

Aufgabe 1

sliding window

Das Sliding Window bezeichnet auf der Senderseite eine logische Beschränkung der Gesamtzahl an Paketen, die noch durch den Empfänger quittiert werden müssen. Das heißt, ein Sender kann nur dann Pakete senden, wenn der durch das sliding window bzw. den entsprechenden window threshold festgesetzte Schwellenwert noch nicht erreicht ist. Allerdings gleitet (slide) das Fenster mit jedem quittierten Paket (ACK) um eins weiter, d.h. der Sender kann mit jeder Quittung mindestens ein weiteres Paket senden, solange der window threshold nicht erreicht ist. So beginnt der Sender z.B. im slow start mode (s.u.) zunächst 1 Paket zu senden, nach empfangen des ACKs wird die Anzahl der Pakete auf 2 erhöht usw. bis der window threshold erreicht ist. In dies der Fall wird für jede Quittung (ACK) stets nur 1 neues Paket gesendet. Die Verwendung eines sliding window durch Sender und Empfänger verhindert somit die Verstopfung (congestion) des Netzwerks.

TCP Tahoe

TCP Tahoe ist eine frühe TCP Version die einen Congestion Control Algorithmus verwendet und Ende der 80er Jahre in Berkeley entwickelt wurde. TCP Tahoe umfasst eine slow start, eine AIMD und eine fast retransmit phase. Die Slow start phase dauert solange, bis das Datenaufkommen den slow start threshold erreicht. Ist der threshold erreicht, verdoppelt der slow start Algorithmus die Größe des sliding window und passt den threshold an. Die slow start Phase dauert solange bis die Größe des Fensters der Größe des thresholds entsprechen.

An die slow start phase schließt sich die AIMD phase an. In dieser Phase wird die Fenstergröße additiv um 1 erhöht und der threshold auf die Hälfte der Fenstergröße verringert.

Die fast retransmit phase wird durch den loss detection Algorithmus von Tahoe ausgelöst, wenn der host 3 DUP ACKs empfängt, d.h. Paketverlust aufgetreten ist.

TCP Reno

TCP Reno ist eine Weiterentwicklung von TCP Tahoe, das um fast recovery ergänzt wurde. Wenn der loss detection Algorithmus 3 DUP ACKs registriert, wird zunächst der fast retransmit Algorithmus aufgerufen. Falls es während des fast retransmit weiterhin zu Paketverlusten kommt, wird die Fenstergröße um 50% reduziert. Der Grund dafür ist, dass 3 DUP ACKs empfangen wurden, d.h. Pakete zum Empfänger gelangen und somit das Netzwerk zwar funktioniert, aber verstopft ist (congested).

Wenn Paketverlust durch den retransmission timeout entdeckt wird, wird die Fenstergröße auf ihren Initialwert zurückgesetzt, da der retransmission timeout eine größere Verstopfung des Netzwerks signalisiert

TCP Vegas

Reno und Tahoe konnten Verstopfungen (congestion) erst bei tatsächlichem Paketverlust feststellen. TCP Vegas kann bereits vor dem Auftreten von Paketverlusten Verstopfungen im Netzwerk registrieren und die Größe des Fensters anpassen. Dafür verwendet TCP Vegas die round trip time der gesendeten Pakete. Dabei wird der tatsächliche und der erwartete Durchsatz mit einem min threshold und einem max threshold verglichen und bei einem Über- oder Unterschreiten dieser Schwellenwerte die Fenstergröße angepasst.

Protokolle

- Ethernet (Ebene 2, Datalink Layer) - weil: arbeitet direkt auf dem physischen Übertragungsmedium, d.i. Netzwerkkarte
- ARP (Ebene 2 - 3, network / data link) - weil: arbeitet direkt auf Ethernet, aber löst IPv4 und MAC Adressen auf
- IP, ICMP (Ebene 3 Network Layer) - weil: logische Adressierung
- NAT (Ebene 3 - 4) - weil: modifiziert IP Adressen und Ports
- UDP, TCP, NCP, QUIC (Ebene 4, Transport layer) - weil: Transportdienste
- SSH, FTP, HTTP, SMTP, DHCP, DNS (Ebene 7, Application Layer)
- weil: Arbeiten auf TCP und sind anwendungsbezogen

Aufgabe 2

a) 4

Befehl: nmap -sn 192.168.10.0/24

Erklärung -sn: ping scan, kein port scan

b) Linux

Output: Running: Linux 5.X

OS CPE: cpe:o:linux:linux_kernel:5

OS details: Linux 5.0 - 5.14

Network Distance: 21 hops

Befehl: nmap -O scanme.nmap.org

Erklärung -O: Enable OS detection (alternativ: *-osscan-guess* aggressivere OS detection)

c) kein Output bekommen

Befehl: nmap -sn --script whois-domain.nse nmap.org

Erklärung: -sn: siehe oben

--script whois-domain.nse führt ein Script aus, dass versucht Informationen über den Domänennamen des Ziels *nmap.org* zu erhalten

d)

