9.7 Ephemeral Ports und Verbindungsgrenzen

Beim Aufbau einer ausgehenden TCP- oder UDP-Kommunikation benötigt das initiierende System normalerweise einen freien lokalen Quellport. Dieser kurzzeitig verwendete Port wird als dynamischer, temporärer oder ephemerer Port bezeichnet.

Beispiel:

```
Client:
192.0.2.100:53124

Server:
198.51.100.25:443

Datenfluss:
192.0.2.100:53124 -> 198.51.100.25:443/TCP
```

Der Server verwendet den bekannten Zielport `443`. Der Client verwendet den dynamisch ausgewählten Quellport `53124`.

Wenn kein geeigneter Quellport, Socket, Dateideskriptor, NAT-Eintrag oder Verbindungseintrag mehr verfügbar ist, können neue Verbindungen fehlschlagen, obwohl:

- das Ziel erreichbar ist;
- die Firewallregel stimmt;
- der Serverdienst läuft;
- bestehende Verbindungen weiterhin funktionieren;
- Ping erfolgreich ist;
- DNS korrekt auflöst.

Ziele

Nach der Bearbeitung dieser Seite soll nachvollziehbar geprüft werden können:

- welcher dynamische Portbereich auf einem System tatsächlich gilt;
- wie viele Ports dieses Bereichs belegt sind;
- welcher Prozess besonders viele Verbindungen erzeugt;
- ob `TIME_WAIT`, `CLOSE_WAIT` oder andere Zustände auffällig sind;
- ob lokale Porterschöpfung oder SNAT-Porterschöpfung vorliegt;
- ob ein NAT-Gateway, Load Balancer oder eine Firewall die Grenze erreicht;
- ob statt Ports eine andere Verbindungsgrenze betroffen ist;
- warum bestehende Sitzungen funktionieren, während neue Verbindungen scheitern;

- wie Verbindungswiederverwendung und Connection Pooling die Last reduzieren;
- wie eine Kapazitätsänderung sicher geplant und verifiziert wird.

Sicherheits- und Änderungsgrundsätze

Port- und Verbindungsgrenzen dürfen nicht ohne Ursachenanalyse verändert werden. Eine größere Grenze kann das Symptom verschieben, während die eigentliche Ursache bestehen bleibt.

Mögliche Ursachen sind:

- fehlerhafte Anwendungsschleife;
- fehlende Verbindungswiederverwendung;
- nicht geschlossene Sockets;
- ungewöhnlich hohe Verbindungsrate;
- langsames oder nicht antwortendes Ziel;
- zu großer Retry-Sturm;
- Portscan;
- Schadsoftware;
- DDoS;
- ungeeignete NAT-Architektur;
- fehlerhafte Timeouts;
- fehlende Kapazitätsplanung.

Nicht als erste Diagnosemaßnahme geeignet sind:

- Server vorsorglich neu starten;
- NAT-Gateway neu erstellen;
- Firewall-Sitzungstabelle vollständig löschen;
- Conntrack-Tabelle leeren;
- dynamischen Portbereich ungeprüft erweitern;
- `TIME_WAIT`-Dauer ungeprüft verkürzen;
- Socket- oder Dateideskriptorgrenzen maximal erhöhen;
- Verbindungsbegrenzungen deaktivieren;
- mehrere öffentliche IP-Adressen ohne Kapazitätsnachweis ergänzen;
- alle Anwendungspools neu starten;
- Keepalive- oder Timeoutwerte pauschal verändern;
- Sicherheitssoftware deaktivieren.

Vor jeder Änderung sind zu dokumentieren:

- Ausgangszustand;
- aktueller Portbereich;
- Anzahl belegter Ports;
- Verbindungszustände;
- betroffene Prozesse;
- häufigste Ziele;

- Verbindungsrate;
- NAT- oder SNAT-Auslastung;
- weitere Ressourcenlimits;
- Risiko;
- Rückweg;
- Erfolgskriterium;
- Testverfahren.

Portnummernbereiche

IANA unterscheidet folgende Bereiche:

Bereich	Bezeichnung	Typische Verwendung
`0–1023`	System Ports	bekannte und besonders geschützte Dienste
`1024–49151`	User Ports	registrierte Anwendungsdienste
`49152–65535`	Dynamic/Private Ports	dynamische oder private Verwendung

Der IANA-Bereich `49152–65535` ist eine allgemeine Einteilung. Er beweist nicht, dass ein Betriebssystem lokal genau diesen Bereich verwendet.

Der tatsächlich verwendete Bereich kann abhängen von:

- Betriebssystem;
- Betriebssystemversion;
- TCP oder UDP;
- IPv4 oder IPv6;
- Netzwerknamespace;
- Anwendung;
- expliziter Socketbindung;
- Systemrichtlinie;
- Containerplattform;
- Cloudplattform;
- NAT-Gerät;
- benutzerdefinierter Konfiguration.

Deshalb muss der aktuelle Bereich auf dem betroffenen System ausgelesen werden.

Ephemeren Quellport und Serverport unterscheiden

Normale Clientverbindung:

```
192.0.2.100:53124 -> 198.51.100.25:443/TCP
```

Port	Rolle
`53124`	dynamischer Quellport des Clients
`443`	Zielport des Serverdienstes

Rückverkehr:

```
198.51.100.25:443 -> 192.0.2.100:53124/TCP
```

Der Server benötigt nicht für jeden eingehenden Client einen neuen lokalen Listenerport. Alle HTTPS-Verbindungen können lokal den Serverport `443` verwenden, weil die vollständigen Verbindungen durch unterschiedliche Adressen und Ports unterscheidbar bleiben.

Das Verbindungstupel

Eine TCP- oder UDP-Kommunikation wird typischerweise über folgende Merkmale unterschieden:

```
Quelladresse
Quellport
Zieladresse
Zielport
Protokoll
```

Beispiel:

```
192.0.2.100:53124 -> 198.51.100.25:443/TCP
```

Eine gleichzeitig bestehende Verbindung kann denselben lokalen Port möglicherweise gegenüber einem anderen Ziel verwenden:

```
192.0.2.100:53124 -> 203.0.113.80:443/TCP
```

Ob und wie ein Betriebssystem lokale Ports zwischen verschiedenen Zielen wiederverwendet, hängt von Implementierung, Bindungsart und Socketoptionen ab.

Deshalb gilt nicht allgemein:

```
Anzahl dynamischer Ports =
maximale Gesamtzahl aller ausgehenden Verbindungen
```

Die Grenze muss immer im Zusammenhang mit Quelladresse, Zieladresse, Zielport, Protokoll, NAT und Verbindungszustand bewertet werden.

Anzahl eines Portbereichs berechnen

Allgemeine Berechnung:

```
Anzahl =
Endport - Startport + 1
```

Beispiel:

```
Startport:
49152

Endport:
65535

Anzahl:
65535 - 49152 + 1
= 16384 Ports
```

Davon können abzuziehen sein:

- reservierte Ports;
- bereits gebundene Ports;
- Ports in aktiven Verbindungen;
- noch nicht wiederverwendbare Verbindungstupel;
- Ports in `TIME_WAIT`;
- anwendungsspezifische Ausschlüsse;
- System- oder Plattformreservierungen;
- Sicherheitsrichtlinien;
- NAT- oder Providerbeschränkungen.

Die theoretische Anzahl ist deshalb keine garantierte Verbindungskapazität.

TCP und UDP getrennt betrachten

TCP und UDP besitzen getrennte Protokollräume. Derselbe numerische Port kann gleichzeitig für TCP und UDP verwendet werden.

Beispiel:

```
TCP 53124
UDP 53124
```

Das sind zwei unterschiedliche Transportendpunkte.

Zu prüfen sind daher getrennt:

- dynamischer TCP-Portbereich;
- dynamischer UDP-Portbereich;
- TCP-Verbindungen;
- UDP-Endpunkte;
- TCP-Time-Wait-Zustände;
- UDP-Tracking- und NAT-Timeouts;
- protokollspezifische Cloudlimits.

Eine funktionierende UDP-Verbindung beweist nicht, dass noch TCP-Ports verfügbar sind.

IPv4 und IPv6 getrennt prüfen

Betriebssysteme und Plattformen können Portbereiche oder Zustände für IPv4 und IPv6 getrennt verwalten.

Zu prüfen sind:

- verwendete Adressfamilie;
- IPv4-Portbereich;
- IPv6-Portbereich;
- Dual-Stack-Verhalten;
- Happy-Eyeballs-Verbindungen;
- getrennte Firewallregeln;
- getrennte NAT- oder SNAT-Pfade;
- IPv4-mapped IPv6 Sockets;
- Anwendung mit mehreren parallelen Verbindungsversuchen.

Eine Anwendung kann bei einem Namensaufruf mehrere IPv6- und IPv4-Verbindungen starten. Dadurch kann die Zahl der Verbindungsversuche höher sein als die Zahl sichtbarer Benutzeranfragen.

