



THOMAS DRILLING

# NETZWERKANALYSE MIT WIRESHARK

Windows   
Windows, Virtualisierung  
und Cloud für Profis **Pro**

## Inhalt

Was ist Wireshark? .....	3
Strafrechtliche Implikationen .....	3
Ethical Hacking .....	3
Promiscuous Mode erforderlich .....	3
Verfügbarkeit .....	4
Vorbereitung und Installation unter Windows .....	6
Capture-Treiber installieren .....	6
Spracheinstellungen ändern .....	10
Wireshark unter Linux und macOS installieren .....	11
Wireshark über Ubuntu-Software einrichten .....	11
Installation unter Debian .....	12
Wireshark starten .....	14
Setup unter macOS .....	15
TCP-IP-Grundlagen für die Netzwerk-Analyse .....	18
Netzwerk-Analyse am Beispiel von Ping .....	21
Aufzeichnung starten .....	21
Aufteilung des Fensters .....	23
MAC- und IP-Adresse auslesen .....	23
Berechnung der Frame-Länge .....	24
TCP-Handshake beim Aufbau einer Sitzung untersuchen .....	29
Mitschnitt beginnen .....	29
Aufzeichnung auswerten .....	30
Die drei Schritte des Handshakes .....	34
TCP-Fehlerkorrektur beobachten .....	36
FTP unter der Lupe .....	40
Berechnung von SEQ und ACK .....	40
FTP-Mitschnitt untersuchen .....	40
TCP-Flusssteuerung als Schema .....	43
Mitschnittfilter versus Anzeigefilter .....	45
Weniger Daten, geringere Flexibilität .....	45
Berkeley Filter .....	45
Mitschnittfilter für DNS .....	45
Anzeigefilter .....	46
Nach IP-Adresse filtern .....	47
Filterhierarchien .....	47
Protokollfilter .....	49
Anzeigefilter für FTP .....	50
Filtervorschläge .....	52
Datenströme verfolgen mit Verbindungsfiltern .....	53

## Was ist Wireshark?

Wireshark ist ein professioneller Netzwerk-Sniffer, den Administratoren, Sicherheitsberater und Hacker gleichermaßen schätzen. Damit kann man auf der Ebene von Frames, Paketen oder Segmenten veranschaulichen, was sich auf den Schichten des OSI-/ISO- bzw. DOD-Modells und auf der physikalischen Leitung bewegt.

Zwar gibt unzählige freie und quelloffene Netzwerk-Sniffer, das Open-Source-Werkzeug *Wireshark* ist aber neben *nmap* unbestritten das wichtigste Tool für Netzwerkadministratoren, Hacker oder Penetration-Tester.

Man kann mit Wireshark

- Netzwerkwerke analysieren
- Kommunikationsprobleme identifizieren
- Schwachstellen aufdecken

### Strafrechtliche Implikationen

Bevor wir uns um die Installation kümmern, kommen wir aufgrund der Mächtigkeit und Leistungsfähigkeit des Tools schon aus rechtlichen Gründen nicht umhin, auf den Hacker-Paragraphen "Vorbereiten des Ausspähens und Abfangens von Daten" (§202c des deutschen StGB) aus dem Jahr 2007 hinzuweisen.

Er stellt die Beschaffung und Verbreitung von Zugangscodes zu geschützten Daten sowie auch die Herstellung und den Gebrauch von Werkzeugen, die diesem Zweck dienlich sind, als Vorbereitung einer Straftat unter Strafe (maximal zwei Jahre). Eine juristische Stellungnahme der European Expert Group for IT Security (EICAR) geht davon aus, dass konstruktive Tätigkeiten (im Dienste der IT-Sicherheit) bei ausführlicher Dokumentation nach diesem Paragraphen nicht strafbar sind.

### Ethical Hacking

Wer Wireshark im Sinne des White- bzw. Ethical-Hackings beispielsweise zur Analyse und Absicherung der eigenen IT-Infrastruktur oder zu Lehr- und Demonstrationszwecken einsetzt, bewegt sich zumindest in einer Common-Sense-Grauzone.

Möchten Sie zu hundert Prozent auf der sicheren Seite sein, dann dürfen Sie Wireshark ausschließlich zur Analyse eigener Rechner und Netze einsetzen, und zwar so, dass andere Netzwerkressourcen davon nicht tangiert sind.

Werden Sie etwa als IT-Consultant oder als Arbeitnehmer offiziell mit Netzwerkanalysen und/oder Pentests beauftragt, sichern Sie sich vorher rechtlich ab (möglichst schriftlich) und dokumentieren Sie akribisch, was Sie tun.

### Promiscuous Mode erforderlich

Wireshark überwacht Netzwerkschnittstellen und schneidet den Datenverkehr auf den angegebenen Interfaces mit. Da stellt sich bereits die Frage, wie das in einem "ge-switch-ten" Netzwerk funktioniert?

Bekanntlich ist Ethernet von seiner Logik her ein Bus-System, auch wenn wir es bei IEEE 802.3u, 802.3ab oder 802.3an physikalisch mit Stern-Verkabelungen zu tun haben. Das alte Bus-System ist quasi nur ins Innere des Hubs oder Switches gewandert.

Vergegenwärtigt man sich, dass ein Hub rein funktional nichts anderes ist als ein Multiport-Repeater und ein Switch eine Multiport-Bridge, dann kann man auf einer Arbeitsstation den gesamten Backbone-Datenverkehr über das verwendete Interface nur beobachten, wenn sie an einen Hub angeschlossen ist, da nur ein solcher sämtliche Frames immer an jeden Port weiterleitet.

Gottlob gib es heute keine Hubs mehr. In ge-switch-ten Netzwerken gelangt mit Ausnahme von Broadcasts immer nur derjenige Traffic an eine bestimmte Schnittstelle, der auch für diese bestimmt ist.

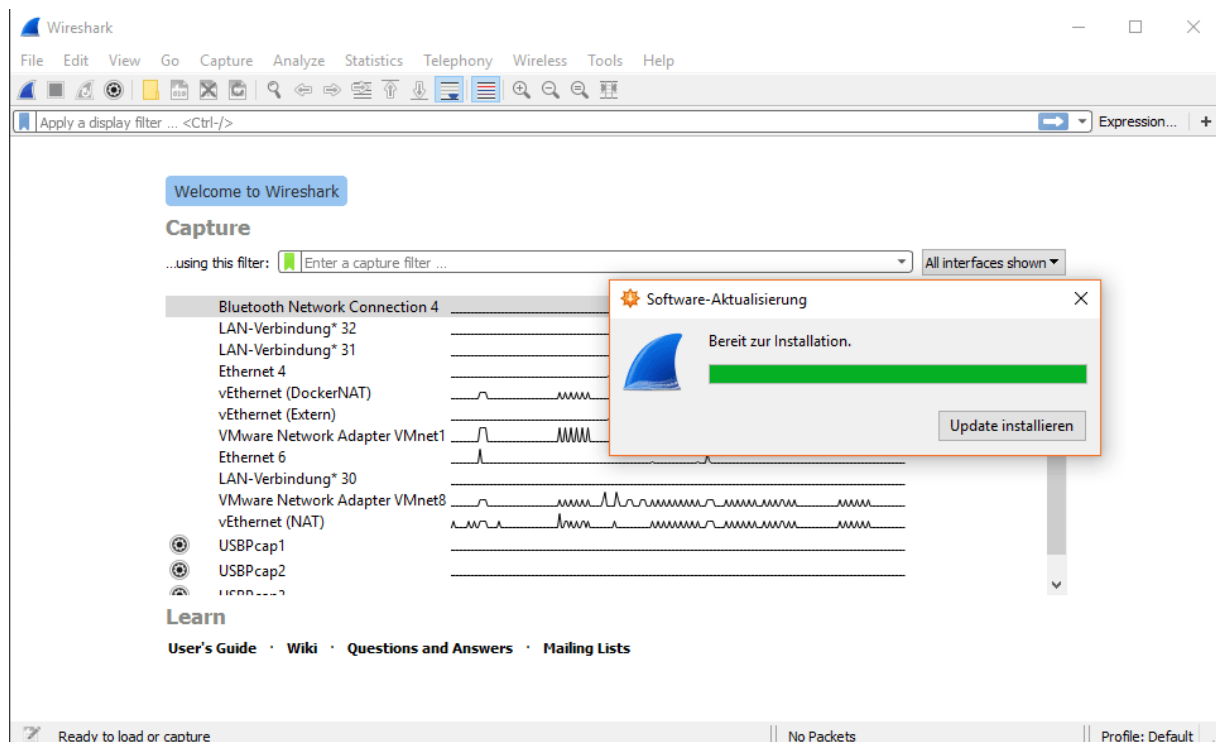
Ein Switch schaltet aufgrund seiner Eigenintelligenz (sie basiert auf der vom ihm geführten und stetig aktualisierten MAC Address Table) intern immer nur die gerade erforderlichen Routen, was Kollisionsdomänen im Unterschied zum Hub auf den jeweiligen Port beschränkt.

Und was hat das nun mit Wireshark zu tun? Um in einen ge-switch-ten Netzwerk auf einem ausgewählten Netzwerk-Interface auch Ethernet-Frames bzw. IP-Pakete oder TCP-Segmente zu beobachten, die nicht für dieses selbst, sondern eine andere Station im Layer-2-Segment bestimmt sind, benötigen wir den Promiscuous Mode.

## Verfügbarkeit

Wireshark ist quelloffene Software und kann daher auch im Source Code sowie als Installationspaket für Windows (64- und 32-bit), MacOS 10.6 oder als 32-Bit portable App [heruntergeladen werden](#). Hier stehen optional auch das aktuelle Development-Release, die stabilen Vorgängerversionen sowie die Dokumentation bereit.

Wireshark verfügt über ein Update-Tool, das die Software anschließend auf dem neuesten Stand hält. Eine neue Version erhält man dann, sofern verfügbar, mit einem Klick auf *Help => Check for Updates*.



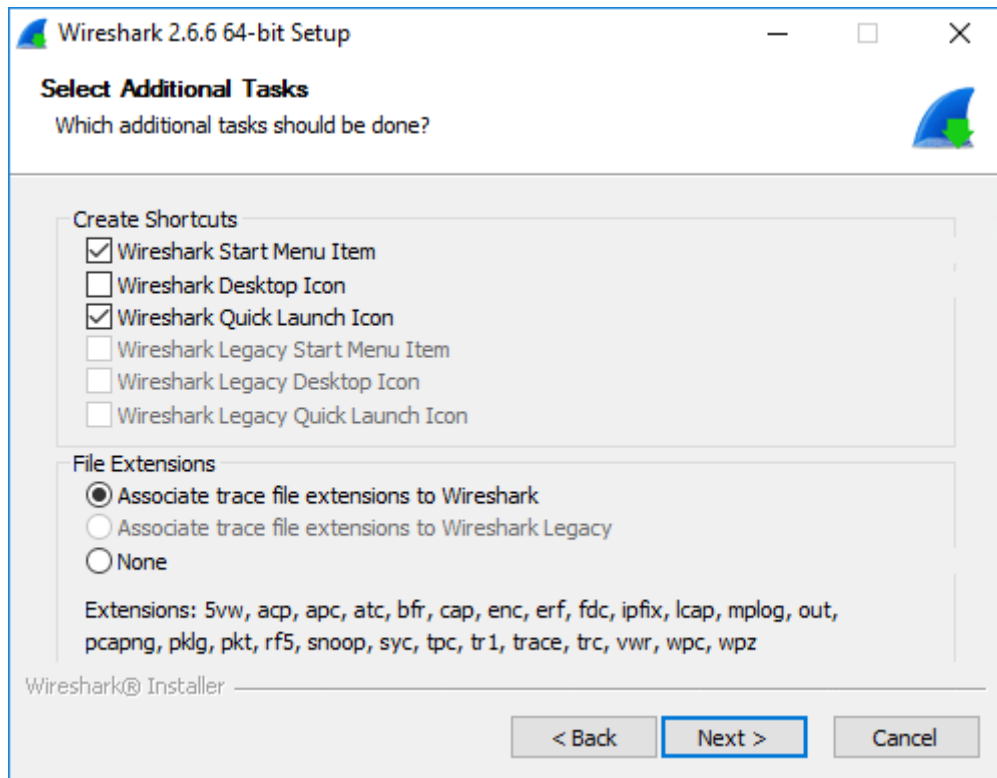
*Wireshark aktualisiert sich über einen Auto-Updater*

Bei den gängigen Linux-Distributionen ist Wireshark in der Regel in den Paketquellen der gängigen Distributionen enthalten und kann über den jeweiligen Paketmanager installiert werden.

Da man Wireshark ohnehin meist in isolierten Umgebungen bzw. am besten aus einer virtuellen Maschine heraus verwendet, empfiehlt sie für den problemlosesten Einsatz von Wireshark eine auf Pen-tests, Forensik und Sicherheitsanalysen spezialisierten Linux-Live-Distribution wie zum Beispiel [Kali-Linux](#). Aus eine Kali-Live-Umgebung heraus lässt sich Wireshark out of the Box nutzen.

## Vorbereitung und Installation unter Windows

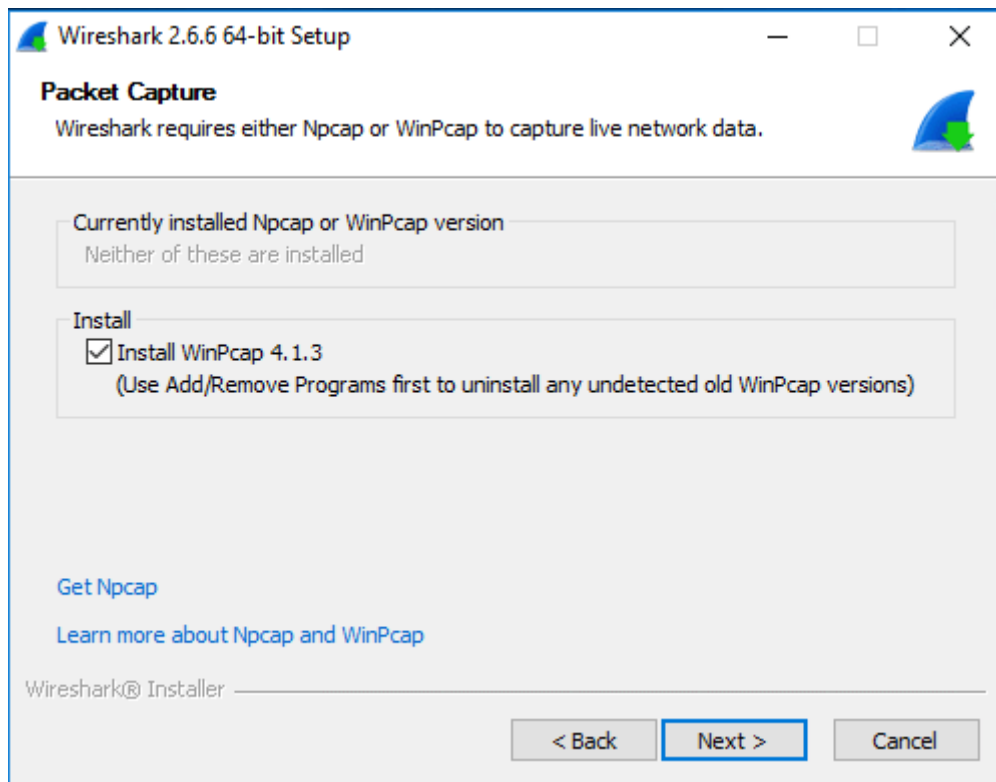
Unter Windows bedarf es dazu einer speziellen Bibliothek, um das betreffende Interface durch einen passenden Capture-Treiber zu modifizieren. Auch wenn die Wireshark-Installation unter Windows und Mac gewohnt simpel verläuft, sollte man deshalb den Setup-Assistenten nicht einfach durchwinken. Bis zu *Select Additional Tasks* können wie allerdings die Vorgaben übernehmen.



Das Setup von Wireshark kann man bis zu diesem Dialog durchlaufen, dann kommt der Capture-Treiber an die Reihe.

### Capture-Treiber installieren

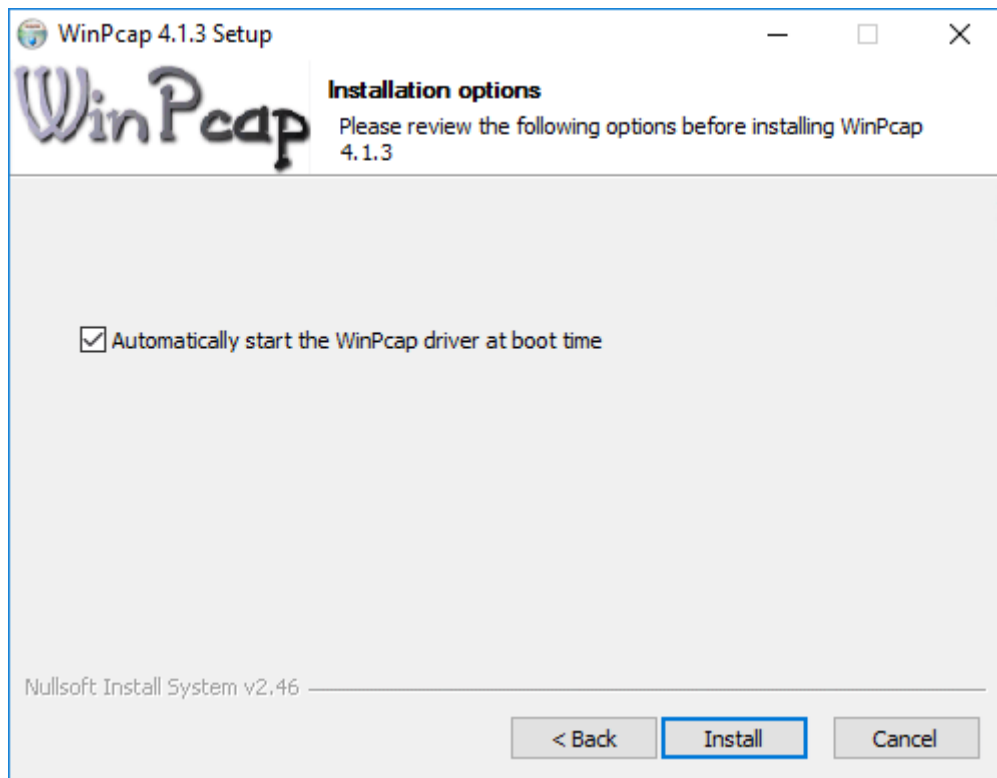
Interessant wird es dann auf der Seite *Packet Capture*. Hier schlägt der Installer vor, die Bibliothek WinPcap 4.1.3 zu installieren, welche auch gleich mitgeliefert wird.



*Wireshark bringt WinPcap mit, so dass man es gleich mitinstallieren kann.*

Aktiviert man diese Option, dann wird das WinPcap-Setup automatisch angestoßen. Sie läuft dann parallel zur Wireshark-Installation ab. Letztere wartet, bis der Capture-Treiber installiert ist.

Hier kommt es dann besonders darauf an, dass Windows den Capture-Treiber beim Systemstart automatisch startet, was aber die Default-Einstellung ist. Sofern auf dem System noch kein Capture-Treiber installiert war, wird dies auch so funktionieren.



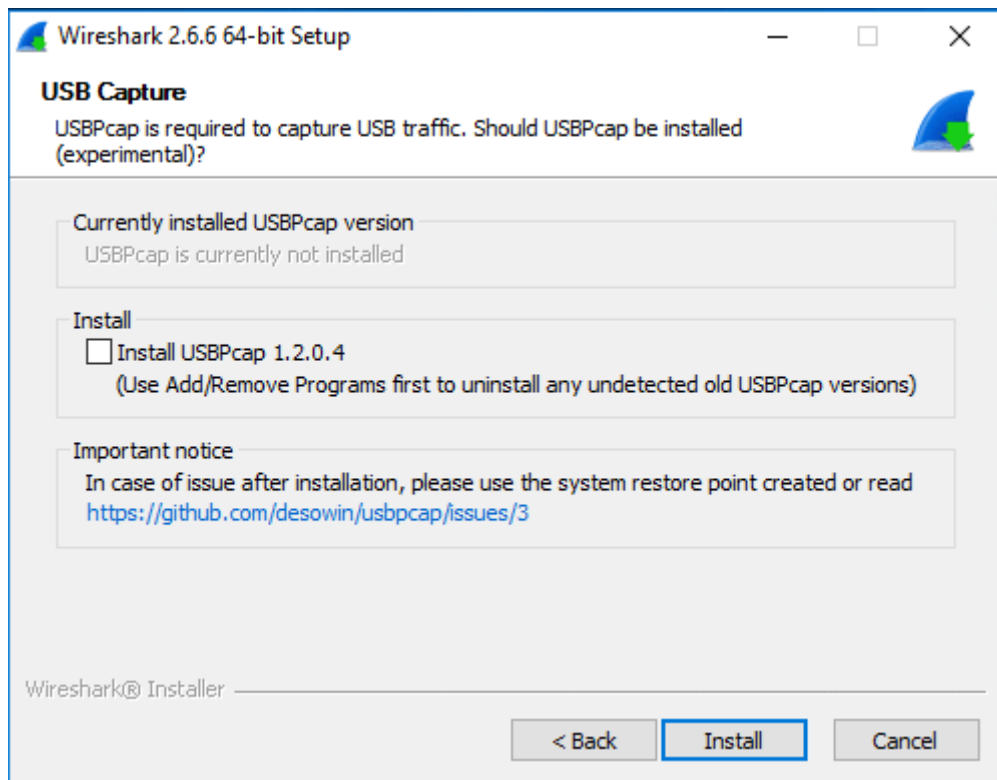
*Im Setup von WinPcap sollte man festlegen, dass der Treiber beim Systemstart automatisch geladen wird.*

Leistungsfähiger und stabiler als der WinPcap-Treiber ist indes der Capture-Treiber des nmap-Projektes namens *Npcap*. Hat man nmap bereits installiert, ist auch der Npcap-Treiber vorhanden und die Installation des WinPcap-Treiber sollte übergangen werden.

Ich hatte übrigens im Test einige Probleme damit, einen älteren WinPcap-Treiber zugunsten eines neueren Npcap rückstandsfrei loszuwerden, um Wireshark einwandfrei installieren zu können. Die Deinstallation von WinPcap funktioniert nur dann vollständig, wenn man den Rechner danach neu startet.

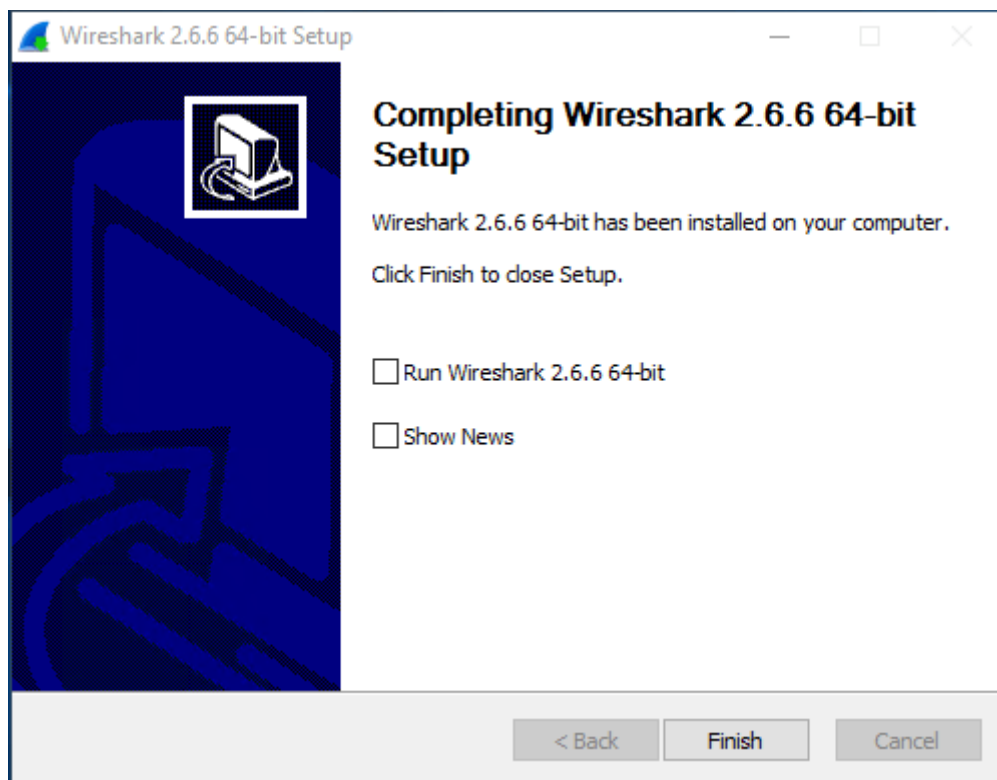
Optional kann man über den Wireshark-Installer dann auch noch USBPcap installieren, was wir hier aber auslassen.





