1 Protokolle der Anwendungsschicht

Konsultieren Sie sogenannte RFCs der IETF (erklären Sie die Bedeutung dieser Abkürzungen) um folgende Fragestellungen zu beantworten:

1.1 Beschreiben Sie die folgenden Protokolle hinsichtlich ihrer Eigenschaften und wesentlichen Unterschiede: HTTP/0.9, HTTP/1.0, HTTP/1.1, HTTP/2.0.

- ◇ **HTTP/0.9** :: Erstellt 1991.
  Es ist ein Subset des vollen HTTP-Protokolls, wie wir es heute kennen.
  Merkmale:
  i. Kein Austausch von Klientenprofil.
  ii. Einfachheit Request/Response-Model
  iii. Keine Sessions oder States
  iv. Weitverbreitete Nutzung: Request of Data through Browser
  v. Nutzt telnet-Protokolstil via TCP-IP

  Beispiel: Verbindungsaufbau
  Der Klient erstellt eine TCP-IP Verbindung zu einem Host via Domänname oder IP und Port-Nummer. Wird kein Port definiert, so wird der default-Wert 80 gesetzt. Der Server akzeptiert die Verbinung und Datentransfer kann stattfinden.
  telnet mailsrv.aau.at 25 ... verbindung zum Mailserver der AAU auf Port 25

  Beispiel: Request & Response
  Ein HTTP-Request wird an einen Server gestellt.
  "GET http://www.uni-klu.ac.at:80/myapp/index.html"
  Der Server überprüft diese Anfrage.
  Gibt es dieses Dokument?
  Darf der User darauf zugreifen?
  Ist es ein dynamisch generiertes Dokument?
  Ist das GET-Format richtig? ... u.s.w.
  Der Server sendet anschließend eine Nachricht zurück zum Anfragensteller.
  Vgl.: HTTP-Statuscodes[1]
  Information - 100++: Continue, Switching Protocols, Processing
  Erfolgreich - 200++: OK, Accepted, Non-Authoritative Information,...
  Umleitungen - 300++: Moved permanently, See other, Use Proxy[2], ...
  Client-Fehler - 400++: Bad Request, Unauthorized, Forbidden, Not Found, ...
  Server-Fehler - 500++: Internal Server Error, Bad Gateway[3], Service N/A,...
  Der Browser des Anfragenstellers erhält das Dokument und rendert gemäß .css/.php/... . Bei vollständiger Übertragung des abgerufenen Protokolls unterbricht der Server die Verbindung zum Klienten.

[1] Siehe https://de.wikipedia.org/wiki/HTTP-Statuscode

[2] A proxy is a forwarding agent, receiving requests for a URI in its absolute form, rewriting all or parts of the message, and forwarding the reformatted request toward the server identified by the URI.

[3] A gateway is a receiving agent, acting as a layer above some other server(s) and, if necessary, translating the requests to the underlying server's protocol.

◇ **HTTP/1.0** :: Erweitert um 1994 Spezifiziert im RFC 1945. Dies erweitert das HTTP/0.9 Protokoll um weitere TCP-IP Verbindungen für multiplen Datentransfer mittels desselben Requests. Dadurch werden neben Texten des eigentlichen Dokuments eingebettete Bilder, anhand deren Domäne, geladen.
Eine Website welche 5 Bilder beinhaltet besitzt somit 6 Separate TCP-Verbindungen:

- 1. Verbindung: Text
- 2. Verbindung: Bild 1
- 3. Verbindung: Bild 2
- ...
- 6. Verbindung: Bild 5

Es unterstützt das Caching via Header durch If-Modified-Since.

◇ **HTTP/1.1** :: Implementiert 1996 Spezifiziert im RFC 2616 Im Rahmen von HTTP/1.1 wurde das Protokoll um mehrere Funktionen erweitert:

- Support persistenter Verbindungen :: Multiple Requests über eine spezifische Leitung
- Support für Packettransfer sowie Kompression & Dekompression
- Virtual Hosting :: Webserver mit 1 IP kann mehrere Domännamen besitzen
- Verbesserung von Byte Transfers: Wiederaufnahme unterbrochener Vermittlungen
- Sprachensupport
- Funktionserweiterungen 'OPTIONS', 'PUT', 'DELETE',...

Für 1.1 braucht es einen spezifizierten Header für den Host, welcher mitunter die Verbindungsart (normal, persistent) mitbestimmt. Dadurch kann es zu Kompatibilitätsproblemen kommen. Durch HTTP-Pipelining können mehrere Requests & Responses innerhalb derselben Verbindung stattfinden.
Im Vergleich zu 1.0, brauchte man quasi per Element eine eigene Verbindung, hier jedoch nicht. Der Transfer aller Elemente geschieht durch diesselbe TCP-Verbindung, was auch der Zeitkomplexität zugute kommt. Man kann im Rahmen des HTTP/1.1 auch Daten dem Server übertragen, mittels PUT und DELETE.

◇ **HTTP/2.0** :: Erstmals 2015 Spezifiziert von der IETF als Nachfolger von HTTP/11 und definiert in RFC 7540 und 7541. Primär auf Optimierungen bezogen, gelten folgende Änderungen:

- Übertragungsbeschleunigung mittels Zusammenfassen mehrerer Requests :: via Multiplex
- Erweiterungen der Datenkompression :: HPACK-Algorithmus, Kompression beinhaltet Kopfdaten
- Binär kodierte Übertragung von Inhalten - Server-initiierte Datenübertragung :: Push-Verfahren

HTTP-Requestmethoden:

GET: Parameterübergabe in der URL. Gekennzeichnet durch das '?' *GET /wiki/Spezial:Search?search=Katzen&go=Artikel*

◇ Public Data

- ◇ Can be cached thus remain in browser history
- ◇ Längenrestriktion (2048 Characters for any URL)
- ◇ Encoded in URL
- ◇ ASCII-Characters only

POST: Parameterübergabe im Kopf des HTTP-Request

POST my/demoformat/index.php HTTP/1.1
Host: streifenmafiadomain.ru
name=putin&pet=vodka&nastrovje=cheers

- ◇ Private Data
- ◇ Never cached, thus do not remain in browser history
- ◇ Have no restriction on Data or Length
- ◇ Encoded in Head/Body. Enabled Multipart encoding for binaries.
- ◇ Any type allowed

HEAD: Same as GET but merely returns HTTP header & no document body. Validates Cached Document
PUT: Uploads a document to the specified URI
DELETE: Deletes the specified resource
OPTIONS: Returns HTTP-Server Methods
CONNECT: Converts the Connection to transparent TCP/IP Tunnel

1.2 Beschreiben Sie die folgenden Protokolle und wie sie benutzt werden: SMTP, POP3, IMAP. Geben Sie ein konkretes Beispiel für SMTP an und demonstrieren Sie dies mit Hilfe von telnet und mailsrv.uni-klu.ac.at.

- ◇ SMTP - Simple Mail Transfer Protocol
  Das SMTP, entstanden aus dem Mail Box Protocol und FTP Mail, ist eine Ansammlung an Internetprotokollen, welche das Senden und Weiterleiten von E-Mails spezifiziert. Standartports sind per Definition 25/TCP, 467/TCP+SSL (veraltet) oder 587/TCP(Senden).