Befehl: nmap -Pn -p<portnumber> -oG <logfilename.gnmap> <target networks>

Erklärung: -Pn: alle hosts werden als *online* betrachtet

-p: festlegen der Portbereiche, die gescannt werden sollen

-oG: Output scan in Grepable Format in eine log file

e) Bei einem Syn Scan schickt nmap lediglich SYN Pakete und bricht die Verbindung bei einem SYN/ACK vom Zielhost ab. SYN Scan wird verwendet um verhältnismässig schnell viele Ports auf einem Zielhost zu scannen, da Syn Scans unauffällig sind. Der Parameter für einen Syn Scan ist -sS.

f) 80: http, 22: ssh, 443: https

Aufgabe 3

capture filter: udp port 67 or udp port 68 or udp port 546 or udp port 547

Aufgabe 4

Initialisierung:

Es werden die Kosten für die von einem Knoten aus unmittelbar erreichbaren Nachbarknoten in die Routingtabelle des Knotens geschrieben:

von A: kann B mit Kosten 3 und C mit Kosten 23 durch je 1 Kante erreicht werden

von B: kann A mit Kosten 3 und C mit Kosten 2 durch je 1 Kante erreicht werden

von C: kann A mit Kosten 23 und B mit Kosten 2 und D mit Kosten 5 durch je 1 Kante erreicht werden

von D: kann C mit Kosten 5 durch 1 Kante erreicht werden

Aktualisierung 1:

Nachdem in die Routingtabelle für jeden Knoten mit den Kosten für die unmittelbar erreichbaren Nachbarn initialisiert ist, werden mithilfe der Routingtabellen der unmittelbaren Nachbarn die Routingtabellen für jeden Knoten für die mittelbaren Nachbarn vervollständigt.

für A

- wird ein Weg zu B via C mit Kosten 25 hinzugefügt. Die Kosten ergeben sich aus $A \xrightarrow{23} C$ und $C \xrightarrow{2} B$
- wird ein Weg zu C via B mit Kosten 5 hinzugefügt. Die Kosten ergeben sich aus $A \xrightarrow{3} B$ und $B \xrightarrow{2} C$
- wird ein Weg zu D via C mit Kosten 28 hinzugefügt. Die Kosten ergeben sich aus $A \xrightarrow{23} C$ und $C \xrightarrow{5} D$

für B

- wird ein Weg zu A via C mit Kosten 25 hinzugefügt. Die Kosten ergeben sich aus $B \xrightarrow{2} C$ und $C \xrightarrow{23} A$
- wird ein Weg zu C via A mit Kosten 26 hinzugefügt. Die Kosten ergeben sich aus $B \xrightarrow{3} A$ und $A \xrightarrow{23} C$
- wird ein Weg zu D via C mit Kosten 7 hinzugefügt. Die Kosten ergeben sich aus $B \xrightarrow{2} C$ und $C \xrightarrow{5} D$

für C

- wird ein Weg zu A via B mit Kosten 5 hinzugefügt. Die Kosten ergeben sich aus $C \xrightarrow{2} B$ und $B \xrightarrow{3} A$
- wird ein Weg zu B via A mit Kosten 26 hinzugefügt. Die Kosten ergeben sich aus $C \xrightarrow{23} A$ und $A \xrightarrow{3} B$

für D

- wird ein Weg zu A via C mit Kosten 10 hinzugefügt. Die Kosten ergeben sich aus $D \xrightarrow{5} C$ und $C \xrightarrow{5} A$
- wird ein Weg zu B via C mit Kosten 7 hinzugefügt. Die Kosten ergeben sich aus $D \xrightarrow{5} C$ und $C \xrightarrow{2} B$

Aktualisierung 2:

In der 2. Aktualisierung werden die verbliebenen leeren Felder der einzelnen Routingtabellen ausgefüllt, d.i.

für A: wird ein Weg zu B via A mit Kosten 26 hinzugefügt.

für B: wird ein Weg zu D via A mit Kosten 31 hinzugefügt.

für C:

- wird ein Weg zu A via D mit Kosten 33 hinzugefügt.
- wird ein Weg zu B via D mit Kosten 12 hinzugefügt.
- wird ein Weg zu D via A mit Kosten 51 hinzugefügt.
- wird ein Weg zu D via B mit Kosten 9 hinzugefügt.

Aktualisierung und endgültiges Ergebnis

Nachdem die Routingtabellen ausgefüllt sind, werden die Verbindungen mit den jeweils kürzesten Pfaden zu den entsprechenden Knoten aktualisiert

Algorithmus

Bestimme für alle Knoten die Nachbarschaft und initialisiere anschließend die Routingtabelle für jeden Knoten mit den Wegen zu den Knoten aus seiner Nachbarschaft. Ergänze anschließend für jeden Knoten die neu hinzugekommenen Wege bis entweder die Tabelle vollständig ausgefüllt ist, oder es keine neuen Wege mehr gibt. Bestimme abschließend zu jedem Weg in der Routingtabelle den jeweils kürzesten Weg.