*Bei Bedarf kann man noch einen Capture-Treiber für USB installieren.*

Wurde der jeweilige Capture-Treiber erfolgreich installiert, setzt auch der Wireshark-Installer seine Arbeit fort.



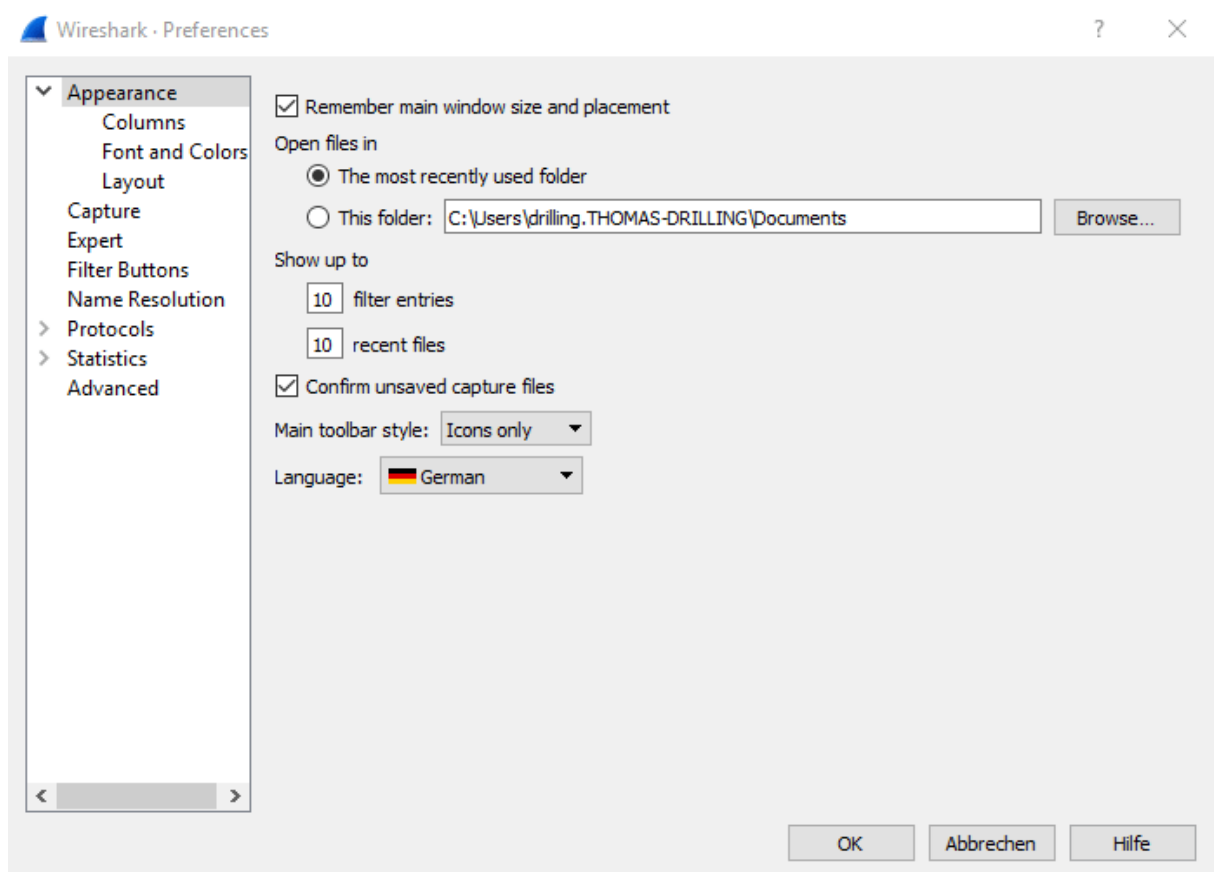
*Erfolgreicher Abschluss der Wireshark-Installation*

Zum Abschluss hat man die Wahl, Wireshark direkt zu starten. Ein weiterer Reboot ist nicht erforderlich.

Der Installer verrät allerdings nicht, dass man Wireshark als Administrator starten muss, um den Promiscuous Mode tatsächlich nutzen zu können.

### Spracheinstellungen ändern

Wenn man das Tool auf eine deutsche Bedienerführung umstellen möchte, kann man das unter *Edit* => *Preferences* => *Appearance* => *Language* tun. Die Übersetzung ist jedoch nicht allzu gut gelungen. Allerdings sind die bei einem Netzwerk-Sniffer obligatorischen Begrifflichkeiten aus den TCP/IP-Kontext in Englisch meist ohnehin eingängiger.



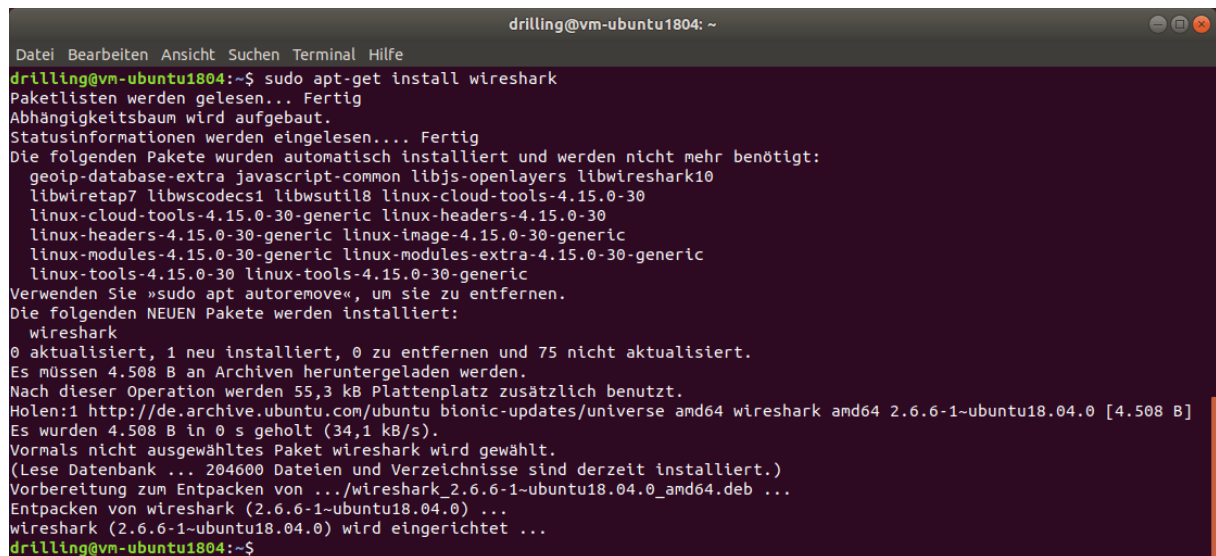
*Umstellen auf die deutsche Oberfläche*

## Wireshark unter Linux und macOS installieren

Wireshark wird häufig aus einer virtuellen Maschine verwendet. Nachdem Linux für Pentests und Sicherheitsanalysen sehr beliebt ist, kann man auch unter Windows überlegen, ein solches OS in einer VM zu nutzen und so Windows-Lizenzen zu sparen. Die folgende Anleitung zeigt, wie man Wireshark dort installiert.

Unter Linux wird Wireshark üblicherweise über den Paket-Manager der jeweiligen Distribution installiert. Auf einem Ubuntu-System öffnen wir dazu ein Terminal bzw. eine Konsole und geben folgendes Kommando ein:

```
sudo apt-get install wireshark
```



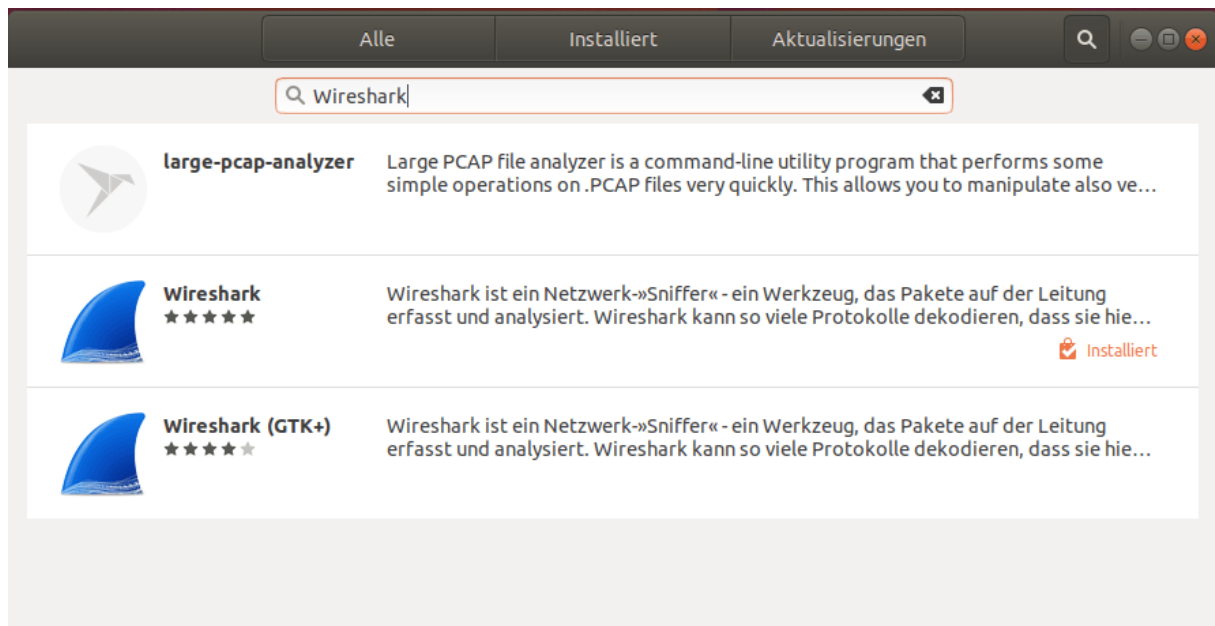
```
drilling@vm-ubuntu1804: ~
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
drilling@vm-ubuntu1804:~$ sudo apt-get install wireshark
Paketlisten werden gelesen... Fertig
Abhängigkeitsbaum wird aufgebaut.
Statusinformationen werden eingelesen... Fertig
Die folgenden Pakete wurden automatisch installiert und werden nicht mehr benötigt:
  geoip-database-extra javascript-common libjs-openlayers libwireshark10
  libwiretap7 libwscodec1 libwsutil8 linux-cloud-tools-4.15.0-30
  linux-cloud-tools-4.15.0-30-generic linux-headers-4.15.0-30
  linux-headers-4.15.0-30-generic linux-image-4.15.0-30-generic
  linux-modules-4.15.0-30-generic linux-modules-extra-4.15.0-30-generic
  linux-tools-4.15.0-30 linux-tools-4.15.0-30-generic
Verwenden Sie »sudo apt autoremove«, um sie zu entfernen.
Die folgenden NEUEN Pakete werden installiert:
  wireshark
0 aktualisiert, 1 neu installiert, 0 zu entfernen und 75 nicht aktualisiert.
Es müssen 4.508 B an Archiven heruntergeladen werden.
Nach dieser Operation werden 55,3 kB Plattenplatz zusätzlich benutzt.
Holen:1 http://de.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 wireshark amd64 2.6.6-1-ubuntu18.04.0 [4.508 B]
Es wurden 4.508 B in 0 s geholt (34,1 kB/s).
Vormals nicht ausgewähltes Paket wireshark wird gewählt.
(Lese Datenbank ... 204600 Dateien und Verzeichnisse sind derzeit installiert.)
Vorbereitung zum Entpacken von .../wireshark_2.6.6-1-ubuntu18.04.0_amd64.deb ...
Entpacken von wireshark (2.6.6-1-ubuntu18.04.0) ...
wireshark (2.6.6-1-ubuntu18.04.0) wird eingerichtet ...
drilling@vm-ubuntu1804:~$
```

*Installation von Wireshark unter Ubuntu 18.04 mit apt-get*

Das war es schon. Da das Paket-Management von Linux unter Ubuntu 18.04 LTS sämtliche benötigten Abhängigkeiten inzwischen auflöst, wird auch der zugehörige Capture-Treiber in Form der Bibliothek *libpcap* automatisch mit installiert. Dass genau diese Abhängigkeit in der Abbildung nicht auftaucht, liegt daran, dass die Library in unserem Fall bereits vorinstalliert ist.

## Wireshark über Ubuntu-Software einrichten

Unter Ubuntu-Desktop, also dem System mit GUI, kann man alternativ auch die Ubuntu-eigene Paketverwaltung *Ubuntu-Software* verwenden und über das Lupensymbol direkt nach "Wireshark" suchen.



Wireshark über die grafische Paketverwaltung von Ubuntu hinzufügen

Das Programm wird dann sowohl in der normalen als auch in einer für die GTK+-GUI-Bibliotheken optimierten Version gefunden und kann per Mausklick installiert werden. Da wir das Tool schon zuvor in der Konsole hinzugefügt haben, wird Wireshark als *installiert* gemeldet.

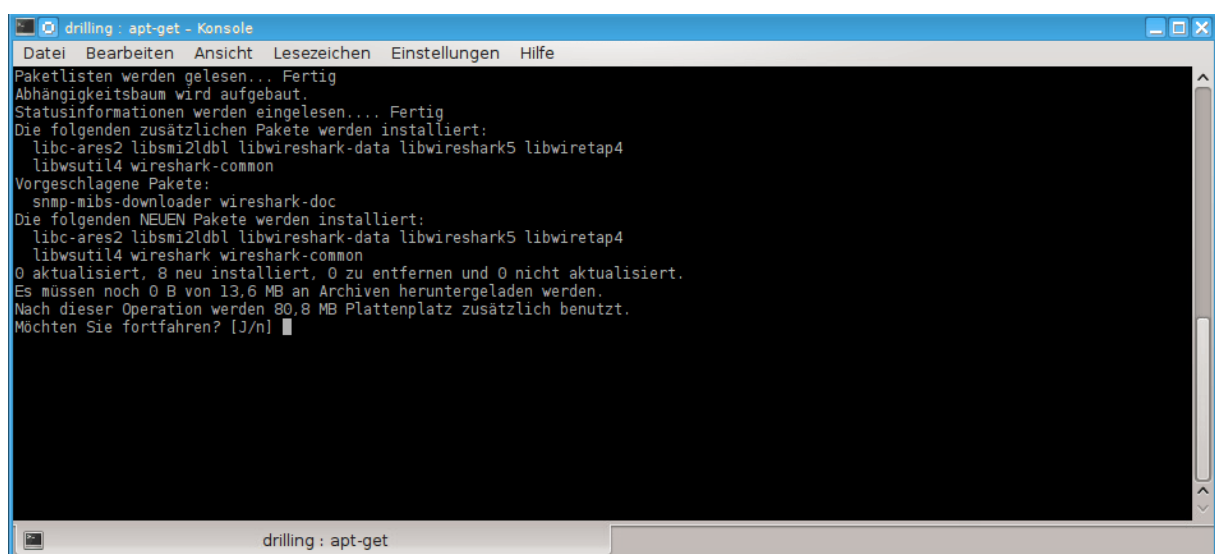
### Installation unter Debian

Auf einem Debian-System werden wir mit

```
su -
```

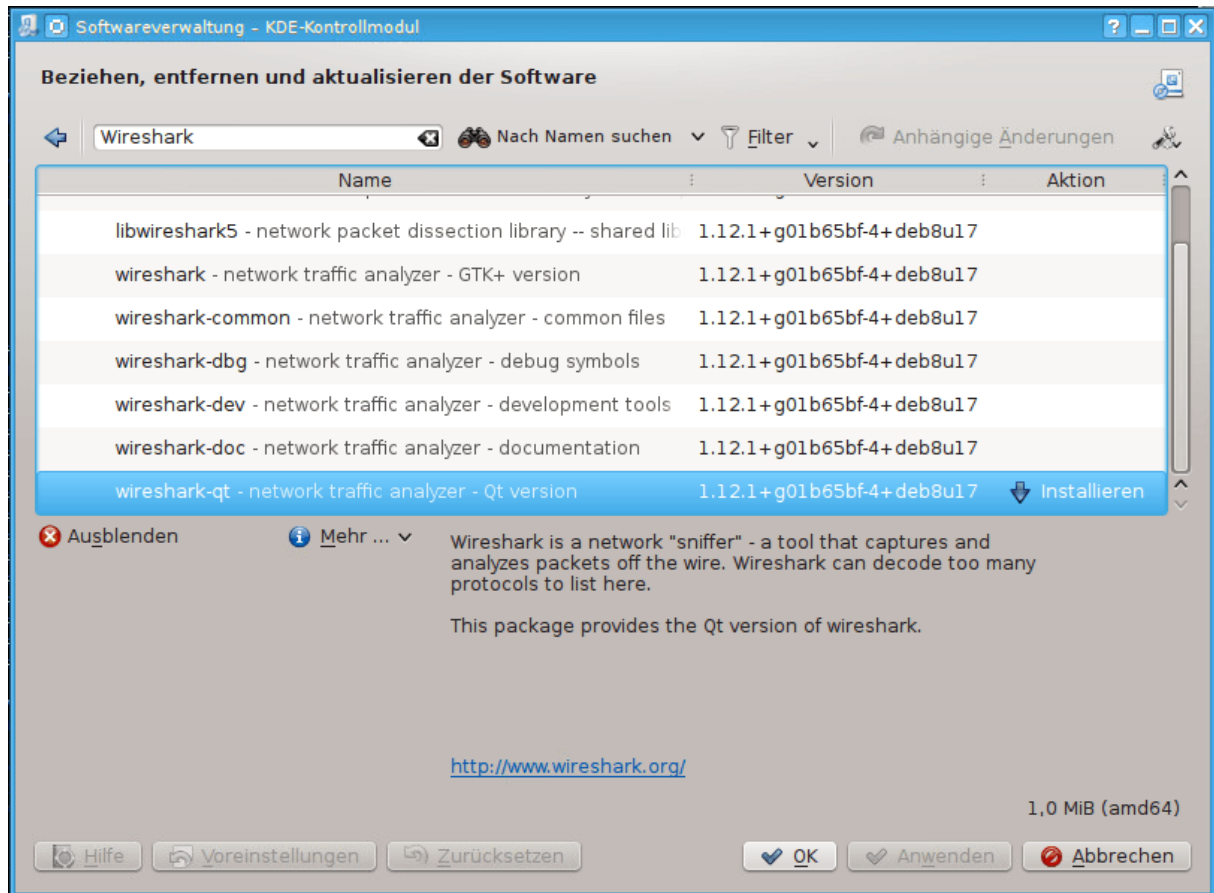
zu root, geben das zugehörige Passwort ein und verwenden den Befehl

```
apt-get install wireshark
```



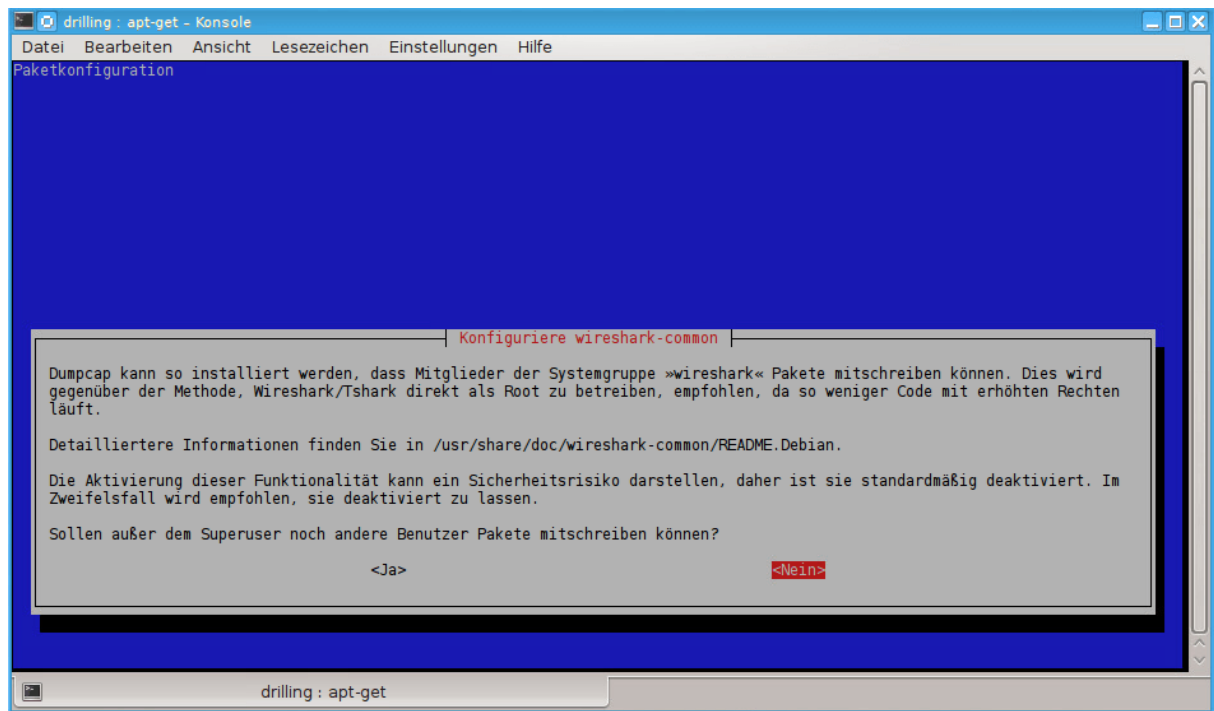
Wireshark über apt-get unter Debian hinzufügen

Auch hier werden Abhängigkeiten automatisch aufgelöst. Optional kann man natürlich auch unter Debian und jeder anderen Distribution einen grafischen Installer verwenden, hier die QT-Version (KDE) unter Debian 8.



*Wireshark über den grafischen Installer von Debian hinzufügen*

Auch hier ist die Abhängigkeit für *libpcap* bereits erfüllt. Bei Debian werden Software-Pakete übrigens gleich im Verlauf der Installation konfiguriert, weshalb man dort noch den Dialog "Konfiguriere wireshark-common" bestätigen muss.



*Entscheiden, ob Wireshark nur von root oder auch von anderen Benutzern verwendet werden darf.*

Hier geht es darum, ob außer dem Superuser (root) noch andere Benutzer in der Lage sein sollen, mit Wireshark Pakete aufzuzeichnen. Das sollte man in der Regel verneinen (Default). Wir müssen also im praktischen Einsatz Wireshark immer mit root-Berechtigungen starten, was auch sinnvoll ist.

