ADMIN:DOCS

Netzwerkprobleme unter Linux systematisch analysieren

Lerne, wie du Netzwerkprobleme unter Linux systematisch analysierst und behebst. Dieser Artikel behandelt IP-Adressierung, Routing, DNS und essenzielle Diagnosewerkzeuge.

Kategorie: Netzwerk **Veröffentlicht:** 06.08.2026

Die systematische Analyse von Netzwerkproblemen unter Linux ist keine optionale Zusatzfähigkeit, sondern eine der zentralen Disziplinen der täglichen Systemadministration. Störungen treten selten isoliert auf. Sie zeigen sich oft als unklare Symptome: ein Dienst antwortet nicht, ein Host ist zeitweise unerreichbar, Namensauflösung schlägt fehl oder die Latenz steigt ohne erkennbaren Grund. Ohne strukturiertes Vorgehen verlierst du schnell Zeit in zufälligen Checks und falschen Annahmen. Dieser Artikel liefert dir deshalb ein klares, wiederholbares Raster, mit dem du Probleme methodisch eingrenzt und belastbare Aussagen triffst.

Das Ziel ist ein praxisorientiertes Nachschlagewerk. Du erhältst kein theoretisches Lehrbuch und keine Sammlung isolierter Befehle. Stattdessen lernst du, wie du die vorhandenen Bordmittel moderner Linux-Systeme gezielt und in sinnvoller Reihenfolge einsetzt. Der Schwerpunkt liegt auf einem Bottom-up-Ansatz, der von der lokalen Host-Konfiguration über Connectivity- und Pfadprüfungen bis zur Paket- und Log-Analyse führt. Jeder Schritt baut auf dem vorherigen auf und reduziert den Suchraum. So vermeidest du typische Umwege und kommst schneller zu einer belastbaren Diagnose.

Dieser Leitfaden führt dich Schritt für Schritt von den Grundlagen zur Tiefenanalyse.

Wir arbeiten uns systematisch von der lokalen Host-Konfiguration über Pfad- und Erreichbarkeitstests bis hin zu Paket-Captures und Kernel-Logs vor. Alle Abschnitte sind so aufgebaut, dass du bei einem akuten Problem auch direkt im passenden Kapitel einsteigen kannst.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

Voraussetzungen für die erfolgreiche Arbeit mit diesem Material sind solide Linux-Systemkenntnisse, sicherer Umgang mit der Shell und erste praktische Erfahrungen in der Systemadministration. Du solltest die grundlegenden Netzwerkkonzepte – Interfaces, Adressierung, Routing, DNS und Firewall – bereits verstanden haben und dich im Terminal ohne ständige Nachschlagehilfe bewegen können. Shell-Scripting-Kenntnisse helfen, wiederkehrende Prüfungen zu automatisieren, sind aber keine zwingende Voraussetzung. Der Text setzt voraus, dass du die Ausgabe der verwendeten Werkzeuge interpretieren und die Ergebnisse in einen Gesamtkontext einordnen kannst.

> ⚠ **Wichtig:** Dieser Artikel richtet sich an Linux-Administratoren und IT-Fachkräfte, die bereits grundlegende Erfahrungen mit der Netzwerkadministration unter Linux haben. Du solltest dich im Terminal sicher bewegen können und die wichtigsten Netzwerkkonzepte verstehen.
>
> Wer diese Grundlagen mitbringt, findet hier ein Werkzeug, das im Betrieb Alltagstauglichkeit besitzt. Der Text bleibt bewusst nah an den realen Bedingungen eines Administrators: begrenzte Zeit, unklare Symptome und der Zwang, schnell zu entscheiden, ob das Problem lokal, auf dem Pfad oder auf der Gegenseite liegt. Genau dafür ist die hier beschriebene Systematik gedacht.

Die folgende Methodik bildet die Grundlage für alle weiteren Schritte.

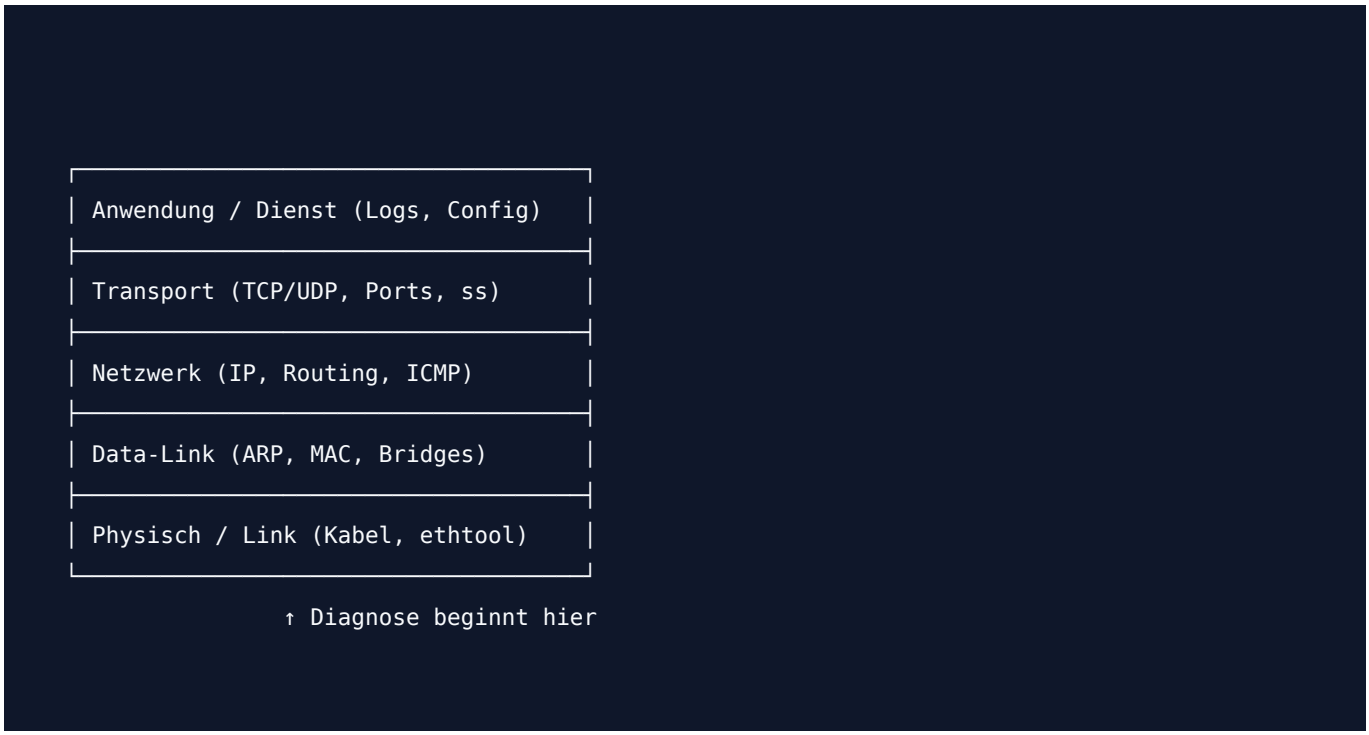
Systematische Methodik der Netzwerkdiagnose

Bottom-up-Ansatz entlang der Netzwerkschichten

Netzwerkprobleme unter Linux entstehen selten isoliert auf einer einzigen Ebene. Jede höhere Schicht setzt voraus, dass die darunterliegenden funktionieren. Deshalb beginnst du die Diagnose stets von unten und arbeitest dich schrittweise nach oben. Dieser Bottom-up-Ansatz verhindert, dass du Zeit mit komplexen Tools verlierst, während die eigentliche Ursache viel näher am physischen oder Link-Layer liegt.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

Die relevanten Schichten orientieren sich am klassischen Modell, werden jedoch an die Linux-Realität angepasst. Auf der untersten Ebene prüfst du den physischen Link und den Treiberstatus. Darauf folgt die Data-Link-Schicht mit MAC-Adressen, ARP und VLAN-Zuordnung. Die Netzwerkschicht umfasst IP-Adressierung und Routing. Darüber liegen Transportprotokolle und Sockets, schließlich die Anwendung selbst.



Beginne immer mit dem Link-Status. Ein Interface, das DOWN ist oder keine Carrier-Erkennung meldet, macht alle weiteren Prüfungen sinnlos. Erst wenn der Link steht und eine gültige Adresse vorhanden ist, prüfst du die Erreichbarkeit des Gateways. Danach testest du die Weiterleitung und erst zum Schluss DNS, Ports und Anwendungsantworten.

> ⚠ Überspringst du die unteren Schichten, riskierst du Fehldiagnosen, bei denen du Firewall-Regeln oder Dienstkonfigurationen suchst, obwohl der Fehler bereits im Treiber oder in der Verkabelung liegt.

Der Ansatz zwingt dich zu Disziplin. Du dokumentierst jedes Ergebnis, bevor du zur nächsten Schicht gehst. Nur so erkennst du Abhängigkeiten und vermeidest parallele, unstrukturierte Checks. In komplexen Umgebungen mit Bridges, Bonding oder Containern wird dieser disziplinierte Weg besonders wertvoll, weil dort mehrere virtuelle Schichten übereinanderliegen.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

> □□ Der Bottom-up-Pfad eignet sich auch dann, wenn Symptome auf der Anwendungsebene auftauchen. Ein Webserver, der keine Verbindungen annimmt, kann genauso gut an einer fehlenden Route oder an einem downen Interface scheitern wie an einer falschen Listen-Adresse.

□□ **Praktisches Beispiel** Stell dir vor, ein Host antwortet nicht auf SSH-Anfragen. Statt sofort `ss` oder `tcpdump` zu starten, beginnst du mit dem Interface:

```
ip link show eth0
ethtool eth0 | grep -E 'Link detected|Speed|Duplex'
ip addr show eth0
ip route show default
```

Erst wenn Link detected: `yes`, eine `gültige IP` und eine `Default-Route` vorhanden sind, gehst du weiter zu `Ping` und Socket-Status. Dieser Ablauf spart in der Praxis oft die Hälfte der Diagnosezeit.