Lokale und übersetzte Ports unterscheiden

Ein NAT- oder SNAT-Gerät kann den lokalen Quellport verändern.

Beispiel:

```
Vor SNAT:
10.10.20.25:53124 -> 203.0.113.80:443

Nach SNAT:
198.51.100.5:61001 -> 203.0.113.80:443
```

Dabei existieren zwei unterschiedliche Portbetrachtungen:

Ebene	Port
lokaler Clientport	`53124`
externer SNAT-Port	`61001`

Der Client kann lokal noch Ports besitzen, während der SNAT-Portpool des Gateways erschöpft ist. Umgekehrt kann das NAT-Gateway noch Kapazität besitzen, während das lokale Betriebssystem keinen geeigneten Quellport mehr findet.

Lokale Porterschöpfung

Lokale Porterschöpfung liegt vor, wenn ein System für eine neue ausgehende Kommunikation keinen geeigneten lokalen Port beziehungsweise kein verwendbares lokales Verbindungstupel mehr zuweisen kann.

Typische Symptome:

- neue ausgehende TCP-Verbindungen schlagen fehl;
- bestehende TCP-Verbindungen funktionieren weiter;
- Fehler tritt nach längerer Laufzeit auf;
- Neustart behebt das Problem nur vorübergehend;
- viele Verbindungen besitzen denselben Prozess;
- viele Einträge befinden sich in `TIME_WAIT`;
- sehr viele Einträge befinden sich in `CLOSE_WAIT`;
- hohe Anzahl von Verbindungen zu demselben Ziel;
- DNS, Kerberos, RPC, Datenbank- oder API-Aufrufe scheitern gleichzeitig;
- eingehende Verbindungen können teilweise weiter funktionieren;
- Fehler verschwindet, sobald Verbindungen aus der Tabelle altern.

Mögliche Fehlermeldungen sind abhängig von Betriebssystem und Anwendung:

```
Cannot assign requested address
Address already in use
No buffer space available
```

```
Too many open files
Only one usage of each socket address is normally permitted
Connection timed out
Connection failed
```

Diese Meldungen sind nicht gleichbedeutend. Beispielsweise kann `Address already in use` auch auf eine einzelne explizite Portkollision hinweisen.

SNAT-Porterschöpfung

Bei SNAT teilen sich mehrere interne Systeme eine oder mehrere externe IP-Adressen.

Beispiel:

```
10.10.20.11 -> 198.51.100.5
10.10.20.12 -> 198.51.100.5
10.10.20.13 -> 198.51.100.5
```

Das NAT-Gateway muss die Verbindungen über externe Quellports eindeutig zuordnen.

Eine hohe Konzentration auf dasselbe Ziel kann besonders kritisch sein:

```
viele interne Systeme
        ->
eine öffentliche NAT-Adresse
        ->
eine Zieladresse
        ->
ein Zielport
```

Beispiel:

```
1000 interne Anwendungsinstanzen
        ->
198.51.100.5
        ->
203.0.113.80:443
```

Mögliche Symptome:

- nur ausgehende Verbindungen über das NAT-Gateway scheitern;
- interne Ziele funktionieren;
- bestehende Verbindungen funktionieren;
- neue Verbindungen zu einem häufig genutzten Ziel schlagen fehl;
- Verbindungen zu anderen Zieladressen funktionieren;
- Fehler tritt nur unter Last auf;
- Cloudmetrik zeigt Port Allocation Errors;
- zusätzliche NAT-IP verbessert das Verhalten;
- Connection Pooling reduziert die Fehler.

Porterschöpfung pro Ziel einordnen

Der kritische Grenzfall ist häufig nicht die Gesamtzahl aller Ziele, sondern eine hohe Zahl gleichzeitiger oder schnell wiederholter Verbindungen zu genau derselben Kombination aus:

```
Ziel-IP-Adresse
Zielport
Protokoll
```

Beispiel:

```
203.0.113.80:443/TCP
```

Ein API-Gateway, Datenbankserver, Proxy oder zentrales Authentifizierungssystem kann dadurch einen besonders konzentrierten Portbedarf erzeugen.

Zu prüfen sind:

- Anzahl unterschiedlicher Zieladressen;
- Anzahl unterschiedlicher Zielports;
- Verbindungen je Ziel;
- neue Verbindungen pro Sekunde;
- durchschnittliche Lebensdauer;
- Anteil wiederverwendeter Verbindungen;
- Retryrate;
- Timeoutdauer;
- DNS-Lastverteilung;
- NAT-IP-Adressen;
- Backend- oder Zielverteilung.

TIME_WAIT

Nach dem regulären Ende einer TCP-Verbindung kann das System das Verbindungstupel für eine bestimmte Zeit im Zustand `TIME_WAIT` halten.

Zweck ist unter anderem:

- verspätete Segmente einer alten Verbindung abzufangen;
- eine neue gleichartige Verbindung nicht mit alten Paketen zu vermischen;
- den abschließenden Verbindungsabbau zuverlässig behandeln zu können.

Viele `TIME_WAIT`-Einträge sind nicht automatisch ein Fehler. Sie zeigen jedoch eine hohe Rate kurzlebiger TCP-Verbindungen.

Kritisch wird die Situation, wenn gleichzeitig:

- neue ausgehende Verbindungen scheitern;
- der dynamische Portbereich stark belegt ist;
- Windows-Ereignisse auf Porterschöpfung hinweisen;
- ein einzelner Prozess besonders viele Verbindungen erzeugt;
- ein einzelnes Ziel stark konzentriert ist;
- die Einträge schneller entstehen als sie freigegeben werden.

Ein hoher `TIME_WAIT`-Bestand ohne Verbindungsfehler beweist keine aktuelle Porterschöpfung.

CLOSE_WAIT

`CLOSE_WAIT` bedeutet vereinfacht:

1. Die Gegenstelle hat das Ende der Verbindung angekündigt.
2. Das lokale Betriebssystem hat dies an die Anwendung gemeldet.
3. Die lokale Anwendung hat ihren Socket noch nicht vollständig geschlossen.

Viele dauerhaft bestehende `CLOSE_WAIT`-Einträge können auf eine Anwendung hinweisen, die Verbindungen nicht korrekt schließt.

Zu prüfen sind:

- betroffener Prozess;
- Alter der Verbindungen;
- Wachstum über die Zeit;
- Remoteziele;
- Anwendungsprotokolle;
- Thread- oder Taskzustand;
- Fehlerbehandlung;
- Connection Pool;
- Dateideskriptorverbrauch.

`CLOSE_WAIT` beweist nicht automatisch eine erschöpfte dynamische Portmenge. Es kann jedoch gleichzeitig Sockets, Dateideskriptoren, Speicher und lokale Verbindungstupel binden.

SYN_SENT

Viele Verbindungen im Zustand `SYN_SENT` bedeuten, dass das lokale System Verbindungsanfragen gesendet hat, aber der TCP-Handshake noch nicht abgeschlossen wurde.

Mögliche Ursachen:

- Ziel antwortet nicht;
- Firewall verwirft;
- Rückroute fehlt;
- Ziel ist überlastet;
- falsche Zieladresse;
- Paketverlust;
- zu lange Anwendungstimeouts;
- Retry-Sturm;
- SYN-Pakete werden schneller erzeugt als sie ablaufen.

Viele lang anhaltende `SYN_SENT`-Verbindungen können Ports und Sockets binden, obwohl keine erfolgreiche Anwendungskommunikation entsteht.

ESTABLISHED

Viele `ESTABLISHED`-Verbindungen können beabsichtigt sein, beispielsweise bei:

- Datenbank-Connection-Pools;
- HTTP-Keepalive;
- HTTP/2;
- WebSockets;
- Message Brokern;
- Streaming;
- Replikation;
- persistenten Agentenverbindungen.

Zu prüfen sind:

- sind die Verbindungen aktiv oder ungenutzt?
- entspricht die Anzahl der Konfiguration?
- werden alte Verbindungen ersetzt, ohne geschlossen zu werden?
- existiert eine obere Poolgrenze?
- besitzt jede Anwendungsinstanz einen eigenen Pool?
- wächst die Zahl dauerhaft?
- passt die Summe aller Instanzpools zur Zielkapazität?

Eine hohe Zahl etablierter Verbindungen ist nicht automatisch Porterschöpfung. Sie kann jedoch andere Grenzwerte erreichen.

FIN_WAIT und LAST_ACK

Viele Verbindungen in Zuständen des Verbindungsabbaus können auf Probleme beim sauberen Beenden hinweisen.

Zu prüfen sind:

- antwortet die Gegenstelle auf den Verbindungsabbau?
- besitzt die Anwendung geeignete Schließ- und Abbruchlogik?
- blockiert eine Firewall abschließende Pakete?
- existiert ein asymmetrischer Rückweg?
- werden Verbindungen durch Timeouts statt kontrolliert beendet?
- treten Paketverluste auf?
- ist das Ziel überlastet?

Die genaue Bedeutung ist anhand der TCP-Zustandsfolge und einer Paketaufzeichnung zu prüfen.