### Wireshark starten

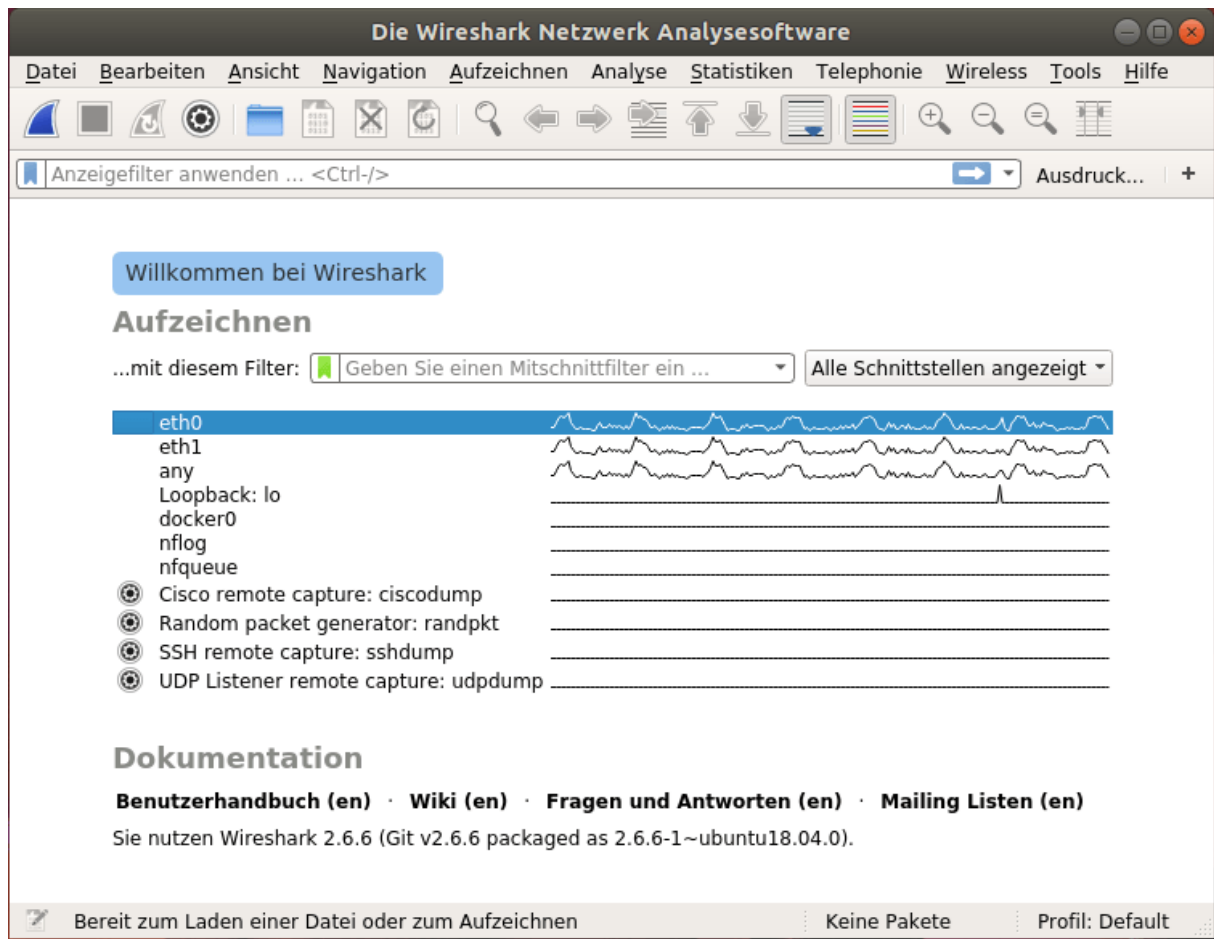
Ist das erledigt, können wir Wireshark unter Debian mit

```
su -
wireshark&
```

oder unter Ubuntu mit

```
sudo wireshark&
```

starten. Es empfiehlt sich, dem Aufruf das &-Zeichen anzuhängen, damit die Konsole weiter eingabefähig bleibt oder geschlossen werden kann.

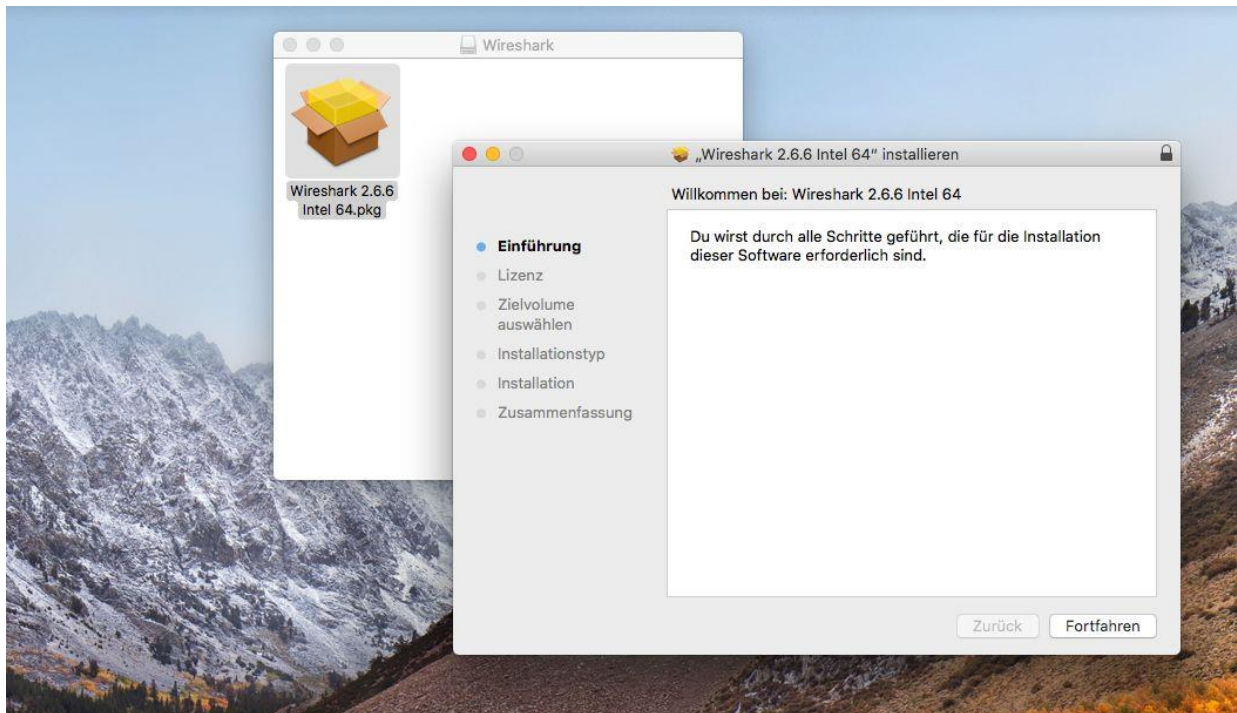


Wireshark unter Linux

## Setup unter macOS

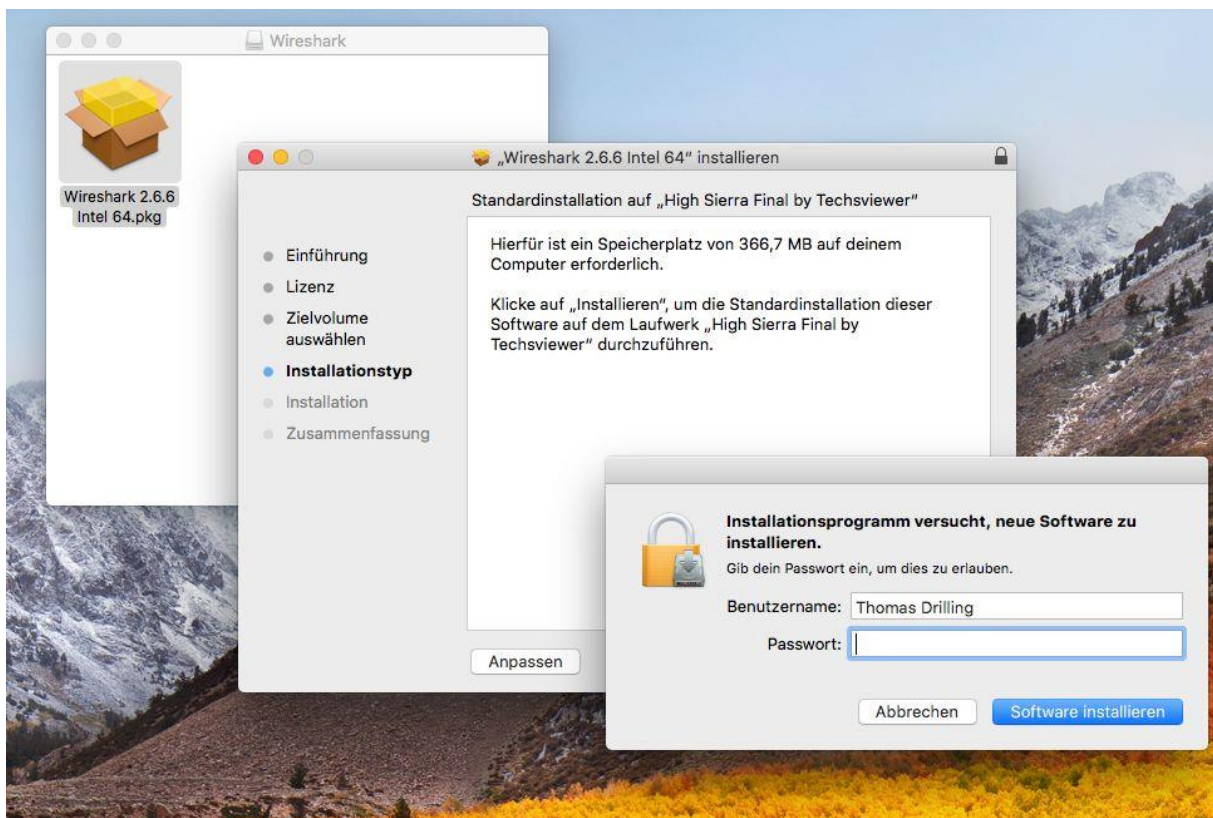
Unter macOS X ist die Wireshark-Installation ein Kinderspiel. Man lädt einfach das DMG-Paket [von der Download-Seite herunter](#), mountet es und klickt doppelt auf den Installer *Wireshark <Version> Intel64.pkg*.





*Wireshark-Setup unter macOS*

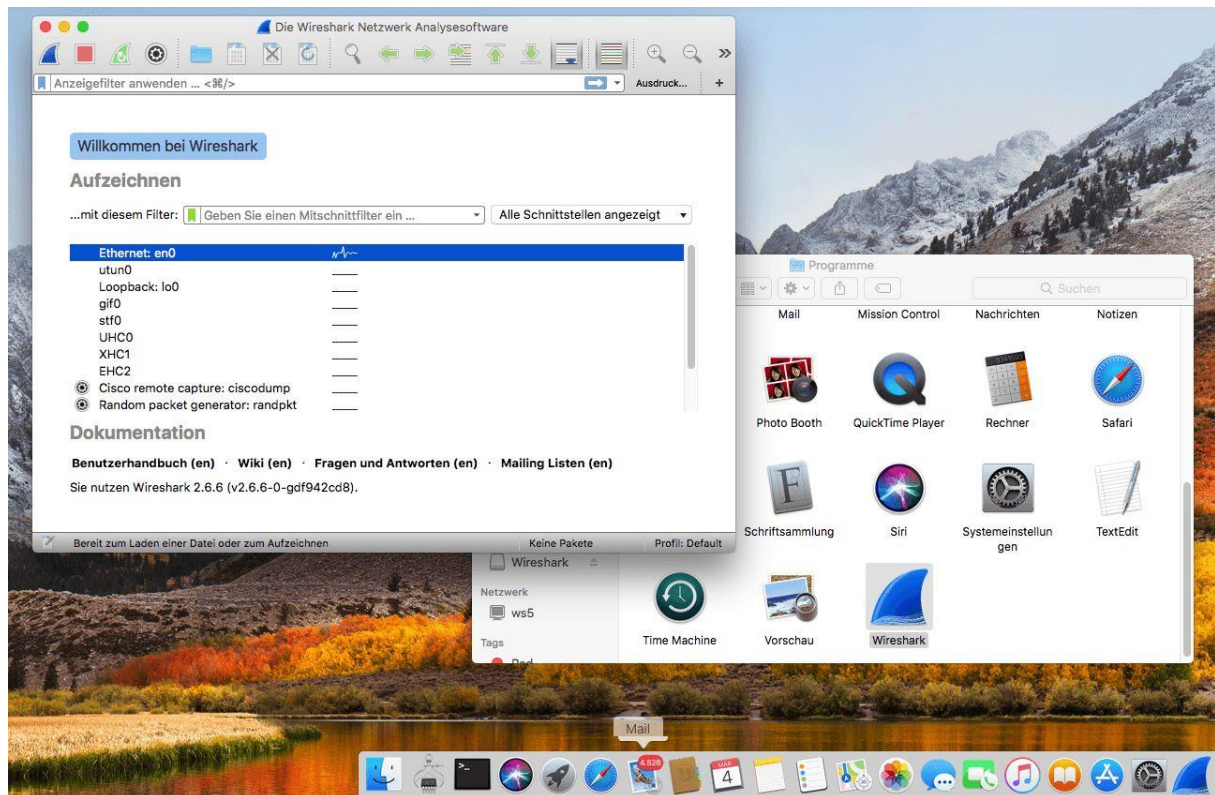
Dann durchläuft man mit *Fortfahren* die einzelnen Schritte im Assistenten, genau wie in der Windows-Version. Für die Installation bedarf es administrativer Rechte, ähnlich wie bei sudo unter Ubuntu gibt man hier sein Passwort ein.



*Der Installationsassistent verlangt nach einem Passwort, danach läuft das Setup durch.*



Danach läuft die Installation ohne weitere Rückfrage durch. Etwaige Abhängigkeiten werden automatisch behoben, sodass man Wireshark jetzt problemlos im Finder per Doppelklick starten kann.



*Wireshark für macOS aus dem Finder starten*

Wir haben damit Wireshark in allen Varianten auf den wichtigsten Systemen installiert.

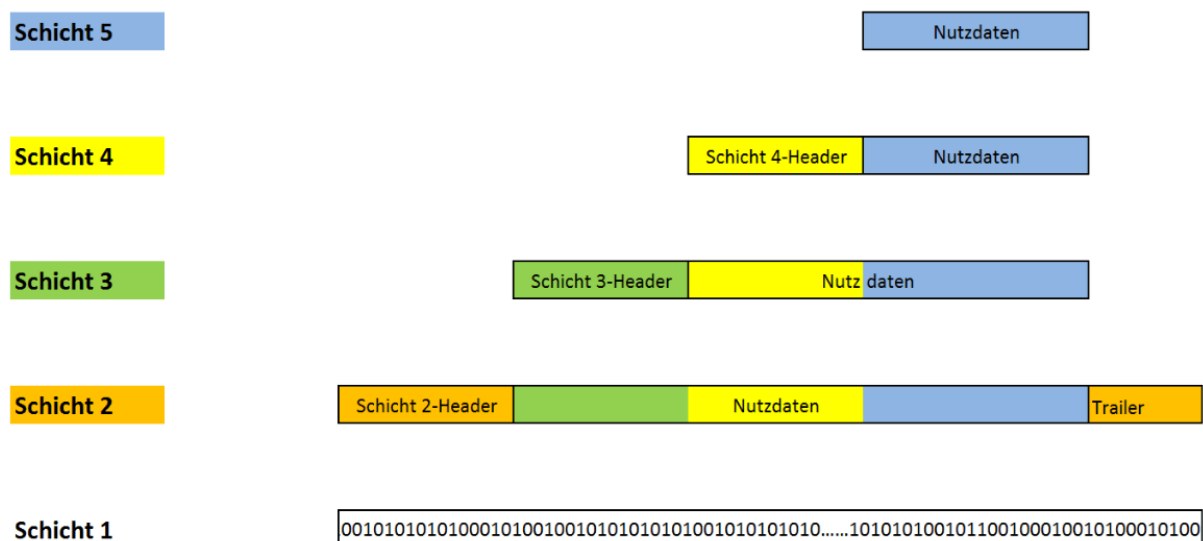
# TCP/IP-Grundlagen für die Netzwerk-Analyse

Für den erfolgreichen Einsatz von Wireshark (oder Sniffen wie tcp-dump) ist es notwendig, dass man sich im TCP/IP- bzw. im OSI-Schichtenmodell sicher bewegt. Die grundlegenden Zusammenhänge kann man sehr schön in Wireshark verifizieren, ganz unabhängig vom transportierten Protokoll und dem Einsatzzweck.

Angenommen wir sind im klassischen Ethernet kabelgebunden (CAT 6) unterwegs und berücksichtigen nur die unteren 4 Schichten. Dann haben wir es mit folgenden Segment-/Packet- und Frame-Größen zu tun, wobei die Header des TCP- und des IP-Protokolls jeweils 20 Byte (ohne Optionen) umfassen.

Die Kombination aus Header und Trailer beim Ethernet-Frame auf Schicht 2 (der einzige Layer, bei dem man eigentlich real von Kapselung sprechen dürfte) macht  $14+4=18$  Byte aus. Die Payload wird ergänzt um den Header der jeweils höheren Schicht für die Nutzdaten der darunter liegenden Schicht:

- **Layer 4: TCP-Segment = MSS = 1480** = 1460 (Payload) + 20 Byte Header (4 Words)
- **Layer 3: IP-Paket = MTU = 1500** = 1480 Payload + 20 Header (4 Words)
- **Layer 2: Ethernet-Frame** = (bei Ethernet) = **1518** = 1500 Payload + 18 (14 Byte Header + 4 Byte Trailer)



*Jede Schicht enthält neben der Payload auch den Header der darüber liegenden Schicht.*

Eine Ausnahme bildet die Verwendung von IEEE 802.1Q-Tags (für VLAN oder Priorisierung). Dort wird der Header um vier Byte erweitert und das Frame damit maximal 1522 Byte groß.

Dazu noch mal zur Erinnerung der Aufbau des **Ethernet-Frame** (Schicht 2):

$$\text{Ziel-MAC (6B)} - \text{Quell-MAC (6B)} - \text{Typ (2B)} - \text{Payload (1500B)} - \text{FCS (4B)} = 1518$$

Das Typ-Feld gibt an, um welche Art von Packet es sich handelt, das hier von Schicht 3 eingekapselt ist. bei IPv4 z. B. 0x0800.

**Bei IP** (Schicht 3) ist der Header etwas komplexer aufgebaut und wird meist in 5 Worten untereinander = 20 Byte dargestellt, wobei die Horizontale genau 32 Bit (4 Byte) umfasst. Eine typische Abbildung ist diese:

Version	IHL	ToS	Paketlänge	
Kennung			Flags	Fragment-Offset
TTL	Protokoll		Header-Checksumme	
Quell-IP-Adresse				
Ziel-IP-Adresse				
Optionen/Füllbits				
Daten....				

#### *Aufbau des Headers auf Schicht 3*

Die Bedeutung der einzelnen Felder sollte bekannt sein. Übrigens werden *Optionen/Füllbits* meist nicht genutzt, sodass es beim Header bei genau 5 Worten x 4 Bytes = 20 Bytes bleibt. Diese Zählweise in Worten ist im IHL-Feld spezifiziert.

Im Gegensatz zum Ethernet-Frame sind die Quell- und Ziel-IP anders als Ziel- und Quell-MAC auf Layer 3 nicht vertauscht und jeweils 32 Bit (4 Byte) lang (bei IPv4). Relevant für die folgenden Betrachtungen ist das *Protocol*-Feld.

Analog zum *Typ*-Feld auf Layer 2 gib es an, welches höherwertige Protokoll eingekapselt bzw. transportiert wird. Beispiele wären die Layer-4-Protokolle TCP (Protocol-Nr. 6), UDP (Protocol-Nr. 17) und ICMP mit der Protocol-Nr. 1. Dieses ist allerdings im Gegensatz zu TCP und UDP ein Layer-3-Protokoll wie IP selbst auch.

Werfen wir der Vollständigkeit halber noch einen Blick auf Layer 4, und zwar den TCP-Header, auch wenn wir unsere Analysen mit ICMP starten wollen.

Quell-Port		Ziel-Port	
Sequenz-Nummer			
Acknowledgement-Nummer			
D. O.	Res.	Flags	Window-Größe
Check-Summe		Urgent-Pointer	
Optionen/Füllbits			
Daten....			

#### *Aufbau des TCP-Headers*

Hier beginnt der Aufbau des ebenfalls 20 Bits langen TCP-Headers (ohne Optionen/Füllbits) mit Quell- und Ziel-Port. Da TCP im Gegensatz zu UDP über eine Flusskontrolle verfügt und immer sicherstellt, dass Segmente auch ankommen, sind für spätere Analysen die Felder *Sequence-Number*, *Acknowledgement-Number*, *Flags* und *Window-Size* von Bedeutung.

Wir werden in künftigen Beispielen den TCP-Authentifizierungs-Handshake verfolgen sowie die Flusssteuerung für eine SSH- oder FTP-Kommunikation.

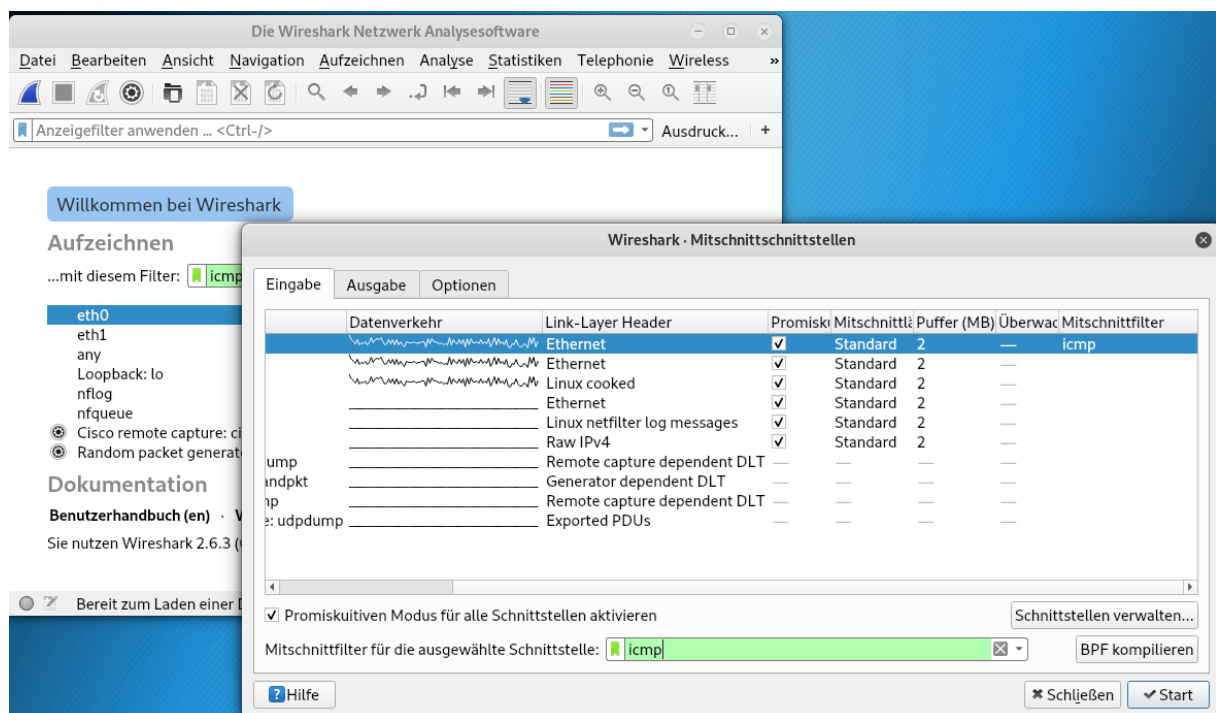
## Netzwerk-Analyse am Beispiel von Ping

Dieses Kapitel zeigt anhand von Ping, wie sich Mitschnitte des Netzwerk-Traffics in Wireshark auswerten lassen. Das Sniffing-Tool erlaubt es dem Anwender durch die Aufteilung des Fensters, sich schnell durch die verschiedenen Schichten des Netzwerk-Stacks zu hangeln und detaillierte Informationen auszulesen.

Für dieses Beispiel verwenden wir der Einfachheit halber und aus Gründen der Sicherheit zwei virtuelle Systeme (Ubuntu und Kali Linux) mit den IP-Adressen 192.160.0.124 (Ubuntu) und 192.168.0.123 (Kali). Auf beiden Systemen ist Wireshark verfügbar, wir können sie daher als Client oder Server verwenden.