Informationssammlung und Reproduzierbarkeit

Eine belastbare Diagnose steht und fällt mit der Qualität der gesammelten Informationen. Bevor du tief in einzelne Schichten einsteigst, sicherst du den aktuellen Zustand des Systems. Nur so kannst du später nachvollziehen, was sich geändert hat, und das Problem unter vergleichbaren Bedingungen erneut auslösen. Ohne diese Grundlage bleiben viele Beobachtungen anekdotisch und schwer zu verifizieren.

Beginne mit einem festen Satz an Basisdaten.

Notiere Zeitpunkt, betroffene Interfaces, aktuelle IP-Adressen, Routing-Tabelle und den Status der relevanten Dienste. Erfasse außerdem die letzten Kernel-Meldungen und die Ausgabe der wichtigsten Netzwerk-Tools. Diese Momentaufnahme bildet den Referenzpunkt. Später kannst du denselben Befehlssatz erneut ausführen und die Differenzen erkennen.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

Reproduzierbarkeit bedeutet, dass du das Symptom unter kontrollierten Bedingungen wieder erzeugen kannst. Das ist besonders bei intermittierenden Störungen entscheidend. **Dokumentiere deshalb genau, unter welchen Umständen das Problem auftritt:** nach einem Neustart, nur zu bestimmten Tageszeiten, nur bei Last oder nur von bestimmten Quelladressen. Halte fest, welche Schritte du bereits unternommen hast und welche Ergebnisse sie geliefert haben.

> ⚠ Ohne diese Disziplin verlierst du schnell den Überblick und wiederholst dieselben Checks mehrfach.

Ein praktischer Weg besteht darin, die Ausgaben in Dateien umzuleiten und mit Zeitstempel zu versehen. So entsteht ein nachvollziehbarer Verlauf. Nutze dafür einfache Shell-Konstrukte und speichere die Ergebnisse in einem eigenen Verzeichnis. Achte darauf, dass die Dateien die relevanten Kontextinformationen enthalten – Hostname, Datum und den genauen Befehl.

> □□ Die Reproduzierbarkeit gewinnt zusätzlich an Wert, wenn mehrere Personen an der Störung arbeiten. Eine klar dokumentierte Ausgangslage verhindert, dass Kollegen dieselben Schritte erneut ausführen oder von veralteten Annahmen ausgehen.

Diagnose-Snapshot 1. Zeitpunkt + Host 2. Link- und Adressstatus

3. Routing und Nachbarn 4. Socket- und Dienststatus 5. Letzte Kernel-/Journal-Meldungen

6. Beobachtetes Symptom + Auslöser

□□ Praktisches Beispiel

Lege ein Verzeichnis an und sichere den Ausgangszustand:

```
mkdir -p /tmp/netdiag/$(date +%Y%m%d-%H%M)
```

```
cd /tmp/netdiag/$(date +%Y%m%d-%H%M)

{
  echo "=== $(hostname) $(date -Iseconds) ==="
  ip -br link
  ip -br addr
  ip route
  ip neigh
  ss -tulpn
} > 01-basis.txt

journalctl -k -b --no-pager | tail -n 100 > 02-kernel.txt
dmesg -T | tail -n 50 > 03-dmesg.txt
```

Führe denselben Satz später erneut aus und vergleiche die Dateien. So erkennst du Änderungen an Interfaces, Routen oder Socket-Zuständen, ohne dich auf das Gedächtnis verlassen zu müssen. Ergänze bei Bedarf gezielte Captures oder zusätzliche Tool-Ausgaben, sobald du die betroffene Schicht eingegrenzt hast.

Die gesammelten Daten bleiben nur dann nützlich, wenn sie vollständig und zeitlich konsistent sind. Vermeide es, einzelne Befehle isoliert und ohne Kontext auszuführen. Arbeite immer mit dem vollständigen Satz und speichere die Ausgaben, bevor du Konfigurationen änderst oder Dienste neu startest.

> Eine häufige Fehlerquelle ist das Überschreiben älterer Snapshots; nummeriere die Dateien deshalb klar und behalte die ursprüngliche Ausgangslage.

Typische Einstiegsfehler und deren Vermeidung

Auch mit sauberer Informationssammlung und reproduzierbaren Snapshots laufen Diagnosen oft in die Irre, wenn die ersten Schritte falsch gewählt werden. Die häufigsten Einstiegsfehler entstehen nicht aus Unwissen über einzelne Befehle, sondern aus der falschen Reihenfolge und aus vorschnellen Annahmen. Wer diese Muster kennt, spart Stunden und vermeidet, dass sich das Symptom durch eigene Eingriffe weiter verändert.

Ein klassischer Fehler besteht darin, sofort auf Anwendungsebene zu suchen.

Viele beginnen mit Port-Scans, Firewall-Regeln oder Dienst-Logs, obwohl das Interface noch DOWN ist oder keine Carrier-Meldung vorliegt. Das führt zu langen Umwegen. Die Vermeidung ist einfach: Jede Diagnose startet mit dem Link- und Adressstatus. Erst wenn diese Basis stimmt, darf die nächste Schicht betrachtet werden. Wer das überspringt, interpretiert später Symptome, die gar nicht zur eigentlichen Ursache gehören.

Häufig werden auch Namensauflösung und Erreichbarkeit verwechselt.

Ein fehlgeschlagener Ping auf einen Hostnamen wird schnell als Netzwerkausfall gewertet, obwohl nur der Resolver stockt. **Die korrekte Trennung lautet:** Zuerst die `IP-Adresse` direkt anpingen, danach den `Hostnamen`. Schlägt nur der Name fehl, liegt das Problem bei `DNS` oder `/etc/hosts`. Schlägt bereits die IP fehl, ist die Schicht darunter betroffen.

> ⚠ Wer diese Unterscheidung nicht trifft, verschwendet Zeit an der falschen Stelle und verdächtigt Routing oder Firewall, obwohl der Fehler in der Auflösung sitzt.

Ebenso verbreitet ist das blinde Vertrauen in veraltete Werkzeuge.

`ifconfig` und `netstat` liefern auf modernen Systemen unvollständige oder irreführende Ausgaben, weil sie Bridge-, Bonding- und Namespace-Informationen nur eingeschränkt zeigen. Die Vermeidung besteht darin, konsequent `ip` und `ss` zu verwenden. Diese Tools lesen direkt aus dem Kernel und geben den tatsächlichen Zustand wieder. Wer weiterhin die Legacy-Befehle nutzt, übersieht häufig virtuelle Interfaces oder Container-Netzwerke.

Ebenfalls problematisch: Das System verändern, bevor der Ausgangszustand gesichert ist. Ein Neustart des Network-Managers, das Flushing der ARP-Tabelle oder das Neuladen von Firewall-Regeln kann das Symptom vorübergehend verschwinden lassen und die Ursache unauffindbar machen. Die Disziplin lautet: Erst dokumentieren, dann eingreifen. Jede Änderung muss nachvollziehbar und idealerweise rückgängig machbar sein.

Auch die Perspektive wird oft falsch gewählt. Viele testen ausschließlich vom betroffenen Host aus und übersehen, dass der Dienst lokal lauscht, aber von außen nicht erreichbar ist – oder umgekehrt. Die Vermeidung erfordert mindestens einen zweiten Beobachtungspunkt: einen anderen Host im selben Segment oder einen gezielten Test von außerhalb. Nur so wird klar, ob das Problem lokal, auf dem Pfad oder auf der Gegenseite liegt.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

□□ Ein subtiler Fehler entsteht, wenn temporäre Zustände ignoriert werden. Stale ARP-Einträge, abgelaufene DHCP-Leases oder kurzlebige Routen können Symptome erzeugen, die nach wenigen Minuten von selbst verschwinden. Wer in diesem Moment bereits tief in die Paketanalyse einsteigt, sucht nach einer Ursache, die inzwischen nicht mehr existiert. Die Gegenmaßnahme ist, den Zeitpunkt des Symptoms genau zu erfassen und denselben Test zeitnah zu wiederholen.

```
Falscher Einstieg                Korrekter Einstieg
─────────────────                ──────────────────
Dienst-Logs prüfen          →    Link-Status prüfen
Firewall-Regeln lesen       →    Adresse und Route prüfen
tcpdump starten             →    Erreichbarkeit trennen
Konfiguration ändern        →    Snapshot sichern
```

□□ **Praktisches Beispiel** Angenommen, ein Host antwortet nicht auf HTTP-Anfragen. Der falsche Einstieg sieht oft so aus:

```
ss -tulpn | grep 80
nft list ruleset
tcpdump -i any port 80
systemctl restart nginx
```

Der korrekte Einstieg beginnt niedriger und belässt den Zustand unverändert:

```
ip -br link
ip -br addr
ip route get 8.8.8.8
ping -c 3 $(ip route | awk '/default/ {print $3}')
ss -tlnp '( sport = :80 )'
```

Nur wenn diese Ausgaben konsistent und unauffällig sind, werden Firewall und Paketcapture hinzugezogen. Diese Reihenfolge verhindert, dass durch den Neustart des Dienstes wertvolle Zustandsmeldungen verloren gehen.

Ein letzter, oft übersehener Fehler ist das Arbeiten mit unzureichenden Rechten oder im falschen Netzwerk-Namespace. Befehle ohne ausreichende Privilegien liefern unvollständige Socket-Listen oder dürfen bestimmte Interfaces nicht abfragen. In Container-Umgebungen kommt hinzu, dass der sichtbare Netzwerkstack nicht dem des Hosts entspricht. Die Vermeidung verlangt, den Kontext explizit zu prüfen: `ip netns list` und gezielte Ausführung innerhalb des richtigen Namespace, falls vorhanden.

> Ohne den korrekten Namespace prüfst du unter Umständen ein ganz anderes Netzwerk-Setup als das betroffene.

