9.3 UDP-Diagnose und ICMP-Fehlermeldungen

Ziel dieser Seite

Diese Seite beschreibt die systematische Diagnose von UDP-Kommunikation und zugehörigen ICMP- beziehungsweise ICMPv6-Fehlermeldungen.

Nach der Bearbeitung muss unterschieden werden können:

- ob ein lokaler UDP-Endpunkt vorhanden ist;
- welcher Prozess den UDP-Port besitzt;
- ob ein Datagramm den Client tatsächlich verlässt;
- ob es den vorgesehenen Server erreicht;
- ob die Serveranwendung das Datagramm verarbeitet;
- ob eine Antwort erzeugt und zurückgesendet wird;
- ob eine Firewall, NAT-Instanz oder andere Zwischenkomponente eingreift;
- ob eine ICMP-Fehlermeldung zurückkommt;
- ob ein Port geschlossen oder lediglich nicht eindeutig prüfbar ist;
- ob Paketgröße, Fragmentierung oder Path MTU beteiligt sind;
- ob Unicast, Broadcast oder Multicast verwendet wird;
- ob der richtige Host-, Container- oder Pod-Namespace untersucht wird.

UDP besitzt keinen Handshake und keine integrierte Empfangsbestätigung. Ein erfolgreich abgesendetes Datagramm beweist deshalb nicht, dass es den Zielhost oder die Zielanwendung erreicht hat.

Sicherheits- und Wirkungsklassen

Kennzeichnung	Bedeutung
LESEND	Erfasst ausschließlich vorhandene Zustände.
NETZAKTIV	Erzeugt Netzwerkverkehr zum geprüften Ziel.
SENSITIV	Kann interne Adressen, Verbindungen oder Nutzdaten sichtbar machen.
ÄNDERND	Verändert Konfiguration oder Laufzeitzustand.
AUSFALLRISIKO	Kann Verbindungen, Dienste oder Netzwerkkomponenten beeinträchtigen.

UDP-Dienste dürfen nur mit gültigen, erwarteten und autorisierten Anfragen getestet werden. Unkontrollierte oder sehr schnelle UDP-Anfragen können:

- Dienste überlasten;
- Rate Limits auslösen;
- Intrusion-Detection-Systeme alarmieren;

- Verstärkungsangriffe begünstigen;
- große Antwortmengen erzeugen;
- produktive Protokollzustände beeinflussen.

Grundlegende Eigenschaften von UDP

UDP ist ein minimales, nachrichtenorientiertes Transportprotokoll.

UDP bietet:

- Quell- und Zielport;
- Datagrammgrenzen;
- Längenfeld;
- Prüfsumme;
- geringen Protokolloverhead;
- Unicast-, Broadcast- und Multicast-Kommunikation.

UDP bietet selbst nicht:

- Verbindungsaufbau;
- Verbindungsabbau;
- Empfangsbestätigung;
- Sequenznummern;
- automatische Wiederholung;
- Reihenfolgegarantie;
- Duplikatschutz;
- Flusskontrolle;
- Überlastkontrolle;
- Sitzungserkennung;
- zuverlässige Erkennung einer ausgefallenen Gegenstelle.

Eine Anwendung oder ein über UDP betriebenes Protokoll kann solche Funktionen selbst implementieren. Beispiele sind:

- DNS mit eigenen Wiederholungen und TCP-Fallback;
- QUIC mit eigener Zuverlässigkeit und Überlastkontrolle;
- anwendungsspezifische Sequenznummern;
- Zeitstempel;
- Request-IDs;
- Heartbeats;
- Bestätigungsdatagramme.

TCP und UDP vergleichen

Merkmal	TCP	UDP

Verbindungsaufbau	Three-Way Handshake	kein Handshake
Datenmodell	Bytestrom	einzelne Datagramme
Bestätigung	TCP-ACK	nicht in UDP enthalten
Reihenfolge	durch TCP sichergestellt	nicht sichergestellt
Wiederholung	durch TCP	nur durch Anwendung
Flusskontrolle	vorhanden	nicht vorhanden
Überlastkontrolle	vorhanden	nicht in UDP selbst
Verbindungszustände	unter anderem `LISTEN`, `ESTABLISHED`	keine entsprechenden TCP-Zustände
Reaktion bei geschlossenem Port	normalerweise TCP Reset	möglicherweise ICMP Port Unreachable
Porttest	TCP-Handshake liefert klares Ergebnis	ohne Protokollantwort häufig mehrdeutig

Ein Test von TCP-Port `53` beweist nicht, dass UDP-Port `53` funktioniert. Umgekehrt beweist eine erfolgreiche UDP-DNS-Abfrage nicht automatisch, dass DNS über TCP funktioniert.

UDP-Header

Der UDP-Header besitzt vier Felder:

```
Quellport
Zielport
Länge
Prüfsumme
```

Die Mindestlänge eines UDP-Datagramms beträgt acht Byte, weil der UDP-Header selbst acht Byte groß ist.

Feld	Größe	Bedeutung
Source Port	16 Bit	Port des Senders
Destination Port	16 Bit	Port des Empfängers
Length	16 Bit	Gesamtlänge von UDP-Header und UDP-Nutzdaten
Checksum	16 Bit	Prüfsumme über relevante IP-, UDP- und Nutzdatenfelder

UDP enthält keine Felder für:

- Sequenznummer;

- Bestätigungsnummer;
- Empfangsfenster;
- SYN;
- FIN;
- RST.

Bei IPv4 kann eine UDP-Prüfsumme gemäß der ursprünglichen UDP-Spezifikation als nicht verwendet gekennzeichnet sein. Bei IPv6 ist eine gültige UDP-Prüfsumme grundsätzlich erforderlich; nur eng definierte Sonderfälle besitzen Ausnahmen.

UDP-Kommunikation eindeutig bestimmen

Ein UDP-Datagramm wird durch folgende Angaben eingeordnet:

```
Quell-IP-Adresse
Quellport
Ziel-IP-Adresse
Zielport
Transportprotokoll UDP
```

Beispiel:

```
Client: 192.0.2.100:53124
Server: 192.0.2.53:53
```

Anfrage:

```
192.0.2.100:53124 → 192.0.2.53:53 UDP
```

Antwort:

```
192.0.2.53:53 → 192.0.2.100:53124 UDP
```

Eine Stateful Firewall oder NAT-Instanz kann diese Angaben verwenden, um einen zeitlich begrenzten Pseudo-Sitzungszustand zu verwalten. UDP selbst erzeugt diesen Zustand nicht.

UDP besitzt keinen Listenerzustand wie TCP

Werkzeuge sprechen bei UDP teilweise trotzdem von „Listening“, weil ein Prozess einen UDP-Endpunkt gebunden hat.

Technisch bedeutet der Befund:

> Ein UDP-Socket ist an eine lokale Adresse und einen lokalen Port gebunden.

Der Befund beweist nicht:

- dass eine bestimmte Anfrage korrekt formatiert ist;
- dass die Anwendung das Datagramm liest;
- dass die Anwendung antwortet;
- dass der Rückweg funktioniert;
- dass die Firewall den Datenverkehr erlaubt;
- dass NAT die Antwort richtig zuordnet;
- dass der Dienst auf Anwendungsebene gesund ist.

Verbundener UDP-Socket

Eine Anwendung kann einen UDP-Socket mit einer Gegenstelle „verbinden“.

Dieser Vorgang führt nicht zu einem Netzwerk-Handshake. Das Betriebssystem kann dadurch lediglich:

- eine Standardzieladresse hinterlegen;
- eingehende Datagramme auf eine Gegenstelle begrenzen;
- bestimmte ICMP-Fehler dem Socket zuordnen;
- normale `send`- und `receive`-Aufrufe ermöglichen.

Ein als verbunden dargestellter UDP-Socket beweist deshalb keine erreichbare Gegenstelle.

Was ein erfolgreicher send-Aufruf beweist

Wenn eine Anwendung ein UDP-Datagramm erfolgreich an das Betriebssystem übergibt, bedeutet das zunächst:

> Der lokale Netzwerkstack hat das Datagramm zur Übertragung angenommen.

Es beweist nicht:

- dass die Netzwerkkarte es gesendet hat;
- dass es die lokale Firewall passiert hat;
- dass es den Zielhost erreicht hat;
- dass ein Prozess am Zielport lauscht;
- dass die Zielanwendung es verarbeitet hat;
- dass eine Antwort erzeugt wurde;

- dass die Antwort den Client erreicht.

Für einen belastbaren Nachweis sind Protokollantwort, Serverprotokoll oder korrelierte Paketaufzeichnung notwendig.