### Aufzeichnung starten

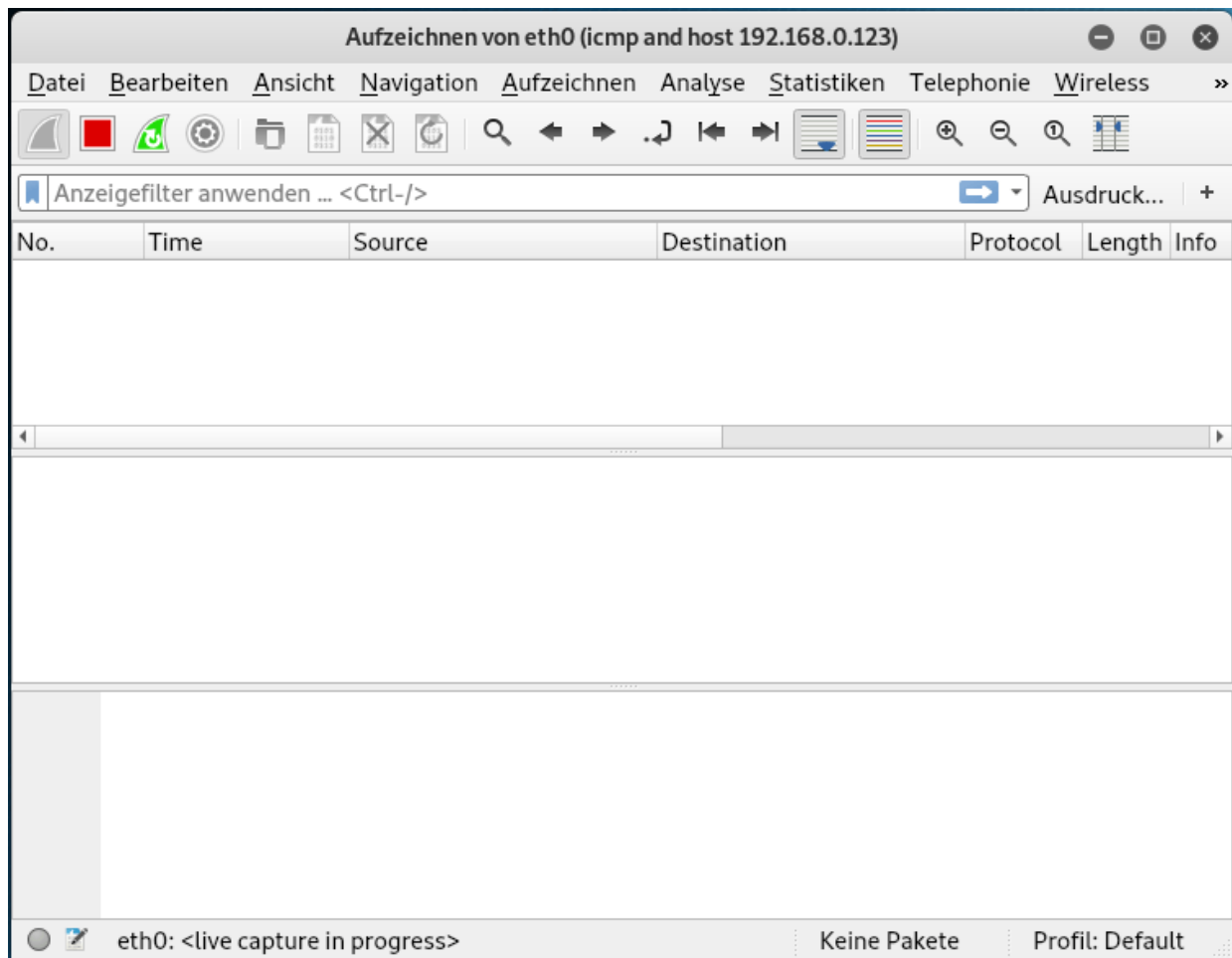
Zuerst klicken wir im Menü *Aufzeichnen* auf *Optionen* und markieren das mitzuschneidende Interface (hier eth0). Wenn Sie etwas länger mit der Maus auf den Eintrag verharren oder das kleine Dreieckssymbol bei Interface aufklappen, wird auch die zugehörige IP-Adresse angezeigt, um ganz sicher zu gehen, das richtige Gerät erwischt zu haben.



*Aufzeichnung des Traffic auf eth0 mit dem Filter icmp in Wireshark starten*

Am Fuß des Dialoges bei "Mitschnittfilter für die ausgewählte Schnittstelle" geben wir den Wert "ICMP" ein.

Sofern es sich um ein gültiges, also von Wireshark akzeptiertes Eingabemuster handelt (erlaubt sind diverse Kombinationen aus Protokollbezeichnung und Ports, die sich auch logisch verknüpfen lassen), wird der Hintergrund grün.



*Leeres Aufzeichnungsfenster in Wireshark vor dem Absenden eines Ping.*

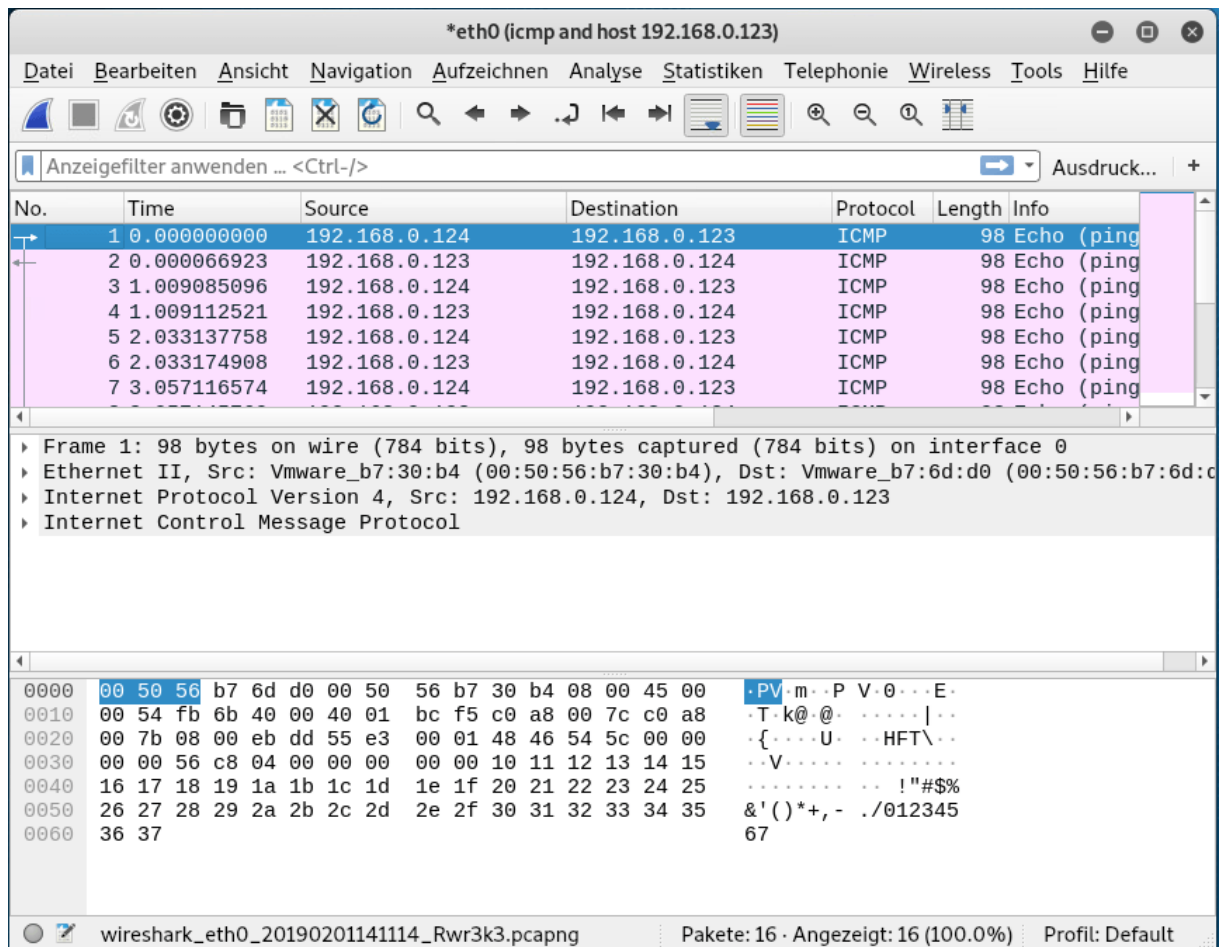
Dann klicken wir auf *Start*. Wireshark wechselt jetzt in den Aufzeichnungsmodus und sollte ein leeres Aufzeichnungsfenster zeigen, sofern Sie sich in einer isolierten Labor-Umgebung befinden.

In realen Umgebungen dagegen wird durchaus auch anderweitiger ICMP-Traffic zu beobachten sein. Dieser muss sich auch nicht explizit an den Host richten, auf dem Wireshark läuft, denn wir benutzen den angegebenen Port ja im Promiscuous-Mode.

Wir können das Ziel im Filter aber auch spezifisch angeben, zum Beispiel mit

```
icmp and host 192.168.0.123
```

Anschließend pingen wir das System von unserem Ubuntu-Client aus. Im Gegensatz zu Windows läuft ein ping unter Linux bis zu einem Benutzer-Timeout. Da wir wenige Pakete benötigen, brechen wir den ping unmittelbar danach mit Strg+C wieder ab und beenden dann den Mitschnitt mit einem Klick auf das rote Quadrat. Das Ergebnis sollte dann so aussehen wie in folgender Abbildung.



Ergebnis der Aufzeichnung eines Ping in Wireshark

Das Beispiel ist trivial, aber wir können die Ausgabe hervorragend nutzen, um den Aufbau der Wireshark-Ausgabe zu erläutern.

### Aufteilung des Fensters

Im obersten Abschnitt des Fensters finden Sie die komplette Liste der aufgezeichneten Pakete. Der mittlere Teil zeigt für das oben ausgewählte Frame detaillierte Informationen zu den einzelnen Protokoll-Headern im OSI-Modell in aufsteigender Reihenfolge.

Der erste Eintrag mit "Frame 1....:" steht dabei allerdings nicht für den Bitübertragungs-Layer 1 im OSI-Modell, sondern enthält eine von Wireshark erstellte Zusammenfassung mit grundlegenden Informationen zum Frame selbst. So zeigt der Eintrag die Frame-Nummer (hier 1) und die Länge des aufgezeichneten Paketes (98 Byte = 784 Bit) sowie die Interface-Nummer.

Umgangssprachlich ist im Zusammenhang mit Wireshark bzw. der Traffic-Analyse im Netzwerk von "Paketen" die Rede. Das ist aber streng genommen nicht ganz richtig, wenn wir uns wie in diesem Beispiel im eigenen lokalen Netzwerk und damit auf Layer-2 bewegen. Daher handelt es sich beim zweiten Eintrag im mittleren Fenster mit der Bezeichnung "Ethernet II" offensichtlich um Details zum von uns aufgezeichneten Ethernet-Frame.

### MAC- und IP-Adresse auslesen

So gibt Wireshark hier die Quell-MAC-Adresse (Src) mit Alias und tatsächlicher Adresse (00:50:56:b7:30:b4) sowie die Ziel-MAC-Adresse an. Der besseren Lesbarkeit wegen (nur hier in der

Zusammenfassung) erfolgt dies nicht in umgekehrter Reihenfolge. Bekanntlich steht im tatsächlichen Frame immer erst die Ziel-MAC.

Aus Alias und Art der MAC-Adresse (die erste 3 Byte sind immer herstellerspezifisch) ist außerdem ersichtlich, dass es sich um generische Adressen von VMware handelt, denn unsere VMs laufen unter ESXi.

Die Bezeichnung "Ethernet II ..." in Zeile 3 steht dann folgerichtig für das von Layer 2 transportierte, bzw. gekapselte Protokoll, also *Internet Protocol Version 4*. Zu erkennen ist hier außerdem bereits die Quell-IP (Src) mit 192.168.0.124 und Ziel-IP (Dst) mit 192.168.0.123.

Zeile 4 bezieht sich dann auf das eingekapselte bzw. im Layer-3 transportierte höherwertige Protokoll. Das muss aber nicht zwingend eines von Layer 4 (TCP, UDP), sondern kann durchaus auch ein höherwertiges Protokoll auf Layer 3 sein wie im Fall von ICMP. Es ist zu erkennen an der Bezeichnung "Internet Control Message Protocol".

Die Paketliste im oberen Teil verrät auch noch so Einiges. Die Nummer in der ersten Spalte wurde von Wireshark selbst der besseren Orientierung wegen angefügt und ist nicht wirklich im Frame enthalten.

Wir erkennen aber auch hier für jedes einzelne Frame auf dem ersten Blick die Source- und Quell-IP, die zeitliche Länge (Dauer) der Aufzeichnung, das übertragene Protokoll (ICMP) und den ICMP-Typ (hier "echo reply" und "echo request") sowie id, ttl und seq-Number.

Der Mitschnitt des Frames mit der Nr. 1 im Layer 2 offenbart, dass ein Client mit der Layer-3-IP-Adresse 192.168.0.124 ein Layer-3-ICMP-Paket an das Ziel 192.168.0.123 vom ICMP-Typ "echo-request" (ping) versendet hat.

### Berechnung der Frame-Länge

Der Frame ist insgesamt 98 Byte lang, wobei man erwähnen muss, dass man die Länge der Test-Pakete, die der ping-Befehl versendet, per Option einstellen könnte. Warum der Frame 98 Byte lang ist, lässt sich auch beantworten.

Die Mindest-Frame-Size im Ethernet ist 64 Byte. Da die ping-Default-Size nur 32 Byte ist, wird auf 64 Byte aufgefüllt. Hinzu kommen 14 Byte Ethernet-Header + 20 Byte IPv4 Header =  $64 + 14 + 20 = 98$  Byte, denn einen TCP-Header gibt es im Fall von ICMP ja nicht.

Im unteren Fensterteil schließlich zeigt Wireshark den kompletten Inhalt des aufgezeichneten Pakets (in diesem Fall 98 Bytes) in Raw-Darstellung an. Man achte dabei auf das folgende interessante Detail. Klickt man im mittleren Fenster

- auf den Eintrag für "Frame 1", ist unten in den Rohdaten das komplette Paket mit seiner Länge von 98 Byte blau markiert.
- nur auf den Layer-2-Header-Eintrag, dann wird unten nur der Teil des Paketes blau markiert, der den Header eines Ethernet-Frames von 14 Byte umfasst.
- hingegen in der Mitte auf den Eintrag für das IP-Paket, werden unten exakt 20 Byte markiert und zwar beginnend mit jenem Byte, das sich an den Ethernet-Frame-Header anschließt, denn der IP-Header ist ja der Kapselung wegen Teil der Payload (Nutzdaten) von Schicht 2 (Ethernet).



\*eth0 (icmp and host 192.168.0.123)

Datei

Bearbeiten

Ansicht

Navigation

Aufzeichnen

Analyse

Statistiken

Telephonie

Wireless

Tools

Hilfe

### Markierte 20 Bytes des IPv4-Headers

Schauen wir nun die einzelnen Ebenen im Detail an, um an weitere Informationen zu gelangen. Markieren wir Zeile 1 im mittleren Fenster, dann werden wir auch im unteren Fenster die gesamten Rohdaten von 98 Byte markiert.

Entfalten wir diese durch einen Klick auf das Dreiecks-Icon vor Frame 1, erhalten wir noch mehr Informationen. Das Ergebnis sollte so aussehen, sofern Sie den Untereintrag für "Interface" ebenfalls noch aufklappen.

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0									
Interface id: 0 (eth0)									
Interface name: eth0									
Encapsulation type: Ethernet (1)									
Arrival Time: Feb 1, 2019 14:14:48.331499894 CET									
[Time shift for this packet: 0.000000000 seconds]									
Epoch Time: 1549026888.331499894 seconds									
[Time delta from previous captured frame: 0.000000000 seconds]									
[Time delta from previous displayed frame: 0.000000000 seconds]									
[Time since reference or first frame: 0.000000000 seconds]									
Frame Number: 1									
Frame Length: 98 bytes (784 bits)									
Capture Length: 98 bytes (784 bits)									
0000	00 50 56 b7 6d d0 00 50	56 b7 30 b4 08 00 45 00	.PV.m..P.V.0...E.						
0010	00 54 fb 6b 40 00 40 01	bc f5 c0 a8 00 7c c0 a8	.T.k@. @. .... ..						
0020	00 7b 08 00 eb dd 55 e3	00 01 48 46 54 5c 00 00	{...U. ..HFT\..						
0030	00 00 56 c8 04 00 00 00	00 00 10 11 12 13 14 15	..V.....						
0040	16 17 18 19 1a 1b 1c 1d	1e 1f 20 21 22 23 24 25	..... !"#\$\$%						
0050	26 27 28 29 2a 2b 2c 2d	2e 2f 30 31 32 33 34 35	&'()*+,-./012345						
0060	36 37		67						

### Detailinformationen und Rohdaten zu Frame 1

Für eine tiefere Analyse entfalten wir jetzt die Details für den zweiten Eintrag im mittleren Fenster "Ethernet II ..", also zum Ethernet-Frame-Header auf Layer-2. Achten Sie dabei auf die unten markierten Rohdaten, die genau dessen 14 Byte umfassen. Im mittleren Fenster lassen sich dann weitere Detailebenen einblenden.

```

▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▼ Ethernet II, Src: Vmware_b7:30:b4 (00:50:56:b7:30:b4), Dst: Vmware_b7:6d:d0 (00:50:56:b7:6d:d0)
  ▼ Destination: Vmware_b7:6d:d0 (00:50:56:b7:6d:d0)
    Address: Vmware_b7:6d:d0 (00:50:56:b7:6d:d0)
    .... ..0. .... = LG bit: Globally unique address (factory default)
    .... ..0. .... = IG bit: Individual address (unicast)
  ▼ Source: Vmware_b7:30:b4 (00:50:56:b7:30:b4)
    Address: Vmware_b7:30:b4 (00:50:56:b7:30:b4)
    .... ..0. .... = LG bit: Globally unique address (factory default)
    .... ..0. .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
▶ Internet Protocol Version 4, Src: 192.168.0.124, Dst: 192.168.0.123
▶ Internet Control Message Protocol
0000  00 50 56 b7 6d d0 00 50 56 b7 30 b4 08 00 45 00  .PV.m..P.V.0...E.
0010  00 54 fb 6b 40 00 40 01 bc f5 c0 a8 00 7c c0 a8  .T.k@.@. ....|..
0020  00 7b 08 00 eb dd 55 e3 00 01 48 46 54 5c 00 00  .{...U..HFT\..
0030  00 00 56 c8 04 00 00 00 00 00 10 11 12 13 14 15  .V.....
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25  .....!#$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35  &'()*+,-./012345
0060  36 37 67

```

#### Details zum Ethernet-Frame Header auf Layer-2

Wir erkennen nun Informationen zur Source- und Destination-MAC-Adresse. Im Layer-2, also im Ethernet-Frame-Header, steht immer zuerst die Ziel-MAC. Wie schon erwähnt lässt sich an den jeweils linken 3 Bytes der 6 Byte langen MAC-Adresse erkennen, dass der Hersteller des Netzwerkgeräts VMware ist, wir uns also offenbar in einer virtualisierten Umgebung bewegen.

Am Ende der Anzeige wird noch das *Type*-Feld des Ethernet-Frames angezeigt, das angibt, welches Protokoll der nächsten höheren Ebene im Frame transportiert wird, hier IPv4 (0x0800).

Noch ein Detail ist interessant. Markieren Sie in der *Details*-Ansicht beispielsweise die Ziel-MAC-Adresse, dann wird auch unten im Rohdaten-Fenster exakt der zugehörige Bereich markiert. Das Gleiche gilt für die direkt dahinter folgende Source-MAC.

```

▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▼ Ethernet II, Src: Vmware_b7:30:b4 (00:50:56:b7:30:b4), Dst: Vmware_b7:6d:d0 (00:50:56:b7:6d:d0)
  ▼ Destination: Vmware_b7:6d:d0 (00:50:56:b7:6d:d0)
    Address: Vmware_b7:6d:d0 (00:50:56:b7:6d:d0)
    .... ..0. .... = LG bit: Globally unique address (factory default)
    .... ..0. .... = IG bit: Individual address (unicast)
  ▼ Source: Vmware_b7:30:b4 (00:50:56:b7:30:b4)
    Address: Vmware_b7:30:b4 (00:50:56:b7:30:b4)
    .... ..0. .... = LG bit: Globally unique address (factory default)
    .... ..0. .... = IG bit: Individual address (unicast)
  Type: IPv4 (0x0800)
▶ Internet Protocol Version 4, Src: 192.168.0.124, Dst: 192.168.0.123
▶ Internet Control Message Protocol
0000  00 50 56 b7 6d d0 00 50 56 b7 30 b4 08 00 45 00  .PV.m..P.V.0...E.
0010  00 54 fb 6b 40 00 40 01 bc f5 c0 a8 00 7c c0 a8  .T.k@.@. ....|..
0020  00 7b 08 00 eb dd 55 e3 00 01 48 46 54 5c 00 00  .{...U..HFT\..
0030  00 00 56 c8 04 00 00 00 00 00 10 11 12 13 14 15  .V.....
0040  16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25  .....!#$%
0050  26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35  &'()*+,-./012345
0060  36 37 67

```

#### MAC-Adresse des Quellrechners

Nun setzen wir die Analyse mit Detailinformationen aus dem Layer-3-Header (Eintrag "Internet Protocol Version 4" im mittleren Fenster) fort. Da der IP-Header mit 20 Byte länger ist und mehr Felder kennt als der Ethernet-Frame, beansprucht das mittlere Fenster auch mehr Platz. Markieren wir hier wieder analog zu oben wieder die Source-IP, dann wird auch nur der zugehörige Bereich in den Rohdaten eingefärbt.

```

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
> Ethernet II, Src: Vmware_b7:30:b4 (00:50:56:b7:30:b4), Dst: Vmware_b7:6d:d0 (00:50:56:b7:6d:d0)
> Internet Protocol Version 4, Src: 192.168.0.124, Dst: 192.168.0.123
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    0000 00.. = Differentiated Services Codepoint: Default (0)
    .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 84
  Identification: 0xfb6b (64363)
  > Flags: 0x4000, Don't fragment
    0... .. = Reserved bit: Not set
    .1.. .. = Don't fragment: Set
    ..0. .. = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0
  Time to live: 64
  Protocol: ICMP (1)
  Header checksum: 0xbcf5 [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.0.124
  Destination: 192.168.0.123
  > Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xebdd [correct]
    [Checksum Status: Good]
    Identifier (BE): 21987 (0x55e3)
    Identifier (LE): 58197 (0xe355)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Response frame: 2]
0010 00 54 fb 6b 40 00 40 01 bc f5 c0 a8 00 7c c0 a8 .T.k@.0. ....|..
0020 00 7b 08 00 eb dd 55 e3 00 01 48 46 54 5c 00 00 .{....U. ...HFT\..
0030 00 00 56 c8 04 00 00 00 00 00 10 11 12 13 14 15 ..V.....
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 ..... !"#%$

Source (ip.src), 4 Bytes
Pakete: 16 - Angezeigt: 16 (100.0%) - Verworfen: 0 (0.0%) - Profil: Default
```

### Detail-Informationen aus dem Layer-3-Header

Analog zum Typ-Feld im Layer 2 gibt es im Layer 3 das "Protocol"-Feld, das hier den Wert 1 hat, was für ICMP, ebenfalls ein Layer-3-Protokoll, steht. Andere populäre "höherwertige" Protokolle wären zum Beispiel die Layer-4-Protokolle TCP mit 6 oder UDP mit 17.

```

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
> Ethernet II, Src: Vmware_b7:30:b4 (00:50:56:b7:30:b4), Dst: Vmware_b7:6d:d0 (00:50:56:b7:6d:d0)
> Internet Protocol Version 4, Src: 192.168.0.124, Dst: 192.168.0.123
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    0000 00.. = Differentiated Services Codepoint: Default (0)
    .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
  Total Length: 84
  Identification: 0xfb6b (64363)
  > Flags: 0x4000, Don't fragment
    0... .. = Reserved bit: Not set
    .1.. .. = Don't fragment: Set
    ..0. .. = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0
  Time to live: 64
  Protocol: ICMP (1)
  Header checksum: 0xbcf5 [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.0.124
  Destination: 192.168.0.123
  > Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xebdd [correct]
    [Checksum Status: Good]
    Identifier (BE): 21987 (0x55e3)
    Identifier (LE): 58197 (0xe355)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Response frame: 2]
0010 00 54 fb 6b 40 00 40 01 bc f5 c0 a8 00 7c c0 a8 .T.k@.0. ....|..
0020 00 7b 08 00 eb dd 55 e3 00 01 48 46 54 5c 00 00 .{....U. ...HFT\..
0030 00 00 56 c8 04 00 00 00 00 00 10 11 12 13 14 15 ..V.....
0040 16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25 ..... !"#%$

Protocol (ip.proto), 1 Byte
Pakete: 16 - Angezeigt: 16 (100.0%) - Verworfen: 0 (0.0%) - Profil: Default
```