UDP und QUIC

UDP besitzt keinen TCP-Verbindungsabbau und keinen TCP-`TIME_WAIT`-Zustand. Trotzdem können UDP-Sockets, NAT-Zuordnungen und Connection-Tracking-Einträge Ports belegen.

Betroffene Anwendungen können sein:

- DNS;
- VoIP;
- Streaming;
- QUIC und HTTP/3;
- VPN;
- Telemetrie;
- Gaming;
- Monitoring.

Zu prüfen sind:

- lokale UDP-Endpunkte;
- NAT-Timeout;
- Conntrack-Timeout;
- Portwiederverwendung;
- Antwortverkehr;
- QUIC-Verbindungsrate;
- Anzahl externer Ziele;

- Firewallzustand;
- anwendungsspezifische Socketverwaltung.

Eine Analyse ausschließlich der TCP-Verbindungen übersieht UDP- oder QUIC-bedingte Engpässe.

RPC-Dynamik nicht mit Clientports verwechseln

Windows RPC kann nach dem Kontakt zum RPC Endpoint Mapper auf TCP-Port `135` einen dynamischen Serverport verwenden.

Vereinfachter Ablauf:

1. Client kontaktiert Ziel auf TCP `135`.
2. Endpoint Mapper nennt den dynamischen Serverport.
3. Client verbindet sich mit diesem Zielport.
4. Der Client verwendet dafür zusätzlich einen eigenen dynamischen Quellport.

Beispiel:

```
Client:
192.0.2.100:53124

RPC-Server:
198.51.100.25:52044

Verbindung:
192.0.2.100:53124 -> 198.51.100.25:52044/TCP
```

Dabei sind beide Ports dynamisch, aber mit unterschiedlichen Rollen:

Port	Rolle
`53124`	dynamischer Clientquellport
`52044`	dynamisch zugewiesener RPC-Serverzielport

Firewallregeln müssen diese Rollen korrekt berücksichtigen.

Porterschöpfung und andere Verbindungsgrenzen unterscheiden

Grenze	Typischer Befund
dynamischer lokaler Portbereich	kein geeigneter Quellport für neue Verbindung
SNAT-Portpool	NAT-Gerät kann keine neue Übersetzung anlegen

Grenze	Typischer Befund
Socketgrenze	Anwendung oder Kernel kann keinen weiteren Socket anlegen
Dateideskriptorgrenze	Prozess erhält beispielsweise `Too many open files`
Prozesshandlegrenze	Betriebssystem oder Prozess erreicht Handlegrenze
Conntrack-Tabelle	Firewall oder Router kann keinen neuen Zustand anlegen
Firewall-Sessionlimit	neue Sitzungen werden verworfen
NAT-Sessionlimit	neue NAT-Zuordnungen schlagen fehl
TCP-SYN-Backlog	neue Handshakes warten oder werden verworfen
Accept-Backlog	vollständig aufgebaute Verbindungen werden nicht schnell genug angenommen
Anwendungslimit	Dienst erlaubt nur definierte Zahl paralleler Verbindungen
Connection-Pool-Limit	Anwendung wartet auf freien Poolplatz
Thread- oder Workerlimit	Verbindungen werden nicht rechtzeitig verarbeitet
Datenbanklimit	maximale Datenbanksitzungen erreicht
Load-Balancer-Limit	Flow-, Port- oder Zielgrenze erreicht
Cloudquote	plattformspezifische Obergrenze erreicht
Zielsystemlimit	Gegenstelle begrenzt Quellen oder Verbindungsrate
API-Rate-Limit	Anwendung lehnt Anfragen trotz funktionierender TCP-Verbindung ab

Ein erfolgreicher TCP-Porttest beweist nicht, dass die Anwendung noch einen Datenbankpoolplatz oder Worker besitzt.

Windows: dynamischen Portbereich anzeigen

IPv4 TCP:

```
netsh int ipv4 show dynamicport tcp
```

IPv4 UDP:

```
netsh int ipv4 show dynamicport udp
```

IPv6 TCP:

```
netsh int ipv6 show dynamicport tcp
```

IPv6 UDP:

```
netsh int ipv6 show dynamicport udp
```

Bei aktuellen unterstützten Windows-Versionen ist der Standardbereich normalerweise:

```
Startport:
49152

Endport:
65535

Anzahl:
16384
```

Der tatsächliche Systemwert ist trotzdem mit `netsh` zu prüfen. TCP, UDP, IPv4 und IPv6 werden getrennt angezeigt.

Windows: TCP-Verbindungen anzeigen

Alle TCP-Verbindungen:

```
Get-NetTCPConnection
```

Etablierte Verbindungen:

```
Get-NetTCPConnection `
  -State Established
```

`TIME_WAIT`:

```
Get-NetTCPConnection `
  -State TimeWait
```

`CLOSE_WAIT`:

```
Get-NetTCPConnection `
  -State CloseWait
```

Ausgehende Verbindungsversuche:

```
Get-NetTCPConnection `
  -State SynSent
```

Nach Ziel filtern:

```
Get-NetTCPConnection `
  -RemoteAddress "198.51.100.25" `
  -RemotePort 443
```

Nach Prozess-ID filtern:

```
Get-NetTCPConnection `
  -OwningProcess <PID>
```

Prozess bestimmen:

```
Get-Process `
  -Id <PID>
```

Windows: Zustände zählen

```
Get-NetTCPConnection |
  Group-Object State |
  Sort-Object Count -Descending |
  Select-Object Count, Name
```

Mögliche Ausgabe:

```
Count Name
----- ----
 8200 TimeWait
```

```
 1450 Established
  200 CloseWait
   35 SynSent
```

Die Werte müssen mit einer Baseline und dem aktuellen Fehlerzeitpunkt verglichen werden.

Windows: Verbindungen je Prozess zählen

```
Get-NetTCPConnection |
  Group-Object OwningProcess |
  Sort-Object Count -Descending |
  Select-Object -First 20 Count, Name
```

Die Spalte `Name` enthält in diesem Fall die Prozess-ID.

Einzelne Prozess-IDs auflösen:

```
Get-Process `
  -Id <PID>
```

Ausführlicher Zusammenhang:

```
Get-NetTCPConnection |
  ForEach-Object {
    $process = Get-Process -Id $_.OwningProcess -ErrorAction SilentlyContinue

    [PSCustomObject]@{
      State         = $_.State
      LocalAddress  = $_.LocalAddress
      LocalPort     = $_.LocalPort
      RemoteAddress = $_.RemoteAddress
      RemotePort    = $_.RemotePort
      PID           = $_.OwningProcess
      Process       = $process.ProcessName
    }
  } |
  Sort-Object Process, State
```

Auf stark belasteten Systemen kann diese Auswertung umfangreich sein. Sie sollte gezielt und während des Fehlerzeitraums durchgeführt werden.

Windows: häufigste Ziele ermitteln

```
Get-NetTCPConnection |
  Where-Object State -ne Listen |
  Group-Object RemoteAddress, RemotePort |
  Sort-Object Count -Descending |
  Select-Object -First 20 Count, Name
```

Damit kann festgestellt werden, ob sich sehr viele Verbindungen auf ein einzelnes Ziel konzentrieren.

Windows: klassische Netstat-Auswertung

Verbindungen, Ports und Prozess-IDs:

```
netstat -ano
```

Mit ausführbarer Datei, administrative Eingabeaufforderung erforderlich:

```
netstat -anob
```

Gebundene Ports einschließlich bestimmter nicht aktiver TCP-Bindungen:

```
netstat -anoq
```

`TIME_WAIT` zählen:

```
netstat -ano | find /c "TIME_WAIT"
```

`CLOSE_WAIT` zählen:

```
netstat -ano | find /c "CLOSE_WAIT"
```

`SYN_SENT` zählen:

```
netstat -ano | find /c "SYN_SENT"
```

Ein einfacher Textzähler ist nur ein Indikator. Für die Zuordnung zu Prozess, Ziel, Zeit und Portbereich ist die vollständige Ausgabe erforderlich.

Windows: Ereignisse prüfen

Windows kann bei Porterschöpfung unter anderem TCP/IP-Ereignisse `4227` und `4231` protokollieren.

Gezielte Suche:

```
Get-WinEvent `
  -FilterHashtable @{
    LogName   = "System"
    Id        = 4227, 4231
    StartTime = (Get-Date).AddHours(-24)
  } |
  Select-Object TimeCreated, Id, ProviderName, Message
```

Die Ereignisse müssen zeitlich mit den tatsächlichen Verbindungsfehlern korreliert werden.

Ein hoher `TIME_WAIT`-Bestand allein bestätigt keine Porterschöpfung. Microsoft empfiehlt zusätzlich, reale ausgehende Verbindungsfehler und passende Ereignisse nachzuweisen.

