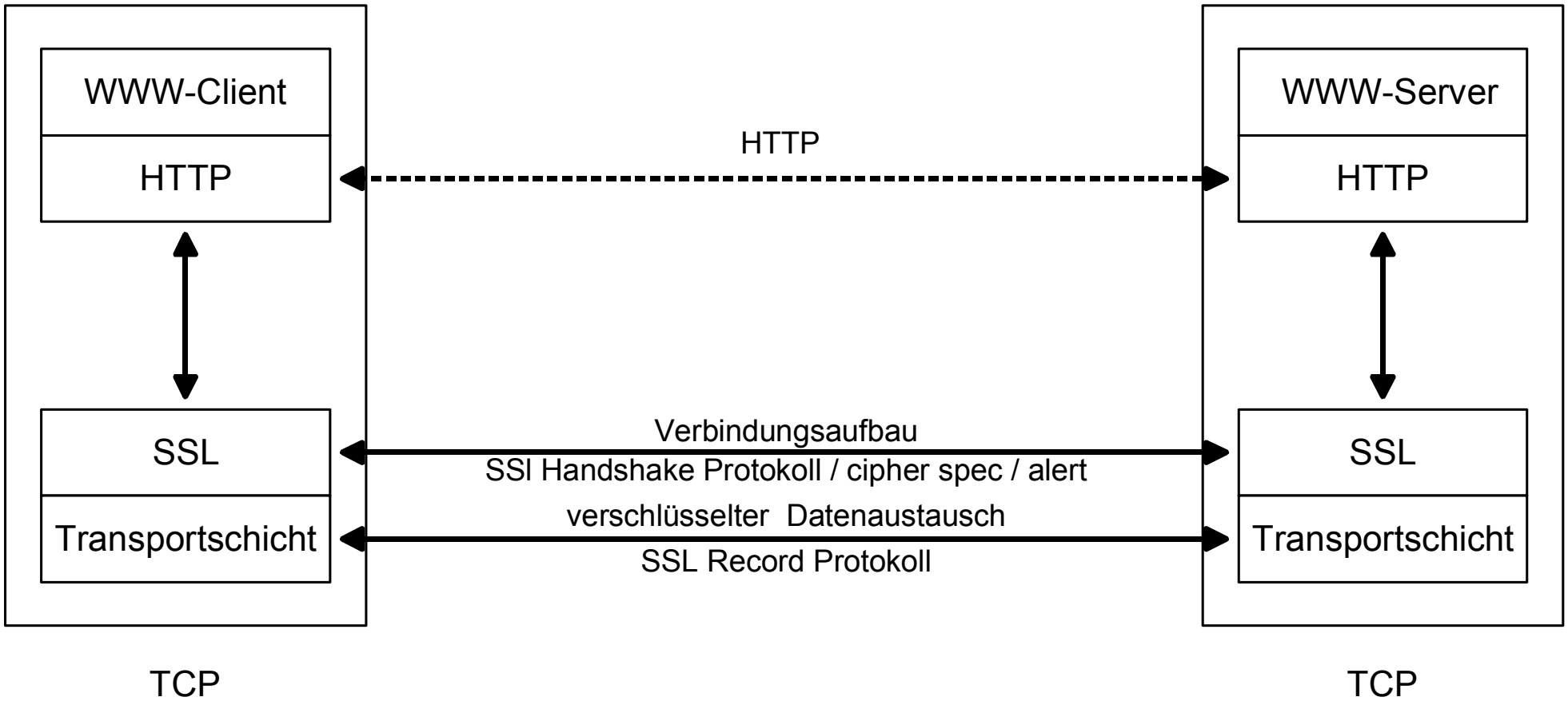
Hypertext Transfer Protocol (HTTP)
→ Transport Layer Security (TLS) oder SSL

- TLS kann eine Vielzahl höherer Protokolle unterstützen (HTTP, FTP, SMTP, Telenet, ...).
- TLS ist in zwei Schichten angeordnet.
- Den Kern des Protokolls bildet eine TLS-Datensatz-Protokollschicht, die einen sicheren Kanal zwischen Client und einem Server implementiert.