  Hinter der Ausführung des SMTP-Protokolls stehen die jeweiligen Webserver, welche Mail-Dienste anbieten.

  Der Nutzer schreibt via User Agent (Outlook, Thunderbird,...) eine Email und sendet diese los.
  Diese Mail wird nun von Webserver zu Webserver weitergeleitet. Basierend auf dem TCP-Protokoll kommt primär der Handshake mit anschließendem Transfer, gefolgt vom Schließen der Verbindung.
  Dies wird so oft wiederholt, bis sich die Mail am Webserver des Empfängers befindet.
- ◇ POP3 - Post Office Protocol Version 3
  Das POP3 ist ein ASCII-Protokoll welches den Mail-Empfang und die Datenübertragung durch Kommandos steuert. Standartports sind 110/TCP und 995/TCP (verschlüsselt)
  Es ermöglicht das Auflisten, Empfangen und Löschen von E-Mails am jeweiligen Mailserver.

Bei POP3 sind keine permanenten Verbindungen zum Webserver notwendig, Verbindung wird je nach Bedarf aufgebaut und nach Anmeldung am Server werden alle Mails vom Server heruntergeladen. Jedoch findet keine Synchronisation zwischen anderen User Agents und POP3 statt. Die Authentifizierung erfolgt mittels Usernamen und Passwort, welche allerdings als Reintext übertragen werden. SASL oder APOP dienen hierbei als Schutz dieser Daten.

◇ IMAP - Internet Message Access Protocol Mithilfe von IMAP, ein Netzwerkprotokoll, kann man Mails effizient verwalten und lesen. Standartports für IMAP sind 143/TCP und 993/TCP+TLS.

Möchte ein User den Inhalt eines Ordners sehen so wird dieser mithilfe des User Agents vom Webserver gezogen. Wenn man eine Mail lesen möchte, so wird explizit diese vom Webserver angefordert. Datenspeicherung und Verwaltung geschieht am Webserver, wodurch man lokale Speicherung vermeiden kann.

Obwohl IMAP von vielen Servern unterstützt werden, variiert der Funktionsumfang. So besitzt Thunderbird oder Outlook eine erweiterte IMAP-Unterstützung an, während Opera oder Apple Mail lediglich ein vereinfachtes IMAP-Protokoll implementieren. Viele Webserver-Anbieter unterdrücken die Unterstützung von IMAP, aufgrund der serverbasierten Speicherung.

Beispiel: SMTP Example

```
.3.0$ telnet mailsrv.aau.at 25
Trying 143.205.180.43...
Connected to mailsrv.aau.at.
Escape character is '^]'.
220 mailsrv.aau.at ESMTP Postfix
EHLO yo
250-mailsrv.aau.at
250-PIPELINING
250-SIZE 41943040
250-VRFY
250-ETRN
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
MAIL FROM: myself@aau.jp
250 2.1.0 Ok
RCPT TO: thauer@edu.aau.at
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
asdfg My first Telnet
.
250 2.0.0 Ok: queued as 131621605ED
QUIT
221 2.0.0 Bye
Connection closed by foreign host.
```

Alternativ:

```
telnet -z ssl smtp.gmail.com 465
HELO
AUTH LOGIN
```

```
Enter mail-adress in Base64
Enter Password in Base64
if Accepted:
MAIL TO: <mail.com>
MAIL FROM: <mail.com>
DATA
Subect: xy
asdasd
.
QUIT
```

1.3 Beschreiben Sie das DNS-Protokoll.
Das Domain Name System-Protokoll beschreibt alles rund um das Mapping von IP-Adressen auf Domännamen. Man vergibt oft Aliase für zusätzliche Hostnamen, um mit mehreren URLs auf dieselbe IP zugreifen zu können. Via Load Distribution kann man intelligentes Hosting ermöglichen, so ist es beispielsweise einen Alias auf eine neue IP ziegen zu lassen. VGL.: Hosting von Streaming-Material innerhalb der Server einer anderen Firma.

Bei einem globalen Ausfall vom DNS gilt es die IP-Adressen einzugeben. Bei Ausfall eines einzelnen DNS-Servers gibt es keine Einschränkungen aufgrund der Vernetzung der ISPs. Prinzipiell hat jeder public Server zumindest 2 Hosts.

Das DNS-Protokoll sieht vor, dass bei der Abfrage eines Server immer der nächstgelegenste ISP kontaktiert wird. Je nach Iterativer oder Rekursiver Implementierung geschieht folgendes:

- ◇ Iterativ: Das Usergerät spricht den nächstgelegenen ISP an, welcher mögliche Kontaktdaten des spezifizierten Ziels zurücksendet. Dieser fragt anschließend den zurückgegebenen Kontakt ab, bis der richtige Server gefunden wurde und eine Übertragung erfolgen kann.
- ◇ Rekursiv: Das Usergerät spricht den ISP an, welcher anschließend einen übergeordneten Server abfragt. Sobald der Zielserver gefunden wurde, reisen die Kontaktdaten über denselben Weg wieder zurück.

2 Wireshark

2.1 Setting up Wireshark

Für Ubuntu-Users funktioniert dies recht schnell. Im Terminal (Strg+Alt+T):

```
sudo apt-get install wireshark
...
sudo dpkg-reconfigure wireshark-common
sudo adduser 'whoami' wireshark
```

Sollte weiterhin kein mitschneiden möglich sein, da der Zugriff auf /usr/bin/dumpcap nicht gestattet wird, so folgt:

```
sudo chmod +x /usr/bin/dumpcap
```