Analyse der lokalen Host-Konfiguration

Interfaces, Link-Status und Adressierung

Die Prüfung der Interfaces bildet den eigentlichen Einstieg in die lokale Host-Analyse. Ohne funktionierenden Link und ohne gültige Adressierung bleiben alle weiteren Tests größtenteils bedeutungslos. Deshalb immer zuerst den physischen und logischen Zustand der Netzwerkschnittstelle erfassen, bevor Routing, DNS oder gar Dienste betrachtet werden.

Der zentrale Befehl für den Überblick ist `ip`.

Mit der Option `-br` erhältst du eine kompakte Übersicht über alle Interfaces, ihren Zustand und die zugeordneten Adressen. Die Ausgabe zeigt unmittelbar, ob ein Interface administrativ UP ist und ob der Kernel einen Carrier erkennt. Fehlt die Flagge `LOWER_UP` oder erscheint `NO-CARRIER`, liegt das Problem unterhalb der IP-Schicht.

```
ip -br link
ip -br addr
```

Für die detaillierte Betrachtung einzelner Interfaces dient `ip link show`.

Hier werden Flags, MTU, Queue-Discipline und der aktuelle Zustand sichtbar. Besonders relevant sind die Kombinationen `UP,LOWER_UP` (*Link steht*). `UP,NO-CARRIER` (*Interface aktiviert, aber kein Signal*). Die MAC-Adresse und der Promiscuous-Mode können ebenfalls Hinweise liefern, etwa bei Bridge- oder Monitoring-Setups.

Ergänzend liefert `ethtool` Informationen, die `ip` nicht abdeckt.

Dazu gehören die ausgehandelte Geschwindigkeit, der Duplex-Modus, die Link-Erkennung und Fehlerzähler. Ein Gigabit-Interface, das nur mit 100 Mbit/s oder im Half-Duplex-Modus arbeitet, deutet fast immer auf ein Kabel-, Switch-Port- oder Autonegotiation-Problem hin. Steigende Fehler- oder Drop-Zähler weisen auf physische Störungen oder Treiberprobleme hin.

```
ethtool eth0
ethtool -S eth0 | grep -E 'error|drop|crc|frame'
```

Die Adressierung wird mit `ip addr show` geprüft.

Neben der IP-Adresse selbst sind Scope, gültige Lebenszeit und der Zuweisungsmodus entscheidend. Dynamische Adressen von `DHCP` tragen immer den Hinweis `dynamic`, bei

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

statischen fehlt dieser Hinweis. Eine Adresse im Bereich `169.254.0.0/16` signalisiert, dass kein DHCP-Server erreicht wurde und das System auf APIPA ausgewichen ist. IPv6-Adressen sollten ebenfalls betrachtet werden, da viele moderne Umgebungen dual-stack betreiben.

> □□ Ein Interface kann administrativ UP sein und dennoch keine sinnvolle Kommunikation ermöglichen, wenn die Adresse fehlt, abgelaufen ist oder einem falschen Scope zugeordnet wurde. In solchen Fällen zeigt `ip` den Link als funktionsfähig, die höhere Schicht bleibt jedoch blockiert.

Die Interface-Namen selbst verdienen auch Aufmerksamkeit.

Moderne Distributionen verwenden vorhersagbare Namen wie `ens3`, `enp0s3` oder `enx` gefolgt von der MAC. Ältere Skripte, die noch `eth0` erwarten, scheitern dann still. In virtualisierten oder containerisierten Umgebungen kommen zusätzlich virtuelle Interfaces, VLANs und Bonding-Devices hinzu. Diese müssen explizit in die Prüfung einbezogen werden, weil sie den tatsächlichen Ausgangsweg bestimmen.

```
Link-Prüfung in der Praxis
──────────────────────────

1. ip -br link          → Zustand aller Interfaces
2. ip link show DEV     → Flags und Details
3. ethtool DEV          → Speed, Duplex, Carrier
4. ethtool -S DEV       → Fehlerzähler
5. ip addr show DEV     → Adressen und Scope
```

> □□ Die Reihenfolge ist bewusst gewählt: Zuerst der administrative und physische Zustand, dann die Adressierung. Nur so wird verhindert, dass eine fehlende Adresse als Routing- oder Firewall-Problem missverstanden wird.

□□ Praktisches Beispiel

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

Ein Host soll externe Ziele erreichen, antwortet aber nicht. Die Prüfung beginnt mit dem vollständigen Satz:

```
ip -br link
ip -br addr
ip link show eth0
ethtool eth0 | grep -E 'Speed|Duplex|Link detected'
ethtool -S eth0 | grep -iE 'err|drop|crc'
ip addr show eth0
```

Erwartete Normalwerte sind `state UP`, `LOWER_UP`, `Link detected: yes`, eine passende Speed/Duplex-Kombination und mindestens eine gültige globale Adresse. Erscheint `NO-CARRIER` oder `Link detected: no`, ist die weitere Diagnose auf der physischen oder Treiber-Ebene fortzusetzen. Fehlt die Adresse oder liegt nur eine Link-Local-Adresse vor, liegt das Problem bei der Adresszuweisung.

Temporäre Zustände beachten

Interfaces können nach einem Treiber-Reload oder nach dem Entfernen eines Kabels kurzzeitig als `DOWN` angezeigt werden und danach wieder `UP` sein, ohne dass die Adresse erneut bezogen wird. In DHCP-Umgebungen muss dann oft ein Lease-Renew erzwungen werden. Statische Konfigurationen hingegen bleiben stabil, solange das Interface nicht explizit neu konfiguriert wird.

> Eine reine Abfrage des administrativen Status reicht nicht aus – ohne Prüfung von Carrier und Fehlerzählern übersieht man physische Paketverluste.

Routing, Nachbarschaftstabellen und DNS-Resolver

Nach dem Link- und Adressstatus stehen Weiterleitung und Namensauflösung im Fokus. Diese Schicht entscheidet, ob Pakete den lokalen Host überhaupt verlassen können und ob Zieladressen korrekt aufgelöst werden. Fehler hier erzeugen oft Symptome, die fälschlich der

Firewall oder dem entfernten Dienst zugeschrieben werden.

Die Routing-Tabelle wird mit `ip route` und gezielten Abfragen untersucht. Zuerst prüfst du die Default-Route. Fehlt sie oder zeigt sie auf ein nicht erreichbares Gateway, ist externe Kommunikation unmöglich. Anschließend testest du den konkreten Pfad zu einem Ziel mit `ip route get`. Dieser Befehl zeigt nicht nur die Route, sondern auch das verwendete Interface und die Quelladresse. Das ist besonders wertvoll bei Systemen mit mehreren Interfaces oder Policy-Routing.

```
ip route
ip route show default
ip route get 8.8.8.8
ip route get 192.168.1.50 from 10.0.0.10
```

Mehrere Routing-Tabellen oder Regeln (`ip rule`) kommen in komplexeren Setups vor. Für die meisten Diagnosen reicht die Haupttabelle. Wichtig ist, dass die Route zum Gateway über ein Interface führt, das tatsächlich UP und adressiert ist. Eine Route über ein DOWN-Interface bleibt wirkungslos, auch wenn sie in der Tabelle steht.

Die Nachbarschaftstabelle (ARP für IPv4, NDP für IPv6) wird mit `ip neigh` betrachtet. Sie zeigt, welche Layer-2-Adressen der Kernel für bekannte IPs gelernt hat. Zustände wie `REACHABLE` und `STALE` sind normal. `FAILED` oder `INCOMPLETE` deuten darauf hin, dass das Gateway oder der Nachbar nicht antwortet. Ein dauerhaft fehlender Eintrag für das Default-Gateway ist ein klares Signal für ein Problem auf der Data-Link-Schicht oder beim Gateway selbst.

```
ip neigh
ip neigh show dev eth0
ip neigh show to 192.168.1.1
```

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

Stale Einträge können temporäre Störungen verursachen. In solchen Fällen hilft ein gezieltes Löschen des Eintrags, damit der Kernel die Adresse neu auflöst. Das sollte jedoch nur nach Dokumentation des Ausgangszustands geschehen.

> ⚠ Eine fehlende oder fehlerhafte Default-Route wird häufig übersehen, weil `ping` auf lokale Adressen noch funktioniert und erst bei externen Zielen scheitert. Die Prüfung mit `ip route get` macht den tatsächlichen Ausgangsweg sichtbar und verhindert Fehldiagnosen.

Der DNS-Resolver bildet die nächste Prüfinstanz. Moderne Systeme nutzen häufig `systemd-resolved`. Der Status und die konfigurierten Server werden mit `resolvectl` abgefragt. Zusätzlich sollte `/etc/resolv.conf` kontrolliert werden, da ältere Anwendungen oder Container weiterhin darauf zugreifen. Ein funktionierender Resolver bedeutet nicht automatisch, dass die Auflösung für alle Anwendungen gilt – insbesondere in Container-Umgebungen können eigene Resolver-Konfigurationen greifen.

```
resolvectl status
resolvectl query example.com
cat /etc/resolv.conf
dig +short example.com
dig @8.8.8.8 example.com
```

Die Trennung von Erreichbarkeit und Namensauflösung bleibt entscheidend. Ein erfolgreicher Ping auf die IP bei gleichzeitigem Fehlschlag des Hostnamens isoliert das Problem klar auf DNS. Umgekehrt zeigt ein fehlgeschlagener IP-Ping, dass die Ursache tiefer liegt und DNS zunächst irrelevant ist.

> □□ Die Kombination aus `ip route get` und einem gezielten `dig` gegen einen bekannten Server liefert in wenigen Sekunden die Aussage, ob Routing oder Auflösung die Störung

> verursacht. Das spart den Umweg über unnötige Firewall- oder Paketanalysen.