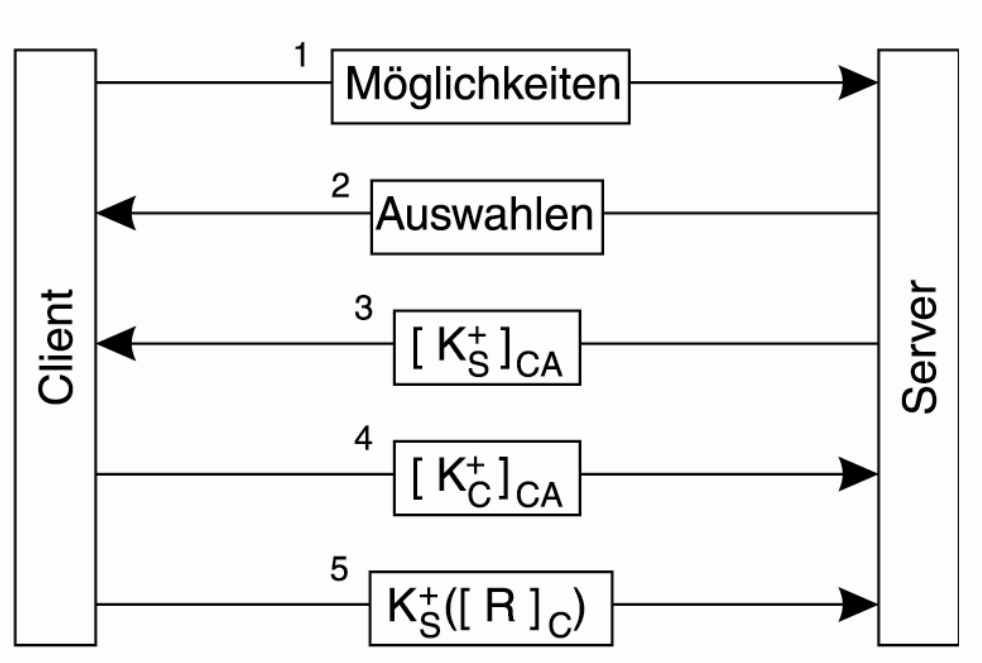
- Die genauen Eigenschaften des Kanals werden bei der Einrichtung festgelegt, können aber Nachrichtenfragmentierung und Komprimierung beinhalten, die in Kombination mit Authentikation, Integrität und Vertraulichkeit angewendet werden.