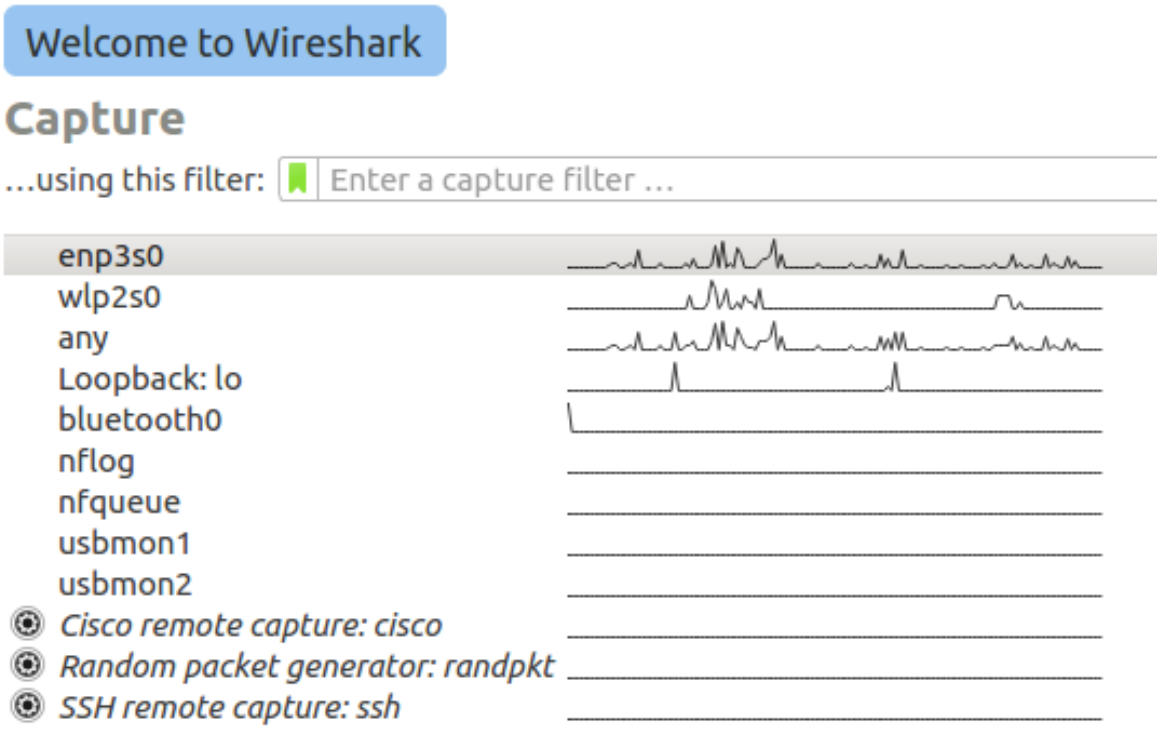
Wireshark wird anschließend über die Apllikationen oder über das Terminal ausgeführt, mithilfe des Befehls

```
wireshark
```

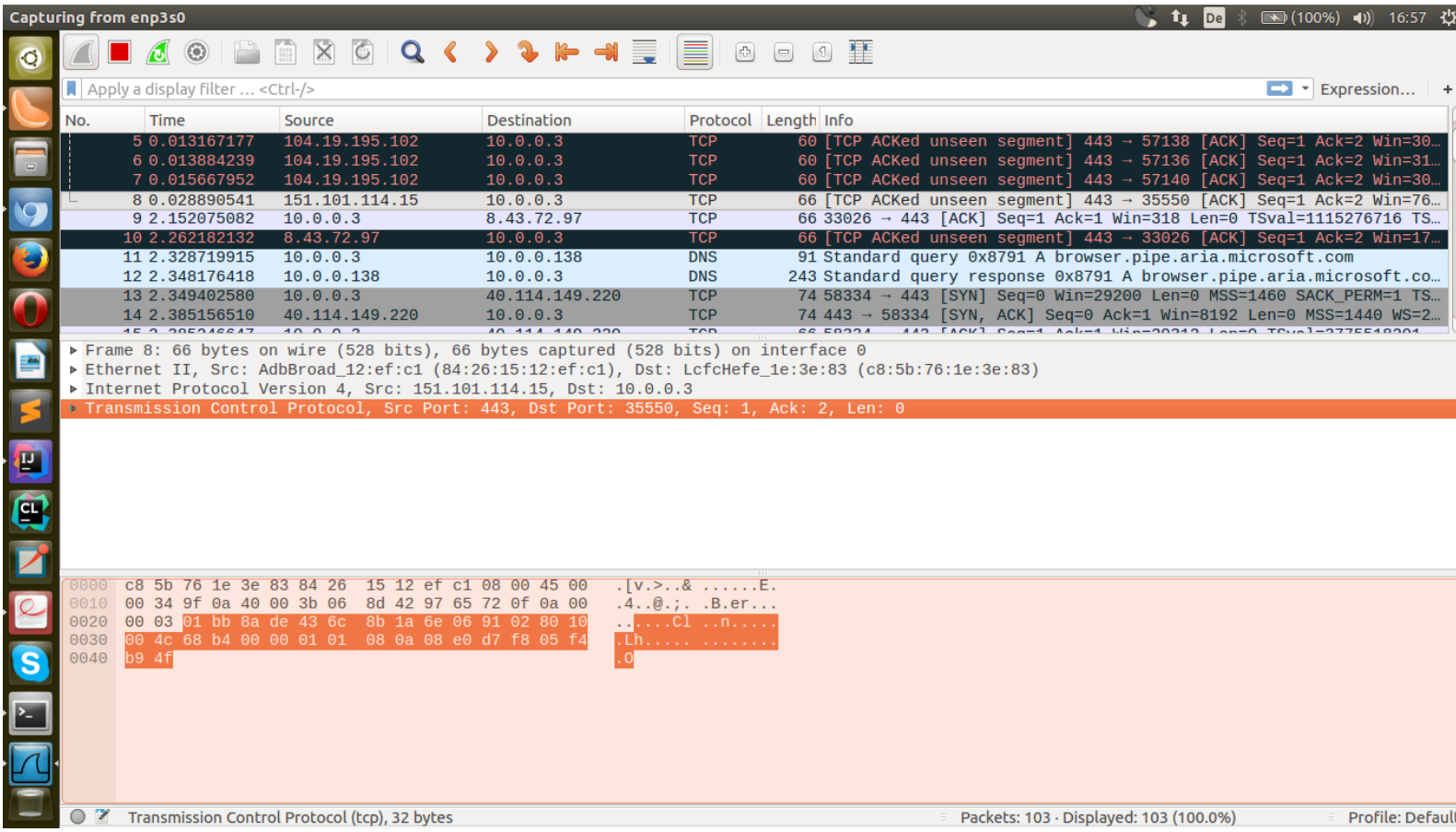
2.2 Funktionalitäten von Wireshark

2.2.1 Grundfunktionalitäten

Wireshark beobachtet stets die eingehenden Packete des jeweiligen Interfaces. Auf dessen Hauptseite kann man bereits die Internetschnittstellen des jeweiligen Gerätes erfassen und beobachten.



Möchte man ein gewisses Interface 'sniffen', so selektiert man dieses und startet die Beobachtung.



Hier beobachtet man eine explizite Schnittstelle zum Internet, mit all dessen Datatransfer. Ist der 'promiscuous mode' aktiviert, so erfasst man alle Packete die sich im derzeitigen Netzwerk bewegen, nicht nur des eigenen Gerätes.

```
Capture > Options > verify "Enable promiscuous mode on all interfaces" checkbox
```

Im oberen Feld sieht man die eingehenden Packete, mitsamt
- Packetnummer (Frame), seit Start
- Dauer der Übertragung
- Ursprungs-IP des Packetes
- Ziel des Packetes
- Das genutzte Protokol der Übertragung (DHCP, ARPA, TCP,...)
- Die Länge das Packetes in Hexadezimalziffern
- Informationen zum Packet, etwa der Request ein

Die Färbungen deuten auf die Packetformate hin. Beispielsweise ist per default TCP pink oder UDP blau markiert. Schwarz gekennzeichnete Packete deuten auf Packete mit Fehlern hin. Für mehr, siehe

```
View > Coloring Rules
```

Im mittleren Feld stehen konkrete Daten zum betrachteten Packet. Für gewöhnlich gliedert sich dieser Teil in

- Frame-Nummer und abgefangenen Bits
... mitsamt Daten über dieses Frame: Protocols in Frame, Marked, Ignored, ...