Erwarteten UDP-Ablauf festlegen

Vor der Diagnose müssen mindestens folgende Angaben dokumentiert werden:

```
Anwendung:
<Dienst oder Protokoll>

Quellhost:
<Hostname und IP-Adresse>

Zielhost:
<Hostname und IP-Adresse>

Quellport:
<fest oder dynamisch>

Zielport:
<UDP-Port>

Adressfamilie:
<IPv4 oder IPv6>

Anfrage:
<erwartete gültige Protokollnachricht>

Erwartete Antwort:
<Antworttyp oder bewusst keine Antwort>

Timeout:
<Anwendungsfrist>

Kommunikationsart:
<Unicast, Broadcast oder Multicast>

Zwischenkomponenten:
```

```
<Firewall, NAT, Load Balancer, Proxy oder Tunnel>
```

Bei einem Protokoll ohne Antwort muss der Nachweis auf der Empfängerseite erfolgen.

UDP-Diagnose benötigt ein protokollspezifisches Werkzeug

Ein allgemeiner UDP-Test kann häufig nur ein Datagramm senden. Er kann nicht sicher bestimmen, ob ein beliebiger UDP-Dienst korrekt arbeitet.

Geeignete Tests sind beispielsweise:

Protokoll	Geeigneter Nachweis
DNS	gültige DNS-Abfrage mit `Resolve-DnsName` oder `dig`
NTP	gültige Zeitabfrage mit vorgesehenem NTP-Werkzeug
SNMP	gültige, autorisierte SNMP-Abfrage
TFTP	kontrollierter TFTP-Protokolltest
Syslog	Empfang im vorgesehenen Logziel bestätigen
RADIUS	autorisierter Test mit passendem RADIUS-Werkzeug
QUIC/HTTP/3	HTTP/3-fähiger Client und Serverprotokolle
anwendungsspezifisches UDP	Herstellerclient, Testfunktion oder definierte Testnachricht

Ein zufälliges oder leeres Datagramm kann von einem korrekt funktionierenden Dienst absichtlich ignoriert werden.

DNS als UDP-Diagnosebeispiel

Windows:

NETZAKTIV

```
Resolve-DnsName `
  -Name "example.test" `
  -Type A `
  -Server "192.0.2.53" `
  -DnsOnly
```

Linux und macOS:

```
dig \
  @192.0.2.53 \
  example.test \
  A
```

Vergleich über TCP:

```
dig \
  @192.0.2.53 \
  example.test \
  A \
  +tcp
```

Auswertung:

UDP	TCP	Mögliche Einordnung
funktioniert	funktioniert	beide Transportwege grundsätzlich nutzbar
fehlerhaft	funktioniert	UDP-Filterung, Fragmentierung, EDNS oder MTU möglich
funktioniert	fehlerhaft	TCP-Port, Firewall oder TCP-Listener prüfen
beide fehlerhaft	beide Pfade oder DNS-Dienst betroffen	Server, Zone, Routing und Firewall prüfen

Der Vergleich ist protokollspezifisch. Er darf nicht auf beliebige UDP-Dienste übertragen werden.

Allgemeine UDP-Porttests richtig bewerten

Ein Aufruf wie:

```
nc -vzu 192.0.2.53 53
```

liefert bei UDP keinen gleichwertigen Nachweis wie ein erfolgreicher TCP-Handshake.

Mögliche Ergebnisse ohne Antwort:

- Port ist geöffnet, aber die Anwendung ignoriert die ungültige Anfrage;
- Port wird gefiltert;
- Datagramm erreicht den Server nicht;

- Antwort erreicht den Client nicht;
- ICMP-Fehler wird gefiltert;
- Zielhost ist ausgefallen;
- Anwendung antwortet grundsätzlich nicht;
- Werkzeug interpretiert fehlende Antwort als möglichen Erfolg.

`nc -u` darf deshalb nur zum gezielten Erzeugen eines autorisierten Datagramms verwendet werden. Für die Funktionsprüfung ist ein protokollspezifischer Test erforderlich.

Windows: UDP-Endpunkte erfassen

LESEND

Alle UDP-Endpunkte:

```
Get-NetUDPEndpoint |
  Sort-Object LocalPort, LocalAddress
```

Bestimmten UDP-Port prüfen:

```
Get-NetUDPEndpoint `
  -LocalPort 53
```

Wesentliche Felder anzeigen:

```
Get-NetUDPEndpoint |
  Select-Object `
    LocalAddress,
    LocalPort,
    OwningProcess |
  Sort-Object LocalPort, LocalAddress
```

Zu prüfen sind:

- lokale Adresse;
- lokaler Port;
- Prozess-ID;
- IPv4 oder IPv6;
- Loopback-, spezifische oder Wildcard-Bindung;
- mehrere Prozesse oder Endpunkte;
- unerwarteter Netzwerk-Namespace.

Windows: Prozess eines UDP-Endpunkts bestimmen

LESEND

```
Get-NetUDPEndpoint `
  -LocalPort 53 |
  ForEach-Object {
    $endpoint = $_
    $process = Get-Process `
      -Id $endpoint.OwningProcess `
      -ErrorAction SilentlyContinue

    [pscustomobject]@{
      LocalAddress = $endpoint.LocalAddress
      LocalPort    = $endpoint.LocalPort
      ProcessId    = $endpoint.OwningProcess
      ProcessName  = $process.ProcessName
      ProcessPath  = $process.Path
    }
  }
```

Einen bekannten Prozess prüfen:

```
Get-Process `
  -Id 1234
```

Einem Windows-Dienst zuordnen:

```
$processId = 1234

Get-CimInstance `
  -ClassName Win32_Service `
  -Filter "ProcessId = $processId" |
  Select-Object `
    Name,
    DisplayName,
    State,
    StartMode,
    ProcessId,
```

```
  PathName
```

Windows: UDP mit netstat prüfen

LESEND

```
netstat -ano -p udp
```

Die Ausgabe zeigt unter anderem:

- lokale Adresse;
- lokalen Port;
- Prozess-ID.

UDP besitzt dabei keinen TCP-Zustand wie `LISTENING` oder `ESTABLISHED`.

Numerische Darstellung ist für die erste Diagnose vorzuziehen, damit keine Namensauflösung die Ausgabe verändert oder verzögert.

Windows: UDP-Statistiken prüfen

LESEND

IPv4:

```
netstat -s -p udp
```

IPv6:

```
netstat -s -p udpv6
```

Verfügbare Leistungsindikatoren:

```
Get-Counter `
  -ListSet "*UDP*"
```

Die Namen der Leistungsindikatoren können abhängig von Systemsprache und Windows-Version abweichen.

Kumulative Zähler müssen über ein definiertes Zeitfenster verglichen werden. Ein hoher Gesamtwert seit dem Systemstart beweist keine aktuelle Störung.

Windows: Test-NetConnection ist kein UDP-Porttest

Dieser Befehl:

```
Test-NetConnection `
  -ComputerName "192.0.2.53" `
  -Port 53
```

prüft TCP-Port `53`.

Er prüft nicht UDP-Port `53`.

Folgende Schlussfolgerung ist daher unzulässig:

```
TcpTestSucceeded = True
also funktioniert UDP-Port 53.
```

Für UDP muss ein passendes Anwendungsprotokoll verwendet oder der Datenweg mit einer Paketaufzeichnung nachgewiesen werden.

Windows: UDP-Test mit PortQry

PortQry ist ein separat bereitzustellendes Microsoft-Diagnosewerkzeug und nicht auf jedem Windows-System vorinstalliert.

NETZAKTIV

```
portqry.exe -n 192.0.2.53 -p UDP -e 53
```

Mögliche Einordnung:

PortQry-Ergebnis	Bedeutung
`LISTENING`	PortQry hat eine verwertbare Protokollantwort erhalten.
`NOT LISTENING`	PortQry hat typischerweise ICMP Port Unreachable erhalten.
`LISTENING OR FILTERED`	Es kam keine eindeutige Antwort; geöffnet und still oder gefiltert sind nicht unterscheidbar.

Für einige bekannte Protokolle sendet PortQry speziell formatierte Anfragen. Bei beliebigen UDP-Diensten bleibt ein fehlendes Ergebnis mehrdeutig.

PortQry darf nur gegen autorisierte Zielsysteme eingesetzt werden.

Linux: UDP-Endpunkte mit ss prüfen

LESEND

Gebundene UDP-Endpunkte:

```
sudo ss -lunp
```

Alle UDP-Sockets:

```
sudo ss -uanp
```

Bestimmten lokalen Port prüfen:

```
sudo ss -lunp \
  'sport = :53'
```

Ausgabe ohne Prozessinformationen:

```
ss -lun
```

Wichtige Felder:

- lokale Adresse;
- lokaler Port;
- Prozess;
- Empfangswarteschlange;
- Sendewarteschlange;
- IPv4 oder IPv6.