Hypertext Transfer Protocol (HTTP)
→ Transport Layer Security (TLS) oder SSL



Hypertext Transfer Protocol (HTTP)
→ Transport Layer Security (TLS) oder SSL

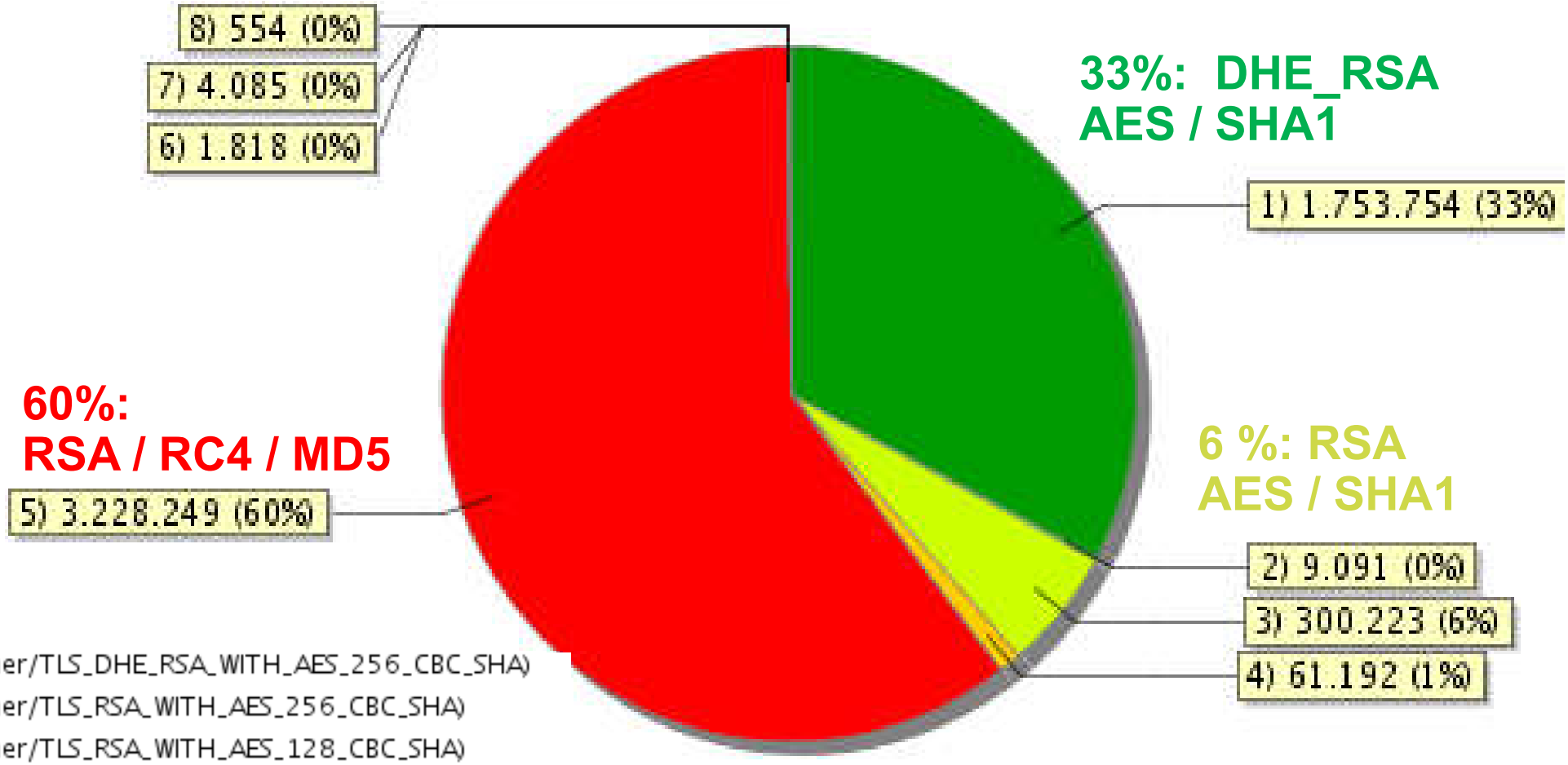
- Die Einrichtung eines sicheren Kanals erfolgt in zwei Phasen.
- Zuerst informiert der Client den Server über die kryptographischen Algorithmen, die er anwenden kann, sowie über ggf. unterstützte Komprimierungsmethoden.
- Die eigentliche Auswahl bleibt dem Server überlassen, der seine Entscheidung an den Client zurückmeldet.
- In der zweiten Phase findet die Authentikation statt.
- usw.



Detaillierte Beschreibung in der Vorlesung „Internet-Sicherheit A“ (ISA)!

Beispiele von Ergebnissen des IAS
→ Wissensbasis: Technologietrend (TLS)

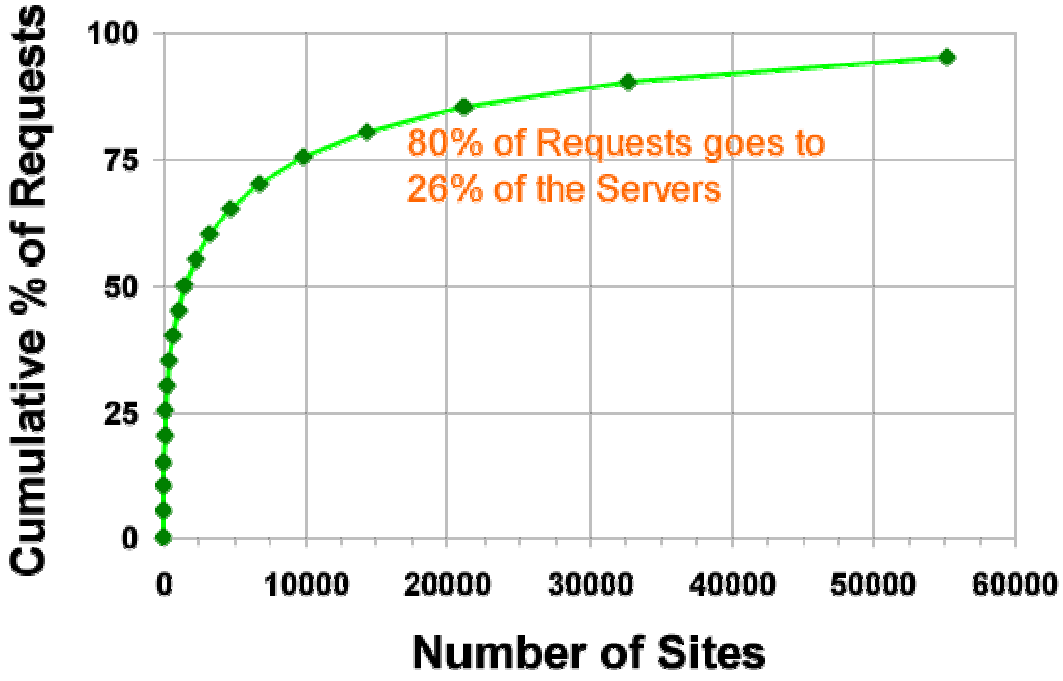
!! 0.1 %: RSA / Export (40) / SHA1 and 0.01 %: RSA / NULL / SHA1 !!