- Verbindungstyp der Übertragung
... inklusive Sender-IP und Empfänger-IP, oftmals in IPv6-Format sowie Übertragungsprotokoll

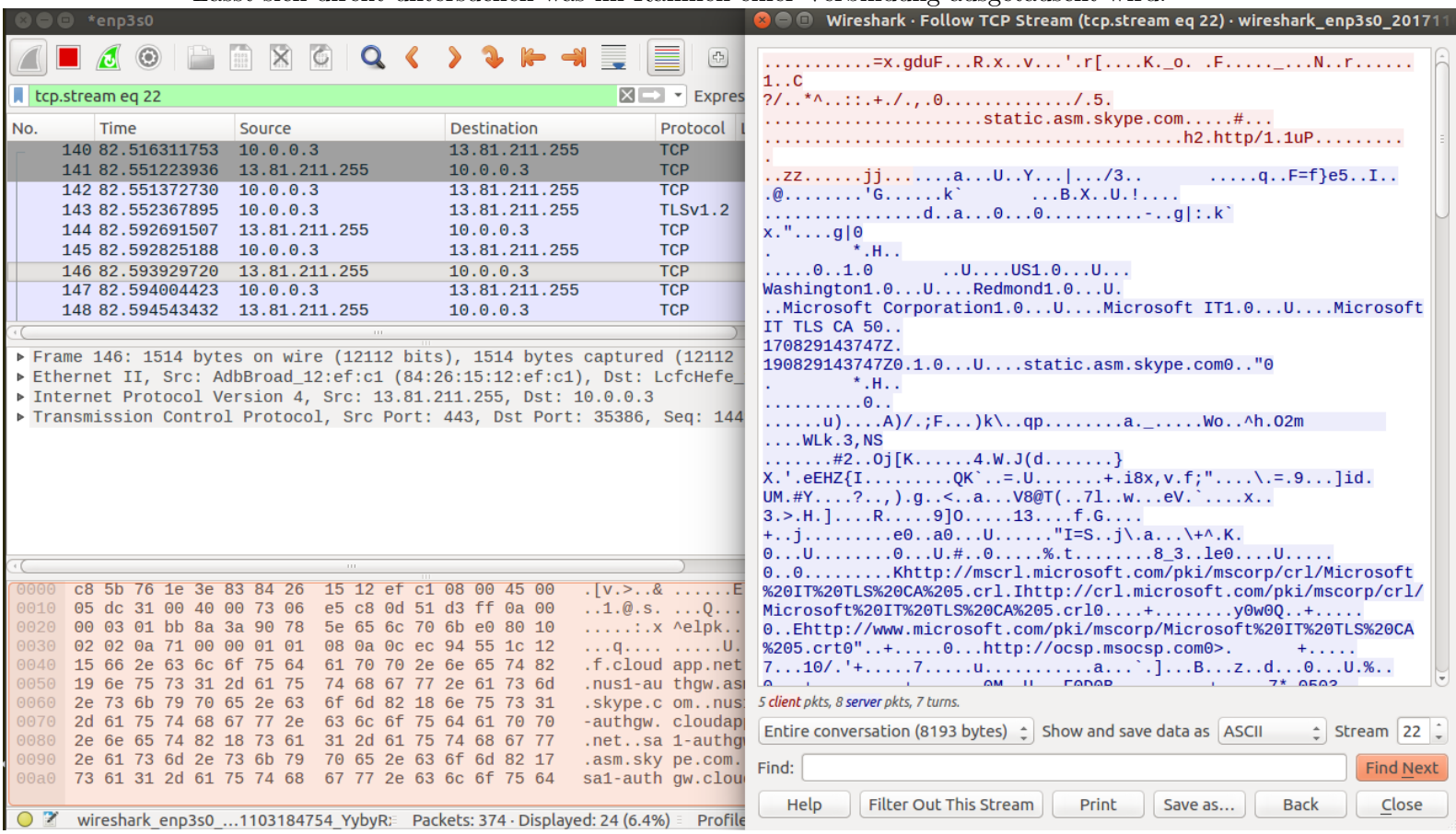
- und unterschiedlichen Informationen, je nach Protokollierungstyp.
... für TCP beispielsweise die Internet Protokol Version und das Transmission Control Protokol, inklusive deren Daten.

Im unteren Feld steht dann der Inhalt der jeweiligen Frames in non-human readable Format.

Eine weitere Interessante Funktion die WireShark bietet ist das direkte Verfolgen einer *-verbindung. Via

```
> Rightclick on Frame > Follow > *-Stream
```

Lässt sich direkt untersuchen was im Rahmen einer Verbindung ausgetauscht wird.



2.2.2 Filtering

Um etwas spezifisches zu untersuchen, bietet WireShark Filteroptionen an. So kann man beispielsweise mit 'dns' explizit die DNS-Packete aus dem ganzen Satz an Packeten herausfiltern. Für weitere Optionen, siehe

```
Analyze > Display Filters
```

Man kann auch eigene Filter erstellen und einbauen.[1]
Übliche Befehlskonfigurationen können wie folgt lauten:

- ◇ tcp.port==80
- – Suche nach allen TCP verbindungen die an Port 80 docken
- ◇ udp contains 64
- – Suche durch alle Packete welche eine x64 Zahl im Frame aufweisen
- ◇ http.connection matches "Keep-Alive"
- – Suche nach allen TCP-Verbindungen mit Keep-Alive Kondition
- ◇ tcp.flags & 0x02
- – Suche durch alle TCPs mit gesetzter SYNchronize-Flag
- ◇ tcp.port in 80 443 8080
- – Suche alle TCPs welche an Port 80 oder 443 oder 8080 docken

2.2.3 Aufzeichnen von Datenverkehr

Das Aufzeichnen des Datenverkehrs erfolgt gewöhnlicherweise direkt mit dem Starten des Sniffings .

Der 'promiscuous mode', wie oben bereits erwähnt, betrachtet auch Packete die an das jeweilige Netzwerk gesendet werden.

Die Aufzeichnungen im 'promiscuous mode' sind in der Regel ungefährlich für alle Beteiligten, da Packete nur empfangen werden wenn sie tatsächlich der Empfänger-IP entsprechen. Jedoch kann man dies mit einem SPAN port, einem Switch Port Analyzer, umgehen, wodurch jegliche Packete durch das Gerät empfangen werden.[2] Aufgrund der Human-Readable-Umschreibung von betrachteten Streams, kann man somit sensible Daten wie Passwörter und IDs abfangen und auslesen. Eine Gegenmaßnahmen dazu sind Enkryptionen durch SSL oder TLS oder eingeschränkte Zutrittsrechte zu Serverräumen.[3] Um einen SPAN aufzusetzen benötigt man physikalischen Zugang zum jeweiligen Router.