Windows: Portbereich eines Tests berücksichtigen

Wenn der ausgelesene TCP-Bereich beispielsweise `49152–65535` lautet:

```
$startPort = 49152
$endPort   = 65535

Get-NetTCPConnection |
  Where-Object {
    $_.LocalPort -ge $startPort -and
    $_.LocalPort -le $endPort
  } |
  Group-Object State |
  Sort-Object Count -Descending |
```

```
Select-Object Count, Name
```

Die Werte müssen an den tatsächlich mit `netsh` ermittelten Bereich angepasst werden.

Linux: lokalen Portbereich anzeigen

```
sysctl net.ipv4.ip_local_port_range
```

Alternative:

```
cat /proc/sys/net/ipv4/ip_local_port_range
```

Beispielausgabe:

```
32768 60999
```

Das ist nur ein Beispiel. Die lokale Ausgabe ist maßgeblich.

Reservierte lokale Ports:

```
sysctl net.ipv4.ip_local_reserved_ports
```

Alternative:

```
cat /proc/sys/net/ipv4/ip_local_reserved_ports
```

`ip_local_port_range` und `ip_local_reserved_ports` sind getrennte Einstellungen. Beide beeinflussen, welche Ports für automatische Zuweisungen zur Verfügung stehen.

Linux: Socketübersicht

```
ss -s
```

Mögliche Bereiche der Zusammenfassung:

- Gesamtzahl der Sockets;
- TCP-Sockets;
- etablierte Verbindungen;

- geschlossene Zustände;
- `TIME_WAIT`;
- UDP-Sockets;
- RAW-Sockets;
- fragmentierte Zustände.

Die genaue Darstellung hängt von der installierten `iproute2`-Version ab.

Linux: TCP-Verbindungen anzeigen

Alle TCP-Sockets:

```
ss -tan
```

Mit Prozessinformationen:

```
sudo ss -tanp
```

Etablierte Verbindungen:

```
ss -tan state established
```

`TIME_WAIT`:

```
ss -tan state time-wait
```

`CLOSE_WAIT`:

```
ss -tan state close-wait
```

`SYN_SENT`:

```
ss -tan state syn-sent
```

Listener:

```
ss -lntp
```

Nach Ziel filtern:

```
ss -tanp \
  dst 198.51.100.25:443
```

Linux: Zustände zählen

```
ss -Htan |
  awk '{print $1}' |
  sort |
  uniq -c |
  sort -nr
```

Nur `TIME_WAIT` zählen:

```
ss -Htan state time-wait |
  wc -l
```

Nur `CLOSE_WAIT` zählen:

```
ss -Htan state close-wait |
  wc -l
```

Nur `SYN_SENT` zählen:

```
ss -Htan state syn-sent |
  wc -l
```

Linux: Kernel-Socketstatistik

IPv4 und allgemeine Socketstatistik:

```
cat /proc/net/sockstat
```

IPv6:

```
cat /proc/net/sockstat6
```

Mögliche Felder sind unter anderem:

- verwendete Sockets;
- TCP in use;
- orphaned;
- `TIME_WAIT`;
- zugewiesener Speicher;
- UDP in use;
- RAW in use;
- fragmentierte Pakete.

Die Werte sind Momentaufnahmen und müssen mit einer Baseline verglichen werden.

Linux: Dateideskriptorgrenzen

Grenze der aktuellen Shell:

```
ulimit -n
```

Grenzen eines laufenden Prozesses:

```
cat /proc/<PID>/limits
```

Offene Dateideskriptoren eines Prozesses zählen:

```
ls -1 /proc/<PID>/fd |
  wc -l
```

Systemweite Dateideskriptorinformationen:

```
cat /proc/sys/fs/file-nr
```

Systemweite Obergrenze:

```
sysctl fs.file-max
```

Sockets werden unter Unix-ähnlichen Betriebssystemen über Dateideskriptoren angesprochen. Ein Prozess kann deshalb die Dateideskriptorgrenze erreichen, obwohl im dynamischen Portbereich noch Ports frei sind.

Linux: Listener- und Backloggrenzen

Aktuelle Listener:

```
ss -lnt
```

Globale Obergrenze des Listen-Backlogs:

```
sysctl net.core.somaxconn
```

TCP-SYN-Backlog:

```
sysctl net.ipv4.tcp_max_syn_backlog
```

Bei einem Listening Socket zeigt `ss` abhängig von Version und Kontext unter anderem aktuelle und maximale Warteschlangenwerte an.

Ein Backlogproblem betrifft eingehende Verbindungen und ist nicht mit ausgehender Porterschöpfung gleichzusetzen.

Linux: Conntrack zusätzlich prüfen

Anzahl aktuell verfolgter Verbindungen:

```
sysctl net.netfilter.nf_conntrack_count
```

Maximalwert:

```
sysctl net.netfilter.nf_conntrack_max
```

Statistiken, sofern `conntrack-tools` installiert ist:

```
sudo conntrack -S
```

Eine volle Conntrack-Tabelle kann neue Verbindungen verhindern, obwohl das lokale Betriebssystem noch Quellports besitzt.

---

macOS: Portbereich prüfen

Verfügbare Portbereichsparameter anzeigen:

```
sysctl -a |
  awk '/net\.inet\.ip\.portrange/'
```

Je nach macOS-Version können unter anderem folgende Werte verfügbar sein:

```
sysctl net.inet.ip.portrange.first
```

```
sysctl net.inet.ip.portrange.last
```

```
sysctl net.inet.ip.portrange.hifirst
```

```
sysctl net.inet.ip.portrange.hilast
```

Die tatsächlich vorhandenen Parameter und Werte sind auf dem betroffenen Mac zu prüfen.

---

macOS: Verbindungen und Prozesse prüfen

TCP-Verbindungen:

```
netstat -anv -p tcp
```

TCP-Sockets mit Prozessen:

```
sudo lsof -nP -iTCP
```

Nur etablierte TCP-Verbindungen:

```
sudo lsof -nP -iTCP -sTCP:ESTABLISHED
```

Nur Listener:

```
sudo lsof -nP -iTCP -sTCP:LISTEN
```

TCP-Statistik:

```
netstat -s -p tcp
```

Interaktive Netzwerksicht:

```
nettop -m tcp
```

Die genaue Ausgabe unterscheidet sich je nach macOS-Version.

Prozess statt nur Portbereich untersuchen

Die wichtigste Frage lautet häufig:

```
Welcher Prozess erzeugt die Verbindungen?
```

Zu dokumentieren sind:

- Prozessname;
- Prozess-ID;
- Dienst;
- ausführbare Datei;
- Benutzerkonto;
- Startzeit;
- Anzahl TCP-Verbindungen;
- Anzahl UDP-Sockets;
- Zustandsverteilung;
- häufigste Ziele;
- neue Verbindungen pro Sekunde;
- Retryrate;
- Fehlerprotokolle;
- Connection-Pool-Konfiguration;
- zuletzt durchgeführte Änderung.

Ein Portbereich sollte nicht erweitert werden, bevor der verursachende Prozess und sein Verbindungsverhalten bekannt sind.

Verbindungsrate und gleichzeitige Verbindungen unterscheiden

Metrik	Bedeutung
gleichzeitige Verbindungen	aktuell parallel bestehende Datenflüsse
neue Verbindungen pro Sekunde	Geschwindigkeit des Verbindungsaufbaus
geschlossene Verbindungen pro Sekunde	Geschwindigkeit des Abbaus
`TIME_WAIT`-Bestand	noch nicht vollständig wiederverwendbare TCP-Zustände
durchschnittliche Lebensdauer	Zeit bis zum Verbindungsende
Retryrate	zusätzliche Versuche nach Fehler oder Timeout
Wiederverwendungsquote	Anfragen pro bestehender Verbindung

Eine Anwendung mit nur wenigen gleichzeitig aktiven Anfragen kann trotzdem sehr viele Ports verbrauchen, wenn sie für jede Anfrage eine neue Verbindung erstellt.

Planungsnäherung für kurzlebige Verbindungen

Vereinfachte Abschätzung:

```
dauerhaft mögliche neue Verbindungen pro Sekunde
≈
verfügbare Portkombinationen / Belegungsdauer
```

Beispiel:

```
verfügbare Ports:
16384

angenommene Belegungsdauer:
240 Sekunden

Näherung:
16384 / 240
≈ 68 neue Verbindungen pro Sekunde
```

Diese Rechnung ist nur ein vereinfachtes Planungsmodell für eine konzentrierte Zielkombination. Das reale Verhalten hängt unter anderem ab von:

- Betriebssystem;
- Portauswahl;

- Portwiederverwendung;
- Zieladressen;
- Zielports;
- Quelladressen;
- NAT;
- TCP-Zustand;
- Socketoptionen;
- Sicherheitsrichtlinien;
- Plattformlimits.

Die Näherung darf nicht als garantierter Grenzwert verwendet werden.

Connection Pooling

Connection Pooling hält eine begrenzte Menge bestehender Verbindungen bereit und verwendet sie für mehrere Vorgänge erneut.

