Lehrstuhl für Betriebssysteme
Fakultät für Informatik
Technische Universität München

Grundlagen Rechnernetze und Verteilte Systeme

IN0010, SoSe 2018

Übungsblatt 8

11. Juni – 15. Juni 2018

Hinweis: Mit * gekennzeichnete Teilaufgaben sind ohne Lösung vorhergehender Teilaufgaben lösbar.

Aufgabe 1 Neighbor Discovery Protocol und IP-Fragmentierung bei IPv6

In Abbildung 1 ist eine Anordnung von Netzkomponenten mit ihren MAC-Adressen dargestellt. PC1 und PC2 seien mittels SLAAC sowohl Link-Local (LL) als auch Global-Unique (GU) Adressen zugewiesen. Für letztere werde das Präfix `2001:db8:1::/64` (PC1/R1) bzw. `2001:db8:2::/64` (PC2/R2) verwendet.

PC1 sendet ein IP-Paket mit 1400 B Nutzdaten an PC2. Die MTU auf dem WAN-Link zwischen R1 und R2 betrage 1280 B[1]. Innerhalb der lokalen Netzwerke gelte die für Ethernet übliche MTU von 1500 B.

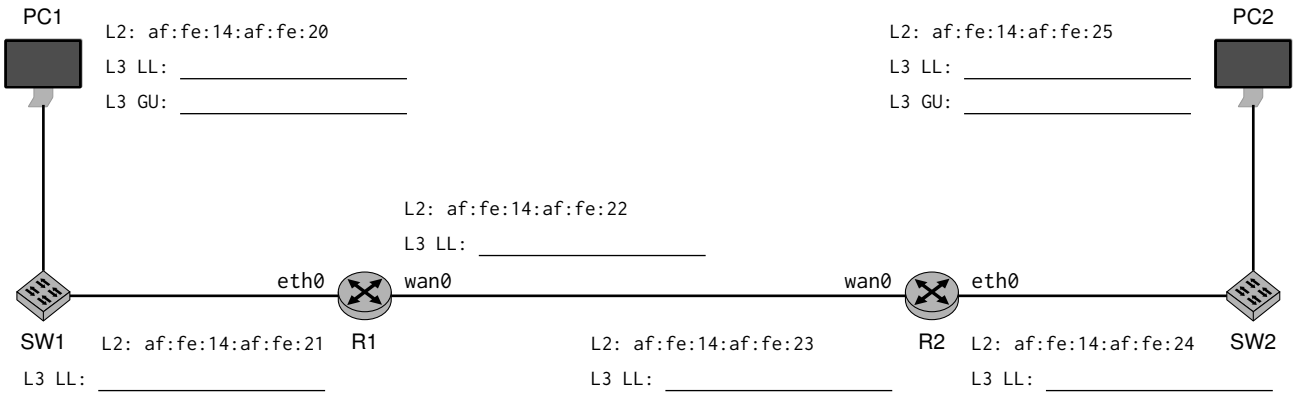


Abbildung 1: Netztopologie

Zunächst soll die Adressvergabe mittels SLAAC nachvollzogen werden

a)* Bestimmen Sie die Link-Local Adressen aller Interfaces.

Siehe Vorlesung:

- PC1: `af:fe:14:af:fe:20` → `fe80::adfe:14ff:feaf:fe20`
- R1.eth0: `af:fe:14:af:fe:21` → `fe80::adfe:14ff:feaf:fe21`
- R1.eth1: `af:fe:14:af:fe:22` → `fe80::adfe:14ff:feaf:fe22`
- R2.eth1: `af:fe:14:af:fe:23` → `fe80::adfe:14ff:feaf:fe23`
- R2.eth0: `af:fe:14:af:fe:24` → `fe80::adfe:14ff:feaf:fe24`
- PC2: `af:fe:14:af:fe:25` → `fe80::adfe:14ff:feaf:fe25`

[1] Dies entspricht der minimalen MTU, die laut RFC 2460 Schicht 2 für IPv6 unterstützen muss.

Hinweis: Das zweite Bit des ersten Oktetts jeder MAC-Adresse wird invertiert.