3 Analysis mit Wireshark

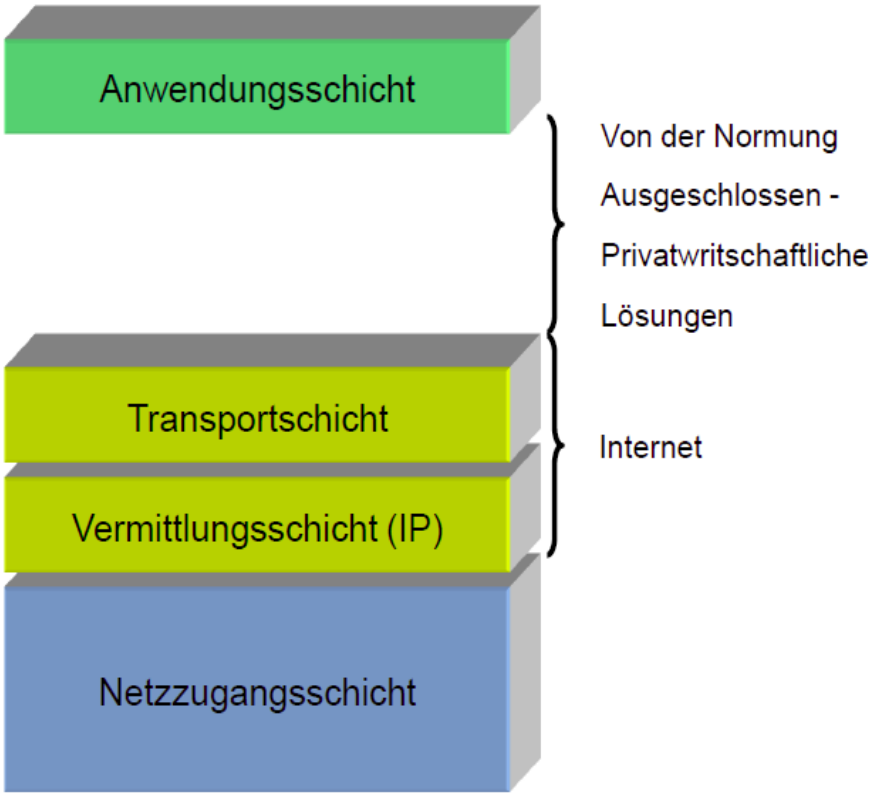
Im Rahmen folgender Fragestellungen analysieren wir mithilfe von Wireshark das beiliegende dump_protocols.pcap.

3.1 Welche Netzwerkprotokolle werden in der Kommunikation verwendet?
Es lassen sich mehrere Protokolle identifizieren, mitunter:

[1] $https://www.wireshark.org/docs/wsug\_html\_chunked/ChWorkBuildDisplayFilterSection.html$
[2] https://blog.packet-foo.com/2016/07/how-to-use-wireshark-to-steal-passwords/
[3] https://blog.packet-foo.com/2016/11/the-network-capture-playbook-part-4-span-port-in-depth/

◇ DHCP - Dynamic Host Configuration Protocol: Zuweisung von Client & Server.
◇ ARP - Address Resolution Protocol: Zuweisung der MAC-Adressen im Lokalen Netzwerk.
◇ ICMP - Internet Control Message Protocol: Handling von Fehler und Status in IP/TCP/UDP
◇ DNS - Domain Name System: Zuweisung von IPs und Hostnamen.
◇ TCP - Transmission Control Protocol: Datenübertragungsprotokoll, orientiert an sicherer Packetzustellung.
◇ HTTP - HyperText Transfer Protocol: User Agent-Rendering gemäß .htmls

3.2 Ordnen Sie jedes der Protokolle einer Schicht im TCP/IP-Referenzmodell zu und stellen Sie die Hierarchie der einzelnen Protokolle graphisch dar.
Zunächst eine Wiederholung des TCP-IP Schichtenmodelles:



TCP/IP - Modell

Die genutzten Protokolle stehen somit in Folgender Relation zum TCP-IP Modell:

- DHCP - Application Layer, als auch für das OSI-Model
  Grund: DHCP vermittelt eine Client-Server verbindung basierend auf dem ausgeführtem Programm. Gibt es kein Programm, so gibt es weder spezifizierten Klienten noch Server.

- ARP - Network Access Layer, or Data Link Layer for the OSI-Model
  Grund: ARP befasst sich mit der Korrektheit der MAC-Adressen. Diese befinden sich im Datalink Layer (OSI) und somit im Network Access Layer des TCP-IP.

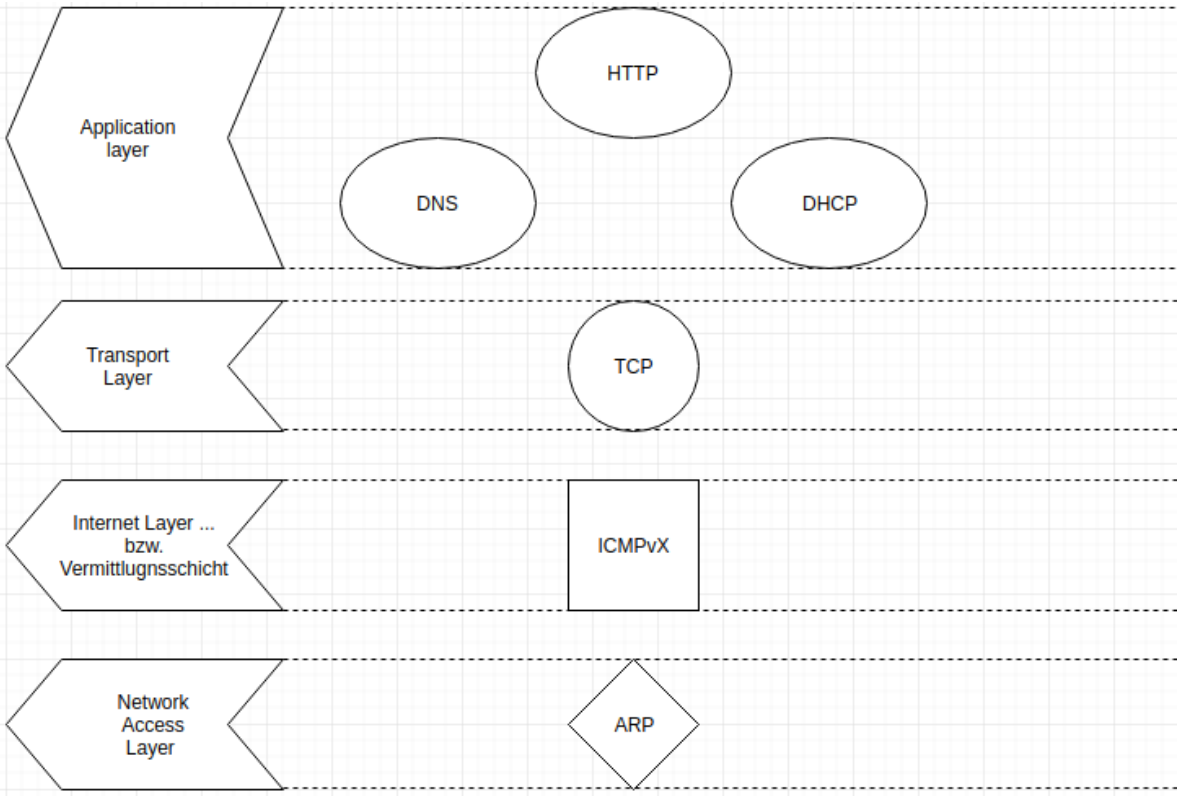
- ICMP - Vermittlungsschicht, or Network Layer für OSI
  Grund: ICMPvX beschäftigt sich mit Fehlercodes des IPvX, weshalb es keinem anderen Layer zuschreibbar ist als dem Network-Layer des OSI-Models. Dementsprechend befindet es sich in der Vermittlungsschicht.