□□ Praktisches Beispiel

Ein Host soll einen internen Dienst unter dem Namen `app.internal` erreichen. Die Prüfung läuft in klarer Reihenfolge:

```
# Routing und Gateway
ip route show default
ip route get 10.20.30.40
ping -c 3 $(ip route | awk '/default/ {print $3}')

# Nachbarschaftstabelle für das Gateway
GATEWAY=$(ip route | awk '/default/ {print $3}')
ip neigh show to $GATEWAY

# DNS-Trennung
ping -c 2 10.20.30.40
ping -c 2 app.internal
dig +short app.internal
resolvectl query app.internal
```

Schlägt der Ping auf die IP fehl, während die Route und der Nachbareintrag korrekt erscheinen, liegt das Problem jenseits des lokalen Hosts. Schlägt nur die Namensauflösung fehl, wird der Resolver und die DNS-Server-Erreichbarkeit geprüft. Ein zusätzlicher Test gegen einen öffentlichen Resolver (`dig @1.1.1.1`) zeigt, ob das Problem lokal oder beim internen DNS-Server liegt.

Bei Hosts mit mehreren Schnittstellen spielt auch Policy-Routing eine Rolle. Bei Hosts mit mehreren Adressen kann `ip route get` eine andere Quell-IP wählen als erwartet. Das führt dazu, dass Antworten über den falschen Pfad zurückkommen oder von Firewalls verworfen werden. In solchen Fällen hilft die explizite Angabe der Quelladresse beim Test.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

```
ip route get 8.8.8.8 from 192.168.10.50
ping -I 192.168.10.50 -c 3 8.8.8.8
```

> Wer Nachbarschaftstabelle und Gateway-Routen übergeht, vermutet oft Fehler in der Anwendung, obwohl bereits Layer 2 oder 3 blockiert sind.

Socket-Status und lokale Dienste mit `ss`

Im nächsten Schritt wird geprüft, ob lokale Dienste auf den erwarteten Sockets lauschen. Denn hier entscheidet sich, ob ein Dienst tatsächlich lauscht, an welche Adresse er gebunden ist und ob Verbindungen aufgebaut werden können. `ss` ist hier das Werkzeug der Wahl, weil es die Socket-Informationen direkt aus dem Kernel liest und deutlich schneller sowie vollständiger arbeitet als das ältere `netstat`.

Der erste und wichtigste Check gilt den lauschenden Sockets.

Mit `ss -tulpn` siehst du alle TCP- und UDP-Listener, die numerischen Ports und die zugehörigen Prozesse. Die Ausgabe zeigt sofort, ob ein Dienst auf `0.0.0.0`, einer spezifischen IP oder nur auf `127.0.0.1` gebunden ist. Eine Bindung ausschließlich an localhost ist einer der häufigsten Gründe, warum ein Dienst lokal funktioniert, von außen aber unerreichbar bleibt.

```
ss -tulpn
ss -tlnp
ss -ulnp
ss -tlnp '( sport = :80 or sport = :443 )'
```

Für detailliertere Filterung unterstützt `ss` Ausdruckssyntax. Damit kannst du gezielt nach Port, Zustand oder Adresse suchen, ohne die gesamte Ausgabe manuell zu durchsuchen. Besonders nützlich ist die Einschränkung auf einen einzelnen Port oder auf Verbindungen in einem bestimmten Zustand.

```
ss -tn state listening
ss -tn state established
ss -tn state syn-sent
ss -tn '( dport = :443 or sport = :443 )'
ss -tnp dst 10.20.30.40
```

Die Zustände selbst liefern wichtige Hinweise.

`LISTEN` bestätigt, dass der Dienst den Port geöffnet hat. `ESTABLISHED` zeigt aktive Verbindungen. `SYN-SENT` deutet auf ausgehende Verbindungsversuche hin, die nicht beantwortet werden. Eine hohe Zahl von `TIME_WAIT`-Einträgen kann bei stark belasteten Servern normal sein, wird aber problematisch, wenn die verfügbaren lokalen Ports erschöpft sind.

> ⚠ Ein Dienst, der in der Prozessliste läuft, aber in `ss` nicht als Listener erscheint, ist entweder an die falsche Adresse gebunden, im falschen Netzwerk-Namespace gestartet oder bereits abgestürzt, ohne dass der Prozessmanager das bemerkt hat.

Zusätzlich zur reinen Socket-Liste liefert `ss -s` eine Zusammenfassung der Socket-Statistiken. Hier siehst du auf einen Blick die Anzahl der Verbindungen pro Zustand. Das hilft, anomale Häufungen schnell zu erkennen, etwa massenhaft `CLOSE-WAIT` oder `SYN-RECV`-Einträge, die auf Ressourcenprobleme oder Angriffe hindeuten können.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

```
ss -s
ss -tn state time-wait | wc -l
ss -tnp | grep -E 'CLOSE-WAIT|FIN-WAIT'
```

In Umgebungen mit Containern oder Netzwerk-Namespaces muss der Kontext beachtet werden. Ein `ss` auf dem Host zeigt nicht die Sockets innerhalb eines Containers. Hier hilft entweder `nsenter` oder die Ausführung von `ss` direkt im Container. Andernfalls diagnostizierst du den falschen Netzwerkstack.

```
# Beispiel für Namespace-Wechsel
ip netns list
ip netns exec myns ss -tulpn
```

> Die Kombination aus Port-Filter und Prozessanzeige (`-p`) spart Zeit, weil du sofort siehst, welcher Prozess den Port hält und ob er unter dem erwarteten Benutzer läuft. Das verhindert Verwechslungen bei mehreren Instanzen desselben Dienstes.

Praktisches Beispiel

Ein Webdienst soll auf Port 8080 erreichbar sein, antwortet aber nicht. Die systematische Prüfung mit `ss` sieht so aus:

```
# Alle Listener anzeigen und gezielt filtern
```

```
ss -tlnp | grep 8080
ss -tlnp '( sport = :8080 )'

# Prüfen, an welche Adresse gebunden wird
ss -tlnp '( sport = :8080 )' | awk '{print $4}'

# Aktive Verbindungen zum Port
ss -tnp '( dport = :8080 or sport = :8080 )'

# Zusammenfassung der Zustände
ss -s
```

Erwartetes Ergebnis bei korrekter Konfiguration ist ein Eintrag in `LISTEN` auf `0.0.0.0:8080` oder der gewünschten Interface-IP sowie der zugehörige Prozessname. Erscheint nur `127.0.0.1:8080`, ist die Bindung falsch. Fehlt der Eintrag vollständig, läuft der Dienst nicht oder lauscht auf einem anderen Port. In diesem Fall werden die Prozessliste und die Dienst-Konfiguration gegengeprüft, bevor weitere Maßnahmen erfolgen.

Besonderheiten bei UDP-Sockets

Im Gegensatz zu TCP gibt es bei UDP keinen Verbindungszustand im klassischen Sinn. `ss -ulnp` zeigt dennoch die gebundenen Ports und Prozesse. Das ist entscheidend für Dienste wie DNS, NTP oder eigene UDP-basierte Anwendungen. Fehlt hier der Eintrag, während der Prozess läuft, liegt fast immer eine fehlerhafte Bindung oder ein Namespace-Problem vor.

```
ss -ulnp
ss -unp '( sport = :53 )'
```

> Veraltete Tools wie `netstat` oder fehlende `-p`-Flags verbergen häufig, welcher Prozess einen Port belegt oder in welchem Namespace er läuft.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

Connectivity- und Pfadanalyse

Erreichbarkeitstests

Stehen Sockets und lokale Dienste fest, verschiebt sich die Diagnose auf den externen Netzwerkpfad. Die Frage lautet jetzt, ob Pakete den Host verlassen, das Ziel erreichen und wieder zurückkommen. Dafür stehen uns `ping`, `mtr` und `tracepath` zur Verfügung und liefern uns unterschiedliche Perspektiven auf denselben Pfad, die sich ergänzen.

`ping` bleibt der schnellste Erreichbarkeitstest.

Er prüft, ob ICMP-Echo-Anfragen beantwortet werden, und liefert Latenz sowie Paketverlust. Wichtig ist die konsequente Trennung von IP- und Namensauflösung. Zuerst wird die reine IP getestet, danach der Hostname. Nur so isolierst du DNS-Probleme von echten Verbindungsstörungen. Die Option `-c` begrenzt die Anzahl der Pakete, `-W` setzt das Timeout pro Paket und `-i` steuert das Intervall.

```
ping -c 5 8.8.8.8
ping -c 5 -W 2 192.168.1.1
ping -c 10 -i 0.2 10.20.30.40
ping -c 5 example.com
```

Ein erfolgreicher Ping auf die IP bei gleichzeitigem Fehlschlag des Hostnamens verweist klar auf den Resolver. Umgekehrt zeigt ein fehlgeschlagener IP-Ping, dass das Problem auf dem Pfad oder beim Ziel liegt. Hohe Latenzschwankungen oder ansteigender Verlust deuten auf Überlastung oder instabile Verbindungen hin.

`mtr` kombiniert die Funktionen von `ping` und `traceroute` und liefert eine fortlaufende Statistik über jeden Hop. Im Report-Modus (`-r` oder `-rw`) erzeugt es eine einmalige Zusammenfassung, die sich gut dokumentieren lässt. Die Option `-c` legt die Anzahl der Zyklen fest. Mit `--tcp` oder `--udp` kannst du statt ICMP andere Protokolle testen, was nützlich ist, wenn ICMP gefiltered wird.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

```
mtr -rwc 50 8.8.8.8
mtr -rwc 30 --tcp --port 443 example.com
mtr -n -rwc 20 10.20.30.40
```

Die Ausgabe zeigt für jeden Hop Verlust und Latenz. Ein Verlust, der ab einem bestimmten Hop abrupt ansteigt und auf den folgenden Hops bestehen bleibt, markiert den problematischen Punkt. Ein Verlust, der nur auf einem einzelnen Hop erscheint und danach wieder verschwindet, ist häufig auf ICMP-Rate-Limiting zurückzuführen und nicht zwingend ein echtes Problem.