### Protocol-Feld auf Layer 3 mit Wert 1 für ICMP

Werfen wir abschließend noch einen Blick auf den Abschnitt "Internet Control Message Protocol" (ICMP), also das nächsthöhere transportierte Protokoll. Offenbar kennt auch dieses verschiedene Typen, die im entsprechenden Feld spezifiziert werden. Unser Frame1 verwendet den Typ 8, der für "echo-request" (ping) steht.

```

> Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
> Ethernet II, Src: Vmware_b7:30:b4 (00:50:56:b7:30:b4), Dst: Vmware_b7:6d:d0 (00:50:56:b7:6d:d0)
> Internet Protocol Version 4, Src: 192.168.0.124, Dst: 192.168.0.123
< Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0xebdd [correct]
  [Checksum Status: Good]
  Identifier (BE): 21987 (0x55e3)
  Identifier (LE): 58197 (0xe355)
  Sequence number (BE): 1 (0x0001)
  Sequence number (LE): 256 (0x0100)
  [Response frame: 2]
  Timestamp from icmp data: Feb  1, 2019 14:14:48.000000000 CET
  [Timestamp from icmp data (relative): 0.331499894 seconds]
< Data (48 bytes)
  Data: 56c8040000000000101112131415161718191a1b1c1d1e1f...
  [Length: 48]

```

0000	00 50 56 b7 6d d0 00 50	56 b7 30 b4 08 00 45 00	PV.m..P V.0...E.
0010	00 54 fb 6b 40 00 40 01	bc f5 c0 a8 00 7c c0 a8	.T.k@.@. .... ..
0020	00 7b 08 00 eb dd 55 e3	00 01 48 46 54 5c 00 00	.{...U. .HFT\..
0030	00 00 56 c8 04 00 00 00	00 00 10 11 12 13 14 15	..V.....
0040	16 17 18 19 1a 1b 1c 1d	1e 1f 20 21 22 23 24 25	..... . !"#\$\$%
0050	26 27 28 29 2a 2b 2c 2d	2e 2f 30 31 32 33 34 35	&'()*+,- ./012345
0060	36 37		67

ICMP Typ 8 steht für "echo-request" (ping)

Vergleichen Sie diesen Eintrag mit jenem im Frame 2, dann werden Sie feststellen, dass das Layer-3-Protocol-Feld zwar selbstverständlich auch auf 1 (ICMP) steht, der ICMP-Typ aber folgerichtig 0 ist, was für "echo-reply" (ping) steht.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.0.124	192.168.0.123	ICMP	98	Echo (ping) request id=0x55e3, seq=1/256, ttl=64 (reply in 2)
2	0.000000023	192.168.0.123	192.168.0.124	ICMP	98	Echo (ping) reply id=0x55e3, seq=1/256, ttl=64 (request in 1)
3	1.009085096	192.168.0.124	192.168.0.123	ICMP	98	Echo (ping) request id=0x55e3, seq=2/512, ttl=64 (reply in 4)

```

> Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
> Ethernet II, Src: Vmware_b7:6d:d0 (00:50:56:b7:6d:d0), Dst: Vmware_b7:30:b4 (00:50:56:b7:30:b4)
> Internet Protocol Version 4, Src: 192.168.0.123, Dst: 192.168.0.124
< Internet Control Message Protocol
  Type: 0 (Echo (ping) reply)
  Code: 0
  Checksum: 0xf3dd [correct]
  [Checksum Status: Good]
  Identifier (BE): 21987 (0x55e3)
  Identifier (LE): 58197 (0xe355)
  Sequence number (BE): 1 (0x0001)
  Sequence number (LE): 256 (0x0100)
  [Request frame: 1]
  [Response time: 0,067 ms]
  Timestamp from icmp data: Feb  1, 2019 14:14:48.000000000 CET
  [Timestamp from icmp data (relative): 0.331566817 seconds]
< Data (48 bytes)
  Data: 56c8040000000000101112131415161718191a1b1c1d1e1f...
  [Length: 48]

```

0000	00 50 56 b7 30 b4 00 50	56 b7 6d d0 08 00 45 00	PV.0..P V.m...E.
0010	00 54 f0 86 00 00 40 01	07 db c0 a8 00 7b c0 a8	.T...@. ....{..
0020	00 7c 30 00 f3 dd 55 e3	00 01 48 46 54 5c 00 00	. ...U. .HFT\..
0030	00 00 56 c8 04 00 00 00	00 00 10 11 12 13 14 15	..V.....
0040	16 17 18 19 1a 1b 1c 1d	1e 1f 20 21 22 23 24 25	..... . !"#\$\$%
0050	26 27 28 29 2a 2b 2c 2d	2e 2f 30 31 32 33 34 35	&'()*+,- ./012345
0060	36 37		67

Wenn der ICMP-Typ im Layer-4-Header 0 ist, dann liegt ein echo-reply (ping) vor.

Warum allerdings die [IANA](#) den Request (echo = 8) numerisch vor den Reply (echo-reply = 0) gestellt hat, weiß wohl nur der Weihnachtsmann.

## TCP-Handshake beim Aufbau einer Sitzung untersuchen

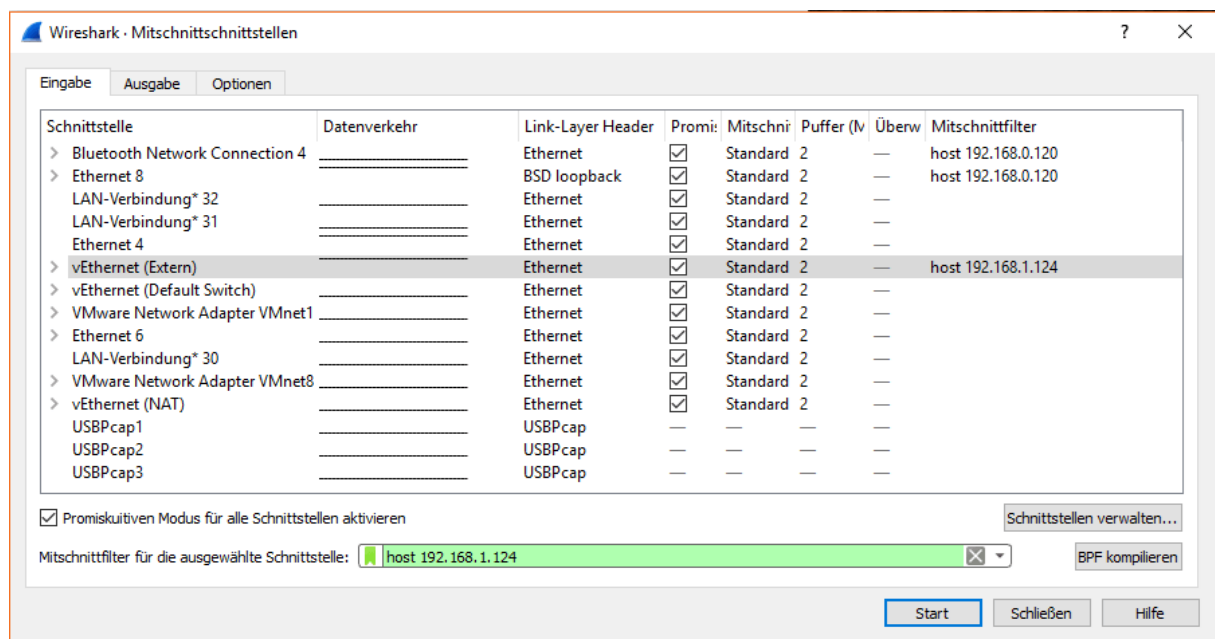
Neben UDP ist TCP das meistverwendete Protokoll der TCP/IP-Familie auf der OSI-Schicht 4. Es arbeitet verbindungsorientiert, wozu es eine Sitzung aufbaut. Dies passiert mit Hilfe eines 3-Wege-Handshakes. Die folgende Anleitung zeigt, wie man diesen Handshake für eine FTP-Kommunikation in Wireshark visualisiert.

Bei unserem Mitschnitt greifen wir mit einem FTP-Client auf einen im lokalen Netzwerk befindlichen FTP-Server zu. Der Client hat die IP-Adresse 192.168.1.200, der Server auf Basis von Ubuntu 18.04 LTS ist unter 192.168.1.124 erreichbar.

### Mitschnitt beginnen

Als Client nutzen wir Filezilla auf einer Windows-Maschine. Wir verwenden den Passiv-FTP-Mode. Dies und der Verzicht auf SFTP (FTP via TLS) muss auch am Client entsprechend konfiguriert werden. Außerdem ist darauf zu achten, dass keine Firewall-Regel auf dem Server oder Client den Verbindungsaufbau verhindert.

Das Mitschneiden in Wireshark soll auf dem Client erfolgen. Wir starten daher dort die Windows-Version und definieren im Menü *Aufzeichnen => Optionen* nach dem Markieren der gewünschten Netzwerkschnittstelle unter dem *Eingabe*-Dialog in der Zeile *Mitschnittfilter für die ausgewählte Schnittstelle* einen neuen Mitschnittfilter.



*Netzwerkschnittstelle für die Aufzeichnung der FTP-Kommunikation auswählen*

Diesmal wollen wir den Mitschnitt auf den gewünschten Ziel-Host beschränken. Das klappt mit dem Schlüsselwort *host* gefolgt von der Ziel-IP, also "host 192.168.1.124".

Dann starten wir den Mitschnitt und initiieren anschließend einen FTP-Zugriff mit Filezilla und betätigen, dass wir unverschlüsseltes FTP auf Port 21 erlauben möchten.

ftpuser@192.168.1.124 - FileZilla

Datei Bearbeiten Ansicht Übertragung Server Lesezeichen Hilfe Neue Version verfügbar!

Server: 192.168.1.124 Benutzername: ftpuser Passwort: ..... Port: Verbinden

Status: Verbindung zum Server getrennt  
 Status: Verbinde mit 192.168.1.124:21...  
 Status: Verbindung hergestellt, warte auf Willkommensnachricht...  
 Status: Unsicherer Server; er unterstützt kein FTP über TLS.  
 Status: Angemeldet  
 Status: Empfange Verzeichnisinhalt...  
 Status: Anzeigen des Verzeichnisinhalts für "/home/ftpuser" abgeschlossen

Lokal: .\Users\drilling.THOMAS-DRILLING\Eigene Dateien\ Server: /home/ftpuser

Dateiname Dateigröße Dat

..

home  
ftpuser

Dateiname	Dateigröße	Dateityp	Zuletzt geändert	Berechtigu...	Besitzer/Gr...
..					
.bash_logout	220	BASH_LOG...	04.04.2018 20:3...	adfrw (0644)	1001 1001
.bashrc	3.771	BASHRC-D...	04.04.2018 20:3...	adfrw (0644)	1001 1001
.profile	807	PROFILE-D...	04.04.2018 20:3...	adfrw (0644)	1001 1001
examples.desktop	8.980	DESKTOP-...	16.04.2018 10:1...	adfrw (0644)	1001 1001

15 Dateien und 69 Verzeichnisse. Gesamtgröße: 12.401.477 Bytes

4 Dateien. Gesamtgröße: 13.778 Bytes

Server/Lokale Datei	Richtung	Datei auf Server	Größe	Priorität	Zeitpunkt
---------------------	----------	------------------	-------	-----------	-----------

Zu übertragende Dateien (1) Fehlgeschlagene Übertragungen Erfolgreiche Übertragungen

*Verzeichnisinhalt des FTP-Users für den Mitschnitt anzeigen*

Nach dem Verbindungsaufbau zeigen wir zum Beispiel den Verzeichnisinhalt des FTP-Users an und beenden die Aufzeichnung.

[Aufzeichnung auswerten](#)

Werfen wir nun einen Blick auf den Mitschnitt.

^Ethernet (Extern) (host 192.168.1.124)

Datei Bearbeiten Ansicht Navigation Aufzeichnen Analyse Statistiken Telefonie Wireless Tools Hilfe

Anzeigefilter anwenden ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.200	192.168.1.124	TCP	54	11215 → 21 [FIN, ACK] Seq=1 Ack=1 Win=2050 Len=0
2	0.002915	192.168.1.124	192.168.1.200	TCP	60	21 → 11215 [FIN, ACK] Seq=1 Ack=2 Win=229 Len=0
3	0.003035	192.168.1.200	192.168.1.124	TCP	54	11215 → 21 [ACK] Seq=2 Ack=2 Win=2050 Len=0
4	0.013000	192.168.1.200	192.168.1.124	TCP	66	11394 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
5	0.013732	192.168.1.124	192.168.1.200	TCP	66	21 → 11394 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
6	0.013843	192.168.1.200	192.168.1.124	TCP	54	11394 → 21 [ACK] Seq=1 Ack=1 Win=525568 Len=0
7	0.305912	192.168.1.124	224.0.0.251	MDNS	85	Standard query 0x0000 A ws1.thomas-drilling.local, "QM" question
8	1.306780	192.168.1.124	224.0.0.251	MDNS	85	Standard query 0x0000 A ws1.thomas-drilling.local, "QM" question
9	3.309255	192.168.1.124	224.0.0.251	MDNS	85	Standard query 0x0000 A ws1.thomas-drilling.local, "QM" question
10	5.030754	Vmware_b7:30:b4	Elitegro_60:e6:57	ARP	60	Who has 192.168.1.200? Tell 192.168.1.124
11	5.030796	Elitegro_60:e6:57	Vmware_b7:30:b4	ARP	42	192.168.1.200 is at c8:9c:dc:60:e6:57
12	5.208846	192.168.1.124	192.168.1.200	FTP	113	Response: 220 ProFTPD 1.3.5e Server (Debian) [::ffff:192.168.1.124]
13	5.209127	192.168.1.200	192.168.1.124	FTP	64	Request: AUTH TLS
14	5.210593	192.168.1.124	192.168.1.200	TCP	60	21 → 11394 [ACK] Seq=60 Ack=11 Win=29312 Len=0
15	5.210594	192.168.1.124	192.168.1.200	FTP	79	Response: 500 AUTH not understood
16	5.210777	192.168.1.200	192.168.1.124	FTP	64	Request: AUTH SSL
17	5.211115	192.168.1.124	192.168.1.200	FTP	79	Response: 500 AUTH not understood
18	5.221081	192.168.1.200	192.168.1.124	FTP	68	Request: USER ftpuser
19	5.223909	192.168.1.124	192.168.1.200	FTP	89	Response: 331 Password required for ftpuser

> Frame 4: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0

> Ethernet II, Src: Elitegro\_60:e6:57 (c8:9c:dc:60:e6:57), Dst: Vmware\_b7:30:b4 (00:50:56:b7:30:b4)

> Internet Protocol Version 4, Src: 192.168.1.200, Dst: 192.168.1.124

> Transmission Control Protocol, Src Port: 11394, Dst Port: 21, Seq: 0, Len: 0

Source Port: 11394

Destination Port: 21

[Stream index: 1]

[TCP Segment Len: 0]

Sequence number: 0 (relative sequence number)

[Next sequence number: 0 (relative sequence number)]

Acknowledgment number: 0

1000 .... = Header Length: 32 bytes (8)

0000 00 50 56 b7 30 b4 c8 9c dc 60 e6 57 00 00 45 00 ..PV.0...W..E..

0010 00 34 21 67 40 00 80 06 00 00 c0 a8 01 c8 c0 a8 ..4lg@.....

0020 01 7c 2c 82 00 15 dc 2e f2 fb 00 00 00 80 02 .|,.....

0030 fa f0 84 bb 00 00 02 04 05 b4 01 03 08 08 01 01 .....  
0040 04 02 ..

### Ergebnis der Aufzeichnung einer FTP-Kommunikation in Wireshark

Relevant für den TCP-Handshake sind hier die 3 Zeilen mit den Flags [SYN], [SYN] und [ACK]. Diese drei Pakete werden zu Beginn jeder TCP-Sitzung ausgetauscht, wie die Abbildung weiter unten illustriert.

Zeile 1 zeigt die Verbindungsaufnahme vom Client (192.168.1.200) zum Server (192.168.1.124). FTP baut den Kontrollkanal über einen vom FTP-Client frei gewählten High-Port (hier 23913) zu Ziel-Port 21 des Servers auf.

Bereits in der Zeilendarstellung ist das SYN-Flag in eckigen Klammern zu erkennen, ebenso wie in der Detaildarstellung im mittleren Fenster, das heißt, es handelt sich um ein SYN-Paket (streng genommen "Segment") im Rahmen des TCP-3-Wege-Handshakes.





*Verlauf eines TCP-Handshakes*

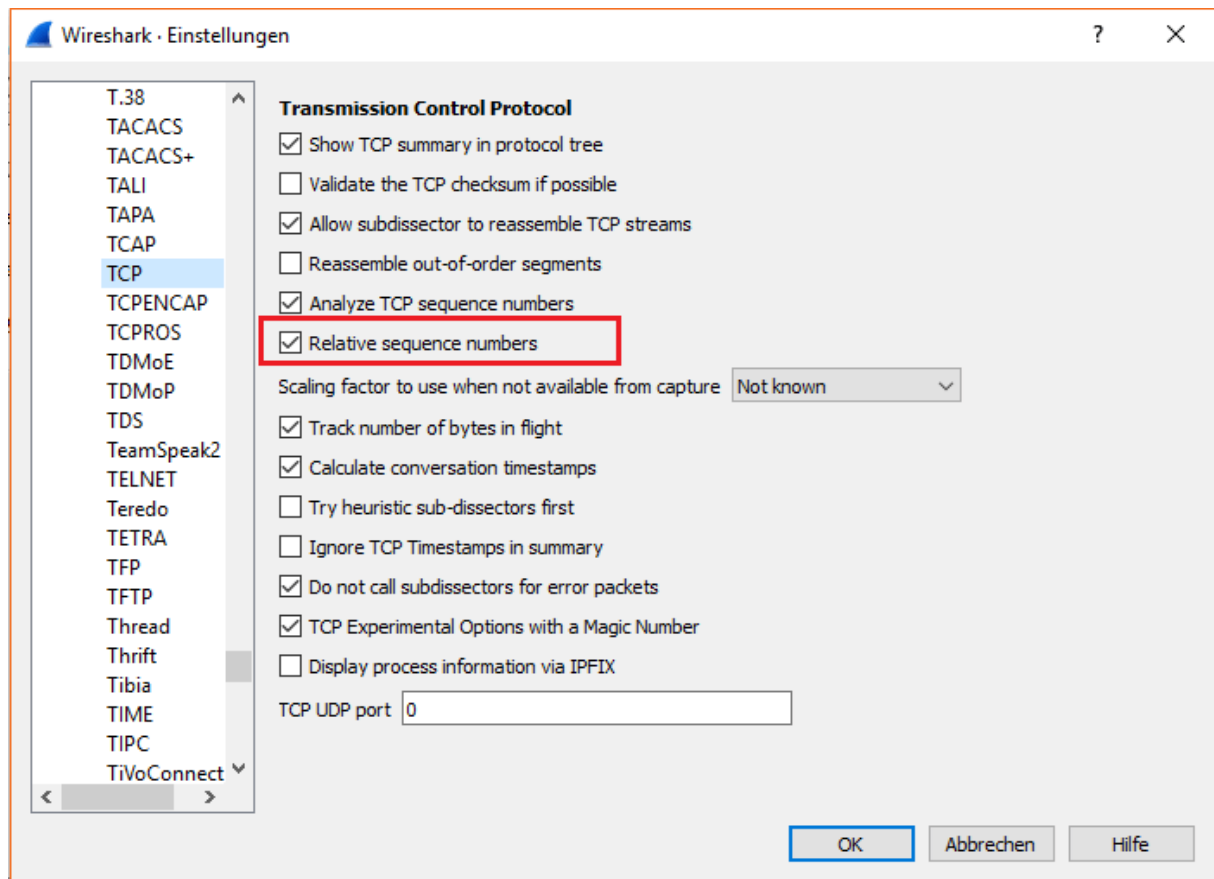
Der Client, der die Verbindung aufbauen möchte, sendet dem Server also ein SYN-Paket (für "synchronize") mit einer so genannten Sequenznummer, hier "0". Übrigens sind während der späteren Datenübertragungsphase (active open) die Rollen von Client und Server (aus TCP-Sicht) symmetrisch, sodass jeder der beiden Partner einen Verbindungsabbau initiieren kann.

Mit Hilfe der Sequenz-Nummern stellt TCP im Rahmen der Verbindungssteuerung die Vollständigkeit einer Übertragung in der korrekten Reihenfolge und ohne Duplikate sicher. Das SYN-Paket ist ein spezielles Paket, bei dem das SYN-Flag (SYN-Bit) im TCP-Header gesetzt ist.

Bei der Start-Sequenznummer handelt es sich eigentlich um eine beliebige Zahl, deren Generierung von der jeweiligen TCP-Implementierung abhängt. Zwecks nutzerfreundlicher Verfolgbarkeit einer TCP-Kommunikation verwendet Wireshark allerdings per Default so genannte relative Sequenznummern, weshalb die Kommunikation hier mit der Sequenznummer 0 beginnt.

Dies lässt sich, falls gewünscht, in der Protokollkonfiguration im Dialog *Wireshark-Einstellungen* unter *Bearbeiten => Einstellungen* im Knoten *Protocols* bei *TCP* ändern.





*Einstellung für relative TCP-Sequenznummern in Wireshark*

Die echte Sequence-Number sollte möglichst zufällig sein, um Sicherheitsrisiken zu vermeiden. Wir bleiben aber bei relativen Sequenznummern und betrachten das Paket im mittleren Fenster etwas genauer.

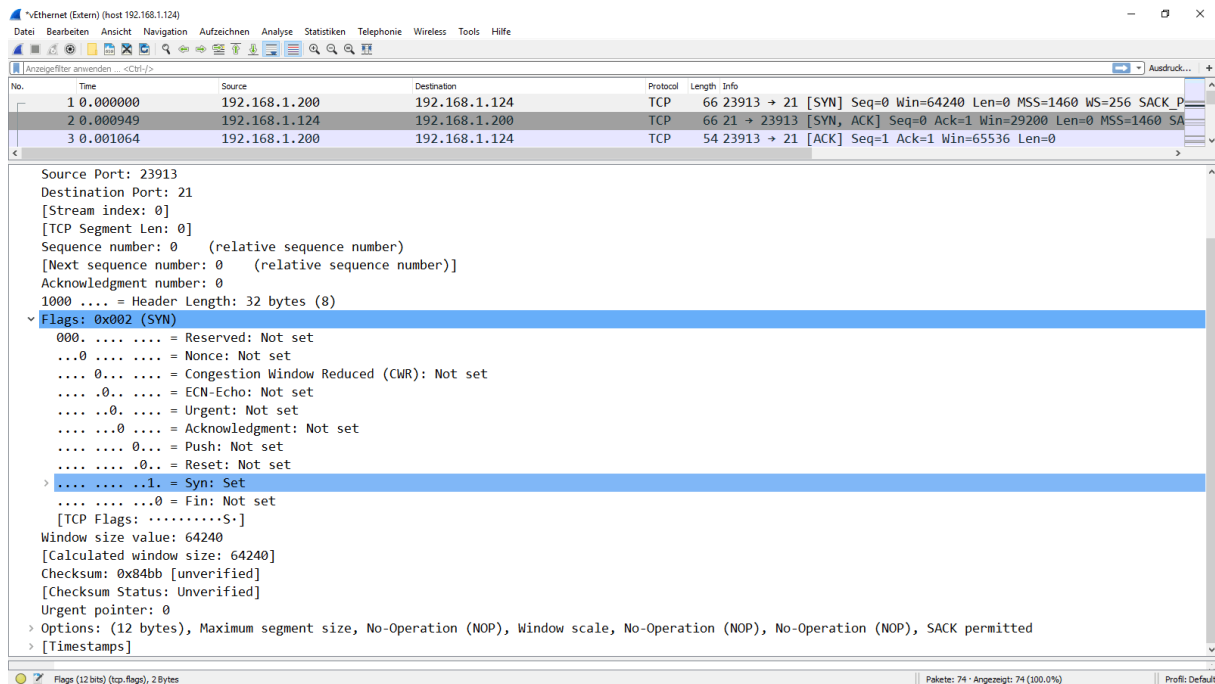
Die Darstellung beginnt mit der Zusammenfassung in der Zeile

Transmission Control Protocol, Src. Port: <Port>, Dst Port: 21, Seq: 0, Len:0

Hat man die Ansicht mit einem Klick auf den Rechtspfeil entfaltet, folgen die Zeilen:

- Source Port:
- Destination Port:
- [Stream index:]
- TCP Segment Len:
- Sequence number:
- Next sequence number:
- Acknowledgement-Number:

Daran anschließend findet sich der Knoten *Flags*, der sich ebenfalls mit einem Klick auf den Rechtspfeil entfalten lässt.



### Flags des TCP-Pakets beim Handshake

Dahinter folgen noch die so genannte *TCP-Window-Size* als Sende- bzw. Empfangspuffer, eine *Checksum*, der *Checksum Status* und der noch eingeklappte Bereich *Options*.