- DNS - Application Layer - parallel mit HTTP, eben so für das OSI-Model
  Grund: DNS beschäftigt sich mit dem Zuschreiben von Aliasen der IP-Adressen, den Hostnamen. Dieser findet jedoch nur im Browser(=Applikationsschicht) nutzen, da aus maschineller Sicht die Hostnamen IP-Adressen wiederspiegeln.

- TCP - Transport Layer für das TCP-IP als auch OSI-Model
  Grund: TCP beschäftigt sich mit dem Datenaustausch über Handshake. Als bekanntes Protokoll dient es der sicheren Übertragung von Frames und ist dementsprechend auf dem Transportlayer angesiedelt.

- HTTP - Application Layer, ebenso für OSI
  Grund: HTTP dient der Übertragung der spezifizierten .htmls eines Servers, gewöhnlicherweise via TCP/IP. Da HTML (& css sowie js oder php, etc.) der Darstellung einer Seite dienen, gilt HTTP der Applikationsschicht.

Graphisch wären die Protokolle etwa so einzugliedern:



Alternativ:

```
Statistics > Protocol Hierarchy
```

4 Analysis mit Wireshark

Im Rahmen dieser Sektion wollen wir eine weitere Datei analysieren, welche mit Wireshark aufgenommen wurde, dump_http.pcap.

4.1 Welche Objekte wurden vom Klienten via HTTP angefordert? Mithilfe von

```
File > Export Objects > HTTP...
```

kann man direkt einsehen, welche Elemente im Rahmen aller HTTP-Protokollierten Frames, innerhalb des derzeitig geladenen .pcap übertragen wurden. In unserem dump_http.pcap wurden die Elemente text(145 Byte, Frame 6), logo.gif(4445 Byte, Frame 22) und TechnikErleben.png(27 kB, Frame 66) angefordert, von 192.168.1.10.

4.2 Recherchieren Sie die Bedeutung der einzelnen Header-Felder bei den Anfragen bzw. Antworten des Servers. TCP-Header bestimmen eine ganze Reihe an diversen Datensätzen. Über 10 Felder von je 20 bytes & zusätzlichen 40 byte an optionalen, informativen Daten, bestimmt man

◇ Sender TCP Port Nummer (2 Byte)

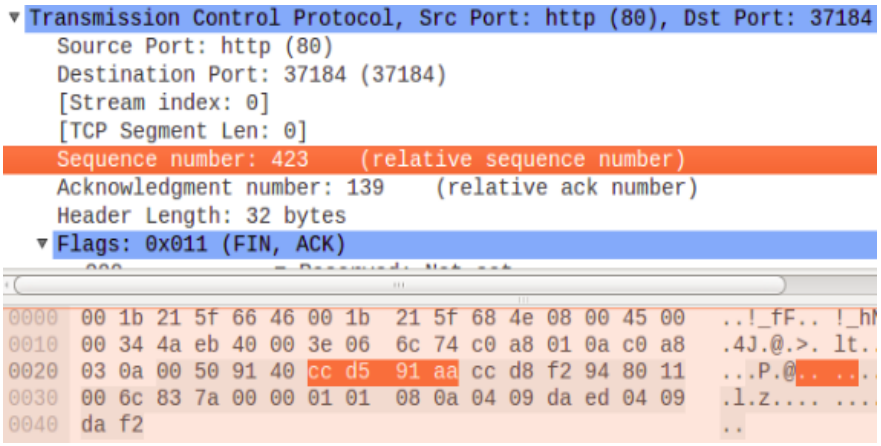
Der spezifizierte Port, von welchem der Sender sendet.

◇ Empfänger TCP Port Nummer (2 Byte)

Der spezifizierte Port, wo der Empfänger die Nachricht erhält.

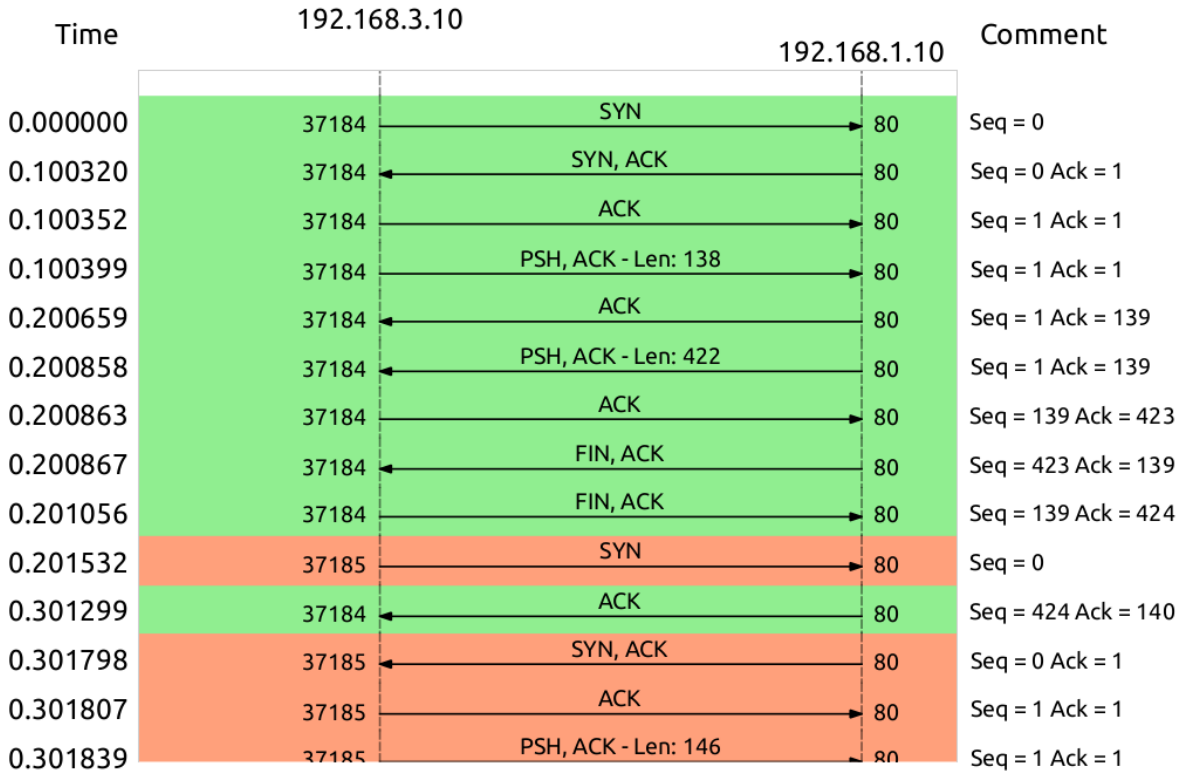
◇ Sequenznummer (4 Byte)

... sie dient der Information über die Menge bereits gesendeter Data. Sie ist in jedem Datenpacket vorhanden und dient mitsamt der Acknowledge Nummer (next §) dazu, dem Sender den erfolgreichen Transfer zu übermitteln. Zwecks Sicherheitsvorkehrungen startet diese immer mit einem zufälligen Wert zwischen 0 und $(2^32) - 1$. Wireshark bietet hinsichtlich der X-hunderten Zahlenwerten eine relative Sequenznummer an, um die Packetidentifikation in mehreren Hinsichten zu vereinfachen.