Mögliche Vorteile:

- weniger TCP-Handshakes;
- weniger TLS-Handshakes;
- geringere Verbindungsrate;
- weniger `TIME_WAIT`;
- geringerer Portverbrauch;
- geringere CPU-Last;
- geringere Latenz;
- weniger Last auf NAT und Firewall.

Zu prüfen sind:

- minimale Poolgröße;
- maximale Poolgröße;
- Pool pro Prozess;
- Pool pro Instanz;
- Pool pro Benutzer;
- Pool pro Ziel;
- Idle-Timeout;
- maximale Lebensdauer;
- Health Check;
- Validierung vor Wiederverwendung;
- Verhalten bei Zielausfall;
- Retrylogik;
- Schließen beim Anwendungsende.

Ein unbegrenzt großer Pool beseitigt keine Kapazitätsgrenzen und kann das Zielsystem überlasten.

HTTP-Verbindungen wiederverwenden

Für HTTP-basierte Anwendungen sind relevant:

- HTTP/1.1 Keep-Alive;
- HTTP/2-Multiplexing;
- HTTP/3 über QUIC;
- Proxyverbindungen;
- maximale parallele Verbindungen pro Ziel;
- DNS-Änderungen;
- TLS-Sitzungswiederverwendung;
- Clientbibliothek;
- Lebensdauer des HTTP-Clients.

Ein häufiges Fehlmuster ist die Erstellung eines neuen HTTP-Clients und einer neuen TCP-/TLS-Verbindung für jede einzelne Anfrage.

Besser ist eine kontrollierte Wiederverwendung mit:

- begrenztem Pool;
- korrekten Timeouts;
- Fehlerbehandlung;
- DNS-Aktualisierung;
- Gesundheitsprüfung;
- sauberem Schließen beim Anwendungsende.

Die konkrete Implementierung hängt von Programmiersprache, Framework und Bibliothek ab.

Retries kontrollieren

Ein langsames oder ausgefallenes Ziel kann einen Retry-Sturm auslösen.

Beispiel:

```
100 Anfragen
× 5 sofortige Wiederholungen
= 500 Verbindungsversuche
```

Bei mehreren Anwendungsinstanzen:

```
500 Versuche
× 20 Instanzen
= 10000 Verbindungsversuche
```

Zu prüfen sind:

- maximale Anzahl von Wiederholungen;
- Wartezeit;
- exponentielles Backoff;
- zufällige Streuung oder Jitter;
- Circuit Breaker;
- Gesamtzeitlimit;
- Wiederholung nur idempotenter Operationen;
- gemeinsames Ziel aller Instanzen;
- Verhalten bei DNS- oder TLS-Fehlern.

Sofortige und unbegrenzte Retries können eine kleine Störung zu Port-, Socket- und Zielüberlastung verstärken.

AWS NAT Gateway

AWS dokumentiert für ein NAT Gateway pro zugeordneter IPv4-Adresse bis zu `55.000` gleichzeitige Verbindungen zu jedem eindeutigen Ziel.

Ein eindeutiges Ziel wird dabei bestimmt durch:

```
Ziel-IP-Adresse
Zielport
Protokoll
```

Zu prüfen sind:

- NAT-Gateway-Typ;
- zugeordnete IPv4-Adressen;
- Anzahl Verbindungen je Ziel;
- CloudWatch-Metriken;
- Port Allocation Errors;
- Verbindungsversuche;
- Idle Timeouts;
- Availability Zone;
- Routing;
- Zielkonzentration.

Mögliche relevante CloudWatch-Metriken sind abhängig vom Ressourcentyp unter anderem:

- `ActiveConnectionCount`;
- `ConnectionAttemptCount`;
- `ErrorPortAllocation`;

- `IdleTimeoutCount`;
- `PacketsDropCount`.

Zusätzliche NAT-IP-Adressen können die Kapazität erhöhen. Vorher muss jedoch geprüft werden, ob die Anwendung unnötig viele neue Verbindungen erstellt.

Azure NAT Gateway

Azure NAT Gateway stellt pro öffentlicher IPv4-Adresse `64.512` SNAT-Ports bereit. Ein NAT Gateway kann mehrere öffentliche IP-Adressen verwenden.

Microsoft dokumentiert außerdem eine Grenze gleichzeitiger Verbindungen je eindeutigem Zielendpunkt. Die jeweils aktuelle Plattformdokumentation ist für die eingesetzte Ressource zu prüfen.

Zu prüfen sind:

- Anzahl öffentlicher IP-Adressen;
- zugeordnete Subnetze;
- SNAT Connection Count;
- Failed SNAT Connection Count;
- Zieladresse und Zielport;
- Verbindungsrate;
- TCP- und UDP-Nutzung;
- Idle Timeouts;
- Load-Balancer-Outbound-Regeln;
- Azure Firewall;
- NAT-Gateway-Zuordnung.

TCP und UDP besitzen beim Azure NAT Gateway getrennte SNAT-Portinventare.

Azure Load Balancer

Ein Azure Load Balancer kann ausgehende SNAT-Ports nach konfigurierten Outbound Rules und Backendanzahl zuteilen.

Zu prüfen sind:

- Load-Balancer-SKU;
- Outbound Rule;
- Anzahl Backendinstanzen;
- zugewiesene SNAT-Ports je Instanz;
- verwendete öffentliche IP-Adressen;
- Idle Timeout;
- Port-Reuse;

- SNAT-Port-Metriken;
- Port Allocation Failures;
- NAT-Gateway-Zuordnung.

Eine ungleichmäßige Verbindungsverteilung kann dazu führen, dass eine einzelne Backendinstanz ihre Portzuteilung erreicht, obwohl andere Instanzen noch Kapazität besitzen.

Google Cloud NAT

Google Cloud NAT kann Ports statisch oder dynamisch auf VMs verteilen.

Zu prüfen sind:

- NAT-IP-Adressen;
- minimale Ports pro VM;
- maximale Ports pro VM;
- dynamische Portzuweisung;
- Endpoint-Independent Mapping;
- Portnutzung je VM;
- verworfene Pakete;
- Zielkonzentration;
- VM- und Podanzahl;
- Verbindungsrate;
- Logging und Monitoring.

Bei dynamischer Portzuweisung kann eine schnelle Laststeigerung vorübergehend Pakete verlieren, während die Portzuweisung erweitert wird. Lasttests sollten deshalb auch den Anstieg der Verbindungsrate berücksichtigen.

Container und Kubernetes

Mehrere Container oder Pods können sich einen Host-, Node- oder NAT-Portpool teilen.

Zu prüfen sind:

- Netzwerkmodus;
- eigener oder gemeinsamer Network Namespace;
- Node-IP;
- Pod-IP;
- Host-NAT;
- kube-proxy;
- CNI-Plugin;
- eBPF-Datenpfad;
- Cloud NAT;
- Service Mesh;

- Sidecar Proxy;
- Zahl der Pods;
- Poolgröße pro Pod;
- Retryverhalten;
- Rolling Deployment;
- Horizontal Scaling.

Beispiel:

```
50 Pods
× 200 ausgehende Poolverbindungen
= 10000 Verbindungen
```

Ein Scale-out kann den Gesamtportbedarf erhöhen, obwohl jede einzelne Instanz unverändert arbeitet.

Proxies und Gateways

Forward Proxies, Reverse Proxies, API Gateways und Service-Mesh-Sidecars initiieren eigene ausgehende Verbindungen.

Dadurch existieren getrennte Verbindungsabschnitte:

```
Client -> Proxy
Proxy  -> Ziel
```

Zu prüfen sind für jeden Abschnitt:

- Quelladresse;
- Quellport;
- Zieladresse;
- Zielport;
- Connection Pool;
- maximale Verbindungen;
- Idle Timeout;
- Retrylogik;
- NAT;
- TLS-Terminierung;
- Zahl der Worker;
- Backendverteilung.

Der Client kann ausreichend Ports besitzen, während der Proxy seinen ausgehenden Portpool erschöpft.

Firewalls und Connection Tracking

Zusätzlich zum Quellport muss eine Stateful Firewall oder ein NAT-Router häufig einen Sitzungseintrag anlegen.

Zu prüfen sind:

- aktuelle Sitzungsanzahl;
- maximales Sitzungslimit;
- Neuverbindungen pro Sekunde;
- NAT-Übersetzungen;
- Verbindungen je Quelle;
- Verbindungen je Ziel;
- halb offene Sitzungen;
- UDP-Pseudositzungen;
- Timeouts;
- Drop-Zähler;
- Tabellenfüllstand;
- Cluster-Synchronisation.

Eine Portbereichserweiterung am Client kann die Firewall stärker belasten und dort die nächste Grenze erreichen.

Serverseitige Verbindungsgrenzen

Wenn eingehende Verbindungen scheitern, sind zusätzlich zu prüfen:

- Listener vorhanden;
- Listeneradresse;
- Listenerport;
- SYN-Backlog;
- Accept-Backlog;
- maximale Clients;
- Workerzahl;
- Threadzahl;
- Prozessgrenze;
- Dateideskriptorgrenze;
- Datenbankpool;
- Zielabhängigkeiten;
- CPU;
- Speicher;
- Garbage Collection;
- Lizenzgrenze;
- Rate Limit;
- Reverse Proxy;

- Load Balancer.