### Die drei Schritte des Handshakes

Im Rahmen des 3-Wege-TCP-Handshakes mit (SYN, SYN/ACK, SYN) muss im ersten Paket des Handshakes immer das SYN-Flag gesetzt sein. Ist der Port geöffnet, antwortet der Server von seinem Port (hier 23913) auf Port 21 des Clients und bestätigt den Erhalt des ersten SYN-Pakets.

Das tut er, indem er dem gewünschten Verbindungsaufbau durch Senden eines SYN/ACK-Paket zustimmt. Dabei handelt es sich ebenfalls um ein spezielles Paket, bei dem im TCP-Header sowohl das SYN-Flag als auch das ACK-Bit (Acknowledgement) gesetzt sind.

- ✓ **Flags: 0x012 (SYN, ACK)**
  - 000. .... = Reserved: Not set
  - ...0 .... = Nonce: Not set
  - .... 0... = Congestion Window Reduced (CWR): Not set
  - .... .0.. = ECN-Echo: Not set
  - .... ..0. = Urgent: Not set
  - .... ...1 = Acknowledgment: Set
  - .... .... 0... = Push: Not set
  - .... .... .0.. = Reset: Not set
  - > .... .... .1. = Syn: Set
  - .... .... ...0 = Fin: Not set
  - [TCP Flags: .....A..S.]

### Flags des SYN/ACK-Pakets beim Handshake in Wireshark

Im dritten Paket des 3-Wege-Handshakes ist dann nur das ACK-Flag gesetzt, womit der Client signalisiert, dass er die Zustimmung des Servers zur Verbindungsaufnahme registriert hat und jetzt bereit für die Übertragung ist. Damit ist die Verbindung aufgebaut und die Sitzung steht. Alles, was hierauf folgt, ist anwendungsspezifisch.

## TCP-Fehlerkorrektur beobachten

In einer TCP-Session erkennt der Empfänger anhand von Sequence Numbers, ob Pakete in der richtigen Reihenfolge bzw. mehrfach eintreffen. Mit Acknowledgments bestätigt er den korrekten Empfang oder veranlasst eine erneute Übertragung. Dieser Vorgang lässt sich in Wireshark gut nachvollziehen.

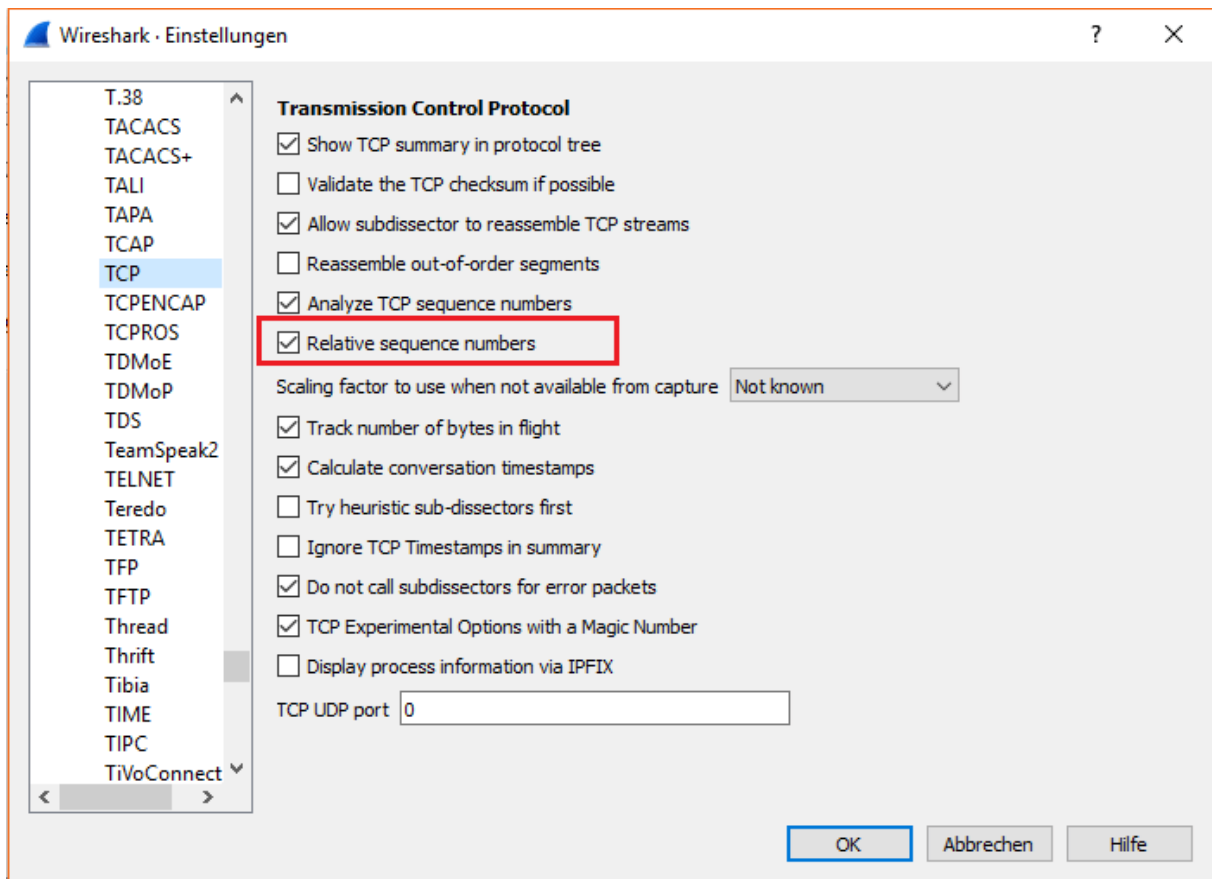
Die folgende Abbildung illustriert die Sequence- und Acknowledgment-Nummern, mit deren Hilfe TCP die Fehlerkorrektur und die Flusskontrolle realisiert.



*Verlauf eines TCP-Handshakes*

Wie weiter oben gezeigt, generiert jeder der Partner beim Aufbau einer TCP-Verbindung eine zufällige, 32-Bit lange Sequence Number. Wireshark ersetzt diese lediglich der Übersicht halber durch relative Sequenznummern beginnend mit 0, und zwar für Client und Server.

Jeder der Partner erhöht im Verlauf der Kommunikation die Sequenznummer um die Anzahl der gesendeten Bytes. Zur Analyse der Flusskontrolle kann es daher hilfreich sein, die relativen Sequenznummern zu deaktivieren.



*Einstellung für relative TCP-Sequenznummern in Wireshark*

Das sieht dann für das erste SYN-Paket so aus:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.200	192.168.1.124	TCP	66	23913 → 21 [SYN] Seq=3759279677 Win=64240 Len=0 MSS=1460 WS=2
2	0.000949	192.168.1.124	192.168.1.200	TCP	66	21 → 23913 [SYN, ACK] Seq=3842592582 Ack=3759279678 Win=29200
3	0.001064	192.168.1.200	192.168.1.124	TCP	54	23913 → 21 [ACK] Seq=3759279678 Ack=3842592583 Win=65536 Len=0

> Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0  
 > Ethernet II, Src: Elitegro\_60:e6:57 (c8:9c:dc:60:e6:57), Dst: Vmware\_b7:30:b4 (00:50:56:b7:30:b4)  
 > Internet Protocol Version 4, Src: 192.168.1.200, Dst: 192.168.1.124  
 > Transmission Control Protocol, Src Port: 23913, Dst Port: 21, Seq: 3759279677, Len: 0

Source Port: 23913  
 Destination Port: 21  
 [Stream index: 0]  
 [TCP Segment Len: 0]  
 Sequence number: 3759279677  
 [Next sequence number: 3759279677]  
 Acknowledgment number: 0  
 1000 .... = Header Length: 32 bytes (8)

> **Flags: 0x002 (SYN)**  
 Window size value: 64240  
 [Calculated window size: 64240]  
 Checksum: 0x84bb [unverified]  
 [Checksum Status: Unverified]  
 Urgent pointer: 0  
 > Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted  
 > [Timestamps]

*Der Client startet mit der zufälligen Sequenz-Nummer 3759279677. Die eigene Acknowledgement-Nummer ist noch 0.*

Dabei vermerkt jeder der Partner in der Zeile "[Next sequence number:]" eine Folge-Sequenz-Nummer. Hierbei handelt es sich immer um genau diejenige Acknowledgment-Nummer, die als nächste von der Gegenstelle erwartet wird.

In der Abbildung oben ist die zufällige Start-Sequenz-Nummer des Clients 3759279677. Die eigene Acknowledgement-Nummer ist noch 0, da der Client noch keine anderen Informationen vom Server erhalten hat.

Die *Next-Sequence-Number* ist identisch mit der eigenen Sequence Number 3759279677, da noch keine Daten vom Client gesendet wurden. Werfen wir nun einen Blick in das zweite Paket:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.200	192.168.1.124	TCP	66	23913 → 21 [SYN] Seq=3759279677 Win=64240 Len=0 MSS=1460 WS=2
2	0.000949	192.168.1.124	192.168.1.200	TCP	66	21 → 23913 [SYN, ACK] Seq=3842592582 Ack=3759279678 Win=29200
3	0.001064	192.168.1.200	192.168.1.124	TCP	54	23913 → 21 [ACK] Seq=3759279678 Ack=3842592583 Win=65536 Len=0

> Frame 2: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0

> Ethernet II, Src: Vmware\_b7:30:b4 (00:50:56:b7:30:b4), Dst: Elitegro\_60:e6:57 (c8:9c:dc:60:e6:57)

> Internet Protocol Version 4, Src: 192.168.1.124, Dst: 192.168.1.200

> Transmission Control Protocol, Src Port: 21, Dst Port: 23913, Seq: 3842592582, Ack: 3759279678, Len: 0

Source Port: 21

Destination Port: 23913

[Stream index: 0]

[TCP Segment Len: 0]

Sequence number: 3842592582

[Next sequence number: 3842592582]

Acknowledgment number: 3759279678

1000 .... = Header Length: 32 bytes (8)

> Flags: 0x012 (SYN, ACK)

Window size value: 29200

[Calculated window size: 29200]

Checksum: 0xf83c [unverified]

[Checksum Status: Unverified]

Urgent pointer: 0

> Options: (12 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted, No-Operation (NOP), Window scale

> [SEQ/ACK analysis]

> [Timestamps]

Die Acknowledgement Number berechnet sich aus der Sequence Number der Gegenstelle.

Die beliebige Start-Sequenz-Nummer des Servers ist 3842592582. Demnach ist der Wert für Next-Sequence-Number ebenfalls 3842592582, da von hier ebenfalls noch keine Daten gesendet wurden.

Die Acknowledgement-Nummer muss allerdings

$$3759279677 + 1 = 3759279678$$

sein, weil diese im Rahmen des TCP-Handshakes immer einfach um 1 erhöht wird. Beim Handshake selbst werden keine weiteren Daten gesendet.

Werfen wir noch einen Blick auf das dritte Paket des Handshakes.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.200	192.168.1.124	TCP	66	23913 → 21 [SYN] Seq=3759279677 Win=64240 Len=0 MSS=1460 WS=2
2	0.000949	192.168.1.124	192.168.1.200	TCP	66	21 → 23913 [SYN, ACK] Seq=3842592582 Ack=3759279678 Win=29200
3	0.001064	192.168.1.200	192.168.1.124	TCP	54	23913 → 21 [ACK] Seq=3759279678 Ack=3842592583 Win=65536 Len=0

> Frame 3: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0

> Ethernet II, Src: Elitegro\_60:e6:57 (c8:9c:dc:60:e6:57), Dst: Vmware\_b7:30:b4 (00:50:56:b7:30:b4)

> Internet Protocol Version 4, Src: 192.168.1.200, Dst: 192.168.1.124

> Transmission Control Protocol, Src Port: 23913, Dst Port: 21, Seq: 3759279678, Ack: 3842592583, Len: 0

Source Port: 23913

Destination Port: 21

[Stream index: 0]

[TCP Segment Len: 0]

Sequence number: 3759279678

[Next sequence number: 3759279678]

Acknowledgment number: 3842592583

0101 .... = Header Length: 20 bytes (5)

> Flags: 0x010 (ACK)

Window size value: 256

[Calculated window size: 65536]

[Window size scaling factor: 256]

Checksum: 0x84af [unverified]

[Checksum Status: Unverified]

Urgent pointer: 0

> [SEQ/ACK analysis]

> [Timestamps]

Das dritte Paket eines TCP-Handshakes

Hier ist die Sequence-Number des Clients nun 3759279678, die Next-Sequence-Number wieder 3759279678 (denn auch der Server hat ja im Rahmen des Handshakes noch keine Daten gesendet) und die Acknowledgment-Number nun

$$842592582 + 1 = 842592583$$

Dabei handelt es sich um eine beim TCP-Three-Way-Handshake um 1 erhöhte Sequenz-Nummer.

## FTP unter der Lupe

Als Basis für diese Analyse dient eine FTP-Sitzung, bei der wir bewusst auf den verschlüsselten Modus ohne TLS oder SSH/Public-Key verzichten. Dies gibt uns Gelegenheit zu demonstrieren, wie einfach Hacker mit Wireshark solche Informationen abfangen und beispielsweise an das FTP-Passwort gelangen können.

Bevor wir uns mit zweckgebundenen Analysen befassen, bleiben wir vorerst noch bei der TCP-Flusssteuerung, diesmal aber im Rahmen der Datenübertragung. Da nun auch Daten transferiert werden, wird die Berechnung der Sequence- und Acknowledgment-Nummern mit realen 32-Bit-Werten aufwändiger.

### Berechnung von SEQ und ACK

Wer deshalb Wireshark wieder auf relative Sequenznummern zurücksetzt, darf trotzdem nicht vergessen, dass Client und Server in Wahrheit mit zufälligen und daher nicht identischen Sequenznummern starten.

Werfen wir jetzt ein Blick auf die weitere Entwicklung der Sequenznummer in Rahmen der Datenübertragung. Grundsätzlich gilt, dass sich die vom Client verwendete Sequence-Number zusammensetzt aus der vorherigen Sequence-Number plus der Anzahl der vom Client im entsprechenden Abschnitt versendeten Daten-Bytes., also

$$\text{SEQ Client} = \text{alte SEQ Client} + \text{Anzahl Datenbytes Client}$$

Die Acknowledgment-Number des Clients errechnet sich dagegen aus der Sequence-Number vom Server plus der Anzahl der Daten-Bytes, die der Server versendet hat.

$$\text{ACK Client} = \text{SEQ Server} + \text{Anzahl Datenbytes Server}$$

Es handelt sich also bei der ACK-Nummer also um eine Art Prüfsumme, mit welcher der Server herausfindet, ob der Client tatsächlich alle vom Server gesendeten Daten empfangen hat.

### FTP-Mitschnitt untersuchen

Überprüfen wir diesen Sachverhalt anhand des FTP-Mitschnitts und betrachten dazu die Zeilen 14 bis 17.

Beginnen wir mit Zeile 14. Sie fragen sich, was davor passiert ist, also im Rahmen der Pakete 4 bis 13? Hier haben Client und Server erst einmal versucht, eine TLS-Verbindung auszuhandeln, was aber aufgrund der vorbereiteten Konfiguration von proftpd und Filezilla scheitern musste.

Da wir den Mitschnittfilter zudem nur auf den Ziel-Host 192.168.1.24 und nicht explizit auf das FTP-Protokoll beschränkt hatten, konnten sich zudem noch einige andere Pakete dazwischenmogeln. Im Vergleich zum bloßen Handshake ist schon in der zusammenfassenden Zeilenansicht zu erkennen, dass das "Protocol" nun FTP ist.



Wireshark v2.10.12 (64-bit) - vEthernet (Extern) (host 192.168.1.124)

Datei Bearbeiten Ansicht Navigation Aufzeichnen Analyse Statistiken Telephonie Wireless Tools Hilfe

Anzeigefilter anwenden ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
14	5.019781	192.168.1.200	192.168.1.124	FTP	68	Request: USER ftpuser
15	5.020843	192.168.1.124	192.168.1.200	FTP	89	Response: 331 Password required for ftpuser
16	5.021047	192.168.1.200	192.168.1.124	FTP	68	Request: PASS ftpuser
17	5.049521	192.168.1.124	192.168.1.200	FTP	82	Response: 230 User ftpuser logged in
18	5.049777	192.168.1.200	192.168.1.124	FTP	68	Request: OPTS UTF8 ON

<

> Frame 14: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0  
 > Ethernet II, Src: Elitegro\_60:e6:57 (c8:9c:dc:60:e6:57), Dst: Vmware\_b7:30:b4 (00:50:56:b7:30:b4)  
 > Internet Protocol Version 4, Src: 192.168.1.200, Dst: 192.168.1.124  
 > Transmission Control Protocol, Src Port: 23913, Dst Port: 21, Seq: 3759279698, Ack: 3842592692, Len: 14  
     Source Port: 23913  
     Destination Port: 21  
     [Stream index: 0]  
     [TCP Segment Len: 14]  
     Sequence number: 3759279698  
     [Next sequence number: 3759279712]  
     Acknowledgment number: 3842592692  
     001 .... = Header Length: 20 bytes (5)  
     > Flags: 0x018 (PSH, ACK)  
     Window size value: 256  
     [Calculated window size: 65536]  
     [Window size scaling factor: 256]  
     Checksum: 0x84bd [unverified]  
     [Checksum Status: Unverified]  
     Urgent pointer: 0  
     > [SEQ/ACK analysis]  
     > [Timestamps]  
     TCP payload (14 bytes)  
     > File Transfer Protocol (FTP)  
     [Current working directory: ]

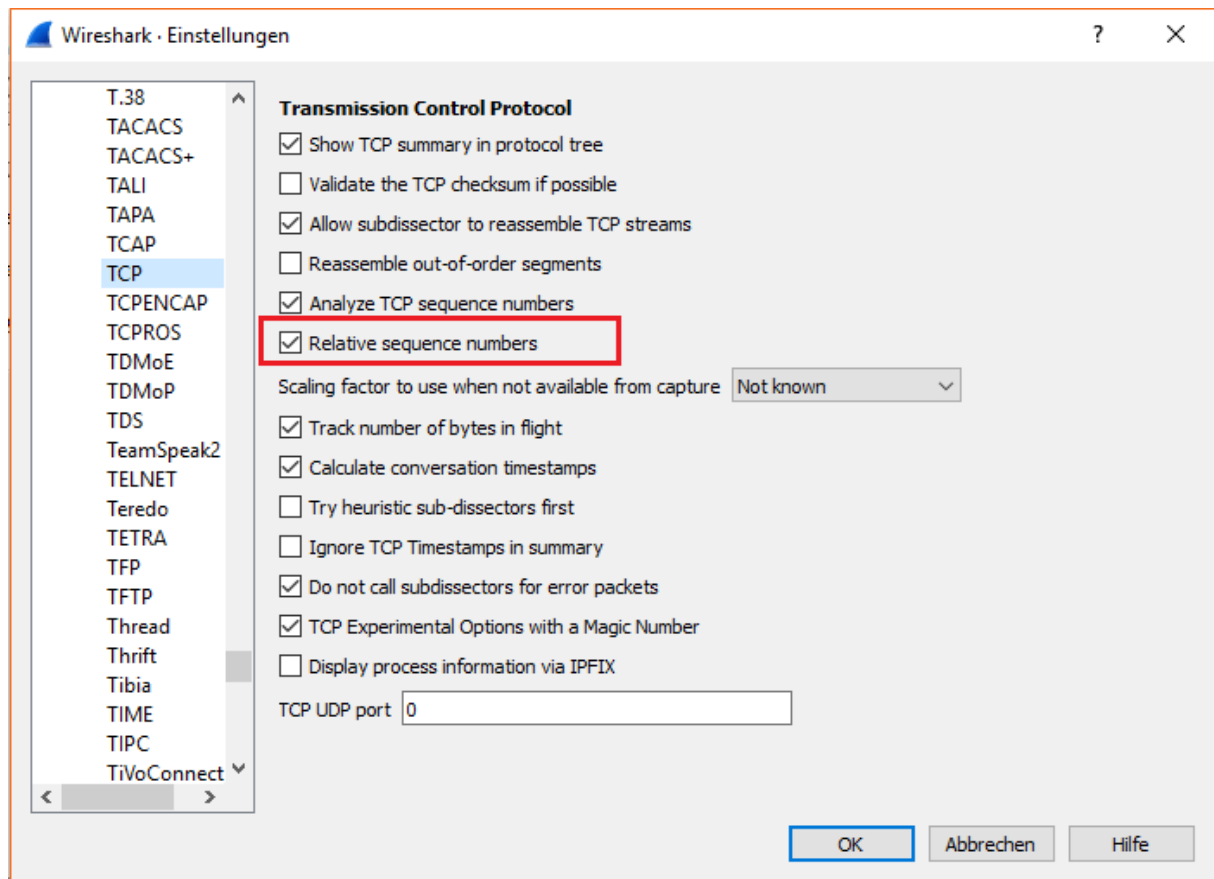
0000 00 50 56 b7 30 b4 c8 9c dc 60 e6 57 08 00 45 00 ·PV· ······w·E·  
 wireshark\_vEthernet (Extern)\_20190304134518\_a23144.pcapng | Paketes: 74 · Angezeigt: 74 (100.0%)

Im FTP-Mitschnitt ist der Name des Users erkennbar

In Zeile 14 übermittelt der Client dem Server dann den gewünschten User-Namen für die Anmeldung: "USER ftpuser". Im Detail-Fenster ist außerdem zu erkennen, dass die Sequenznummer des Clients inzwischen 3759279698 ist.

Die "Next sequence Number" hingegen ist 3842592692, da der Client 14 Byte übertragen hat, was das Len-Feld verrät.

Wir können die Länge durch Subtrahieren der beiden Sequenz-Nummern überprüfen, schalten dazu aber wieder auf relative Sequenznummern um und rechnen  $35-21=14$ , was exakt dem Feld "Len:" für die Länge der Nutzdaten entspricht.



*Einstellung für relative TCP-Sequenznummern in Wireshark*

Diese Anzeige taucht noch einmal auf in der letzten Zeile des Detailfensters bei "TCP payload (14 bytes)" auf. Markiert man im mittleren Teil an genau dieser Stelle den Payload-Teil, dann werden die Nutzdaten auch unten in der hexadezimalen Rohdatenansicht angezeigt und tatsächlich ist hier, wie zu erwarten, das übergebene Passwort "ftpuser" zu erkennen.

~Ethernet (Extern) (Host 192.168.1.124)

Datei Bearbeiten Ansicht Navigation Aufzeichnen Analyse Statistiken Telefonie Wireless Tools Hilfe

Anzeigefilter anwenden... <Ctrl>F

No.	Time	Source	Destination	Protocol	Length	Info
10	5.010804	192.168.1.124	192.168.1.200	TCP	60	21 → 23913 [ACK] Seq=60 Ack=11 Win=29312 Len=0
11	5.010805	192.168.1.124	192.168.1.200	FTP	79	Response: 500 AUTH not understood
12	5.010953	192.168.1.200	192.168.1.124	FTP	64	Request: AUTH SSL
13	5.011234	192.168.1.124	192.168.1.200	FTP	79	Response: 500 AUTH not understood
14	5.019781	192.168.1.200	192.168.1.124	FTP	68	Request: USER ftpuser

Source Port: 23913  
Destination Port: 21  
[Stream index: 0]  
[TCP Segment Len: 14]  
Sequence number: 21 (relative sequence number)  
[Next sequence number: 35 (relative sequence number)]  
Acknowledgment number: 110 (relative ack number)  
0101 .... = Header Length: 20 bytes (5)  
> Flags: 0x018 (PSH, ACK)  
Window size value: 256  
[Calculated window size: 65536]  
[Window size scaling factor: 256]  
Checksum: 0x84bd [unverified]  
[Checksum Status: Unverified]  
Urgent pointer: 0  
> [SEQ/ACK analysis]  
> [Timestamps]  
TCP payload (14 bytes)  
> File Transfer Protocol (FTP)

```

0000  00 50 56 b7 30 b4 c8 9c dc 60 e6 57 08 00 45 00  ·PV·0· · · · ·W· ·E·
0010  00 36 4e 42 40 00 80 06 00 00 c0 a8 01 c8 c0 a8  ·6NB@· · · · ·
0020  01 7c 5d 69 00 15 e0 12 0e 52 e5 09 4f b4 50 18  ·|i· · · · ·R· ·O·P·
0030  01 00 84 bd 00 00 55 53 45 52 20 66 74 70 75 73  · · · · ·US ER ftpus
0040  65 72 0d 0a  ·er· · · · ·

```

The TCP payload of this packet (tcp.payload), 14 Bytes

Pakete: 74 · Anzeigert: 74 (100.0%) · Vervorfe

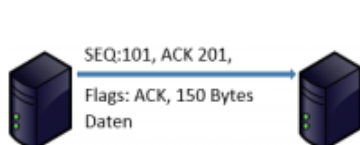
Aus der unverschlüsselten Verbindung lässt sich das FTP-Passwort auslesen.