◇ Acknowledge number (4 Byte)

Diese dient dazu den Datentransfer zu bestätigen. Prinzipiell gilt, dass die geantwortete ACK zur neuen SEQ wird, siehe auch



... Via Wireshark, lässt sich ein solcher Graph mittels

```
Statistics > Flow Graph > Flow Type: TCP flow
```

erstellen.

◇ TCP Data Offset (X words ... 8 Bytes each)

.. dieser dient zur Spezifikation des eigentlichen Dateistarts. Anhand von 'Header Length: xx Byte' lässt sich dies feststellen.

◇ Reserved Data (4 Bit)

Das Reserved Data-Feld ist für zukünftige Verwendungen reserviert. Alle Bits müssen null sein.

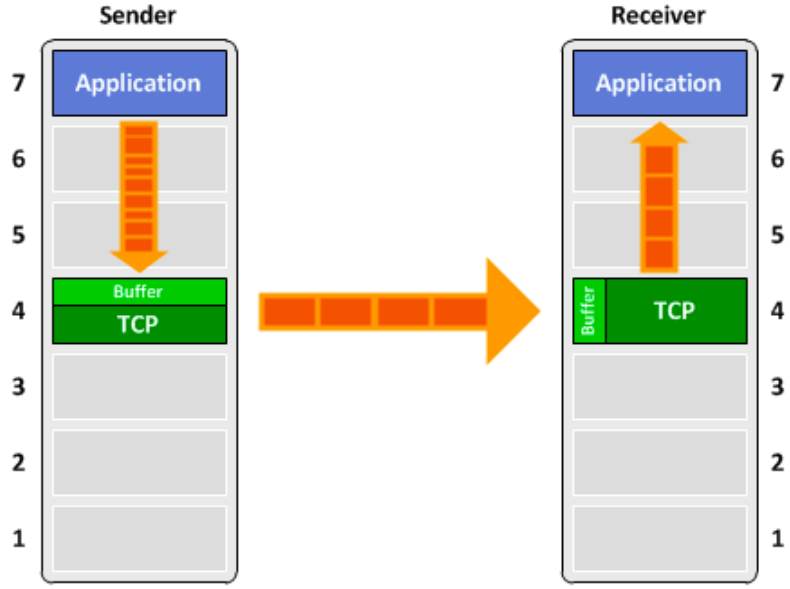
◇ Control Flags (8 Bit)

Beschrieben durch 2 Variablen oder 4 Hexa-Werten. Zu den möglichen Flags gehören:

- ECE - Explicit Congestion Notification (ECN-ECHO): Bei Netzwerküberlastung
- CWR - Congestion Window Reduced: Revertiert ECE
- URG - Urgent: Selten genutzt, oftmals als Interrupt-Signal
- ACK - Acknowledgement: Setzt Gültigkeit für Sequenz/Acknowledge-Values

- PSH - Push: Umgehung lokaler Puffer für schnellere Übertragung
- RST - Reset: Zum Abbruch von Verbindungen
- SYN - Initiieren der Verbindung. Üblicherweise beantwortet mit SYN+ACK oder RST
- FIN - Finish: Schlussflag, dient zur Freigabe der Verbindung & Bestätigt vollständige Übertragung.

Bezüglich PSH: Datenverkehr & Pufferfunktion bei Übertragung



◇ Window size (2 Bytes)

Bestimmt die Puffergröße bezüglich der zu sendenden Daten. Dies steht in Relation mit ACK, da der Datensatz erst mit der ACK-Nummer bestätigt wird. e.g.: Bei zu klein gewähltem Puffer muss der Sender eines Elementes auf die Antwort des Empfängers warten, bevor dieser weitere Packete des jeweiligen Elementes senden darf.

◇ TCP checksum[1] (2 Byte)

Die Prüfsumme des TCP-Headers dient der Erkennung von Fehlübertragungen, bestehend aus Empfänger-IP, Sender-IP, TCP-Protokollerkennung, Länge des TCP-Headers und Nutzdaten.

◇ Urgent Pointer (2 Byte)

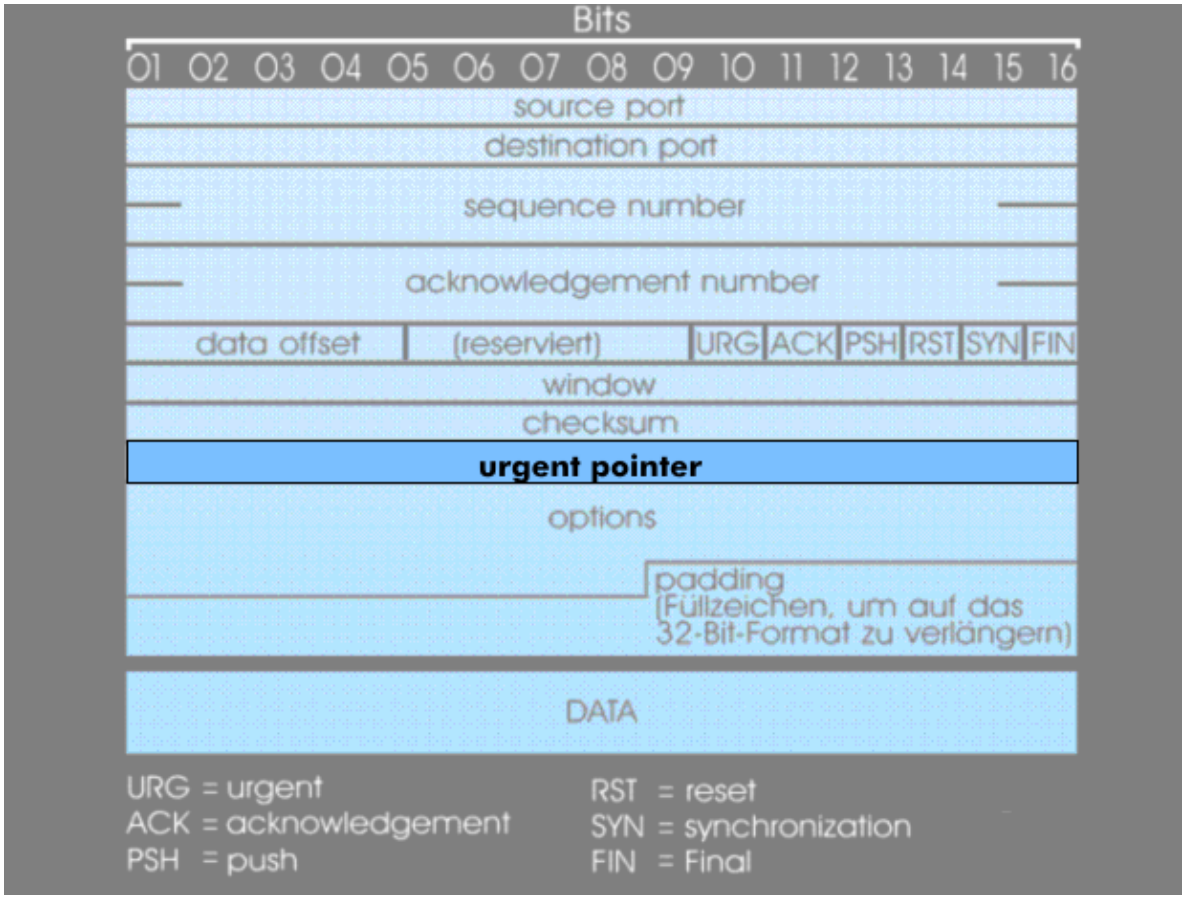
Der Urgent-Pointer zeigt auf das Ende der dringenden Daten. Der Wert des Pointers ist ein positiver Offset der Sequenznummer. Gilt nur wenn die URG-Flag gesetzt ist.