Prozessinformationen können erhöhte Berechtigungen erfordern.

Linux: Prozess mit lsof bestimmen

LESEND

Alle UDP-Sockets:

```
sudo lsof \
  -nP \
  -iUDP
```

Bestimmten UDP-Port prüfen:

```
sudo lsof \
  -nP \
  -iUDP:53
```

Prozess prüfen:

```
ps -fp <PID>
```

Bei systemd:

```
systemctl status <Dienstname>
```

Ein laufender Dienst beweist nicht, dass er den erwarteten UDP-Port im richtigen Netzwerk-Namespace gebunden hat.

Linux: UDP-Zähler prüfen

LESEND

Alle Netzwerkstatistiken:

```
nstat -az
```

Ausgewählte UDP-Zähler:

```
nstat -az UdpInDatagrams
```

```
nstat -az UdpNoPorts
```

```
nstat -az UdpInErrors
```

```
nstat -az UdpRcvbufErrors
```

```
nstat -az UdpSndbufErrors
```

Klassische Übersicht:

```
netstat -su
```

Schnittstellenzähler:

```
ip -s link
```

Mögliche Hinweise:

Zähler	Mögliche Einordnung
`UdpInDatagrams`	empfangene und zugestellte UDP-Datagramme
`UdpNoPorts`	Datagramme für nicht belegte UDP-Ports
`UdpInErrors`	allgemeine UDP-Empfangsfehler
`UdpRcvbufErrors`	Empfangspuffer konnte Datagramme nicht aufnehmen
`UdpSndbufErrors`	Sendepufferfehler
Schnittstellen-Drops	Verlust an oder nahe der Netzwerkschnittstelle

Die genaue Zählerverfügbarkeit hängt von Kernel und Werkzeugversion ab.

---

macOS: UDP-Endpunkte prüfen

LESEND

```
sudo lsof \
  -nP \
  -iUDP
```

Bestimmten UDP-Port prüfen:

```
sudo lsof \
  -nP \
  -iUDP:53
```

Socketübersicht:

```
netstat -anv -p udp
```

UDP-Statistiken:

```
netstat -s -p udp
```

Netzwerkschnittstellen:

```
ifconfig
```

Interaktive Netzwerkansicht:

```
nettop
```

`lsof` ist unter macOS für die Zuordnung eines UDP-Ports zu einem Prozess normalerweise geeigneter als `netstat`.

Empfangswarteschlange und Anwendungsgeschwindigkeit

Ein Datagramm kann den Host erreichen, aber verloren gehen, bevor die Anwendung es verarbeitet.

Mögliche Ursachen:

- Anwendung liest nicht schnell genug;
- Empfangspuffer ist voll;
- CPU-Überlastung;
- Prozess hängt;
- Workerpool ist erschöpft;
- zu hohe Datagrammrate;
- Datagramme sind größer als erwartet;
- Kernel- oder Schnittstellenpuffer laufen über;
- Container besitzt zu geringe Ressourcen;
- Anwendung verwirft ungültige oder unerwartete Nachrichten.

Benötigte Nachweise:

- Paketaufzeichnung am Empfänger;
- UDP- und Schnittstellenzähler;
- Socketwarteschlange;
- Prozess-CPU und Speicher;
- Anwendungsprotokoll;
- Rate und Größe der Datagramme;
- Vergleich mit einer Baseline.

Wenn ein Datagramm in der Paketaufzeichnung des Servers sichtbar ist, beweist das noch nicht, dass die Anwendung es aus dem Socket gelesen hat.

ICMP ist nicht nur Ping

ICMP dient zur Übertragung von Kontroll- und Fehlermeldungen für IP.

Ping verwendet:

- ICMP Echo Request;
- ICMP Echo Reply.

ICMP umfasst jedoch zusätzlich Meldungen wie:

- Destination Unreachable;
- Port Unreachable;
- Time Exceeded;
- Fragmentation Needed;
- Packet Too Big;
- Parameter Problem;
- Redirect.

ICMPv4 und ICMPv6 sind nicht identisch. Sie verwenden unterschiedliche Typen, Codes und Protokollmechanismen.

Das vollständige Blockieren von ICMP kann Fehlerdiagnose und Path MTU Discovery beeinträchtigen. Bei IPv6 besitzt ICMPv6 darüber hinaus grundlegende Bedeutung für mehrere IPv6-Funktionen.

ICMP-Fehler einer UDP-Anfrage zuordnen

Eine ICMP-Fehlermeldung enthält Teile des Pakets, das den Fehler ausgelöst hat.

Dadurch können Betriebssystem, Firewall oder Analysewerkzeug die Meldung beispielsweise folgender Kommunikation zuordnen:

```
192.0.2.100:53124
→ 192.0.2.53:53 UDP
```

Zu prüfen sind im eingebetteten ursprünglichen Paket:

- ursprüngliche Quelladresse;
- ursprüngliche Zieladresse;
- UDP-Quellport;
- UDP-Zielport;
- Protokollnummer;
- gegebenenfalls weitere Anwendungsdaten.

Bei NAT muss auch die ICMP-Fehlermeldung korrekt zur ursprünglichen internen Kommunikation zurückübersetzt werden.

Häufige ICMPv4-Typen

Typ	Bezeichnung	Typische Bedeutung
`0`	Echo Reply	Antwort auf Echo Request
`3`	Destination Unreachable	Ziel, Protokoll oder Port nicht erreichbar
`5`	Redirect	Hinweis auf einen anderen nächsten Router
`8`	Echo Request	Ping-Anfrage
`11`	Time Exceeded	TTL abgelaufen oder Fragmentwiederherstellung zu langsam
`12`	Parameter Problem	Fehler in einem IP-Headerfeld

ICMP Source Quench ist veraltet und darf nicht als moderner Überlastkontrollmechanismus verwendet werden.

ICMPv4 Destination Unreachable

ICMPv4 verwendet für Destination Unreachable den Typ `3`.

Code	Bedeutung
`0`	Network Unreachable
`1`	Host Unreachable
`2`	Protocol Unreachable

Code	Bedeutung
3	Port Unreachable
4	Fragmentation Needed and DF Set
5	Source Route Failed
9	Network Administratively Prohibited
10	Host Administratively Prohibited
13	Communication Administratively Prohibited

Nicht jedes Gerät verwendet alle Codes. Firewalls können Fehler außerdem still verwerfen oder andere Meldungen erzeugen.

ICMPv4 Port Unreachable

Typischer Ablauf:

```
Client → Server:
UDP-Datagramm an Port 9999

Server → Client:
ICMP Destination Unreachable
Type 3, Code 3
Port Unreachable
```

Mögliche Einordnung:

- Zielhost wurde erreicht;
- auf dem angesprochenen UDP-Port ist häufig kein passender Endpunkt vorhanden;
- eine Firewall oder Zwischenkomponente kann die Meldung erzeugt haben;
- NAT- oder Portweiterleitungsziel besitzt keinen passenden UDP-Dienst.

Der genaue Erzeuger muss anhand von Aufzeichnungen und Protokollen bestimmt werden.

ICMPv4 Fragmentation Needed

Typischer Befund:

```
ICMP Type 3, Code 4
Fragmentation Needed and DF Set
```

Bedeutung:

- ein IPv4-Paket ist für den nächsten Link zu groß;
- das Paket darf aufgrund gesetztem DF-Bit nicht fragmentiert werden;
- die Meldung kann eine verwendbare MTU enthalten;
- der Sender muss die Paketgröße anpassen.

Wenn diese ICMP-Meldung gefiltert wird, kann ein Path-MTU-Black-Hole entstehen:

kleine Datagramme funktionieren

große Datagramme schlagen fehl

keine verwertbare Fehlermeldung erreicht den Sender

ICMPv4 Time Exceeded

ICMPv4 verwendet Typ `11`.

Code	Bedeutung
`0`	TTL während der Weiterleitung abgelaufen
`1`	Zeit für Fragmentwiederherstellung abgelaufen

TTL-Ablauf wird unter anderem von Traceroute verwendet, um Zwischenrouter sichtbar zu machen.

Ein Stern in einer Traceroute-Ausgabe beweist nicht, dass der betreffende Router ausgefallen ist. Er kann ICMP-Antworten filtern oder begrenzen und trotzdem normalen Datenverkehr weiterleiten.