Inhalt

HTTP - Caching
→ Motivation (1/2)

- Eine einfache Methode der Leistungssteigerung ist, angeforderte Seiten für den Fall zu speichern, dass sie erneut verwendet werden.
- Diese Technik ist besonders bei Seiten effektiv, die häufig besucht werden.
- Wie ist die Verteilung auf Zugriffe von Web-Seiten?



- **50% der Zugriffe gehen auf 1% der Web-Seiten**
- 80% der Zugriffe gehen auf 26% der Web-Seiten

HTTP - Caching
→ Motivation (2/2)

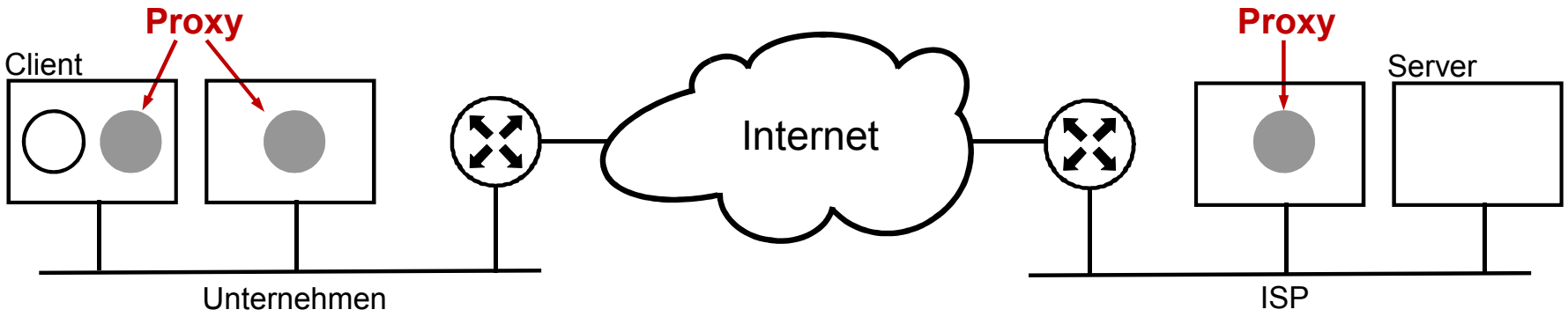
- Die Aufbewahrung von Seiten für zukünftige Zugriffe wird als **Caching** bezeichnet (Ablegen von Seiten im Cache).
- Die übliche Vorgehensweise ist, dass ein Prozess, der als Proxy bezeichnet wird, den Cache verwaltet.
- Um Caching zu verwenden, kann der Browser so konfiguriert werden, dass alle Seitenanforderungen an einen Proxy statt an den eigentlichen Server der Seite gesendet werden.
- Wenn der Proxy die Seite besitzt, wird diese sofort zurückgegeben, wenn nicht, wird die Seite vom Server geholt und im Cache für zukünftige Zugriffe abgelegt und sodann an den anfordernden Client übertragen.
- Beim Caching stellen sich zwei wichtige Fragen:
  - Wer sollte das Caching ausführen?
  - Wie lange sollten Seiten im Cache gehalten werden?

HTTP - Caching
→ Wer soll das Caching ausführen? (1/2)

- Die Browser auf den Rechnersystemen führen oftmals Proxies aus, so dass sie schnell die kürzlich besuchten Seiten nachschlagen können.
- In einem Unternehmen ist der Proxy oft ein Rechner, der von allen Rechnern im Unternehmen gemeinsam genutzt wird.
- Wenn ein Benutzer eine bestimmte Seite anzeigt und ein anderer Benutzer im gleichen Unternehmen möchte diese Seite ebenfalls ansehen, kann diese aus dem Cache des Unternehmens-Proxy abgerufen werden.
- Viele ISPs setzen ebenso Proxies ein, um den Zugriff für alle Kunden zu beschleunigen.
- Oftmals arbeiten alle Cache-Speicher gleichzeitig, so dass Anforderungen zuerst zum lokalen Browser-Cache gehen.
- Wenn dies nicht erfolgreich ist, fragt der lokale Cache den Unternehmens-Proxy.
- Wenn dieser nicht erfolgreich ist, fragt der Unternehmens-Proxy den ISP-Proxy.