◇ Options (0 .. 40 Byte)

Das Options-Feld ist immer unterschiedlich groß, je nach Zusatzinformationen. Eine Option muss stets die RFC-Bedingung von 8 Bit erfüllen, dementsprechend muss

[1] http://www.roman10.net/2011/11/27/how-to-calculate-iptcpudp-checksumpart-1-theory/

man potenzielle Optionswerte mit Nullwerten padden. Man kann beispielsweise die Maximum Segment Size, MSS, beeinflussen. Per Default ist dieser: IP Datagram mit 576 Byte - 40 Byte TCP Header = 536 Byte.



Dementsprechen lassen sich dann die Antworten und Nachfragen des Servers interpretieren.

4.3 Wie viele TCP-Verbindungen werden insgesamt aufgebaut? Wie unterscheidet sich das von dump_protocols.pcap?
Mithilfe von

```
Analyze > Follow > TCP Stream
```

können wir durch die verschiedenen TCP-Verbindungen tooglen. Im Rahmen dessen können wir auch einsehen, was genau übertragen wurde.
Für unser .pcap stehen 3 unterschiedliche TCP-Verbindungen.

◦ Stream 0: Beschreibt GET /test/ ◦ Stream 1: Beschreibt GET /test/logo.gif ◦ Stream 2: Beschreibt GET /test/TechnikErleben.png

Es handelt sich dabei um eine altbewährte Form der HTTP/1.0 Datenübertragung - 1 eigene TCP-Verbindung per Grafik.

4.4 Bestimmen Sie, wie viele Bytes in jeder Verbindung ausgetauscht werden und wie lange die einzelnen Verbindungen bestehen. Via

```
Statistics > Conversations ... TCP
```

können wir alles rund um die TCP-Verbindungen feststellen.

5 Analyse mit Wireshark

Nun werden wir heutigen Datenverkehr aufnehmen und analysieren. Via Youtube catchen wir einige Datenpackete welche wir anschließend begutachten, gemäß folgender Fragen:

1. Versuchen Sie die Verbindung über unterschiedliche Zugangsnetzwerke (z.B. LAN, WLAN, 3/4G sofern möglich) herzustellen und dokumentieren Sie allfällige Unterschiede.
2. Welche Objekte werden vom Client via HTTP angefordert? Hinweis: nur jene beim Videostreaming, andere Objekte (z.B. HTML, Text, Bilder) können vernachlässigt werden. Wir setzen Wireshark auf unser Netzwerk an und starten den Stream. Wir beenden die Aufnahme alsbald und analysieren unsere .pcapng's mithilfe von

```
- Analyze > Follow > TCP Stream / UDP Stream
- Statistics > Conversations
- Filter wie 'tcp.ack == 1 && tcp.seq == 1'
- u.s.w.
```

5.1 Stream eines Youtube-Videos via Ethernetverbindung
Wenn wir Info betrachten, so können wir direkt den TCP-Handshake des Klienten und des Servers verifizieren, ebenso dass der Server uns nicht via RST abgewiesen hat. Ebenso können wir mittels Filter die genutzten Protokolle identifizieren: ARP, SNMP, TCP und TSLv1.2, ICMP, DNS

TSLv1.2 beschreibt eine SSL-Enkryption vom TCP-Austausch über Secure Sockets (Spezielle Ports) SNMP als das Zuweisungsprotokoll innerhalb unseres Heimnetzwerkes

Weiters können wir feststellen, dass es beim Verbindungsaufbau zur URL bis hin zur Beendigung des Videos insgesamt

- 6 unterschiedliche IPs miteinander kommunizierten und im Rahmen dieser
  - 19 unterschiedliche IPv4 Konversationen erfolgten
  - 16 Konversationen über TCP verliefen
  - 4 UDP Transfers stattfanden

Ein Objektzugriff über HTTP erfolgte hier erstaunlicherweise nicht, was jedoch mit der Implementierung seitens Youtube zusammenhängt.

Wir starten einen neuen Sniff-Versuch, nun jedoch über das WLan unseres Heimnetzwerkes.

5.2 Stream eines Youtube-Videos via WLan-Verbindung
Der Stream über WLan erfolgte über weniger Packeten als wie beim Stream durch direkte Ethernetverbindung. Die Protokolltypen blieben die gleichen, während es einen Umschwung bei den Konversationen gab. Diesmal kommunizierten lediglich

- 2 unterschiedliche IPs miteinander, samt
  - 12 unterschiedlichen IPv4 Konversationen
  - 26 TCP Konversationen
  - und 1 UDP Transfer

Bei beiden Versuchen handelte es sich um denselben Anfangszustand (Videoauswahl Youtube), dasselbe Video und dieselbe Videoqualität.

Wir versuchen uns diesmal an einem anderen Anbieter:

5.3 Stream von anilinkz.io[1] über Ethernet Wir umgehen hierbei triviale Daten und starten direkt mit dem Stream selbst, welcher jedoch wiederum Werbung hervorruft (dementsprechend die abgebrochenen TCP-Packete am Start).
Über die Zeitdauer von 5 Minuten ergaben sich ein gutes Dutzend an Konversationen. Verteilt über

- 12 unterschiedliche IPs ergaben sich
  - 138 IPv4 und 5 IPv6 Konversationen
  - 187 TCP Datentransfers und
  - 9 UDP Übertragungen

Der Hauptverbindungsträger ist www11.mp4upload.com, eine klassifizierte Subdomain eines Hosts, ein Hinweis auf implementierte Server Load Balancing.
Es erfolgten hierbei Konversationen über Protokolle wie TLS1.2, TCP, UDP, DNS, MDNS, ARP, IGMPv2 ...

[1] http://anilinkz.to/no-game-no-life-episode-1?src=8

- MDNS oder Multicast DNS dient der IP-Adressierungen in kleineren Netzwerken.