Häufige ICMPv6-Typen

Typ	Bezeichnung
`1`	Destination Unreachable
`2`	Packet Too Big
`3`	Time Exceeded
`4`	Parameter Problem
`128`	Echo Request
`129`	Echo Reply

ICMPv6 enthält außerdem wichtige Mechanismen für IPv6. Es darf nicht pauschal wie optionaler Ping-Verkehr behandelt werden.

ICMPv6 Destination Unreachable

ICMPv6 verwendet Typ `1`.

Code	Bedeutung
`0`	No Route to Destination
`1`	Communication Administratively Prohibited
`2`	Beyond Scope of Source Address
`3`	Address Unreachable
`4`	Port Unreachable
`5`	Source Address Failed Ingress/Egress Policy
`6`	Reject Route to Destination

Port Unreachable bei IPv6:

```
ICMPv6 Type 1, Code 4
```

ICMPv6 Packet Too Big

ICMPv6 verwendet:

```
Type 2, Code 0
```

Die Meldung enthält die MTU des nächsten Links.

IPv6-Router fragmentieren weitergeleitete Pakete nicht. Wenn ein Paket zu groß ist, muss der sendende Endpunkt seine Paketgröße anpassen oder selbst geeignete Fragmentierung verwenden.

Das Blockieren von ICMPv6 Packet Too Big kann dazu führen, dass:

- kleine UDP-Datagramme funktionieren;
- größere Datagramme verschwinden;
- Tunnel oder VPN-Pfade betroffen sind;
- Anwendungen in Timeouts laufen;
- die Ursache fälschlich beim UDP-Dienst gesucht wird.

ICMPv6 Time Exceeded

ICMPv6 verwendet Typ `3`.

Code	Bedeutung
`0`	Hop Limit während der Weiterleitung abgelaufen
`1`	Fragment Reassembly Time Exceeded

Das IPv6 Hop Limit entspricht funktional dem IPv4-TTL-Konzept.

ICMPv6 Parameter Problem

ICMPv6 verwendet Typ `4`.

Code	Bedeutung
`0`	fehlerhaftes Headerfeld
`1`	unbekannter Next-Header-Typ
`2`	unbekannte IPv6-Option

Die Meldung kann auf das fehlerhafte Feld innerhalb des ursprünglichen Pakets verweisen.

Keine ICMP-Antwort ist mehrdeutig

Wenn auf ein UDP-Datagramm weder eine Anwendungsantwort noch eine ICMP-Fehlermeldung folgt, sind unter anderem möglich:

- UDP-Port ist geöffnet, Anwendung antwortet aber nicht auf diese Anfrage;
- Anfrage ist ungültig;
- Anwendung antwortet grundsätzlich nicht;
- Datagramm wurde unterwegs verworfen;
- lokale Firewall verwirft es;
- Netzwerkfirewall verwirft es;
- Zielhost ist nicht erreichbar;
- Antwortweg ist fehlerhaft;
- ICMP wird gefiltert;
- ICMP wird durch Rate Limiting unterdrückt;
- NAT-Zustand fehlt;
- Server ist überlastet;
- Empfangspuffer ist voll;
- Anwendung verwirft das Datagramm;
- Paketaufzeichnung wurde am falschen Punkt durchgeführt.

Das Ergebnis darf nicht ohne weiteren Nachweis als „Port offen“ oder „Port geschlossen“ dokumentiert werden.

ICMP Rate Limiting berücksichtigen

Hosts und Router können ICMP-Fehlermeldungen begrenzen.

Folgen:

- der erste Test liefert ICMP Port Unreachable;
- spätere Tests liefern keine Meldung;
- nur ein Teil der verlorenen Datagramme erzeugt ICMP;
- ein UDP-Scan zeigt uneinheitliche Ergebnisse;
- die Abwesenheit von ICMP wird fälschlich als Filterung interpretiert.

Diagnosetests müssen mit geringer, kontrollierter Rate durchgeführt werden.

ICMP-Fehler müssen nicht die Anwendung erreichen

Ob eine Anwendung einen ICMP-Fehler wahrnimmt, hängt ab von:

- Betriebssystem;
- Socket-API;
- verbundenem oder unverbundenem UDP-Socket;
- Fehlerwarteschlange;
- Anwendungscode;
- NAT;
- Firewall;
- Zuordenbarkeit zum ursprünglichen Datagramm.

Eine Anwendung kann deshalb einen Timeout melden, obwohl in einer Paketaufzeichnung eine ICMP-Fehlermeldung sichtbar ist.

Umgekehrt kann das Betriebssystem einen Socketfehler melden, obwohl die Anwendung selbst keine ICMP-Pakete auswertet.

UDP, MTU und Fragmentierung

Ein UDP-Datagramm bleibt für die Anwendung eine einzelne Nachricht. Auf IP-Ebene kann das zugehörige Paket jedoch fragmentiert werden.

Probleme großer UDP-Datagramme:

- Verlust eines Fragments verwirft das gesamte Datagramm;
- Firewalls verwerfen Fragmente;
- NAT kann Fragmente nicht korrekt zuordnen;
- Tunnel reduzieren die nutzbare Path MTU;
- Fragmentwiederherstellung läuft in einen Timeout;
- ICMP Fragmentation Needed oder Packet Too Big wird blockiert;
- Anwendung besitzt keine geeignete Größenanpassung;

- unterschiedliche Pfade besitzen unterschiedliche MTUs.

Typischer Befund:

```
kleine Anfrage funktioniert
kleine Antwort funktioniert
große Antwort schlägt fehl
über VPN tritt der Fehler häufiger auf
Paketaufzeichnung zeigt Fragmente oder ICMP Packet Too Big
```

IP-Fragmentierung sollte nicht vorsorglich erzwungen werden. Die Anwendung muss geeignete Nachrichtengrößen und Path-MTU-Verfahren verwenden.

UDP und Checksummen diagnostizieren

Eine ungültige UDP-Prüfsumme kann dazu führen, dass ein Datagramm ohne Anwendungsantwort verworfen wird.

Zu prüfen sind:

- wird die Prüfsumme in der Aufzeichnung als gültig angezeigt?
- stammt die Aufzeichnung vom sendenden Host?
- ist Checksum Offloading aktiv?
- zeigt eine externe Aufzeichnung denselben Befund?
- steigen UDP-Checksum- oder Eingangsfehlerzähler?
- tritt der Fehler nur auf einer Schnittstelle auf?

Wie bei TCP kann eine lokal vor der Netzwerkkarte aufgenommene Prüfsumme scheinbar fehlerhaft sein, obwohl sie später durch die Hardware korrekt berechnet wird.

Eine einzelne lokale Anzeige `Bad Checksum` ist deshalb kein ausreichender Fehlernachweis.

Unicast, Broadcast und Multicast unterscheiden

Kommunikationsart	Ziel
Unicast	genau ein IP-Endpunkt
Broadcast	alle geeigneten IPv4-Teilnehmer eines Broadcastbereichs
Multicast	Mitglieder einer Multicastgruppe

Bei Broadcast und Multicast sind zusätzlich zu prüfen:

- richtige Zieladresse;

- richtige lokale Schnittstelle;
- Broadcastberechtigung des Sockets;
- Multicastgruppenmitgliedschaft;
- IGMP bei IPv4;
- MLD bei IPv6;
- Switch-Snooping;
- VLAN;
- TTL beziehungsweise Hop Limit;
- Routerunterstützung;
- Firewallregeln;
- Anwendung mit mehreren Schnittstellen;
- erwartetes Antwortverhalten.

Ein Dienst kann auf Unicast funktionieren und auf Broadcast oder Multicast dennoch fehlschlagen.

IPv6 verwendet keinen Broadcast. Vergleichbare Aufgaben werden dort durch Multicastmechanismen umgesetzt.

Stateful Firewall und UDP

Eine Stateful Firewall kann für UDP einen temporären Zustand anhand des Datenflusses anlegen.

Beispiel:

```
Client 192.0.2.100:53124
→ Server 192.0.2.53:53

erwartete Antwort:
Server 192.0.2.53:53
→ Client 192.0.2.100:53124
```

Zu prüfen sind:

- Quell- und Zieladresse;
- Quell- und Zielport;
- UDP-Protokoll;
- Richtung des ersten Datagramms;
- Zustandstimer;
- erwartete Antwortadresse;
- erwarteter Antwortport;
- NAT-Übersetzung;
- asymmetrischer Rückweg;
- Rate Limits.

Wenn die Antwort erst nach Ablauf des Firewallzustands eintrifft, kann sie verworfen werden.