> ⚠ ICMP wird auf vielen Strecken gefiltert oder limitiert. Ein fehlgeschlagener Ping oder hohe Verluste in `mtr` bedeuten daher nicht automatisch, dass TCP- oder UDP-Verkehr ebenfalls blockiert ist. In solchen Fällen sind zusätzliche Tests mit TCP-basiertem `mtr` oder gezielten Port-Checks erforderlich.

`tracepath` ergänzt die beiden Werkzeuge um die Pfad-MTU-Entdeckung. Es zeigt nicht nur die Hops, sondern auch die maximale Paketgröße, die ohne Fragmentierung durchgestellt werden kann. Das ist entscheidend bei Problemen mit großen Paketen, VPN-Tunneln oder bestimmten Cloud-Verbindungen. Im Gegensatz zu klassischem `traceroute` benötigt `tracepath` keine Root-Rechte.

```
tracepath 8.8.8.8
tracepath -n 10.20.30.40
tracepath -b example.com
```

Die Kombination der drei Werkzeuge ergibt ein belastbares Bild. `ping` liefert die schnelle Ja/Nein-Aussage und die Grundlatenz. `mtr` zeigt, wo auf dem Pfad Verlust oder Verzögerung entstehen. `tracepath` klärt, ob die MTU den Verkehr einschränkt. Gemeinsam verhindern sie,

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

dass ein einzelnes Symptom überinterpretiert wird.

> □□ Die Reihenfolge sollte bewusst gewählt werden: Zuerst ein kurzer `ping` auf die IP, dann `mtr` für die Hop-Statistik und abschließend `tracepath`, wenn der Verdacht auf MTU-Probleme besteht. So bleibt der Diagnoseaufwand gering und die Aussagekraft hoch.

```
Erreichbarkeitstest — sinnvolle Reihenfolge
──────────────────────────────────────────
1. ping -c 5 <IP>           → Grundsatzentscheid
2. ping -c 5 <Hostname>     → DNS-Trennung
3. mtr -rwc 30 <Ziel>       → Hop-Statistik
4. tracepath <Ziel>         → MTU-Pfad
```

□□ Praktisches Beispiel

Ein interner Server unter `10.50.1.20` soll erreicht werden, die Anwendung meldet Timeouts.

Die Prüfung läuft wie folgt:

```
# Grundsatzentscheid und DNS-Trennung
ping -c 5 10.50.1.20
ping -c 5 app.internal.example

# Hop-Analyse mit ausreichender Stichprobe
mtr -rwc 50 -n 10.50.1.20
mtr -rwc 30 --tcp --port 443 10.50.1.20

# MTU-Pfad prüfen
tracepath -n 10.50.1.20
```

```
# Zusätzlicher Test mit kleinerem Intervall bei Verdacht auf intermittierende Verluste
ping -c 20 -i 0.2 10.50.1.20
```

Steigt der Verlust in `mtr` ab Hop 3 dauerhaft an, liegt die Störung dort. Bleibt der Verlust bei `ICMP` hoch, während der TCP-Test mit `--tcp` sauber durchläuft, ist `ICMP` gefiltert. Zeigt `tracepath` eine unerwartet niedrige `pMTU`, müssen große Pakete oder Tunnelkonfigurationen untersucht werden.

Achte bei der Pfadanalyse auch auf asymmetrische Routen. `ping` und `mtr` messen den Hin- und Rückweg. Wenn der Rückweg anders verläuft und dort Pakete verworfen werden, erscheint das Ziel unerreichbar, obwohl der Hinweg intakt ist. In solchen Fällen hilft der Vergleich von Tests von beiden Seiten oder die Betrachtung der Routing-Tabellen auf den beteiligten Hosts.

> Eine reine ICMP-Prüfung täuscht leicht, da viele Firewalls oder Router ICMP-Pakete gezielt drosseln oder filtern.

Firewall- und Filterregeln

nftables`, `firewalld`, `ufw`

Lokale Paketfilter entscheiden häufig darüber, ob ein ansonsten intakter Pfad am Zielhost endet. Die drei gängigen Verwaltungswege – `nftables`, `firewalld` und `ufw` – müssen deshalb systematisch ausgelesen werden, sobald Erreichbarkeitstests einen Bruch nahelegen.

`nftables` bildet die unterste sichtbare Schicht.

Der Befehl `nft list ruleset` gibt die gesamte aktive Konfiguration aus. Interessant sind vor allem die Counter an den einzelnen Regeln. Steigt der Zähler einer Drop- oder Reject-Regel, während Accept-Regeln für denselben Port bei null bleiben, ist die Ursache lokal gefunden. Die Handle-Nummern (`-a`) erlauben später das präzise Entfernen einzelner Einträge, ohne die komplette Basis neu zu laden.

```
nft list ruleset
nft list ruleset -a
nft list table inet filter
nft list chain inet filter input
nft list chain inet filter forward
```

Bei dual-stack-Systemen müssen IPv4- und IPv6-Tabellen getrennt betrachtet werden. Eine Regel, die nur in der einen Familie existiert, erklärt asymmetrische Symptome.

`firewalld` organisiert den Zugriff über Zonen.

Jedes Interface hängt in genau einer Zone, und jede Zone bringt eigene Default-Policies sowie erlaubte Ports und Services mit. Weicht die Zone des betroffenen Interfaces von der erwarteten ab, greifen plötzlich restriktivere Regeln. Runtime- und Permanent-Konfiguration können zudem auseinanderlaufen; nur die permanenten Einstellungen überleben einen Neustart des Dienstes.

```
firewall-cmd --state
firewall-cmd --get-active-zones
firewall-cmd --list-all
firewall-cmd --list-ports
firewall-cmd --query-port=8443/tcp
firewall-cmd --permanent --list-all
```

`ufw` liefert eine nummerierte, leicht lesbare Übersicht.

Die Default-Policies für Incoming und Outgoing stehen ganz oben. Die Nummern der Regeln ermöglichen gezieltes Löschen, ohne die restliche Konfiguration anzufassen.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

```
ufw status verbose
ufw status numbered
ufw app list
```

Unabhängig vom eingesetzten Frontend gilt dieselbe Prüfreihenfolge: Default-Policy der relevanten Kette, explizite Allow-Regeln für Port und Protokoll, Zuordnung des Interfaces zu einer Zone (bei `firewalld`), abschließend die Counter oder Log-Einträge. Nur so wird sichtbar, ob ein Paket die Regelbasis überhaupt erreicht hat oder bereits vorher verworfen wurde.

> ⚠ Die Position einer Regel innerhalb der Kette ist entscheidend. Eine späte Accept-Regel bleibt wirkungslos, wenn eine frühere Drop-Regel denselben Verkehr bereits erfasst.

Der Unterschied zwischen `drop` und `reject` beeinflusst die Symptomlage stark. `drop` erzeugt Timeouts, `reject` liefert dem Absender eine klare Rückmeldung. Viele Diagnose-Tools werten ein ausbleibendes Reject als generellen Verbindungsabbruch und verdecken dadurch die eigentliche Filterwirkung.

```
Filterdiagnose – kritische Punkte
──────────────────────────────────
• Policy der Input- und Forward-Kette
• Explizite Allow-Regel für Port/Protokoll
• Interface-Zonen-Zuordnung (firewalld)
• Counter-Stände an Drop-Regeln
• Übereinstimmung Runtime ↔ Permanent
```

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

□□ Counter und Log-Meldungen liefern den einzigen objektiven Beleg dafür, dass ein Paket tatsächlich von einer bestimmten Regel erfasst wurde. Ohne diese Information bleibt jede Vermutung über blockierten Verkehr unbestätigt.

□□ Praktisches Beispiel

Port `8443` ist lokal im Listen-Zustand, externe Verbindungen scheitern jedoch. Die Filterprüfung läuft so ab:

```
# nftables – Regeln und Counter
nft list ruleset -a
nft list chain inet filter input | grep -E '8443|drop|reject|accept'

# firewalld – Zone und Port
firewall-cmd --get-active-zones
firewall-cmd --list-all
firewall-cmd --query-port=8443/tcp
firewall-cmd --permanent --query-port=8443/tcp

# ufw – falls aktiv
ufw status numbered | grep 8443
```

Eine Accept-Regel mit steigendem Counter und eine passende Zonen-Zuordnung sprechen für eine intakte Filterkonfiguration. Fehlt die Regel oder steigt der Counter einer Drop-Regel, liegt die Ursache in der lokalen Policy. Bei `firewalld` muss zusätzlich geprüft werden, ob die Regel sowohl runtime als auch permanent gesetzt ist.

Auf Systemen, die gleichzeitig routen oder bridgen, greift die Forward-Kette. Eine Allow-Regel in der Input-Kette genügt dann nicht. Der Verkehr, der den Host nur durchquert, wird separat gefiltert und muss eigens freigegeben werden.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

```
nft list chain inet filter forward
firewall-cmd --list-all --zone=internal
```

> Forward-Regeln werden leicht übersehen, weil die meisten Diagnose-Skripte nur die Input-Kette auslesen. Auf Router-, Bridge- oder Container-Hosts führt das zu falschen Entwarnungen.

Dienst- und Port-Erreichbarkeit von außen

Die lokale Bestätigung eines offenen Ports und freigegebener Filterregeln reicht nicht aus. Erst der Test von einem entfernten System zeigt, ob der Dienst tatsächlich von außen erreichbar ist. Unterschiede zwischen lokalem Listen-Zustand und externer Erreichbarkeit entstehen durch vorgelagerte Filter, asymmetrische Routen oder Bindungen, die nur den lokalen Verkehr zulassen.