Typische Unterscheidung:

Befund	Mögliche Einordnung
SYN erhält keine Antwort	Netzwerk, Firewall, Backlog oder Überlastung
SYN erhält RST	kein Listener oder aktive Ablehnung
TCP-Handshake erfolgreich, Anwendung wartet	Worker-, Pool- oder Backendproblem
bestehende Clients funktionieren, neue nicht	Backlog, Verbindungslimit oder Ressourcenerschöpfung
nur ausgehende Serveraufrufe scheitern	lokale Ports, NAT, Pool oder Zielproblem

---

Baseline erfassen

Eine Port- und Verbindungsdiagnose benötigt Vergleichswerte.

Sinnvolle Baseline:

```
dynamischer Portbereich
TCP-Verbindungen gesamt
UDP-Sockets gesamt
ESTABLISHED
TIME_WAIT
CLOSE_WAIT
SYN_SENT
LISTEN
Verbindungen je Prozess
Verbindungen je Ziel
neue Verbindungen pro Sekunde
Dateideskriptoren je Prozess
Conntrack-Auslastung
NAT-Sitzungen
SNAT-Portnutzung
Fehlerzähler
CPU und Speicher
```

Messpunkte:

- Normalbetrieb;
- Lastspitze;
- unmittelbar vor der Störung;

- während der Störung;
- nach der Erholung;
- nach einer kontrollierten Maßnahme.

Hypothese und Gegenbeweis formulieren

Beispiel:

```
Hypothese:
Der Anwendungsdienst erstellt für jeden API-Aufruf eine neue
HTTPS-Verbindung zum Ziel 203.0.113.80:443. Die Verbindungen
verbleiben anschließend in TIME_WAIT, bis der dynamische
TCP-Portbereich erschöpft ist.

Erwarteter Befund:
Während der Störung schlagen neue ausgehende Verbindungen fehl.
Windows protokolliert Ereignis 4227 oder 4231.
Die Anzahl der TIME_WAIT-Verbindungen zum Ziel
203.0.113.80:443 ist sehr hoch und gehört überwiegend zum
gleichen Prozess.

Gegenbeweis:
Der dynamische Portbereich besitzt während der Störung ausreichend
freie Kapazität, es existieren keine passenden TCP/IP-Ereignisse,
und der Verbindungsfehler tritt bereits vor der lokalen Portzuweisung
oder ausschließlich am Zielsystem auf.

Testmethode:
Portbereich, Get-NetTCPConnection, Prozesszuordnung,
Systemereignisse und eine kontrollierte Testverbindung zeitlich
korrelieren.

Risiko:
Nur lesende Diagnose.

Erfolgskriterium:
Portbelegung, verursachender Prozess, Zielkonzentration und
Verbindungsfehler sind für denselben Zeitpunkt nachgewiesen.
```

Kontrollierte Maßnahmen

Maßnahme	Voraussetzung	Risiko	Rückweg
Connection Pooling aktivieren	hohe Rate kurzlebiger Verbindungen bestätigt	fehlerhafte Altverbindungen können wiederverwendet werden	vorherige Clientkonfiguration
HTTP-Verbindungen wiederverwenden	neue Verbindung pro Anfrage nachgewiesen	DNS- und Lebensdauerverhalten beachten	vorheriges Clientverhalten
Poolgröße begrenzen	unkontrolliert großer Pool bestätigt	Wartezeiten bei Last	vorherige Poolgröße
Retries begrenzen	Retry-Sturm nachgewiesen	einzelne Anfrage schlägt früher fehl	vorherige Retrykonfiguration
Backoff und Jitter ergänzen	gleichzeitige Wiederholungen vieler Instanzen	längere Wiederherstellungszeit	alte Retryrichtlinie
Anwendung korrigieren	nicht geschlossene Sockets bestätigt	Deploymentrisiko	vorherige Anwendungsversion
Portbereich erweitern	tatsächliche lokale Porterschöpfung bestätigt	Überschneidungen und höhere Folgelast	vorherigen Bereich wiederherstellen
zusätzliche Quell-IP verwenden	Kapazitätsbedarf und Architektur bestätigt	Routing, Firewall und Freigaben ändern sich	alte IP-Zuordnung
zusätzliche NAT-IP verwenden	SNAT-Erschöpfung bestätigt	Kosten und Ziel-Allowlisting	zusätzliche IP entfernen
NAT-Gateway skalieren	Plattformmetrik bestätigt Engpass	Kosten und Architekturänderung	vorherige Ressourcengröße
Timeouts anpassen	ungeeigneter Timeout nachgewiesen	veraltete Zustände oder höhere Last	alten Timeoutwert wiederherstellen
Dateideskriptorgrenze erhöhen	FD-Limit nachgewiesen und Speicher ausreichend	höherer Ressourcenverbrauch	alten Grenzwert wiederherstellen
Conntrack-Kapazität erhöhen	Tabellenlimit nachgewiesen	höherer Speicherverbrauch	alten Wert wiederherstellen
Serverpool skalieren	Zielkapazität nachgewiesen	zusätzliche Verbindungen und Kosten	Skalierung zurücknehmen

Die Ursachenbehebung in der Anwendung ist einer dauerhaften rein infrastrukturellen Vergrößerung vorzuziehen, wenn die Anwendung unnötig viele Verbindungen erzeugt.

Vollständiger Diagnoseablauf

1. Exakte Fehlermeldung erfassen.
2. Datum, Uhrzeit und Zeitzone dokumentieren.
3. Betroffene Anwendung bestimmen.

4. Ausgehende und eingehende Verbindungen unterscheiden.
5. Quell- und Zielsystem bestimmen.
6. Quelladresse und Zieladresse erfassen.
7. Quellport und Zielport erfassen.
8. TCP, UDP und QUIC unterscheiden.
9. IPv4 und IPv6 unterscheiden.
10. Umfang der Störung bestimmen.
11. Prüfen, ob bestehende Verbindungen weiter funktionieren.
12. Prüfen, ob nur neue Verbindungen scheitern.
13. Dynamischen Portbereich auslesen.
14. Reservierte und ausdrücklich gebundene Ports berücksichtigen.
15. Anzahl aktueller TCP-Verbindungen erfassen.
16. UDP-Sockets erfassen.
17. TCP-Zustände gruppieren.
18. `TIME_WAIT` auswerten.
19. `CLOSE_WAIT` auswerten.
20. `SYN_SENT` auswerten.
21. Verbindungen nach Prozess gruppieren.
22. Verbindungen nach Ziel gruppieren.
23. Verbindungsrate messen.
24. Retryrate bestimmen.
25. Connection-Pool-Konfiguration prüfen.
26. Dateideskriptoren und Handles prüfen.
27. Listener- und Backloggrenzen prüfen.
28. Anwendungs- und Workerlimits prüfen.
29. Conntrack-Auslastung prüfen.
30. Firewall- und NAT-Sitzungsgrenzen prüfen.
31. Vor-NAT- und Nach-NAT-Ports dokumentieren.
32. SNAT-Portnutzung prüfen.
33. Cloudmetriken und Port Allocation Errors prüfen.
34. Container-, Node- und Podkontext berücksichtigen.
35. Proxy- und Sidecar-Verbindungen getrennt prüfen.
36. Anwendungsprotokolle auswerten.
37. Betriebssystemereignisse auswerten.
38. kontrollierte Testverbindung durchführen.
39. bei Bedarf Paketaufzeichnung durchführen.
40. Hypothese und Gegenbeweis formulieren.
41. genau eine kontrollierte Maßnahme vorbereiten.
42. Risiko, Rückweg und Erfolgskriterium dokumentieren.
43. Maßnahme freigeben und umsetzen.
44. neue Verbindungen unter gleicher Last testen.
45. Portbelegung und Zustände erneut messen.
46. SNAT-, Conntrack- und Fehlerzähler erneut prüfen.
47. ursprüngliche Anwendung verifizieren.
48. Nebenwirkungen auf andere Dienste prüfen.
49. temporäre Diagnosemaßnahmen zurücknehmen.