NAT und UDP

NAT kann eine interne Kommunikation wie folgt übersetzen:

```
intern:
192.0.2.100:53124

extern:
203.0.113.10:62000

Ziel:
198.51.100.53:53
```

Die Antwort muss an die übersetzte Adresse und den übersetzten Port zurückkehren:

```
198.51.100.53:53
→ 203.0.113.10:62000
```

Mögliche Fehler:

- NAT-Zustand läuft zu früh ab;
- Antwort verwendet unerwartete Quelladresse;
- Antwort kommt von anderem Serverport;
- asymmetrischer Rückweg umgeht die NAT-Instanz;
- ICMP-Fehler wird nicht korrekt zurückübersetzt;
- mehrere Clients kollidieren durch fehlerhafte Portzuordnung;
- Portweiterleitung wurde nur für TCP eingerichtet;
- interne und externe UDP-Ports stimmen nicht überein.

TCP- und UDP-Portweiterleitungen müssen getrennt konfiguriert werden.

Load Balancer und UDP

Ein UDP-Load-Balancer kann Datagramme anhand des Flows einem Backend zuordnen.

Zu prüfen sind:

- Frontendadresse;
- Frontendport;
- Protokoll UDP;

- Backendport;
- Health-Check-Protokoll;
- Flow-Hash;
- Quellportverhalten;
- UDP-Idle-Timeout;
- Rückweg über denselben Load Balancer;
- Direct Server Return;
- Backendzustand;
- Antwortquelladresse.

Wenn ein Client bei jedem Datagramm einen anderen Quellport verwendet, kann ein Load Balancer unterschiedliche Backends auswählen.

Ein erfolgreicher Health Check beweist nicht automatisch, dass der produktive UDP-Pfad funktioniert.

Docker: UDP-Veröffentlichung prüfen

Docker-Portveröffentlichungen müssen das Transportprotokoll berücksichtigen.

Beispiel:

```
53:53/udp
```

Ohne `/udp` wird eine Veröffentlichung typischerweise als TCP-Veröffentlichung behandelt.

Laufende Container und Ports:

LESEND

```
docker ps \
  --format 'table {{.Names}}\t{{.Status}}\t{{.Ports}}'
```

Portzuordnung:

```
docker port <Containername>
```

Ausführliche Konfiguration:

```
docker inspect \
  <Containername> \
```

```
  --format '{{json .NetworkSettings.Ports}}'
```

UDP-Endpunkt im Container:

```
docker exec \
  <Containername> \
  ss -lunp
```

Compose-Beispiel:

```
ports:
  - "53:53/udp"
```

Zu prüfen sind:

- wurde UDP ausdrücklich veröffentlicht?
- stimmt der Hostport?
- stimmt der Containerport?
- lauscht die Anwendung im Container?
- ist sie an Loopback oder an die Containeradresse gebunden?
- verwendet der Client Hostadresse oder Containeradresse?
- blockiert die Hostfirewall?
- existiert eine passende NAT-Regel?

`EXPOSE 53/udp` im Image veröffentlicht den Port nicht automatisch auf dem Host.

Kubernetes: UDP-Service prüfen

Wenn `protocol` nicht angegeben wird, verwendet ein Kubernetes-Service standardmäßig TCP. Für UDP muss das Protokoll ausdrücklich passen.

Beispiel:

```
ports:
  - name: dns-udp
    protocol: UDP
    port: 53
    targetPort: 53
```

Service anzeigen:

```
kubectl get service \
  <Servicename> \
  -n <Namespace> \
  -o yaml
```

Service beschreiben:

```
kubectl describe service \
  <Servicename> \
  -n <Namespace>
```

EndpointSlices:

```
kubectl get endpointslice \
  -n <Namespace> \
  -l kubernetes.io/service-name=<Servicename> \
  -o wide
```

UDP-Endpunkt im Pod:

```
kubectl exec \
  -n <Namespace> \
  <Podname> \
  -- ss -lunp
```

Bei mehreren Containern:

```
kubectl exec \
  -n <Namespace> \
  <Podname> \
  -c <Containername> \
  -- ss -lunp
```

Zu prüfen sind:

- `protocol: UDP`;
- Service-`port`;
- `targetPort`;

- Pod-Endpunkt;
- Service-Selektor;
- EndpointSlices;
- Readiness;
- NetworkPolicy;
- CNI-Implementierung;
- NodePort oder Load Balancer;
- Rückweg;
- UDP-Idle-Timeout.

Ein eingetragener `containerPort` erzeugt keinen UDP-Endpunkt.

Windows-Paketaufzeichnung mit pktmon

`pktmon` verändert den Aufzeichnungszustand und erstellt Dateien.

SENSITIV · ÄNDERND

Vorhandene Aufzeichnung stoppen:

```
pktmon stop
```

Vorhandene Filter entfernen:

```
pktmon filter remove
```

UDP-Port `53` erfassen:

```
pktmon filter add UDP53 -t UDP -p 53
```

ICMPv4 ergänzen:

```
pktmon filter add ICMPv4 -t ICMP
```

ICMPv6 ergänzen:

```
pktmon filter add ICMPv6 -t ICMPV6
```

Aufzeichnung starten:

```
pktmon start --capture --pkt-size 0 --file-name C:\Temp\udp53.etl
```

Nach reproduziertem Fehler stoppen:

```
pktmon stop
```

In PCAPNG konvertieren:

```
pktmon etl2pcap C:\Temp\udp53.etl --out C:\Temp\udp53.pcapng
```

Filter zurücknehmen:

```
pktmon filter remove
```

Das Zielverzeichnis muss vorhanden sein. Aufzeichnung, Filter und erzeugte Dateien müssen nach der Diagnose kontrolliert behandelt werden.

Paketaufzeichnung unter Linux

UDP und ICMP gemeinsam:

SENSITIV · LESEND

```
sudo tcpdump \
  -ni any \
  '(udp port 53) or icmp or icmp6' \
  -c 200
```

Auf bestimmte Gegenstelle begrenzen:

```
sudo tcpdump \
  -ni any \
  'host 192.0.2.53 and ((udp port 53) or icmp or icmp6)' \
  -c 200
```

In Datei schreiben:

```
sudo tcpdump \
  -ni any \
  -s 0 \
  -w /tmp/udp53.pcap \
  'host 192.0.2.53 and ((udp port 53) or icmp or icmp6)'
```

Nach reproduziertem Fehler mit `Strg+C` beenden.

Paketaufzeichnung unter macOS

Verfügbare Schnittstellen:

```
tcpdump -D
```

Auf konkreter Schnittstelle:

```
sudo tcpdump \
  -ni en0 \
  'host 192.0.2.53 and ((udp port 53) or icmp or icmp6)' \
  -c 200
```

In Datei schreiben:

```
sudo tcpdump \
  -ni en0 \
  -s 0 \
  -w /tmp/udp53.pcap \
  'host 192.0.2.53 and ((udp port 53) or icmp or icmp6)'
```

Bei lokalem Verkehr muss gegebenenfalls `lo0` statt der physischen Schnittstelle verwendet werden.

Wireshark-Filter für UDP und ICMP

Alle UDP-Pakete:

```
udp
```

Bestimmter UDP-Port:

```
udp.port == 53
```

Bestimmte Adresse und Port:

```
ip.addr == 192.0.2.53 && udp.port == 53
```

ICMPv4:

```
icmp
```

ICMPv6:

```
icmpv6
```

ICMPv4 Destination Unreachable:

```
icmp.type == 3
```

ICMPv4 Port Unreachable:

```
icmp.type == 3 && icmp.code == 3
```

ICMPv4 Fragmentation Needed:

```
icmp.type == 3 && icmp.code == 4
```

ICMPv4 Time Exceeded:

```
icmp.type == 11
```

ICMPv6 Port Unreachable:

```
icmpv6.type == 1 && icmpv6.code == 4
```

ICMPv6 Packet Too Big:

```
icmpv6.type == 2
```

ICMPv6 Time Exceeded:

```
icmpv6.type == 3
```

UDP und beide ICMP-Versionen:

```
udp || icmp || icmpv6
```

Eine ICMP-Meldung muss zusammen mit dem darin eingebetteten ursprünglichen Paket ausgewertet werden.