`nc` eignet sich als schneller Verbindungsprüfer. Im Zero-Modus baut das Werkzeug lediglich den TCP-Handshake auf und beendet die Verbindung sofort wieder. Die Rückmeldung ist eindeutig: Erfolg oder Timeout bzw. Connection refused. Für UDP-Dienste wird der Schalter `-u` gesetzt. Der Timeout lässt sich mit `-w` begrenzen, damit hängende Tests nicht unnötig lange blockieren.

```
nc -zv 10.50.1.20 8443
nc -zv -w 4 10.50.1.20 443
nc -zvu -w 3 10.50.1.20 53
nc -zv app.internal.example 8080
```

Schlägt der Versuch von einem entfernten Host fehl, während derselbe Befehl auf dem Zielsystem selbst erfolgreich ist, liegt die Störung auf dem Weg dazwischen.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

`nmap` liefert neben dem reinen Verbindungsstatus zusätzliche Zustandsinformationen. Der Connect-Scan (`-sT`) arbeitet ohne Root-Rechte und meldet `open` , `closed` oder `filtered` . Der SYN-Scan (`-sS`) ist präziser, erfordert jedoch erhöhte Privilegien. Mit `-Pn` wird die Host-Discovery übersprungen, falls ICMP unterwegs verworfen wird. Die Service-Erkennung (`-sV`) versucht zudem, das auf dem Port laufende Protokoll zu identifizieren.

```
nmap -p 8443 10.50.1.20
nmap -sT -Pn -p 80,443,8443 10.50.1.20
nmap -sS -p 8443 10.50.1.20
nmap -sV -p 8443 10.50.1.20
```

`closed` bedeutet, dass ein Reset zurückkam – der Host antwortet, der Port ist jedoch nicht belegt. `filtered` signalisiert ausbleibende Antworten und damit eine Filterung auf dem Pfad oder am Ziel.

Für HTTP- und HTTPS-Endpunkte ergänzt `curl` die Transportebene um die Anwendungsschicht. Verbindungsfehler, TLS-Probleme und HTTP-Statuscodes werden getrennt sichtbar. Die Option `--connect-timeout` verhindert lange Wartezeiten, `--resolve` erzwingt die Nutzung einer bestimmten IP und umgeht DNS-Einflüsse beim Test.

```
curl -v --connect-timeout 5 https://10.50.1.20:8443/
curl -v -o /dev/null -w "%{http_code}\n" http://10.50.1.20:8080/health
curl -v --resolve api.example.com:8443:10.50.1.20 https://api.example.com:8443/
```

> ⚠ Lokale Erfolge mit `ss` und gleichzeitigem Fehlschlag von `nc` oder `nmap` von außen weisen auf eine Störung hin, die außerhalb des Zielhosts liegt. Die weitere Suche muss dann auf dem Pfad oder an zwischengeschalteten Systemen fortgesetzt werden.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

Die Wahl des Testsystems beeinflusst das Ergebnis spürbar. Ein Host im selben Layer-2-Segment sieht andere Filter und Routen als ein System in einem entfernten Netz oder hinter einem NAT-Gateway. Deshalb sollten Tests von mindestens zwei unterschiedlichen Standorten aus durchgeführt werden, sofern die Topologie das erlaubt.

uch die verwendete Quelladresse kann relevant sein, wenn auf dem Ziel oder dazwischen zugriffslistenbasierte Regeln greifen.

```
Externe Prüfung – sinnvolle Kombination
──────────────────────────────────────
1. nc -zv          → schneller Connect-Test
2. nmap -sT/-sS    → Zustandsinformation
3. curl -v         → Anwendungsschicht (HTTP/S)
4. Vergleich       → gleiches vs. fremdes Subnetz
```

> □□ Ein Port kann in `nmap` als `open` erscheinen und dennoch auf Anwendungsebene fehlerhaft reagieren. Erst der protokollspezifische Test mit `curl` oder einem geeigneten Client liefert die endgültige Aussage über die Dienstverfügbarkeit.

□□ Praktisches Beispiel

Ein Dienst auf Port 8443 ist lokal gebunden und in den Filterregeln freigegeben. Clients aus dem Unternehmensnetz melden Verbindungsfehler. Die externe Prüfung wird von zwei unterschiedlichen Hosts aus durchgeführt:

```
# Test von Host A (gleiches Subnetz)
nc -zv -w 3 10.50.1.20 8443
nmap -sT -Pn -p 8443 10.50.1.20
```

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

```
# Test von Host B (anderes Netz)
nc -zv -w 3 10.50.1.20 8443
nmap -sT -Pn -p 8443 10.50.1.20
nmap -sV -p 8443 10.50.1.20

# Anwendungstest
curl -v --connect-timeout 5 https://10.50.1.20:8443/health
curl -v --resolve service.example.com:8443:10.50.1.20
https://service.example.com:8443/health
```

Erfolgreiche Connect-Tests von beiden Seiten bestätigen die Transportebene. Scheitert anschließend der `curl`-Aufruf, liegt die Ursache in TLS, Zertifikat oder Anwendungslogik. Bleibt `nc` bereits ohne Verbindung, muss der Pfad zwischen den beteiligten Hosts erneut untersucht werden.

Auf Systemen mit mehreren Adressen oder Interfaces kann die Antwort über einen anderen Weg zurücklaufen als den Hinweg. In solchen Fällen kommen die SYN-Pakete an, die SYN-ACK-Pakete verlassen den Host jedoch über ein Interface, das der Client nicht erwartet. Der Vergleich der Paketflüsse auf dem Zielhost mit `tcpdump` macht diese Asymmetrie sichtbar.

> Tests, die ausschließlich vom Zielhost selbst aus gestartet werden, blenden systematisch alle Probleme aus, die nur auf dem Weg zwischen Client und Server auftreten.

Paketanalyse, Logs und erweiterte Diagnose

Gezielte Captures

Wenn externe Erreichbarkeitstests und lokale Prüfungen keine eindeutige Ursache liefern, bleibt der Blick in die tatsächlichen Pakete. `tcpdump` erfasst den Verkehr auf dem gewünschten Interface und macht sichtbar, ob SYN-Pakete ankommen, ob Antworten den Host verlassen und welche Flags oder Optionen gesetzt sind. Die Kunst liegt darin, die Capture-Filter so eng zu setzen, dass nur der relevante Verkehr erscheint und die Ausgabe auswertbar bleibt.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

Der grundlegende Aufruf spezifiziert das Interface und einen Filterausdruck. Ohne Filter wird schnell zu viel erfasst. Mit `-n` werden Adressen und Ports numerisch dargestellt, was die Ausgabe beschleunigt und DNS-Nachfragen während des Captures verhindert. `-nn` unterdrückt zusätzlich die Auflösung von Portnamen. Für die spätere Analyse in Wireshark oder tshark wird die Ausgabe in eine Datei geschrieben (`-w`).

```
tcpdump -i eth0 -n port 8443
tcpdump -i any -nn host 10.50.1.20 and port 8443
tcpdump -i eth0 -nn -w /tmp/capture.pcap host 10.40.1.15 and port 443
```

Filterausdrücke folgen der BPF-Syntax. Sie können nach Host, Netz, Port, Protokoll und Richtungsangaben kombiniert werden. Die Operatoren `and`, `or` und `not` erlauben präzise Einschränkungen. Für TCP-Handshakes sind Flags besonders nützlich: `tcp[tcpflags] & tcp-syn != 0` erfasst SYN-Pakete, weitere Flags lassen sich analog ansprechen.

```
tcpdump -i eth0 -nn 'tcp port 8443 and (tcp[tcpflags] & tcp-syn != 0)'
tcpdump -i eth0 -nn 'host 10.50.1.20 and (port 80 or port 443)'
tcpdump -i eth0 -nn 'src net 10.40.0.0/16 and dst port 8443'
tcpdump -i eth0 -nn 'tcp[tcpflags] & (tcp-syn|tcp-ack) != 0'
```

Die Option `-c` begrenzt die Anzahl der erfassten Pakete und verhindert unkontrolliert wachsende Dateien. `-s 0` oder `-s 65535` stellt sicher, dass vollständige Pakete und nicht nur Header gespeichert werden. Bei Verdacht auf Fragmentierung oder ungewöhnliche Optionen ist das unverzichtbar. Für länger laufende Captures empfiehlt sich ein Ringpuffer über `-C` und `-W`, damit ältere Dateien automatisch überschrieben werden.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

```
tcpdump -i eth0 -nn -c 100 -w /tmp/short.pcap port 8443
tcpdump -i eth0 -nn -s 0 -w /tmp/full.pcap host 10.50.1.20
tcpdump -i eth0 -nn -C 50 -W 5 -w /tmp/ring.pcap port 443
```

> ⚠ Zu breite Filter erzeugen schnell unübersichtliche Mengen an Paketen und machen die Auswertung unnötig schwer. Jeder Capture sollte auf den konkreten Verdachtsmoment und die beteiligten Adressen sowie Ports begrenzt werden.

Das Lesen einer gespeicherten Capture-Datei erfolgt ebenfalls mit `tcpdump`. Dieselbe Filtersyntax kann nachträglich angewendet werden, um die gespeicherten Daten weiter einzuschränken. Die Option `-tttt` erzeugt lesbare Zeitstempel, `-X` oder `-A` zeigt den Payload in Hex bzw. ASCII.

```
tcpdump -nn -r /tmp/capture.pcap
tcpdump -nn -tttt -r /tmp/capture.pcap 'tcp[tcpflags] & tcp-syn != 0'
tcpdump -nn -A -r /tmp/capture.pcap port 80
tcpdump -nn -X -c 20 -r /tmp/capture.pcap
```

Bei der Interpretation zählt die Reihenfolge der Pakete. Ein ankommendes SYN ohne nachfolgendes SYN-ACK deutet auf eine Filterung oder einen nicht antwortenden Dienst hin. Ein SYN-ACK, das den Host verlässt, aber beim Client nie ankommt, verweist auf Probleme auf dem Rückweg. Doppelte SYN-Pakete oder unerwartete RST-Flags liefern weitere Hinweise auf Timeout-Verhalten oder aktive Zurückweisungen.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

□□ Die Kombination aus engem Filter, begrenzter Paketanzahl und anschließender gezielter Auswertung der pcap-Datei verhindert, dass relevante Informationen in einer Flut irrelevanter Pakete untergehen.