## TCP-Flusssteuerung als Schema

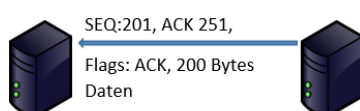
Hier das Prüfsummen-Szenario noch einmal in einer Schema-Ansicht mit angenommenen relativen Sequenznummern. Da wir uns jetzt nicht mehr in der Phase des Verbindungsaufbaus befinden, reden wir auch nicht mehr explizit von Client und Server, sondern von Host A und B. Der linke Host A startet mit SEQ 101, der rechte Host B mit 201.

Noch mal zur Erinnerung: Grundsätzlich ist

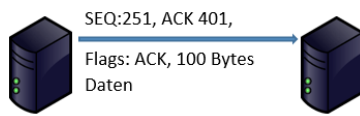
- die SEQ-Nummer, die Host A nutzt, immer die vorherige SEQ-Nummer plus die Anzahl der versendeten Daten-Bytes
- und die ACK-Nummer von Host A die SEQ-Nummer von B plus die Anzahl der von B versendeten Daten-Bytes.



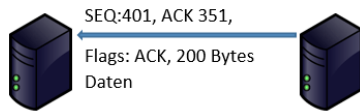
Host A setzt seine SEQ auf 101, seine ACK auf 201, also der Wert, den er als nächsten vom Host B als SEQ erwartet, und sendet dann 100 Byte Daten. Zudem ist das ACK-Flag gesetzt, was Host B signalisiert, dass nun ein Wert kommt, den B überprüfen soll (Prüfsumme).



Host B verwendet nun, wie von A erwartet, die SEQ 201, weil dies seine aktuelle SEQ ist. Zudem muss B nun seine ACK so setzen, dass sie die nächste erwartete SEQ von A enthält, also nach der Formel vorherige SEQ von A + Anzahl der von A gesendeten Datenbytes. Das sind  $101 + 150 = 251$ . Das ist wiederum der Wert, den Host B als nächstes von Host A als SEQ erwartet. Host B sendet nun 200 Bytes Daten.



Dass Host B 200 Bytes Daten versendet, bedeutet, dass Host A nun als ACK die alte SEQ von B + die Anzahl der versendeten Bytes, also  $201+200=401$  verwendet. Die SEQ, die A nun verwendet, ist genau die, die er vorher als ACK vom Host B zurückgespiegelt bekommen hat, also 251, da ja Host B die 100 Bytes Daten korrekt erhalten hat. Nun sendet A seinerseits wieder 100 Bytes Daten.



Das wiederum hat zur Folge, dass Host B nun eine ACK von 251 (SEQ A) + 100 Byte = 351 verwendet. Da er seinerseits vorher 200 Byte Daten versendet hat, setzt er seine neue SEQ auf SEQ B (alt) + 200 =  $201 + 200 = 401$  und versendet wieder 200 Bytes an Daten.

## Mitschnittfilter versus Anzeigefilter

Mittschnitt- bzw. Capture-Filter werden in Wireshark primär verwendet, um die Größe einer Paketerfassung zu reduzieren, sind aber weniger flexibel. Anzeigefilter dagegen blenden im Anschluss an einen (vollständigen) Mitschnitt bestimmte Pakete wieder aus. Dieser Abschnitt zeigt, wie man diese Filtertypen nutzt.

Grundsätzlich handelt es sich bei Mitschnittfiltern um eine Art Server-seitiges Filtern, man zeichnet also von vorne herein nur diejenigen Pakete einer Kommunikation auf, für die man sich interessiert. Der Vorteil ist, dass sich die Menge der Daten gezielt eingrenzen und so erheblich reduzieren lässt.

### Weniger Daten, geringere Flexibilität

Der Nachteil besteht darin, dass alles, was nicht mitgeschnitten wurde, nachträglich nicht mehr zugänglich ist, weil die Daten nicht aufgezeichnet wurden. Außerdem weisen Mitschnittfilter einige Einschränkungen hinsichtlich der Formulierbarkeit bestimmter Ausdrücke auf.

Die Capture-Filter finden sich im Menü *Aufzeichnen => Optionen*. Dort markiert man im oberen Bereich das gewünschte Interface und formuliert am Fuß des Dialoges bei *Mitschnittfilter für die ausgewählte Schnittstelle* den gewünschten Filter gemäß der BPF-Syntax. BPF steht dabei für *Berkeley Paket Filter* oder kurz *Berkeley Filter*.

### Berkeley Filter

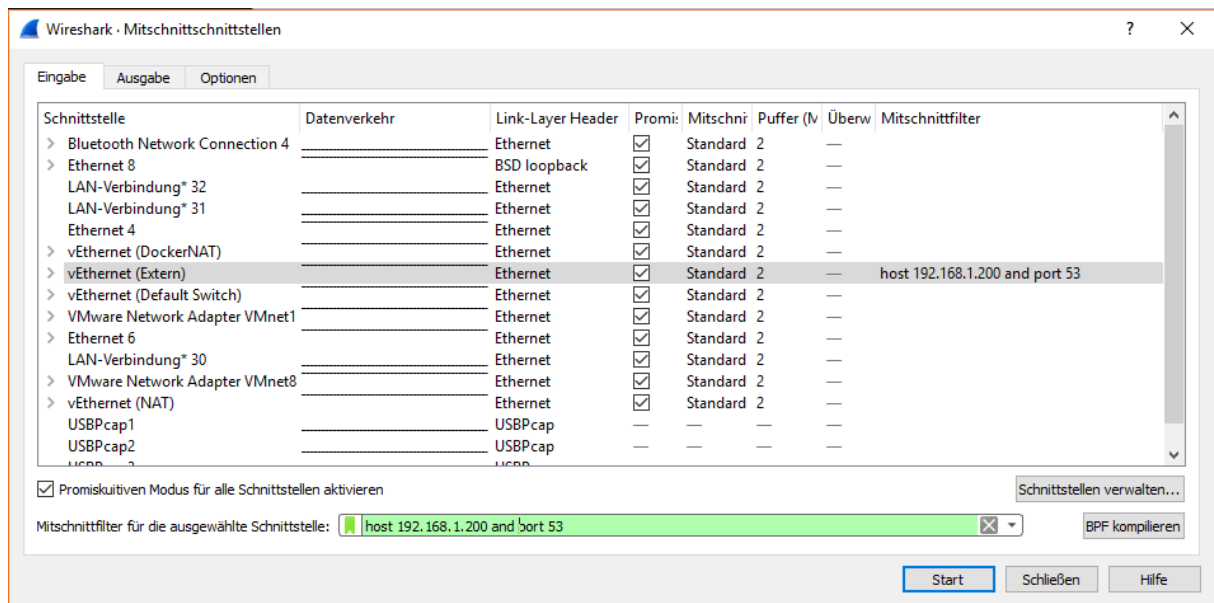
Die Funktion der Berkeley Filter ist als Interpreter in Maschinensprache für die BPF-VM implementiert. So können Anwendungen auf einfache Weise Daten aus dem Paket lesen, arithmetische Operationen darauf ausführen, das Ergebnis mit der Filterdefinition vergleichen und das Paket dann akzeptieren oder verwerfen.

Erstmals mit WinPcap wird zur Leistungsverbesserung auch eine Just-in-time-Kompilierung unterstützt. Seit diese auch für Linux funktioniert, hat sich BPF in der Unix-Welt zu einer universalen virtuellen Maschine im Kernel entwickelt, so dass BPF sogar offizielles Back-End für LLVM ist.

Unter dem Strich verwendet Wireshark also für Capture-Filter dieselbe Syntax wie tcpdump, WinDump, Analyzer und jedes andere Programm, das die Libpcap oder WinPcap-Libs verwendet.

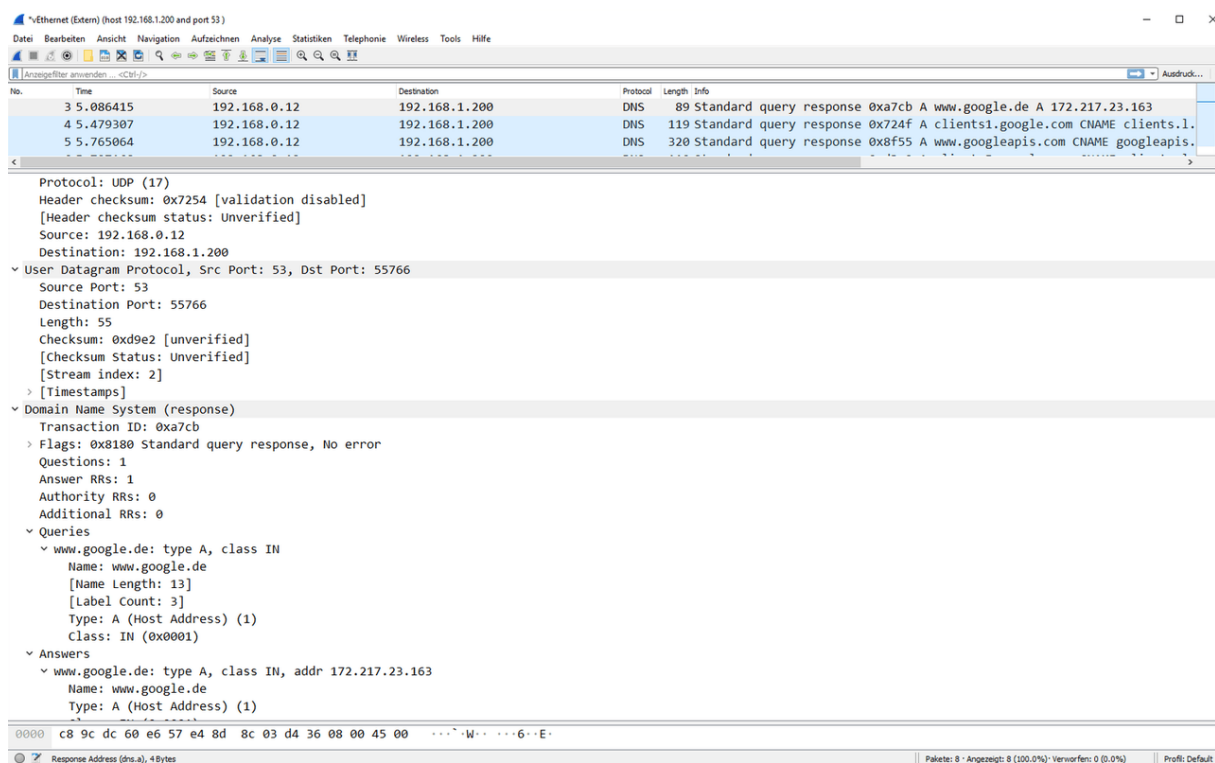
### Mitschnittfilter für DNS

Im folgenden Beispiel filtern wir die DNS-Kommunikation des Clients 192.168.1.200 beim Aufruf einer Webseite (google.de).



### Wireshark-Mitschnittfilter für die DNS-Kommunikation

Hier liefert der DNS-Server im lokalen Netz (192.168.0.12) über UDP Port 53 (DNS-Anfrage/Antwort) den A-Record-Eintrag für `www.google.de` mit der IP-Adresse 172.217.23.163 an den anfragenden Client (192.168.1.200). Der Source-Port des DNS-Servers ist wie erwartet 53.



### Ergebnis der DNS-Aufzeichnung auf Basis eines Capture-Filters

#### Anzeigefilter

Anzeigefilter werden im Anschluss an einen bereits erfolgten (ggf. nicht gefilterten) Mitschnitt nach dem Muster `TCP-Port == 80` formuliert. Dies geschieht in der Eingabezeile direkt unterhalb der Werkzeugleiste.

Hacker, denen es etwa gelingt, ein bestimmtes Interface aufzuzeichnen, werden eher einen vollständigen Mitschnitt bevorzugen, diesen speichern, entwenden und im eigenen Wireshark neu laden, um in aller Ruhe mit Anzeigefiltern experimentieren zu können.

Funktionen zum Speichern und Exportieren bzw. Laden und Importieren von ganzen Mitschnitten oder speziellen Paketen (so genannten Paket-Dissektionen) finden sich im Menü *Datei*.

Wir gehen in der Folge davon aus, dass wir den Mitschnitt aus dem obigen DNS-Beispiel zur Verfügung haben und darauf nun jederzeit nachträglich Anzeigefilter anwenden können.

### Nach IP-Adresse filtern

Mit

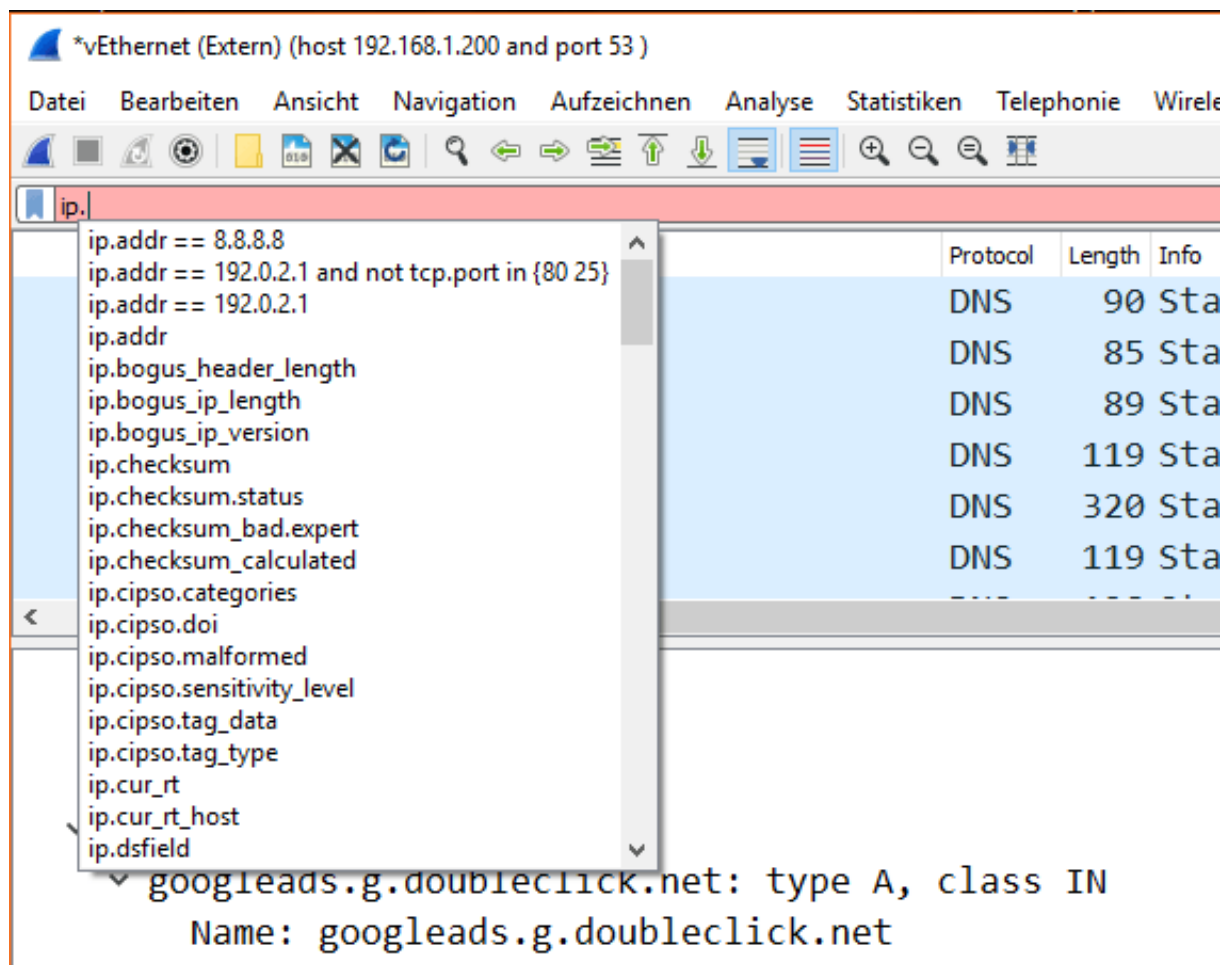
```
ip.addr == 8.8.8.8
```

reduzieren wir die Anzeige auf alle Pakete im Mitschnitt, die entweder die IP-Adresse 8.8.8.8 als Absender oder als Zieladresse verwenden. Identisch mit der Verwendung eines Mitschnittfilters ist, dass die Anzeige beim Formulieren eines validen Filterausdrucks grün wird.

In unserem Beispiel erwischen wir damit genau ein Paket, nämlich die DNS-Antwort des Google-DNS-Servers (8.8.8.8) im Rahmen einer UDP-basierten DNS-Abfrage durch den Client (192.168.1.200) nach der IP-Adresse (A-Record) von *googleleads.g.doubleclick.net*.

### Filterhierarchien

Gehen wir einige weitere Anzeigefilter durch: Beginnt man mit dem Formulieren eines Filters, dann unterstützt der Eingabeassistent den User mit der Anzeige der möglichen Kontexte in verschiedenen Ebenen, jeweils getrennt durch einen Punkt, ähnlich wie bei der objektorientierten Programmierung.

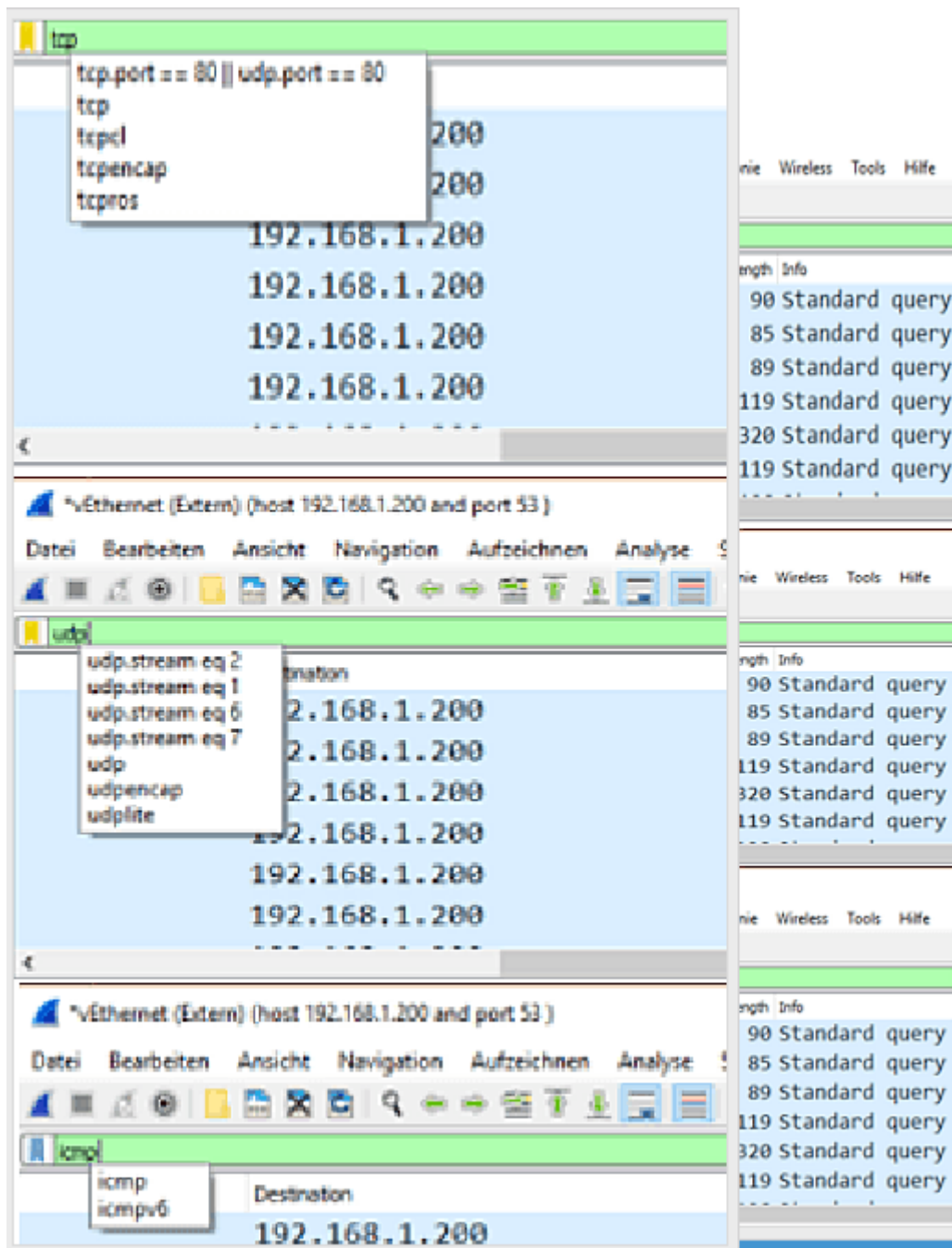


Wireshark bietet bei der Formulierung von Anzeigefiltern eine Type-ahead-Unterstützung.

Wir reden also von einem Filterobjekt mit dem Namen "ip", das wiederum eine Reihe von "Methoden" (im Prinzip sind das Unterfilter) kennt.

Fahren wir damit fort, beim Filtern andere Protokolle anzuwenden, wie "tcp", "udp" oder "icmp", ähnlich wie wir es bereits bei den Mitschnittfiltern getan haben.





Die von Wireshark angezeigten Subfilter hängen vom gewählten Protokoll ab.

## Protokollfilter

Beim Filtern nach Protokollen besteht also offenbar kein Unterschied zum Mitschnittfilter. Allerdings können Mitschnittfilter nicht auf Anwendungsprotokolle filtern. Das Eingrenzen etwa auf HTTP (Layer 7) ist dort nur indirekt durch Angabe der Ports (80, 443 etc.) möglich. Bei den Anzeigefiltern hingegen können wir statt Port 53 auch direkt nach der DNS-Kommunikation filtern, indem wir "DNS" eingeben.

The screenshot shows the Wireshark network protocol analyzer interface. The top pane, titled 'dns', lists several captured packets. A red arrow points to the first packet. The middle pane displays the details of the selected packet (Stream index: 7), showing the Domain Name System (response) structure, including transaction ID, flags, and query details for 'googleads.g.doubleclick.net'. The bottom pane shows the raw packet data in hexadecimal and ASCII format.

### Anzeigefilter für die DNS-Kommunikation

Wir sehen jetzt nur noch DNS-Pakete, allerdings sowohl TCP- als auch UDP-Pakete. Die weit verbreitete Annahme, die DNS-Kommunikation basiere nur auf UDP gilt nur für DNS-Anfragen und Antworten zwischen Client und Server. Der Austausch zwischen DNS-Servern untereinander - etwa bei Zonen-Transfers - findet via TCP-Port 53 statt.