Client- und Serveraufzeichnung vergleichen

Clientaufzeichnung	Serveraufzeichnung	Mögliche Einordnung
UDP-Anfrage sichtbar	Anfrage nicht sichtbar	Verlust oder Filterung vor dem Server
Anfrage sichtbar	Anfrage sichtbar	Hinweg bis zum Aufzeichnungspunkt funktioniert
keine Antwort	Anwendung protokolliert Anfrage nicht	Socket, lokaler Filter oder Anwendungsverarbeitung
keine Antwort	Anwendung protokolliert Anfrage	Anwendung erzeugt keine oder verspätete Antwort
Antwort verlässt Server	Antwort erreicht Client nicht	Rückweg, NAT oder Firewall
ICMP Port Unreachable am Client	Server erzeugt ICMP	Zielport wahrscheinlich nicht gebunden
ICMP nur am Client sichtbar	Server zeigt keine Meldung	Zwischenkomponente als Erzeuger möglich
ICMP verlässt Server	Client erhält es nicht	Rückweg oder ICMP-Filterung
kleine Datagramme funktionieren	große erreichen Server nicht	MTU, Fragmentierung oder Filterung
Anfrage und Antwort sichtbar	Anwendung meldet Timeout	Socketzuordnung, Frist oder Anwendungslogik

---

Typische UDP-Paketfolgen

Erfolgreiche Anfrage mit Antwort

```
Client → Server  UDP-Anfrage
Server → Client  UDP-Antwort
```

Geschlossener UDP-Port

```
Client → Server  UDP-Anfrage
Server → Client  ICMP Port Unreachable
```

Stilles Verwerfen

```
Client → Server  UDP-Anfrage
keine Antwort
keine ICMP-Meldung
```

Server verarbeitet Anfrage nicht

```
Client → Server  UDP-Anfrage
Anfrage erreicht Serverschnittstelle
keine Anwendungsantwort
```

Rückwegfehler

```
Client → Server  UDP-Anfrage
Server → Client  UDP-Antwort
Antwort erreicht Client nicht
```

Path-MTU-Problem

```
kleine UDP-Datagramme funktionieren
großes UDP-Datagramm wird verworfen
ICMP Fragmentation Needed oder Packet Too Big
```

Traceroute und ICMP Time Exceeded

Traceroute nutzt schrittweise erhöhte TTL- beziehungsweise Hop-Limit-Werte. Zwischenrouter können darauf mit ICMP Time Exceeded antworten.

Linux mit UDP-Probes:

NETZAKTIV

```
traceroute \
  -U \
  -p 33434 \
  192.0.2.53
```

IPv6:

```
traceroute \
  -6 \
  2001:db8::53
```

Die genaue Standardmethode von Traceroute hängt von Betriebssystem und Implementierung ab.

Ein unvollständiger Traceroute beweist keinen vollständigen Pfadausfall. Router können:

- ICMP-Antworten filtern;
- Antworten begrenzen;
- nur Datenverkehr weiterleiten;
- für Antwort und Weiterleitung unterschiedliche Richtlinien verwenden;
- auf einem asymmetrischen Rückweg antworten.

Hypothese und Gegenbeweis

Beispiel für einen geschlossenen UDP-Port:

```
Hypothese:
Auf dem Server ist kein UDP-Endpunkt an Port 9999 gebunden.

Erwarteter Befund:
Die lokale Socketliste zeigt keinen UDP-Endpunkt auf Port 9999.
Die Serveraufzeichnung zeigt die eingehende UDP-Anfrage.
Der Server sendet ICMP Port Unreachable zurück.

Gegenbeweis:
Ein Prozess besitzt UDP-Port 9999 und protokolliert die eingehende Anfrage.
```

Testmethode:

Socketliste, Prozesszuordnung und korrelierte Paketaufzeichnung.

Risiko:

Die Anfrage erzeugt Netzwerkverkehr und muss dem erwarteten Protokoll entsprechen.

Beispiel für einen Rückwegfehler:

Hypothese:

Die UDP-Antwort verlässt den Server, wird aber durch eine Firewall

auf dem Rückweg verworfen.

Erwarteter Befund:

Die Serveraufzeichnung zeigt die ausgehende Antwort.

Die Clientaufzeichnung zeigt diese Antwort nicht.

Die Firewall protokolliert einen Drop für den Antwortflow.

Gegenbeweis:

Die Clientaufzeichnung zeigt die Antwort vollständig.

Testmethode:

Zeitgleiche Aufzeichnung auf Client und Server sowie Firewallprotokoll.

Kontrollierte Maßnahmen

Maßnahme	Voraussetzung	Risiko
UDP-Endpunkt starten oder Bindung korrigieren	fehlender Endpunkt bestätigt	geänderte Erreichbarkeit
Firewallregel für UDP korrigieren	UDP-Drop nachgewiesen	unbeabsichtigte Freigabe
ICMP-Regel korrigieren	benötigte Fehlermeldung wird nachweislich blockiert	zusätzliche Kontrollmeldungen werden zugelassen
NAT-Regel um UDP ergänzen	nur TCP oder falsches Protokoll bestätigt	öffentliche Erreichbarkeit ändert sich
UDP-Idle-Timeout abstimmen	ablaufender Sitzungszustand bestätigt	mehr Zustands- und Speicherverbrauch
Antwortquelladresse korrigieren	falsche Quelladresse bestätigt	Routing und Dienstbindung ändern sich
Datagrammgröße reduzieren	MTU- oder Fragmentierungsproblem bestätigt	Protokollverhalten oder Leistung ändert sich

Maßnahme	Voraussetzung	Risiko
Receive Buffer anpassen	Pufferüberlauf nachgewiesen	höherer Speicherverbrauch
Anwendungsgeschwindigkeit verbessern	Empfangsverlust durch Verarbeitung bestätigt	Anwendungsänderung erforderlich
Docker-Publish auf `/udp` korrigieren	fehlende UDP-Veröffentlichung bestätigt	Container muss eventuell neu erstellt werden
Kubernetes-Service auf UDP korrigieren	falsches Serviceprotokoll bestätigt	produktiver Servicepfad ändert sich
Multicastmitgliedschaft korrigieren	fehlende Gruppenmitgliedschaft bestätigt	zusätzlicher Multicastempfang
Rate Limit anpassen	legitimer Verkehr wird nachweislich begrenzt	Überlastungs- und Missbrauchsrisiko

Vor jeder Änderung sind Ausgangszustand, Risiko, Rückweg und Erfolgskriterium zu dokumentieren.

Systematischer Diagnoseablauf

1. Exakte Fehlermeldung und Zeitpunkt aufnehmen.
2. Anwendung und Protokoll bestimmen.
3. Bestätigen, dass tatsächlich UDP verwendet wird.
4. Quell- und Zieladresse dokumentieren.
5. Quell- und Zielport dokumentieren.
6. IPv4 und IPv6 unterscheiden.
7. Unicast, Broadcast oder Multicast bestimmen.
8. Erwartete Anfrage und Antwort beschreiben.
9. Lokalen UDP-Endpunkt auf dem Server prüfen.
10. Besitzenden Prozess bestimmen.
11. Richtige Bindungsadresse prüfen.
12. Host-, Container- und Pod-Namespace unterscheiden.
13. Anwendungsprotokolle auf dem Server prüfen.
14. Protokollspezifischen Test vom Client durchführen.
15. UDP-Anfrage auf dem Client erfassen.
16. Prüfen, ob die Anfrage den Server erreicht.
17. Prüfen, ob die Anwendung die Anfrage verarbeitet.
18. Prüfen, ob eine Antwort erzeugt wird.
19. Prüfen, ob die Antwort den Client erreicht.
20. Quelladresse und Quellport der Antwort prüfen.
21. ICMPv4- und ICMPv6-Meldungen auswerten.
22. ICMP-Erzeuger und eingebettetes Originalpaket prüfen.
23. Stateful Firewall und UDP-Zustand prüfen.
24. NAT-Übersetzung und Rückweg prüfen.
25. Idle-Timeouts berücksichtigen.
26. UDP- und Schnittstellenzähler vergleichen.

27. Empfangspuffer und Anwendungsleistung prüfen.
28. Kleine und große gültige Datagramme vergleichen.
29. MTU, Fragmentierung und Tunnel berücksichtigen.
30. Bei Multicast Gruppenmitgliedschaft und Switch prüfen.
31. Hypothese und Gegenbeweis formulieren.
32. Genau eine kontrollierte Maßnahme durchführen.
33. Identischen Test wiederholen.
34. Anwendung und mehrere Versuche verifizieren.
35. Temporäre Aufzeichnung und Filter zurücknehmen.
36. Ursache und Prävention dokumentieren.