□□ Praktisches Beispiel

Externe Clients erreichen Port 8443 nicht, obwohl der Dienst lokal lauscht und die Firewall den Port freigibt. Der Capture wird auf dem Zielhost gestartet, während ein Client den Verbindungsversuch ausführt:

```
# Capture starten (in einem separaten Terminal)
tcpdump -i eth0 -nn -s 0 -w /tmp/8443.pcap host 10.40.1.15 and port 8443

# Parallel vom Client:
nc -zv 10.50.1.20 8443

# Capture beenden (Ctrl+C) und auswerten
tcpdump -nn -tttt -r /tmp/8443.pcap
tcpdump -nn -r /tmp/8443.pcap 'tcp[tcpflags] & tcp-syn != 0'
tcpdump -nn -r /tmp/8443.pcap 'tcp[tcpflags] & tcp-ack != 0'
```

Erscheinen SYN-Pakete vom Client, aber keine SYN-ACK-Pakete vom Server, liegt die Ursache auf dem Zielhost selbst (Dienst, lokale Filter oder Bindung). Verlassen SYN-ACK-Pakete den Host, erreichen den Client jedoch nicht, muss der Rückweg untersucht werden. Fehlen bereits die SYN-Pakete, greift eine Filterung vor dem Zielhost.

Für UDP-Dienste ändert sich die Betrachtungsweise, weil es keinen Handshake gibt. Hier werden Anfrage- und mögliche Antwortpakete direkt verglichen. Auch die Größe der Pakete und mögliche ICMP-Fehler (Fragmentation needed, Port unreachable) liefern Hinweise.

```
tcpdump -i eth0 -nn -s 0 -w /tmp/dns.pcap port 53
tcpdump -nn -r /tmp/dns.pcap
```

> Captures ohne vorherige Eingrenzung auf Host und Port erzeugen Dateien, die in der Praxis kaum noch sinnvoll auszuwerten sind und zudem sensible Daten enthalten können.

Log- und Kernel-Analyse

Paketmitschnitte zeigen den Verkehr auf dem Draht. Die System- und Kernel-Logs erklären, warum dieser Verkehr zustande kommt oder abreißt. Beide Informationsquellen gehören zusammen: Ein fehlendes SYN-ACK in der Capture-Datei wird erst dann verständlich, wenn der Kernel zeitgleich einen Treiberfehler, einen Netfilter-Drop oder einen Link-Verlust meldet.

`journalctl` ist der zentrale Einstieg auf Systemen mit systemd. Die Einheit `NetworkManager`, `systemd-networkd` oder der jeweilige Dienst können gezielt abgefragt werden. Zeitfilter grenzen die Ausgabe auf den relevanten Zeitraum ein. Die Option `-k` beschränkt die Anzeige auf Kernel-Meldungen, `-b` zeigt nur den aktuellen Boot.

```
journalctl -u NetworkManager --since "10 min ago"
journalctl -u systemd-networkd -b
journalctl -k --since "2024-01-01 14:30:00" --until "2024-01-01 14:45:00"
journalctl -k -b | grep -iE 'eth0|link|carrier|netfilter|nf_'
```

Für netzwerkspezifische Ereignisse lohnt die Suche nach Begriffen wie `link`, `carrier`, `timeout`, `refused` oder dem Interface-Namen. Viele Treiber schreiben hier die Gründe für einen Link-Down oder für zurückgewiesene Pakete.

`dmesg` liefert denselben Kernel-Ringpuffer, oft etwas direkter und mit lesbaren Zeitstempeln über `-T`. Die Ausgabe kann nach demselben Muster gefiltert werden. Bei Verdacht auf Treiberprobleme oder Firmware-Meldungen ist `dmesg` häufig der schnellere Weg.

```
dmesg -T | tail -n 100
dmesg -T | grep -iE 'error|fail|timeout|eth0|enp'
dmesg -T --level=err,warn
```

Netfilter selbst schreibt bei aktiviertem Logging eigene Einträge. Regeln mit dem Target `LOG` oder `NFLOG` erzeugen Meldungen, die in den Kernel-Logs oder über `journalctl` sichtbar werden. Die Priorität und das Prefix der Log-Regel bestimmen, wie leicht die Einträge später zu finden sind.

```
journalctl -k | grep -i 'NETFILTER\|NFLOG\|DROP'
dmesg -T | grep -i 'DROP\|REJECT'
```

> ⚠ Kernel-Meldungen zu Link-Verlusten oder Treiber-Resets erscheinen oft nur kurz und werden bei hohem Log-Aufkommen schnell aus dem Ringpuffer verdrängt. Die Erfassung muss daher zeitnah zum beobachteten Symptom erfolgen.

Die Korrelation mit dem Paketmitschnitt erfolgt über Zeitstempel. Ein SYN-Paket, das in der Capture-Datei ankommt, und eine zeitgleiche Kernel-Meldung über einen Drop in der Input-Kette erklären den fehlenden SYN-ACK. Ebenso können Treiber-Meldungen über CRC-Fehler oder Overruns den Verlust von Paketen erklären, die `tcpdump` bereits nicht mehr gesehen hat.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

```
Log-Quellen und ihr Fokus
_________________________

journalctl -u <dienst>   → Dienst- und Network-Manager-Ereignisse
journalctl -k / dmesg    → Treiber, Link, Netfilter, Hardware
Dienst-Logs              → Anwendungsfehler, Bind-Probleme
```

> □□ Die Kombination aus engem Zeitfilter und gezielter Begriffssuche reduziert die Log-Menge auf die Einträge, die zum konkreten Vorfall gehören. Ohne diese Eingrenzung gehen relevante Meldungen in der allgemeinen Systemaktivität unter.

□□ Praktisches Beispiel

Ein Dienst auf Port 8443 antwortet intermittierend nicht. Der Paketmitschnitt zeigt ankommende SYN-Pakete ohne SYN-ACK. Parallel werden die Logs des relevanten Zeitfensters geprüft:

```
# Kernel- und Netfilter-Meldungen im Zeitraum
journalctl -k --since "14:32:00" --until "14:35:00" | grep -iE
'eth0|drop|reject|link|error'

# Network-Manager bzw. networkd
journalctl -u systemd-networkd --since "14:32:00" --until "14:35:00"
journalctl -u NetworkManager --since "14:32:00" --until "14:35:00"

# dmesg mit lesbaren Zeitstempeln
dmesg -T | grep -iE 'eth0|enp|error|fail|timeout' | tail -n 50

# Dienst-Log parallel
journalctl -u myapi.service --since "14:32:00" --until "14:35:00"
```

Erscheinen im selben Zeitfenster Meldungen über Carrier-Verlust oder Netfilter-Drops, ist die Ursache gefunden. Fehlen solche Einträge, während der Dienst selbst Fehler beim Annehmen von Verbindungen meldet, liegt das Problem in der Anwendung oder in Ressourcenengpässen.

Treiber- und Firmware-Meldungen verdienen besondere Aufmerksamkeit. Viele Netzwerkkarten schreiben bei Überlastung, fehlerhafter Autonegotiation oder defekten Kabeln wiederkehrende Einträge. Diese Meldungen korrelieren oft mit steigenden Fehlerzählern auf dem Interface und erklären intermittierende Verluste, die in reinen Connectivity-Tests nur sporadisch sichtbar werden.

```
dmesg -T | grep -iE 'firmware|reset|watchdog|tx_|rx_|overrun'
ethtool -S eth0 | grep -iE 'error|fail|drop|crc|over'
```

> Logs, die ohne Bezug zum konkreten Zeitpunkt und ohne parallelen Paketmitschnitt gelesen werden, führen leicht zu falschen Ursachenzuschreibungen, weil sie Ereignisse aus völlig anderen Kontexten enthalten.

Fehlerzähler, Performance und MTU-Probleme

Quantitative Zähler auf Interface- und Protokollebene machen sichtbar, was einzelne Paketmitschnitte und Log-Ausschnitte nur andeuten. Sie beantworten die Frage, wie häufig und in welchem Umfang Fehler, Verluste oder Begrenzungen auftreten.

Hardware-nahe Zähler liefert `ethtool -S`. Hier erscheinen CRC-Fehler, Alignment-Probleme, Overruns, missed packets und Collision-Zähler der Netzwerkkarte. Diese Werte stammen direkt aus den Registern des Controllers und bleiben unabhängig von der Kernel-Statistik. Parallel dazu führt `ip -s link` die softwareseitigen RX- und TX-Zähler, inklusive Drop- und Error-Werten, die der Kernel selbst verbucht.

```
ethtool -S eth0
ethtool -S eth0 | grep -E 'crc|align|over|miss|error|drop|coll|fail'
ip -s link show eth0
ip -s -s link show eth0
```

Aussagekräftig wird die Betrachtung erst durch wiederholte Abfragen. Zwei Messungen im Abstand von ein bis fünf Minuten und die Differenz der relevanten Felder zeigen, ob die Zähler aktuell weiterlaufen. Ein hoher Absolutwert aus der Vergangenheit ist bedeutungslos, wenn er sich im Beobachtungszeitraum nicht verändert.

Auf TCP-Ebene liefert `ss` die Retransmit- und Congestion-Informationen pro Verbindung. Die Optionen `-ti` oder `-tai` blenden RTO, Retransmits, Congestion Window und weitere Variablen ein. Häufen sich Retransmits bei gleichzeitig stabilen Interface-Zählern, liegt der Verlust mit hoher Wahrscheinlichkeit auf dem Pfad und nicht am lokalen Host.

```
ss -ti
ss -tai
ss -ti state established
ss -tai dst 10.50.1.20 | grep -E 'retrans|rto|cwnd|rtt'
```

MTU-Begrenzungen erzeugen eine eigene Klasse von Symptomen. Zu große Pakete werden verworfen, wenn zwischengeschaltete Systeme keine Fragmentierung erlauben und ICMP-Meldungen unterdrückt werden. `tracepath` ermittelt die Pfad-MTU, `ping` mit gesetztem Don’t-Fragment-Bit und definierter Größe prüft konkrete Schwellenwerte.