HTTP - Caching
→ Wer soll das Caching ausführen? (1/2)

- Dieser muss erfolgreich sein, entweder durch Abruf des Caches oder vom gewünschten Server direkt.
- Ein Schema, das die Abfrage mehrerer Cache-Speicher nacheinander beinhaltet, wird als ***hierarchisches Caching*** bezeichnet.



HTTP - Caching
→ Wie lange sollten Seiten im Cache sein? (1/4)

- Einige Seiten sollten überhaupt nicht im Cache abgelegt werden.
  - So ändert sich z.B. eine Seite, die die Preise der 50 am meisten gehandelten Aktien enthält, jede Sekunde.
  - Wenn diese im Cache abgelegt wird, erhält der Benutzer bei einer Kopie aus dem Cache veraltete Daten.

- Wenn andererseits die Börse am Abend geschlossen hat, bleibt diese Seite für Stunden oder Tage aktuell, bis die nächste Handelsrunde beginnt.
- D.h. die Eignung einer Seite für die **Aufnahmen in den Cache** kann sich **im Laufe der Zeit verändern**.
- Die zentrale Frage, bei der Bestimmung, wann eine Seite aus dem Cache entfernt wird, ist, wie viel veraltete Inhalte Benutzer in Kauf nehmen (da gecachete Seiten auf der Platte gehalten werden, ist der belegte Speicherplatz selten ein Thema).

HTTP - Caching
→ Wie lange sollten Seiten im Cache sein? (2/4)

- Wenn ein Proxy die Seiten zu schnell herauswirft, wird er selten eine veraltete Seite zurückgeben, aber auch nicht sehr effektiv sein (niedrige Trefferquote).
- Wenn er die Seite zu lange aufbewahrt, kann er eine hohe Trefferquote haben, wobei hier aber oftmals veraltete Seiten zurückgegeben werden.
- Ein Ansatz, dieses Problem zu lösen, ist die Verwendung einer **heuristischen Methode**, um zu schätzen, wie lange jede Seite aufbewahrt wird.
- Eine gängige Vorgehensweise ist es, die Aufbewahrungszeit aufgrund des Headers *Last-Modified* zu bestimmen.
  - Wurde eine Seite eine Stunde zuvor modifiziert, wird sie für eine Stunde im Cache behalten.
  - Wenn sie vor einem Jahr verändert wurde, ist es offensichtlich eine sehr stabile Seite (z.B. Liste der Götter aus der griechischen Mythologie), so dass sie ein Jahr aufbewahrt werden kann, ohne dass man befürchten muss, dass sie sich im Laufe des Jahres ändert.
  - Diese heuristische Vorgehensweise hat sich in der Praxis bewährt, ab und zu werden aber veraltete Seiten zurückgeliefert.

HTTP - Caching
→ Wie lange sollten Seiten im Cache sein? (3/4)

- Eine weitere Möglichkeit ist die Nutzung des Anforderungs-Header *If-Modified-Since*, den ein Proxy an den Server senden kann.
  - Er gibt die Seite an, die der Proxy möchte, und den Zeitpunkt, zu dem die gecachte Seite zuletzt modifiziert wurde (anhand des Headers *Last-Modified*).
  - Wenn die Seite seitdem nicht verändert wurde, sendet der Server eine kurze *Not Modified*-Meldung zurück (Status 304), die den Proxy „anweist“, die gecachte Seite zu verwenden (siehe PM Beispiel 1).
  - Wurde die Seite seitdem modifiziert, wird die neue Seite zurückgeliefert.
- Die beiden Ansätze können auch kombiniert werden.

- Web-Seiten mit dynamischen Inhalten (z.B. PHP-Scripts), sollen nie gecacht werden, da die Parameter das nächste mal anders sein können (siehe Beispiel 2b).