50. Ursache, Maßnahme und Prävention dokumentieren.

Befundmatrix

Befund	Mögliche Einordnung	Nächster Nachweis
bestehende Verbindungen funktionieren, neue nicht	Port-, NAT-, Conntrack- oder Backloggrenze	jeweilige Auslastungszähler prüfen
viele `TIME_WAIT`	hohe Rate kurzlebiger TCP-Verbindungen	Prozess, Ziel und Verbindungsrate
viele `CLOSE_WAIT`	Anwendung schließt Sockets nicht	Prozess und Anwendungscode prüfen
viele `SYN_SENT`	Ziel antwortet nicht oder Rückweg fehlt	Paketaufzeichnung und Route
viele `ESTABLISHED`	Pool oder Langzeitverbindungen	Aktivität, Poolgrenze und Alter
Windows-Ereignis 4227	lokaler Endpunkt kann nicht sicher wiederverwendet werden	Portbestand und Prozesszuordnung
Windows-Ereignis 4231	Zuweisung eines ephemeren Ports fehlgeschlagen	dynamischen Bereich und Belegung prüfen
`Cannot assign requested address`	kein verwendbarer lokaler Endpunkt oder falsche Bindung	Ports, lokale IP und Anwendung
`Too many open files`	Dateideskriptorgrenze	Prozesslimits und offene FDs
lokale Ports frei, Cloudfehler bleibt	SNAT- oder Plattformgrenze	Cloudmetrik und NAT-Gateway
nur ein Ziel betroffen	Verbindungen pro Zielkombination erschöpft	Ziel-IP, Port und Protokoll gruppieren
andere Ziele funktionieren	zielbezogene NAT- oder Gegenstellenbegrenzung	Zielkonzentration und Plattformlimit
Neustart hilft nur vorübergehend	Zustände werden geleert, Ursache bleibt	Wachstum über die Zeit erfassen
zusätzliche NAT-IP hilft	SNAT-Kapazität wahrscheinlich betroffen	Port Allocation Error vor und nach Änderung
mehr Pods verschlechtern Fehler	Gesamtpool und Verbindungsrate steigen	Poolgröße × Podanzahl berechnen
nur Lastspitzen betroffen	Portzuweisung oder Rate überschritten	Verbindungen pro Sekunde messen
TCP funktioniert, UDP nicht	getrennte UDP-Port- oder NAT-Grenze	UDP-Sockets und NAT-Timeout
HTTP/1.1 problematisch, HTTP/2 besser	Multiplexing reduziert Verbindungen	Verbindungsrate vergleichen
Firewall-Sessions voll	Connection Tracking oder Sessionlimit	Firewallstatistik und Drops
Serverhandshake gelingt, Anfrage wartet	Worker-, Pool- oder Backendgrenze	Anwendungsmetriken

Befund	Mögliche Einordnung	Nächster Nachweis
Listener-Queue wächst	Anwendung akzeptiert zu langsam	CPU, Worker und Backlog
Pool erschöpft, Ports frei	Anwendungslimit statt Portlimit	Poolwartezeit und Poolbelegung
viele Retries	Fehler verstärkt Verbindungslast	Backoff- und Retrykonfiguration
Portbereich angepasst, Fehler wandert zur Firewall	nachgelagerte Grenze erreicht	Firewall- und Conntrack-Kapazität

---

Typische Diagnosefehler

- IANA-Portbereich ungeprüft als lokalen Betriebssystembereich annehmen.
- Quellport und Zielport verwechseln.
- RPC-Zielports und Clientquellports gleichsetzen.
- TCP und UDP nicht getrennt prüfen.
- IPv4 und IPv6 nicht getrennt prüfen.
- Lokalen Port und SNAT-Port verwechseln.
- Portanzahl als garantierte Gesamtverbindungszahl interpretieren.
- Zieladresse und Zielport bei der Kapazität ignorieren.
- Nur Gesamtverbindungen und nicht Verbindungen je Ziel prüfen.
- `TIME_WAIT` allein als Beweis der Porterschöpfung verwenden.
- `CLOSE_WAIT` als normalen Portablauf behandeln.
- `SYN_SENT` ohne Netzwerkprüfung als Portproblem bewerten.
- Prozesszuordnung nicht durchführen.
- Verbindungsrate nicht messen.
- Retries und Backoff nicht prüfen.
- Connection Pooling nicht berücksichtigen.
- Poolgröße nur pro Instanz und nicht über alle Instanzen berechnen.
- Container, Pods oder Sidecars übersehen.
- Proxy- und Zielverbindung als eine einzige Verbindung betrachten.
- Dateideskriptorgrenze mit Portgrenze verwechseln.
- Conntrack-Tabelle nicht prüfen.
- Firewall-Sessionlimit ignorieren.
- NAT-Gateway-Metriken nicht prüfen.
- Cloudlimits aus einer anderen Ressource übertragen.
- Portbereich als erste Maßnahme erweitern.
- `TIME_WAIT`-Dauer vorsorglich verkürzen.
- Server neu starten, bevor Beweisdaten gesichert sind.
- Alle Sitzungen löschen.
- Anwendungslimit durch Infrastrukturvergrößerung verdecken.
- Nur eine einzelne Testverbindung durchführen.
- Last- und Langzeittest nach der Maßnahme auslassen.
- Temporäre Diagnoseaufzeichnungen aktiv lassen.

Verifikation

Nach einer Maßnahme müssen mindestens folgende Punkte geprüft werden:

- tatsächlicher dynamischer Portbereich ist dokumentiert;
- reservierte Ports überschneiden sich nicht unerwartet;
- neue Verbindungen werden zuverlässig aufgebaut;
- bestehende Verbindungen bleiben stabil;
- `TIME_WAIT`-Bestand entspricht der erwarteten Verbindungsrate;
- `CLOSE_WAIT` wächst nicht dauerhaft;
- `SYN_SENT` wächst nicht dauerhaft;
- verursachender Prozess ist eindeutig bestimmt;
- Connection Pool arbeitet innerhalb seiner Grenze;
- Verbindungen werden wiederverwendet;
- Retryrate entspricht der Vorgabe;
- Dateideskriptorverbrauch bleibt unterhalb des Limits;
- Listener- und Accept-Backlogs sind unauffällig;
- Conntrack-Tabelle besitzt ausreichende Reserve;
- Firewall-Sitzungslimit wird nicht erreicht;
- NAT-Sitzungen besitzen ausreichende Reserve;
- SNAT-Portfehler treten nicht mehr auf;
- Cloudmetriken zeigen keine fehlgeschlagenen Portzuweisungen;
- TCP und UDP funktionieren, sofern erforderlich;
- IPv4 und IPv6 funktionieren, sofern erforderlich;
- alle Anwendungsinstanzen funktionieren;
- mehrere Ziele funktionieren;
- konzentrierte Verbindungen zum Hauptziel funktionieren;
- Lastspitze wird erfolgreich verarbeitet;
- ursprüngliche Anwendung funktioniert;
- keine neue Sicherheitsumgehung entstand;
- keine unnötige allgemeine Portfreigabe besteht;
- temporäre Diagnosen wurden zurückgenommen;
- Ursache, Maßnahme und Prävention wurden dokumentiert.

Eine erfolgreiche Einzelverbindung direkt nach einem Neustart ist keine ausreichende Verifikation. Porterschöpfung muss unter repräsentativer Last und über einen ausreichend langen Zeitraum ausgeschlossen werden.

Dokumentationsvorlage

```
Störung:
<exakte Beschreibung>

Datum und Uhrzeit:
<Zeitpunkt mit Zeitzone>
```

Betroffene Anwendung:
<Anwendung oder Dienst>

Betroffenes System:
<Hostname und IP-Adresse>

Datenfluss:
<Quell-IP:Port -> Ziel-IP:Port/Protokoll>

Adressfamilie:
<IPv4 oder IPv6>

Dynamischer Portbereich:
<Startport bis Endport>

Reservierte Ports:
<Portliste oder keine>

Verbindungen gesamt:
<Anzahl>

ESTABLISHED:
<Anzahl>

TIME_WAIT:
<Anzahl>

CLOSE_WAIT:
<Anzahl>

SYN_SENT:
<Anzahl>

UDP-Sockets:
<Anzahl>

Hauptverursachender Prozess:
<Name und PID>

Häufigstes Ziel:
<IP-Adresse und Port>

Verbindungsrate:
<neue Verbindungen pro Sekunde>

Connection Pool:
<Min, Max, Nutzung und Wiederverwendung>

Retryverhalten:
<Anzahl, Backoff und Jitter>

Dateideskriptoren:
<verwendet und Limit>

Conntrack:
<Count, Max und Fehler>

Firewall-Sitzungen:
<verwendet und Limit>

NAT vor Übersetzung:
<Quell-IP:Port>

NAT nach Übersetzung:
<öffentliche IP und SNAT-Port>

SNAT-Portnutzung:
<Metrik und Auslastung>

Plattformlimit:
<Ressource und dokumentierte Grenze>

Hypothese:
<vermutete Ursache>

Erwarteter Nachweis:
<messbarer Befund>

```
Gegenbeweis:
<widerlegender Befund>

Nachgewiesene Ursache:
<technischer Nachweis>

Durchgeführte Maßnahme:
<genau eine kontrollierte Änderung>

Risiko:
<mögliche Nebenwirkungen>

Rückweg:
<Rollback>

Verifikation:
<Lasttest, Portnutzung und Anwendungstest>

Prävention:
<Monitoring, Pooling oder Kapazitätsplanung>
```