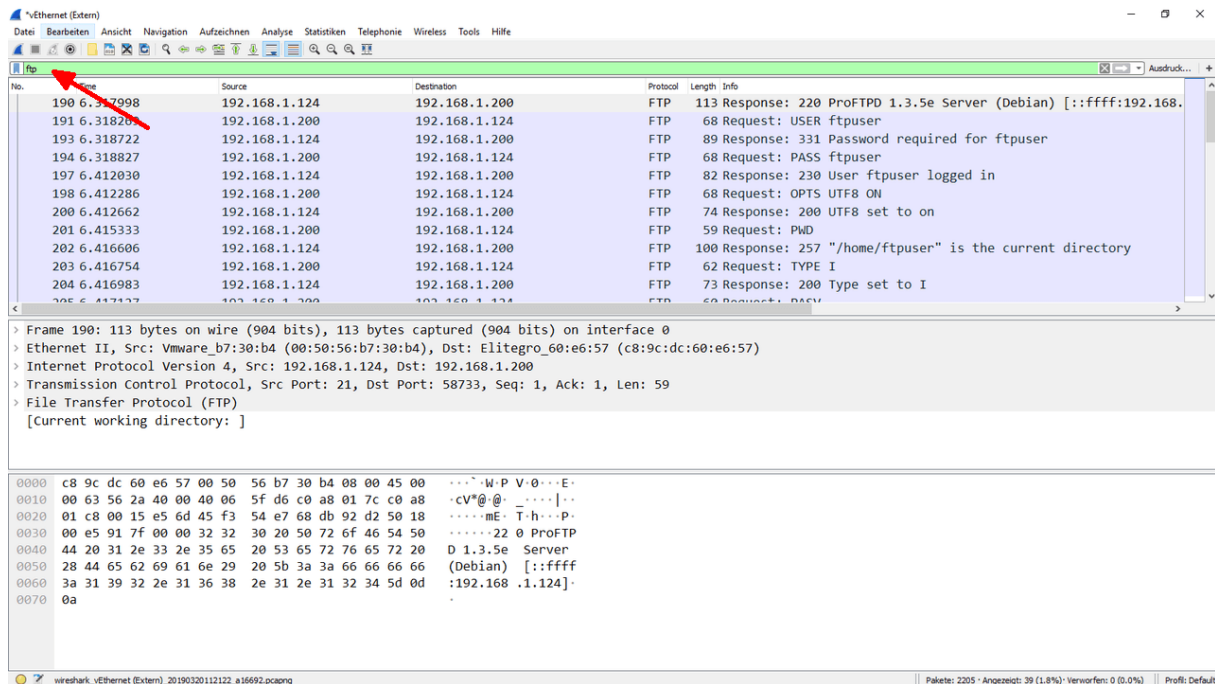
Wir können jetzt auch direkt nach HTTP filtern, würden dann aber tatsächlich nur Pakete sehen, die HTTP verwenden und nicht etwa die gesamte HTTP-Session im Browser mitsamt ACK-Paketen und dem TCP-Handshake. Auch OCSP-Pakete im Rahmen der Zertifikatsbehandlung würden dann nicht erfasst.

### Anzeigefilter für FTP

Für folgendes Beispiel starten wir eine ungefilterte Aufzeichnung einer FTP-Kommunikation. Dafür bauen wir eine nicht SSL-gesicherte FTP-Sitzung zum FTP-Server 192.168.1.124 (Ubuntu) vom Windows-Client 192.168.1.200 auf, führen einen Datei-Upload mittels Filezilla durch und beenden die Aufzeichnung.

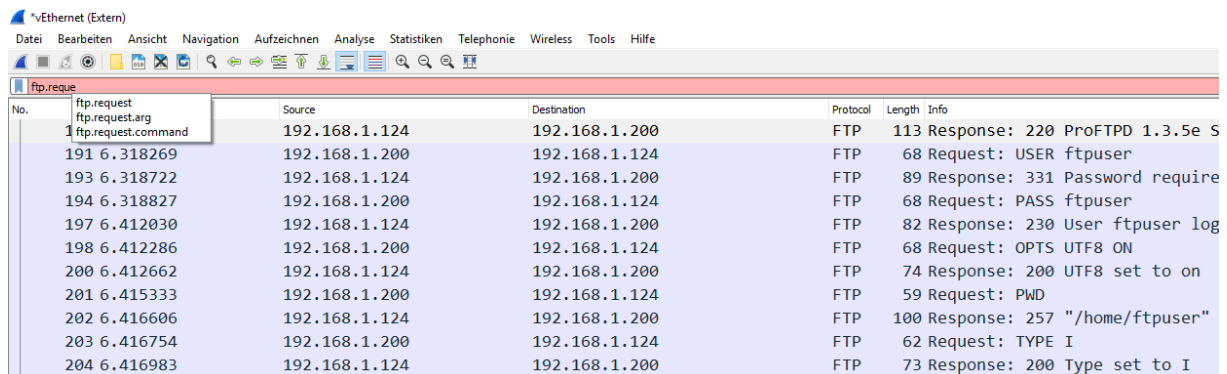
Je nachdem, wie viel andere Partner in dem entsprechenden Netzwerksegment aktiv sind und wie lange man für die beschriebenen Aktionen benötigt, kann die Aufzeichnung mehr oder weniger Pakete enthalten, in unserem Fall 2205 Stück.

Nun können wir gezielt Anzeigefilter verwenden. Bei einem Mitschnittfilter wären wir dabei auf die Port-Ebene (21 für die Control-Port, 20 für die Data-Ports der einzelnen Übertragungs-Sitzungen) beschränkt. Jetzt können wir aber gezielt auf das Anwendungsprotokoll FTP filtern.



### Anzeigefilter für die FTP-Kommunikation

Wir sehen hier alle Pakete, bei denen das Anwendungsprotokoll "FTP" beteiligt ist. Die Anzeige lässt sich aber nun über die geschilderte objektähnliche Hierarchie weiter verfeinern, indem wir mit "ftp.request" nur nach FTP-Paketen fahnden, die einen "Request" repräsentieren.



### Einschränken des Anzeigefilters auf FTP-Requests

Das klappt nicht nur auf der Anwendungs-, sondern auch auf der Netzwerkebene. So filtert

```
tcp.flags.syn == 1
```

nach allen TCP-Paketen, bei denen das SYN-Flag gesetzt ist, also nach der jeweils ersten Antwort beim TCP-Handshake.

**\*vEthernet (Extern)**

Datei Bearbeiten Ansicht Navigation Aufzeichnen Analyse Statistiken Telefonie Wireless Tools Hilfe

tcp.flags.syn == 1

No.	Time	Source	Destination	Protocol	Length	Info
88	1.312530	192.168.1.200	192.168.1.124	TCP	66	58733 → 21 [SYN] Seq=0 Win=6424
89	1.312997	192.168.1.124	192.168.1.200	TCP	66	21 → 58733 [SYN, ACK] Seq=0 Ack
98	1.545231	200.29.32.231	192.168.1.200	TCP	66	60539 → 3389 [SYN, ECN, CWR] Se
99	1.545361	192.168.1.200	200.29.32.231	TCP	66	3389 → 60539 [SYN, ACK] Seq=0 A
208	6.418151	192.168.1.200	192.168.1.124	TCP	66	58734 → 33271 [SYN] Seq=0 Win=6
209	6.418600	192.168.1.124	192.168.1.200	TCP	66	33271 → 58734 [SYN, ACK] Seq=0
232	7.164320	192.168.1.200	40.77.226.249	TCP	66	58735 → 443 [SYN] Seq=0 Win=642
233	7.201842	40.77.226.249	192.168.1.200	TCP	66	443 → 58735 [SYN, ACK] Seq=0 Ac
419	15.844401	192.168.1.200	192.168.0.10	TCP	66	58737 → 5480 [SYN] Seq=0 Win=64
420	15.844774	192.168.1.200	192.168.0.10	TCP	66	58738 → 5480 [SYN] Seq=0 Win=64
421	15.844776	192.168.0.10	192.168.1.200	TCP	66	5480 → 58737 [SYN, ACK] Seq=0 A
422	15.845000	192.168.0.10	192.168.1.200	TCP	66	5480 → 58738 [SYN, ACK] Seq=0 A

...0 .... = Nonce: Not set  
 .... 0... = Congestion Window Reduced (CWR): Not set  
 .... .0... = ECN-Echo: Not set  
 .... ..0. = Urgent: Not set  
 .... ...1 = Acknowledgment: Set  
 .... .... 0... = Push: Not set  
 .... .... .0.. = Reset: Not set  
 > .... .... ..1. = Syn: Set

0000 08 96 d7 41 bd f9 c8 9c dc 60 e6 57 08 00 45 00 ...A... ..W..E.  
 0010 00 34 10 dd 40 00 80 06 00 00 c0 a8 01 c8 c8 1d ..4...@... ..  
 0020 20 e7 0d 3d ec 7b 16 09 e2 6f 00 bf 13 98 80 12 ..={... ..O.....  
 0030 fa 00 ab 9b 00 00 02 04 05 b4 01 03 03 00 01 01 .....  
 0040 04 02 ..

Einschränken der Anzeige auf TCP-Pakete, bei denen das SYN-Flag gesetzt ist.

## Filtervorschläge

### Filtern auf MAC-Adressen

eth.addr == 00:50:56.14.a2.fb

eth.src == 00:50:56.14.a2.fb

eth.dst == 00:50:56.14.a2.fb

### Das Filtern auf IP-Adressen

ip.addr == 192.168.1.33

ip.src == 192.168.1.33

ip.dst == 192.168.1.33

### Negieren von Filtern

!ip.addr == 192.168.1.33 bzw. not ip.addr == 192.168.1.33

### Logisches Verknüpfen

snmp || icmp || dns

snmp or icmp or dns

tcp.window\_size == 0 && tcp.flags.reset != 1

### TCP-Pakete nach Strings filtern

tcp matches "String"

tcp contains "String"

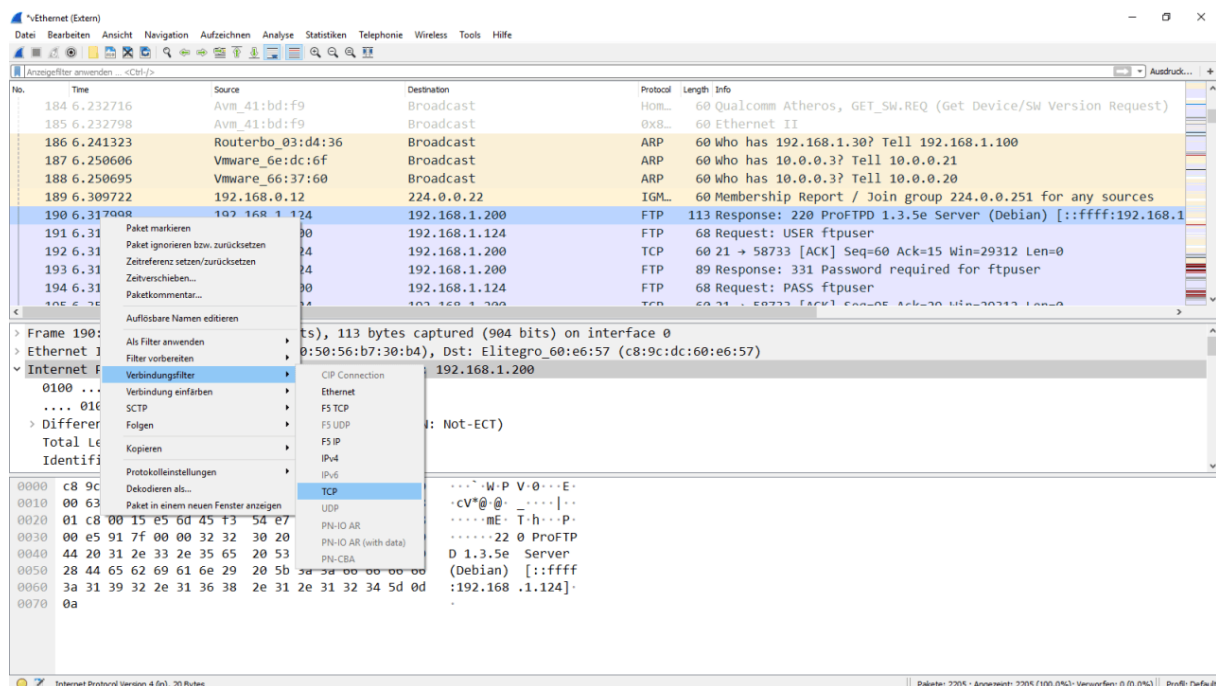
http.request.uri matches "gl=se\$"

## Datenströme verfolgen mit Verbindungsfiltern

Mit Wireshark kann man auch komplette Kommunikationsströme nachverfolgen, also alle Pakete filtern, die eine spezifische Interaktion zwischen zwei Systemen darstellen. Dazu muss man erst ein Paket finden, das zur Interaktion zwischen den beiden Endpunkten gehört und darauf einen Verbindungsfilter anwenden.

Wir gehen im ersten Beispiel von einem bestehenden ungefilterten Mitschnitt einer FTP-Sitzung aus, bei der wir auf eine Verschlüsselung verzichtet haben. Zuerst suchen wir ein Paket aus, von dem wir wissen, dass es zu einem ganz bestimmten Kommunikationsstrom gehört und klicken dann im Kontextmenü auf *Verbindungsfilter*.

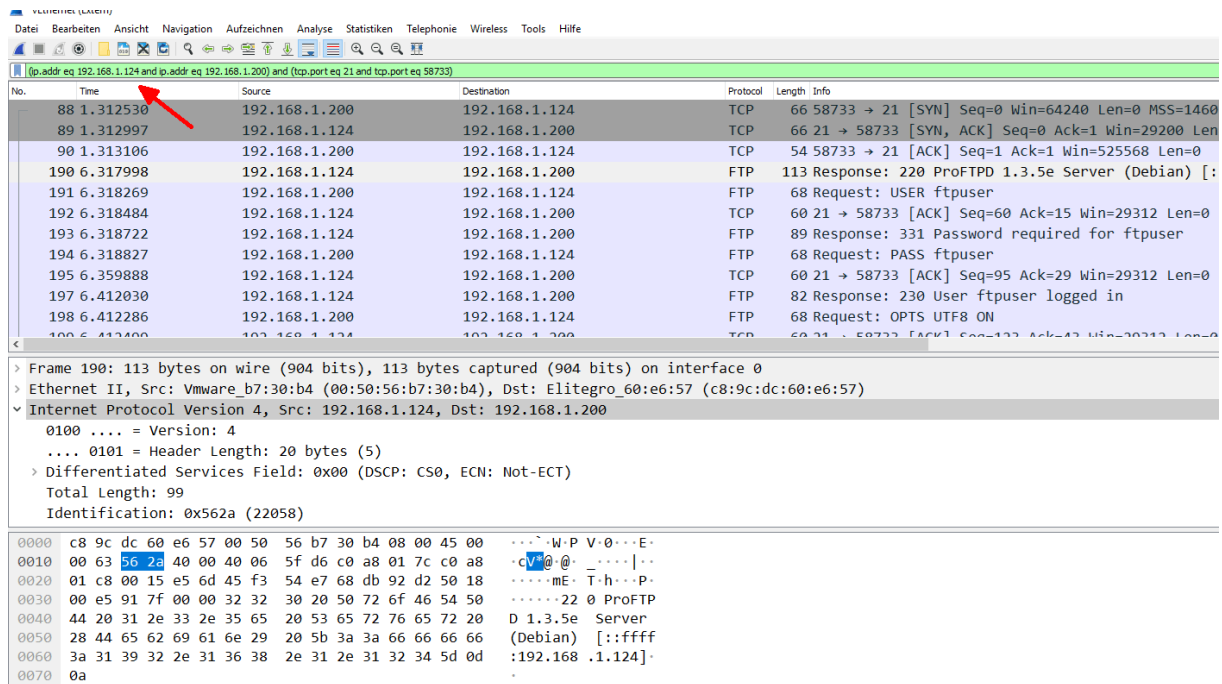
Hier besteht die Möglichkeit, auf MAC-Ebene ("Ethernet" = Layer 2), IPv4-Ebene (Layer 3) oder TCP-Ebene (Layer 4) zu filtern. In diesem Beispiel testen wir mit TCP, weil Layer 4 meist die exakteste Form der Filterung erlaubt.



*TCP-Verbindungsfilter erzeugen für eine FTP-Kommunikation*

In Abhängigkeit von dieser Auswahl erstellt Wireshark jetzt in der Kopfzeile automatisch den passenden Filter, in diesem Fall

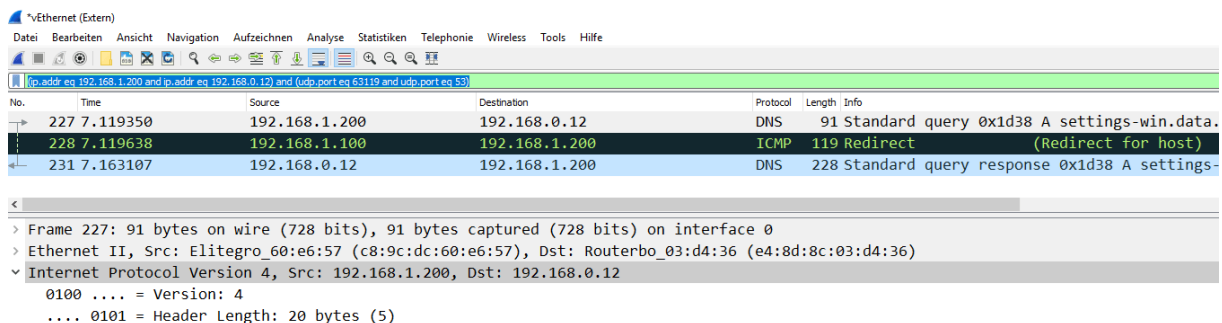
(ip.addr eq 192.168.1.124 and ip.addr eq 192.168.1.200) and (tcp.port eq 21 and tcp.port eq 58733)



Der Verbindungsfilter macht die Pakete der FTP-Kommunikation sichtbar.

In einem weiteren Beispiel können wir bei einem Paket, das erkennbar zu einer DNS-Kommunikation gehört, den Verbindungsfilter auf UDP einstellen. Der generierte Ausdruck sieht dann so aus:

(ip.addr eq 192.168.1.200 and ip.addr eq 192.168.0.12) and (udp.port eq 63119 and udp.port eq 53)



DNS-Kommunikation aus dem Mitschnitt filtern

Diese Vorgehensweise ist zwar ganz nett, Wireshark bietet aber auch noch bessere Methoden.

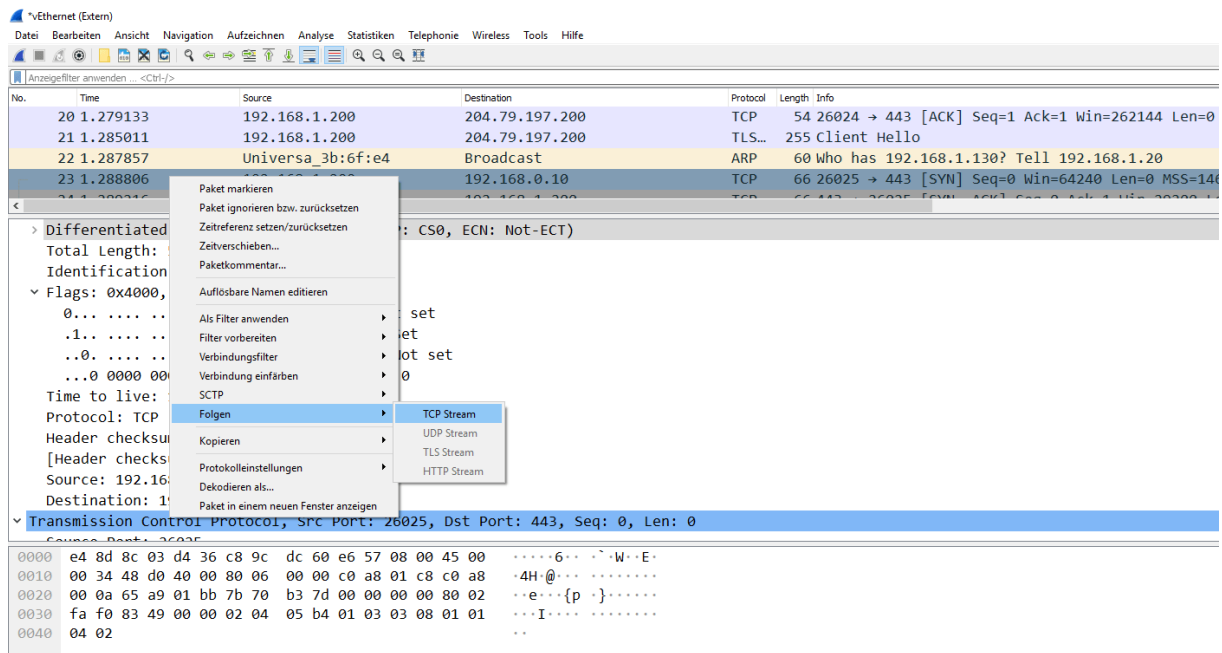
Im nächsten Beispiel haben wir die HTTPS-Kommunikation von Client 192.168.1.200 mit der URL eines VMware vCenter Web-Clients unter der IP 192.168.0.10 aufgezeichnet und uns mit dem SSO-Account

*Administrator@vsphere.local*

authentifiziert.

Nun markieren wir im ungefilterten Mitschnitt ein Paket, das offensichtlich zu diesem Kommunikations-Stream gehört (hier Paket Nr. 23), und wählen im Kontextmenü den Eintrag *Folgen => TCP Stream*.





*TCP-Stream für eine HTTPS-Kommunikation folgen*

Wir erhalten dann folgende Ausgabe:



*Aus einer verschlüsselten Verbindung lassen sich erwartungsgemäß keine verwertbaren Informationen entnehmen.*

Da die Verbindung zum vCenter-Server SSL-verschlüsselt ist, sieht man, wie erwartet, nicht viel. Im folgenden Screenshot hingegen folgen wir einem HTTP-Stream beim Anzeigen der Website [www.vmword.com](http://www.vmword.com).

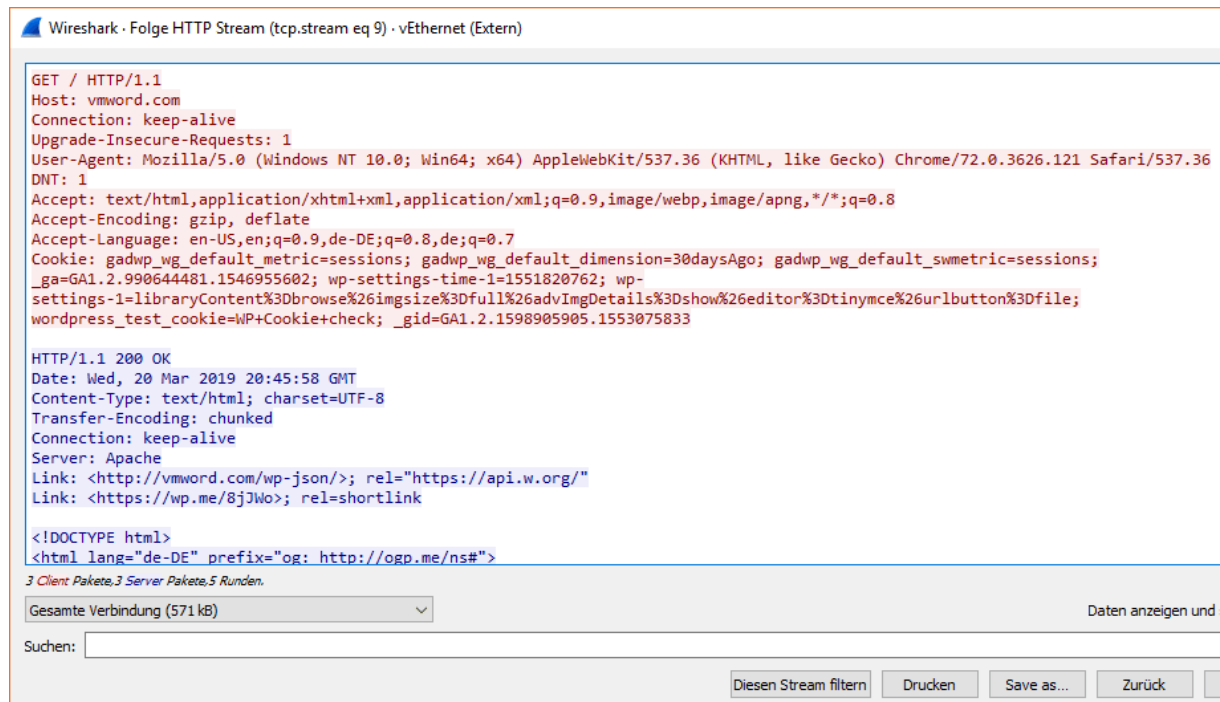
Um ein erstes zu dieser HTTP-Kommunikation passendes Paket überhaupt zu finden, verwenden wir erst den Filter

`tcp contains "vmword.com"`

und markieren dann eines der gefundenen HTTP-Pakete. Danach wählen wir im Kontextmenü *Folgen* => *HTTP Stream*, was den Filter

tcp.stream eq 9

erzeugt. Das Ergebnis sieht nun so aus:



Die Analyse einer HTTP-Verbindung zeigt den HTTP-Header und die Nutzdaten.

Auf diese Weise erhält man den kompletten HTTP-Header und findet unter anderem sehr schnell heraus, welcher Browser verwendet wurde, welche Sprache der Agent akzeptiert und dass auf dem Server Wordpress läuft.