HTTP - Caching
→ Wie lange sollten Seiten im Cache sein? (4/4)

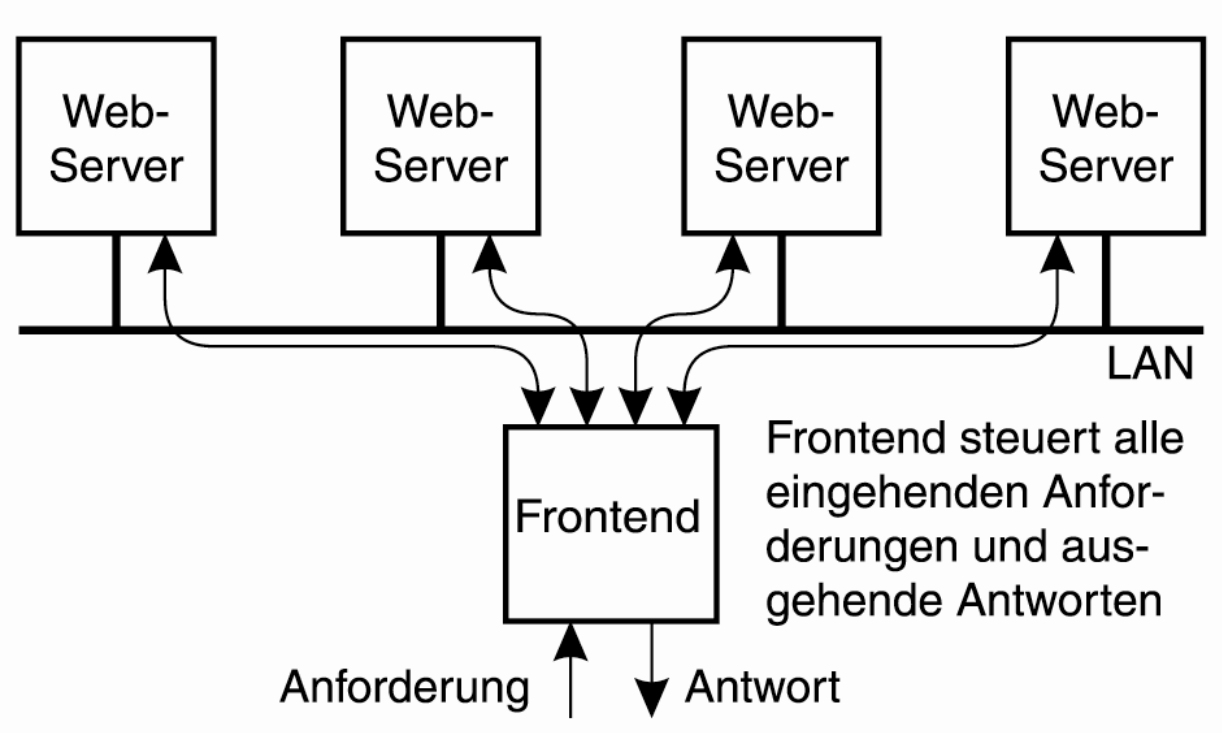
- Ein weiterer Ansatz zur Leistungsoptimierung ist **proaktives Caching**.
- Wenn ein Proxy eine Seite vom Server abruft, kann er die Seite untersuchen, um zu prüfen, ob Hyperlinks vorhanden sind.
- Wenn ja, kann er Anforderungen an den betreffenden Server senden, um den Cache mit den verwiesenen Seiten vorab zu laden, für den Fall, dass sie gebraucht werden.
- Diese Technik kann die Zugriffszeit für die nachfolgende Anforderung reduzieren, aber auch die Übertragungsleitung mit Seiten überfüllen, die nie benötigt werden.

- **Web-Caching ist also alles andere als trivial!**

Inhalt

HTTP - Server-Cluster
→ Das Problem

- Ein wichtiges Problem der Client-Server-Natur des Webs ist, dass ein Web-Server überlastet werden kann.
- Eine praktische Lösung, die in vielen Entwürfen verwendet wird, ist die Replikation eines Servers auf ein Cluster aus Workstations, und einen Frontend zu verwenden, um die Client-Anforderung an eine der parallelen Web-Server weiterzuleiten.



HTTP - Server-Cluster
→ Grundlagen (1/2)

- Ein wichtiger Aspekt dieser Anordnung ist der Entwurf des Frontends, weil es einfach zu einem ernsthaften Leistungsengpass werden kann.
- Im Allgemeinen wird zwischen einem Frontend unterschieden,
  - das als Schalter auf der Transportebene arbeitet, und
  - das auf der Ebene der Applikationsschicht arbeitet.