Quelle: https://admindocs.de/de/netzwerk/netzwerkprobleme-unter-linux-systematisch-analysieren

```
tracepath -n 10.50.1.20
ping -c 4 -M do -s 1472 10.50.1.20
ping -c 4 -M do -s 1400 10.50.1.20
ping -c 4 -M do -s 1200 10.50.1.20
ip link show eth0
```

Der Wert 1472 testet die klassische Ethernet-MTU von 1500 Byte abzüglich der Header. Scheitert dieser Test und kleinere Größen passieren, existiert eine MTU-Einschränkung auf dem Pfad.

> ⚠ CRC- und Alignment-Fehler steigen in der Regel nur bei physischen Problemen: beschädigtes Kabel, defekter Stecker, fehlerhafter Switch-Port oder defekte Netzwerkkarte. Reine Software-Eingriffe lassen diese Zähler unberührt.

Weiterführende Performance-Indikatoren finden sich in den Softnet-Statistiken und den Queue-Disciplines. `/proc/net/softnet_stat` zeigt verworfene Pakete in den Empfangs-Queues der CPUs. `tc qdisc show` listet die aktive Warteschlangenstrategie und mögliche Drop-Verhaltensweisen. `nstat` aggregiert Protokoll-Zähler wie TcpRetransSegs oder IpInDiscards.

```
cat /proc/net/softnet_stat
tc qdisc show dev eth0
nstat -az | grep -E 'TcpRetrans|TcpLoss|IpInDiscards|Ip6InDiscards'
```

> □□ Die Steigungsrate der Zähler innerhalb eines definierten Zeitfensters ist der eigentliche Indikator. Absolute Werte ohne Bezug zum Beobachtungszeitraum führen regelmäßig zu

> Fehlinterpretationen.

Praktisches Beispiel

Sporadische Abbrüche treten auf, Captures zeigen vereinzelte Retransmits. Die Zähler werden vor und während der Reproduktion gelesen:

```
# Messpunkt 1
ethtool -S eth0 | grep -E 'crc|error|drop|over|miss' > /tmp/e1.txt
ip -s link show eth0 > /tmp/i1.txt
ss -ti > /tmp/s1.txt
nstat -az > /tmp/n1.txt

# Störung reproduzieren, danach Messpunkt 2
ethtool -S eth0 | grep -E 'crc|error|drop|over|miss' > /tmp/e2.txt
ip -s link show eth0 > /tmp/i2.txt
ss -ti > /tmp/s2.txt
nstat -az > /tmp/n2.txt

# Vergleiche
diff /tmp/e1.txt /tmp/e2.txt
diff /tmp/i1.txt /tmp/i2.txt
diff /tmp/n1.txt /tmp/n2.txt | grep -E 'TcpRetrans|Discards'

# MTU-Prüfung
ping -M do -s 1472 -c 5 10.50.1.20
tracepath -n 10.50.1.20
```

Laufen die CRC- oder Drop-Zähler während der Störung hoch, liegt die Ursache am Interface oder am Medium. Bleiben sie konstant und steigen nur die TCP-Retransmits, liegt der Verlust weiter entfernt. Scheitert der DF-Ping mit 1472 Byte, muss die MTU auf dem Pfad oder an Tunnel-Endpunkten angepasst werden.

Bei Bonding- oder VLAN-Konfigurationen sind die Zähler auf jeder beteiligten Ebene zu lesen. Fehler, die nur auf dem logischen Bond-Device erscheinen, deuten auf Asymmetrien zwischen

den Slave-Interfaces oder auf Probleme in der Bonding-Logik hin.

```
ethtool -S bond0
ethtool -S eth0
ethtool -S eth1
ip -s link show bond0
ip -s link show eth0
```

> Zählerstände ohne vorherige und nachfolgende Messung im konkreten Zeitfenster des Problems erlauben keine zuverlässige Aussage über die aktuelle Fehleraktivität.

Wichtige Ressourcen und Checklisten

Notfall-Checkliste: Bottom-up in 6 Schritten

Wenn im laufenden Betrieb ein Netzwerkproblem gemeldet wird, gehe die Diagnose ohne Hektik entlang dieser Reihenfolge durch:

Schritt	Diagnoseebene	Empfohlene Befehle	Prüfziel / Fokus
Link & Hardware	Physikalisch & Driver	`ip -br link` `ethtool eth0`	Carrier `LOWER_UP`? Duplex & Speed korrekt?
IP & Gateway	Netzwerkschicht & Routing	`ip -br addr` `ip route get <Ziel>`	IP vorhanden? Gültige Default-Route gesetzt?
Local Listener	Transport & Sockets	`ss -tulpn` `ss -tlnp '( sport = :<Port> )'`	Lauscht Dienst auf `0.0.0.0` (nicht nur `127.0.0.1`)?

Schritt	Diagnoseebene	Empfohlene Befehle	Prüfziel / Fokus
Pfad & Transport	Erreichbarkeit & Port	`ping -c 3 <IP>` `nc -zv <Ziel> <Port>` `mtr -rwc 30 <Ziel>`	Erreichbarkeit von DNS getrennt? Port offen?
Firewall & Rules	Paketfilter & Policies	`nft list ruleset` `firewall-cmd --list-all`	Steigen Zähler an Drop-/Reject-Regeln?
Captures & Logs	Paketfluss & Kernel	`tcpdump -i any -nn host <IP>` `journalctl -k -b`	Kommen SYN-Pakete an? Treiber- / Drop-Logs?

Referenz-Tools: Moderne Bordmittel vs. Legacy

Auf modernen Linux-Distributionen solltest du veraltete Befehle meiden, da sie virtuelle Netzwerkstacks, Namespaces oder Bonding-Interfaces oft nicht korrekt anzeigen:

Diagnosebereich	Modernes Tool (Empfohlen)	Veraltet (Meiden)	Wichtige Man-Page
Interfaces & IP	`ip link`, `ip addr`	`ifconfig`	`man ip-link`, `man ip-address`
Routing & Tabellen	`ip route`, `ip neigh`	`route`, `arp`	`man ip-route`, `man ip-neighbour`
Sockets & Sockets	`ss -tulpn`	`netstat`	`man ss`
Hardware & Speed	`ethtool eth0`	`mii-tool`	`man ethtool`
Hop & Pfadanalyse	`mtr`, `tracepath`	`traceroute`	`man mtr`, `man tracepath`
Paket-Captures	`tcpdump`, `tshark`	`ethereal`	`man tcpdump`
Filter & Firewall	`nft`, `firewall-cmd`	`iptables`	`man nft`

Empfohlene Man-Pages und Dokumentation

Für tiefergehende Konfigurationsdetails und Kernel-Parameter lohnen sich folgende Referenzen:

- **Man-Pages im Terminal**:

- `man 8 ip` – Die umfassende Dokumentation der `iproute2`-Suite.
- `man 8 ss` – Detaillierte Beschreibung aller Filter-Ausdrücke für Sockets.
- `man 8 tcpdump` – BPF-Filter-Syntax und Anzeigeoptionen.
- `man 8 nft` – Syntax und Tabellenstruktur moderner `nftables`-Setups.

Kernel-Dokumentation:

- Linux Kernel Networking Documentation

Praxis-Checklisten für den täglichen Einsatz

Bei akuten Störungen helfen diese kurzen Ablaufpläne, um den Fehler ohne Suchen direkt auf die richtige Schicht einzugrenzen:

Szenario 1: Dienst läuft, ist aber von außen nicht erreichbar

- `ss -tlnp '( sport = :<Port> )'` – Prüfen, ob Bindung auf `0.0.0.0` (nicht nur `127.0.0.1`) steht.
- `nft list ruleset` / `firewall-cmd --list-all` – Prüfen, ob Port in der `input`-Kette / Zone erlaubt ist.
- `nc -zv <Server-IP> <Port>` (von externem Host) – Prüfen, ob der Handshake durchkommt.
- `tcpdump -i any -nn host <Client-IP> and port <Port>` – Prüfen, ob `SYN` am Server ankommt.

Szenario 2: Namensauflösung schlägt fehl oder ist langsam

- `ping -c 2 <IP>` vs. `ping -c 2 <Hostname>` – Erreichbarkeit von Namensauflösung trennen.
- `cat /etc/resolv.conf` | `resolvectl status` – Konfigurierte Nameserver & Status prüfen.
- `dig +short <Hostname>` vs. `dig @1.1.1.1 <Hostname>` – Lokalen vs. öffentlichen DNS-Server vergleichen.
- `time dig <Hostname>` – Latenz und Timeouts beim primären DNS prüfen.

Szenario 3: Sporadische Abbrüche oder Paketverlust

- `ethtool eth0` & `ethtool -S eth0 | grep -iE 'err|drop|crc'` – Physischen Link & Fehlerzähler prüfen.
- `mtr -rwc 50 -n <Ziel-IP>` – Dauerhaften Verlust pro Hop isolieren (ICMP vs. TCP `--tcp`).
- `ping -M do -s 1472 <Ziel-IP>` – Pfad-MTU auf Fragmentierungsprobleme testen.
- `ss -ti` – TCP-Retransmits und RTO der aktiven Verbindung beobachten.

Weiterführende Quellen und Community-Ressourcen

Dokumentation & Spezifikationen:

Linux Kernel Networking Documentation RFC 793 (TCP Specification) Red Hat Enterprise Linux Network Guide

Blogs:

Brendan Gregg's Blog Cloudflare Networking Blog

Communities & Mailinglisten:

Netdev Mailing List r/networking r/sysadmin