Checkliste

- [ ] exakte Fehlermeldung dokumentiert
- [ ] Datum, Uhrzeit und Zeitzone erfasst
- [ ] betroffene Anwendung bestimmt
- [ ] Quelladresse erfasst
- [ ] Quellport erfasst
- [ ] Zieladresse erfasst
- [ ] Zielport erfasst
- [ ] TCP und UDP unterschieden
- [ ] IPv4 und IPv6 unterschieden
- [ ] ausgehende und eingehende Verbindung unterschieden
- [ ] Fehlerumfang bestimmt
- [ ] bestehende Verbindungen getestet
- [ ] neue Verbindungen getestet

- [ ] dynamischen TCP-Portbereich ausgelesen
- [ ] dynamischen UDP-Portbereich ausgelesen
- [ ] reservierte Ports geprüft
- [ ] theoretische Portanzahl berechnet
- [ ] tatsächliche Portbelegung erfasst
- [ ] TCP-Verbindungen gezählt
- [ ] UDP-Sockets gezählt
- [ ] `ESTABLISHED` gezählt
- [ ] `TIME_WAIT` gezählt
- [ ] `CLOSE_WAIT` gezählt
- [ ] `SYN_SENT` gezählt
- [ ] FIN-Zustände berücksichtigt
- [ ] Prozesszuordnung durchgeführt
- [ ] häufigste Prozesse bestimmt
- [ ] häufigste Ziele bestimmt
- [ ] Verbindungen je Ziel bestimmt
- [ ] Verbindungsrate gemessen
- [ ] Retryrate gemessen
- [ ] Connection Pool geprüft
- [ ] Poolgröße über alle Instanzen berechnet
- [ ] HTTP-Verbindungswiederverwendung geprüft
- [ ] Dateideskriptorgrenze geprüft
- [ ] offene Dateideskriptoren gezählt
- [ ] Prozesshandles berücksichtigt
- [ ] Listener geprüft
- [ ] SYN-Backlog geprüft
- [ ] Accept-Backlog geprüft
- [ ] Worker- und Threadgrenzen geprüft
- [ ] Anwendungslimit geprüft
- [ ] Datenbankverbindungslimit geprüft
- [ ] Conntrack-Auslastung geprüft
- [ ] Firewall-Sitzungsgrenze geprüft
- [ ] NAT-Sitzungsgrenze geprüft
- [ ] Vor-NAT-Port dokumentiert

- [ ] Nach-NAT-Port dokumentiert
- [ ] SNAT-Portpool geprüft
- [ ] Cloudmetriken geprüft
- [ ] Port Allocation Errors geprüft
- [ ] Cloudressource und Ressourcentyp bestimmt
- [ ] Container- oder Podkontext berücksichtigt
- [ ] Proxy- und Sidecar-Verbindungen berücksichtigt
- [ ] Betriebssystemereignisse geprüft
- [ ] Anwendungsprotokolle geprüft
- [ ] Baseline verglichen
- [ ] Hypothese formuliert
- [ ] Gegenbeweis definiert
- [ ] Risiko dokumentiert
- [ ] Rückweg dokumentiert
- [ ] nur eine kontrollierte Maßnahme durchgeführt
- [ ] Einzeltest durchgeführt
- [ ] Lasttest durchgeführt
- [ ] Langzeitverhalten geprüft
- [ ] ursprüngliche Anwendung verifiziert
- [ ] Nebenwirkungen geprüft
- [ ] temporäre Diagnosen zurückgenommen
- [ ] Ursache und Prävention dokumentiert

Schnellreferenz

Aufgabe	Befehl
Windows IPv4-TCP-Portbereich	`netsh int ipv4 show dynamicport tcp`
Windows IPv4-UDP-Portbereich	`netsh int ipv4 show dynamicport udp`
Windows IPv6-TCP-Portbereich	`netsh int ipv6 show dynamicport tcp`
Windows IPv6-UDP-Portbereich	`netsh int ipv6 show dynamicport udp`
Windows TCP-Verbindungen	`Get-NetTCPConnection`
Windows etablierte Verbindungen	`Get-NetTCPConnection -State Established`
Windows `TIME_WAIT`	`Get-NetTCPConnection -State TimeWait`
Windows `CLOSE_WAIT`	`Get-NetTCPConnection -State CloseWait`

Aufgabe	Befehl
Windows `SYN_SENT`	`Get-NetTCPConnection -State SynSent`
Windows Netstat	`netstat -ano`
Windows gebundene Ports	`netstat -anoq`
Windows relevante Ereignisse	`Get-WinEvent -FilterHashtable @{LogName="System"; Id=4227,4231}`
Linux-Portbereich	`sysctl net.ipv4.ip_local_port_range`
Linux-reservierte Ports	`sysctl net.ipv4.ip_local_reserved_ports`
Linux-Socketübersicht	`ss -s`
Linux-TCP-Sockets	`ss -tan`
Linux-TCP mit Prozessen	`sudo ss -tanp`
Linux `TIME_WAIT`	`ss -tan state time-wait`
Linux `CLOSE_WAIT`	`ss -tan state close-wait`
Linux `SYN_SENT`	`ss -tan state syn-sent`
Linux-Listener	`ss -lntp`
Linux-Socketstatistik	`cat /proc/net/sockstat`
Linux-FD-Limit der Shell	`ulimit -n`
Linux-Prozesslimits	`cat /proc/<PID>/limits`
Linux-systemweites FD-Limit	`sysctl fs.file-max`
Linux-Listen-Backlog	`sysctl net.core.somaxconn`
Linux-SYN-Backlog	`sysctl net.ipv4.tcp_max_syn_backlog`
Linux-Conntrack-Anzahl	`sysctl net.netfilter.nf_conntrack_count`
Linux-Conntrack-Maximum	`sysctl net.netfilter.nf_conntrack_max`
macOS-Portparameter	`sysctl -a | awk '/net\.inet\.ip\.portrange/'`
macOS-TCP-Verbindungen	`netstat -anv -p tcp`
macOS-TCP-Prozesse	`sudo lsof -nP -iTCP`
macOS-Netzwerkübersicht	`nettop -m tcp`

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

```
netsh int ipv4 set dynamicport ...
netsh int ipv6 set dynamicport ...
sysctl -w net.ipv4.ip_local_port_range=...
sysctl -w net.ipv4.tcp_fin_timeout=...
```

```
TcpTimedWaitDelay ungeprüft verändern
MaxUserPort ungeprüft verändern
ulimit pauschal maximal erhöhen
LimitNOFILE ungeprüft erhöhen
fs.file-max ungeprüft erhöhen
net.core.somaxconn ungeprüft erhöhen
net.ipv4.tcp_max_syn_backlog ungeprüft erhöhen
nf_conntrack_max ungeprüft erhöhen
conntrack -F
Firewall-Sitzungen vollständig löschen
NAT-Sitzungen vollständig löschen
Server vorsorglich neu starten
NAT-Gateway vorsorglich neu erstellen
öffentliche IP-Adressen ohne Kapazitätsplanung ergänzen
Connection Pool unbegrenzt vergrößern
Timeouts pauschal verkürzen
Sicherheitsbegrenzungen deaktivieren
```

Quellen

Standards und Portregistrierung

- IANA – Service Name and Transport Protocol Port Number Registry
- RFC 6335 – Internet Assigned Numbers Authority Procedures for Service Names and Transport Protocol Port Numbers
- RFC 7605 – Recommendations on Using Assigned Transport Port Numbers
- RFC 6056 – Recommendations for Transport-Protocol Port Randomization
- RFC 9293 – Transmission Control Protocol
- RFC 768 – User Datagram Protocol
- RFC 4787 – NAT Behavioral Requirements for UDP
- RFC 7857 – Updates to NAT Behavioral Requirements

Microsoft Windows

- Microsoft Learn – TCP/IP Port Exhaustion Troubleshooting
- Microsoft Learn – Get-NetTCPConnection
- Microsoft Learn – Configure RPC Dynamic Port Allocation with Firewalls

Linux Kernel und Manpages

- Linux Kernel Documentation – IP Sysctl
- Linux Kernel Documentation – Netfilter Conntrack Sysfs Variables
- Linux man-pages – `IP_LOCAL_PORT_RANGE`
- Linux man-pages – `ip(7)`
- Linux man-pages – `tcp(7)`
- Linux man-pages – `socket(7)`
- iproute2 – `ss(8)`
- Netfilter – Conntrack Tools User Manual

Amazon Web Services

- AWS – NAT Gateway Basics
- AWS – Work with NAT Gateways
- AWS – Troubleshoot NAT Gateways

Microsoft Azure

- Microsoft Learn – Source Network Address Translation with Azure NAT Gateway
- Microsoft Learn – Azure NAT Gateway Resource
- Microsoft Learn – Troubleshoot Azure NAT Gateway Connectivity
- Microsoft Learn – Azure Load Balancer Outbound Connections
- Microsoft Learn – Troubleshoot Azure Load Balancer Outbound Connectivity

Google Cloud

- Google Cloud – Cloud NAT IP Addresses and Ports
- Google Cloud – Troubleshoot Cloud NAT

Für diese Seite wurden keine Community-Beiträge oder Hersteller-Social-Media-Aussagen als fachlicher Nachweis verwendet.