- **Schalter auf der Transportebene**
  - Wenn ein Client eine HTTP-Anforderung absetzt, richtet er eine TCP-Verbindung zum Frontend ein.
  - Ein Schalter auf der Transportebene gibt einfach die über die TCP-Verbindung gesendeten Daten an einen der gleichen Server weiter, abhängig von der Auslastung der verschiedenen Server.
  - Nachteil dieses Ansatzes ist, dass der Schalter nicht die Inhalte der HTTP-Anforderung berücksichtigen kann.
  - Er kann die Weiterleitung nur von der Server-Auslastung abhängig machen.

HTTP - Server-Cluster
→ Grundlagen (2/2)

- Eine **inhaltsabhängige Verteilung der Anforderungen** ist im Allgemeinen ein besserer Ansatz, wobei das Frontend eine eingehende HTTP-Anforderung zuerst auswertet und dann entscheidet, an welchen Server diese Anforderung weitergegeben werden soll.
- Diese Methode hat mehrere Vorteile.
  - Gibt das Frontend z.B. Anforderungen desselben Dokumentes immer an denselben Server weiter, kann dieser Server das Dokument **effektiv im Cache** ablegen, was zu **besseren Antwortzeiten** führt.
  - Darüberhinaus ist es möglich, die verschiedenen **Dokumente auf die Server zu verteilen**, statt auf jedem Server jedes Dokument zu replizieren.
  - Dieser Ansatz nutzt die verfügbare Speicherkapazität effizienter und erlaubt die Verwendung von **dedizierten Servern**, um spezielle Dokumente, wie z.B. für Audio und Video, bereitzustellen.
- Ein Problem bei der inhaltsabhängigen Verteilung ist, dass das Frontend eine Menge Arbeit übernehmen muss.

Inhalt

- Ziele und Einordnung
- Das World Wide Web
- HTTP - Hypertext Transfer Protocol
  - HTTP - Methoden / Operationen
  - HTTP - Nachrichten
  - HTTP - Verbindungen
- Transport Layer Security (TLS) oder SSL (hier nur Idee)
- HTTP - Caching
- HTTP - Server-Cluster
- **Zusammenfassung**

Hypertext Transfer Protocol (HTTP)
→ Zusammenfassung (1/2)

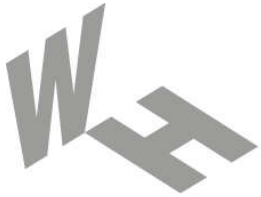
- Das **World Wide Web (WWW)** ist ein riesiges ***verteiltes System***, das aus Millionen von Clients und Servern besteht, die auf verknüpfte Dokumente zugreifen.

- Die ***Server verwalten die Dokumente (Web-Seiten)***, während die ***Clients (Browser)*** den Benutzern eine einfache Schnittstelle für die ***Darstellung und den Zugriff auf diese Dokumente*** bereitstellt.

- Die gesamte Kommunikation im Web zwischen Client und Server basiert auf dem ***Hypertext Transfer Protocol (HTTP)***.
  - HTTP ist ein einfaches Client-Server-Protokoll.
  - HTTP basiert auf TCP und nutzt den Port 80.
  - HTTP kennt Methoden mit denen gewünschte Operationen (GET, POST, ...) durchgeführt werden können.
  - HTTP-Kommunikation findet über Nachrichten mit Headern (z.B. *Accept-Encoding*-Nachrichten-Header) statt.

Hypertext Transfer Protocol (HTTP)
→ Zusammenfassung (2/2)

- **TLS/SSL** ist notwendig, wenn **vertrauenswürdige Kommunikation** über das Web erfolgen soll.
- Mit Hilfe von **Caching** kann eine **hohe Leistungssteigerung** im Web erzielt werden.
- **Server-Cluster** helfen, Web-Server aufzubauen, die sehr **viele Abfragen** beantworten müssen.



Westfälische Hochschule
Gelsenkirchen Bocholt Recklinghausen
University of Applied Sciences

Hypertext Transfer Protocol (HTTP)

Vielen Dank für Ihre Aufmerksamkeit
Fragen ?

Prof. Dr. (TU NN)
Norbert Pohlmann

Institut für Internet-Sicherheit – if(is)
Westfälische Hochschule, Gelsenkirchen
http://www.internet-sicherheit.de

if(is) internet-sicherheit.