Befundmatrix

Befund	Mögliche Einordnung	Nächster Nachweis
kein lokaler UDP-Endpunkt	Dienst fehlt oder falscher Namespace	Prozess und Dienststart prüfen
Endpunkt nur auf Loopback	nur lokale Erreichbarkeit	Sollbindung prüfen
Endpunkt vorhanden, Anfrage erreicht Server nicht	Netzwerk oder Firewall	Client- und Zwischenaufzeichnung
Anfrage erreicht Server, Anwendung sieht sie nicht	Socket, Firewall oder Puffer	Prozesslog und UDP-Zähler
Anwendung sieht Anfrage, antwortet nicht	Protokoll oder Anwendung	Anwendungslogik prüfen
Antwort verlässt Server, erreicht Client nicht	Rückweg, NAT oder Firewall	Client- und Firewallaufzeichnung
ICMP Port Unreachable	Zielport nicht gebunden oder aktive Ablehnung	lokale Socketliste
ICMP Administratively Prohibited	Richtlinie blockiert	Firewall- oder Routerkonfiguration
keine Antwort und kein ICMP	geöffnet/still oder gefiltert	serverseitige Aufzeichnung
kleine Datagramme funktionieren	Basispfad funktioniert	größere gültige Nachricht testen
große Datagramme scheitern	MTU, Fragmentierung oder Puffer	ICMP und Fragmente erfassen
ICMP Fragmentation Needed	IPv4-Paket zu groß und DF gesetzt	Path MTU prüfen
ICMPv6 Packet Too Big	IPv6-Paket überschreitet Path MTU	gemeldete MTU auswerten
`UdpNoPorts` steigt	Datagramme erreichen ungebundene Ports	Zielport und Dienst prüfen
`UdpRcvbufErrors` steigt	Empfangspuffer überlastet	Datenrate und Anwendung prüfen
nur nach Inaktivität fehlerhaft	UDP-Zustand abgelaufen	Firewall-/NAT-Timeout
neue Anfrage funktioniert sofort	alter Pseudo-Sitzungszustand fehlerhaft	Flow und Quellport vergleichen
Docker zeigt nur TCP-Publish	UDP nicht veröffentlicht	Portkonfiguration prüfen

Befund	Mögliche Einordnung	Nächster Nachweis
Kubernetes-Service nutzt TCP	falsches Serviceprotokoll	Manifest korrigieren
Service hat keine Endpunkte	Selektor oder Readiness	Pods und EndpointSlices
Multicast nur auf einem Host fehlerhaft	Gruppenmitgliedschaft oder Schnittstelle	IGMP/MLD und Socket prüfen
Wireshark zeigt Bad Checksum nur lokal	Checksum Offloading möglich	externe Aufzeichnung
PortQry meldet `LISTENING OR FILTERED`	Ergebnis nicht eindeutig	gültiger Protokolltest und Servertrace

---

Typische Diagnosefehler

- TCP- und UDP-Port mit derselben Nummer gleichsetzen.
- `Test-NetConnection -Port` als UDP-Test verwenden.
- Einen erfolgreichen Ping als UDP-Nachweis bewerten.
- Einen UDP-Endpunkt als vollständigen Funktionsnachweis behandeln.
- Von einem erfolgreichen send-Aufruf auf die Zustellung schließen.
- `nc -u` als eindeutigen Porttest verwenden.
- Eine ungültige Testnachricht an einen Dienst senden.
- Aus fehlender Antwort auf einen geschlossenen oder gefilterten Port schließen.
- `LISTENING OR FILTERED` als sicher geöffnet interpretieren.
- ICMP ausschließlich mit Ping gleichsetzen.
- Alle ICMP-Meldungen blockieren.
- ICMPv4 und ICMPv6 verwechseln.
- ICMP Port Unreachable nicht dem eingebetteten Paket zuordnen.
- ICMP-Quelladresse ohne weiteren Nachweis als Erzeuger behandeln.
- ICMP Rate Limiting ignorieren.
- Nur auf dem Client aufzeichnen.
- Antwortquelladresse und Antwortport nicht prüfen.
- Stateful Firewallzustand bei UDP ignorieren.
- UDP-Idle-Timeout nicht berücksichtigen.
- TCP-Portweiterleitung als UDP-Portweiterleitung betrachten.
- `/udp` bei Docker-Portveröffentlichung vergessen.
- Kubernetes-`protocol` nicht prüfen.
- `containerPort` als echten Endpunkt bewerten.
- Container- und Host-Namespace verwechseln.
- kleine und große Datagramme nicht vergleichen.
- Fragmentierung und Tunnel-MTU ignorieren.
- eine lokale Bad-Checksum-Anzeige ungeprüft als Fehler bewerten.
- Empfangspuffer und Anwendungsleistung ignorieren.
- Unicast, Broadcast und Multicast nicht unterscheiden.
- Multicastgruppenmitgliedschaft nicht prüfen.
- Netzwerkaufzeichnungen unbegrenzt laufen lassen.
- UDP-Tests mit hoher Rate durchführen.

- Firewall vorsorglich vollständig deaktivieren.
- mehrere Variablen gleichzeitig verändern.
- nur einen einzelnen erfolgreichen Antwortversuch verifizieren.

Verifikation

Nach einer Maßnahme müssen mindestens folgende Punkte geprüft werden:

- der richtige UDP-Endpunkt ist vorhanden;
- der richtige Prozess besitzt den Port;
- richtige Bindungsadresse wird verwendet;
- IPv4 funktioniert, sofern vorgesehen;
- IPv6 funktioniert, sofern vorgesehen;
- der Client verwendet den richtigen Zielport;
- die Anfrage verlässt den Client;
- die Anfrage erreicht den Server;
- die Anwendung verarbeitet die Anfrage;
- die Anwendung erzeugt die erwartete Antwort;
- die Antwort verwendet die richtige Quelladresse;
- die Antwort verwendet den richtigen Quellport;
- die Antwort erreicht den Client;
- keine unerwarteten ICMP-Fehler entstehen;
- ICMP Fragmentation Needed oder Packet Too Big funktioniert;
- kleine und repräsentativ große Datagramme funktionieren;
- keine UDP-Empfangspufferfehler entstehen;
- Firewallzustand bleibt ausreichend lange bestehen;
- NAT übersetzt Anfrage, Antwort und ICMP korrekt;
- Docker veröffentlicht ausdrücklich UDP;
- Kubernetes-Service verwendet `protocol: UDP`;
- `port` und `targetPort` stimmen;
- EndpointSlices enthalten die vorgesehenen Pods;
- Broadcast oder Multicast funktioniert, sofern erforderlich;
- mehrere aufeinanderfolgende Versuche funktionieren;
- ursprüngliche Anwendung funktioniert;
- weitere repräsentative Clients funktionieren;
- temporäre Filter wurden entfernt;
- Aufzeichnungen wurden geschützt oder kontrolliert gelöscht;
- Ursache, Maßnahme und Prävention wurden dokumentiert.

Eine einzelne sichtbare UDP-Antwort ist keine ausreichende Verifikation für alle Clients, Paketgrößen und Netzwerkpfade.

Dokumentationsvorlage

Störung:

<exakte Beschreibung>

Zeitpunkt:

<Datum, Uhrzeit und Zeitzone>

Anwendung:

<Dienst oder Protokoll>

Client:

<Hostname und IP-Adresse>

Server:

<Hostname und IP-Adresse>

Adressfamilie:

<IPv4 oder IPv6>

Kommunikationsart:

<Unicast, Broadcast oder Multicast>

Quellport:

<Port>

Zielport:

<Port>

Erwartete Anfrage:

<Protokollnachricht>

Erwartete Antwort:

<Antwort oder keine Antwort vorgesehen>

UDP-Endpunkt:

<Adresse, Port, Prozess und PID>

Clientaufzeichnung:

<Anfrage gesendet, Antwort oder ICMP>

Serveraufzeichnung:

<Anfrage empfangen und Antwort gesendet>

ICMP-Befund:

<Typ, Code, Absender und eingebettetes Paket>

Paketgröße:

<UDP- und IP-Größe>

Fragmentierung:

<ja, nein oder unbekannt>

Firewallzustand:

<Befund>

NAT-Zuordnung:

<interne und externe Adressen und Ports>

Timeout:

<gemessene Dauer und verantwortliche Komponente>

UDP-Zähler:

<relevante Differenzen>

Nachgewiesene Ursache:

<technischer Befund>

Gegenbeweis ausgeschlossen durch:

<Test und Ergebnis>

Durchgeführte Maßnahme:

<genau eine kontrollierte Änderung>

Risiko und Rückweg:

<Beschreibung>

Verifikation:

<identischer Test, Anwendung und weitere Clients>

Prävention:

<Monitoring oder Konfigurationsverbesserung>

Checkliste

- [ ] exakte Fehlermeldung dokumentiert
- [ ] Zeitpunkt und Zeitzone erfasst
- [ ] Anwendung bestimmt
- [ ] UDP als Transport bestätigt
- [ ] TCP und UDP unterschieden
- [ ] Client und Server bestimmt
- [ ] Quelladresse dokumentiert
- [ ] Zieladresse dokumentiert
- [ ] Quellport dokumentiert
- [ ] Zielport dokumentiert
- [ ] IPv4 und IPv6 unterschieden
- [ ] Unicast, Broadcast oder Multicast bestimmt
- [ ] erwartete Anfrage beschrieben
- [ ] erwartete Antwort beschrieben
- [ ] lokalen UDP-Endpunkt geprüft
- [ ] Bindungsadresse geprüft
- [ ] Prozessbesitzer bestimmt
- [ ] Dienstzuordnung geprüft
- [ ] Host-, Container- und Pod-Namespace unterschieden
- [ ] protokollspezifischen Test verwendet
- [ ] `Test-NetConnection` nicht als UDP-Nachweis verwendet
- [ ] Clientaufzeichnung durchgeführt
- [ ] Serveraufzeichnung durchgeführt
- [ ] ausgehende Anfrage bestätigt
- [ ] Eingang am Server bestätigt
- [ ] Verarbeitung durch Anwendung bestätigt
- [ ] ausgehende Antwort bestätigt
- [ ] Eingang der Antwort am Client bestätigt
- [ ] Antwortquelladresse geprüft
- [ ] Antwortquellport geprüft

- [ ] ICMPv4 geprüft
- [ ] ICMPv6 geprüft
- [ ] ICMP-Typ und Code dokumentiert
- [ ] eingebettetes ursprüngliches Paket geprüft
- [ ] ICMP-Erzeuger eingegrenzt
- [ ] ICMP Rate Limiting berücksichtigt
- [ ] Stateful Firewallzustand geprüft
- [ ] NAT-Zuordnung geprüft
- [ ] UDP-Idle-Timeout berücksichtigt
- [ ] asymmetrischen Rückweg berücksichtigt
- [ ] UDP-Zähler ausgewertet
- [ ] Schnittstellen-Drops geprüft
- [ ] Empfangspuffer geprüft
- [ ] Anwendungslast geprüft
- [ ] Checksum Offloading berücksichtigt
- [ ] kleine Datagramme getestet
- [ ] repräsentativ große Datagramme getestet
- [ ] MTU berücksichtigt
- [ ] Fragmentierung geprüft
- [ ] ICMP Packet Too Big berücksichtigt
- [ ] Docker-Protokoll bei Bedarf geprüft
- [ ] Kubernetes-Serviceprotokoll bei Bedarf geprüft
- [ ] EndpointSlices bei Bedarf geprüft
- [ ] Multicastgruppenmitgliedschaft bei Bedarf geprüft
- [ ] Hypothese formuliert
- [ ] Gegenbeweis festgelegt
- [ ] Risiko und Rückweg dokumentiert
- [ ] nur eine kontrollierte Maßnahme durchgeführt
- [ ] identischen Test wiederholt
- [ ] ursprüngliche Anwendung getestet
- [ ] weitere repräsentative Clients geprüft
- [ ] temporäre Filter entfernt
- [ ] Aufzeichnungen geschützt oder entfernt
- [ ] Ursache und Prävention dokumentiert

Schnellreferenz

Aufgabe	Befehl oder Filter
Windows-UDP-Endpunkte	`Get-NetUDPEndpoint`
Windows-UDP-Port	`Get-NetUDPEndpoint -LocalPort <Port>`
Windows-UDP-netstat	`netstat -ano -p udp`
Windows-UDP-Statistik	`netstat -s -p udp`
Windows-UDP-PortQry	`portqry.exe -n <Ziel> -p UDP -e <Port>`
Windows-Paketmonitor	`pktmon`
Windows-DNS-UDP-Test	`Resolve-DnsName -Name <Name> -Server <DNS-IP> -DnsOnly`
Linux-UDP-Endpunkte	`sudo ss -lunp`
Linux-alle UDP-Sockets	`sudo ss -uanp`
Linux-UDP-Port	`sudo ss -lunp 'sport = :<Port>'`
Linux-Prozess zu UDP-Port	`sudo lsof -nP -iUDP:<Port>`
Linux-UDP-Zähler	`nstat -az`
Linux-Schnittstellenzähler	`ip -s link`
macOS-UDP-Sockets	`sudo lsof -nP -iUDP`
macOS-UDP-netstat	`netstat -anv -p udp`
macOS-UDP-Statistik	`netstat -s -p udp`
DNS-UDP-Test	`dig @<DNS-IP> <Name> <Typ>`
DNS-TCP-Vergleich	`dig @<DNS-IP> <Name> <Typ> +tcp`
Linux-Aufzeichnung	`sudo tcpdump -ni any '(udp port <Port>) or icmp or icmp6' -c 200`
macOS-Aufzeichnung	`sudo tcpdump -ni <Interface> '(udp port <Port>) or icmp or icmp6' -c 200`
Wireshark UDP-Port	`udp.port == <Port>`
Wireshark ICMPv4	`icmp`
Wireshark ICMPv6	`icmpv6`
ICMPv4 Port Unreachable	`icmp.type == 3 && icmp.code == 3`
ICMPv4 Fragmentation Needed	`icmp.type == 3 && icmp.code == 4`
ICMPv4 Time Exceeded	`icmp.type == 11`
ICMPv6 Port Unreachable	`icmpv6.type == 1 && icmpv6.code == 4`
ICMPv6 Packet Too Big	`icmpv6.type == 2`
ICMPv6 Time Exceeded	`icmpv6.type == 3`

Befehle und Maßnahmen, die nicht unkontrolliert als erste Diagnose verwendet werden dürfen

```
Stop-Process
taskkill
Stop-Service
Restart-Service
Set-NetFirewallRule
New-NetFirewallRule
Deaktivieren der Windows-Firewall
netsh int ip reset
netsh winsock reset
kill
kill -9
systemctl stop
systemctl restart
sysctl -w
iptables
nft
ufw disable
firewall-cmd --permanent
docker restart
docker stop
kubectl delete
kubectl rollout restart
unkontrollierte UDP-Portscans
UDP-Lasttests ohne Freigabe
ungefilterte Broadcasts
ungefilterte Multicasttests
pauschales Zulassen aller ICMP-Typen
vollständiges Blockieren von ICMP oder ICMPv6
willkürliche Erhöhung von Socketpuffern
unkontrollierte Änderung von UDP-Idle-Timeouts
```

Ein Neustart kann Socketzustände, Zähler, NAT-Zuordnungen und den für die Diagnose wichtigen Ausgangszustand zerstören.

Quellen

Standards

- RFC 768 – User Datagram Protocol
- RFC 8085 – UDP Usage Guidelines
- RFC 1122 – Requirements for Internet Hosts – Communication Layers
- RFC 792 – Internet Control Message Protocol
- RFC 1812 – Requirements for IP Version 4 Routers
- RFC 4443 – ICMPv6 for IPv6
- RFC 8200 – Internet Protocol, Version 6 Specification
- RFC 1191 – Path MTU Discovery
- RFC 8201 – Path MTU Discovery for IPv6
- RFC 4821 – Packetization Layer Path MTU Discovery
- RFC 8899 – Datagram PLPMTUD
- RFC 4890 – Recommendations for Filtering ICMPv6 Messages in Firewalls
- RFC 6633 – Deprecation of ICMP Source Quench Messages

Offizielle Microsoft-Dokumentation

- Microsoft Learn – Get-NetUDPEndpoint
- Microsoft Learn – netstat
- Microsoft Learn – PortQry command-line tool
- Microsoft Learn – Packet Monitor
- Microsoft Learn – Pktmon command formatting
- Microsoft Learn – Resolve-DnsName

Offizielle Linux- und Projektdokumentation

- Linux man-pages – ss(8)
- Linux man-pages – udp(7)
- Linux man-pages – icmp(7)
- Linux man-pages – ipv6(7)
- Linux man-pages – tcpdump(8)
- Linux man-pages – traceroute(8)
- ISC BIND 9 – Manual Pages und dig

Offizielle Wireshark-Dokumentation

- Wireshark Display Filter Reference – UDP
- Wireshark Display Filter Reference – ICMP

- Wireshark Display Filter Reference – ICMPv6
- Wireshark User’s Guide

Offizielle Containerdokumentation

- Docker Docs – Port publishing and mapping
- Docker Docs – Networking overview
- Kubernetes – Service
- Kubernetes – EndpointSlices
- Kubernetes – Network Policies

Für diese Seite wurden keine Community-Berichte oder Social-Media-Aussagen als technische Nachweise verwendet.