Grund: Manuell vergebene IPv6-Adressen haben häufig einen Interface-Identifier der Form `::abcd`, d.h. die ersten 48 bit sind 0. Schließt man nun von einer solchen IPv6-Adresse auf die zugrundeliegende MAC-Adresse, wäre das vorletzte Bit deren ersten Oktetts 0, was auf eine global eindeutige MAC-Adresse hinweise würde – was offensichtlich falsch ist, da diese ja von einer manuell vergebenen IPv6-Adresse stammt.
Würde man dieses Bit nicht invertieren, müssten alle manuell vergebenen Interface-Identifier von der Form `200::????` sein (oder, falls die einmalige Abkürzung mehrerer Nullgruppen bereits im Subnet-Identfier lag, von der Form `200:0:0:????`).

b) Bestimmen Sie die Global-Unique Adressen von PC1 und PC2. Nehmen Sie dazu an, dass Router R1 mit dem Präfix `2001:db8:1::/64` und Router R2 mit `2001:db8:2::/64` konfiguriert ist.

Die Herleitung geschieht analog zu den Link-Local-Adressen, allerdings mit dem Präfix des jeweiligen Routers, welche über Router Advertisements PC1 und PC2 bekannt gemacht werden.

- `af:fe:14:af:fe:20` → `2001:db8:1:0:adfe:14ff:feaf:fe20`
- `af:fe:14:af:fe:25` → `2001:db8:2:0:adfe:14ff:feaf:fe25`

c)* An welcher Stelle im Netzwerk wird die Fragmentierung stattfinden?

Direkt an PC1, da bei IPv6 keine Fragmentierung an Routern stattfindet.

d)* In wie viele Fragmente muss das Paket mindestens aufgeteilt werden?

Die MTU (Maximum Transmission Unit) ist die maximale Größe eines Pakets auf Schicht 3 inkl. Header. Sie entspricht also genau der maximalen Größe der Payload auf Schicht 2. Im Falle von Fragmentierung werden die einzelnen Fragmente jeweils einen IPv6-Header der Länge 40 B sowie einen Fragment Header der Länge 8 B tragen. Sofern keine weiteren Extension Header zum Einsatz kommen, erhalten wir demnach:

$$N = \left\lceil \frac{1400\,\mathrm{B}}{1280\,\mathrm{B} - 40\,\mathrm{B} - 8\,\mathrm{B}} \right\rceil = 2$$

e) Bestimmen Sie die Größe der L3-SDU für jedes Fragment.

Pro Fragment können 1280 B − 40 B − 8 B = 1232 B Nutzdaten übertragen werden. Da es sich dabei auch um ein Vielfaches von acht handelt (Fragment Offset ist in Vielfachen von 8 B angegeben), entspricht dies auch der tatsächlich übertragbaren Nutzdatenmenge.

Das erste Fragment hat daher eine eine Payload von 1232 B und das zweite eine Payload von 168 B.

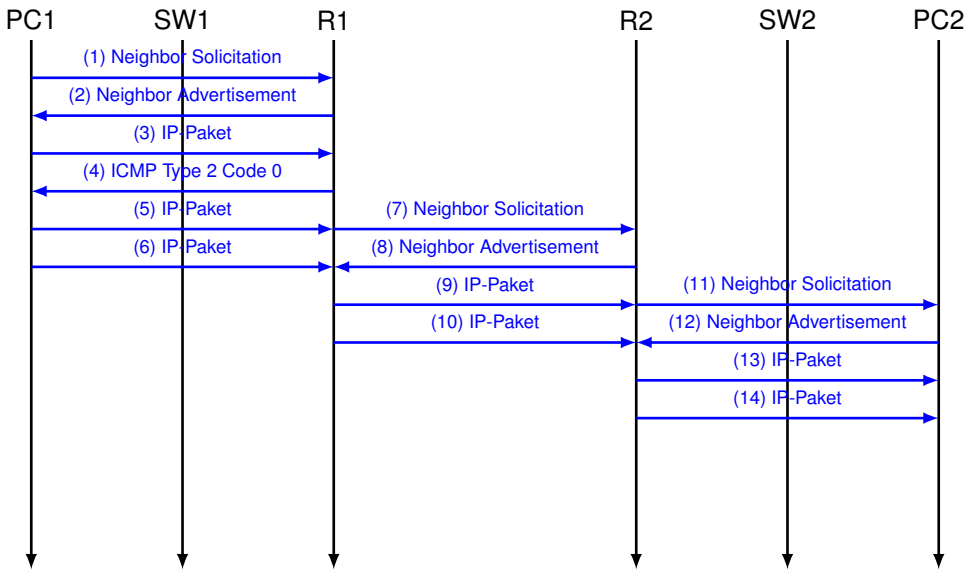
f)* An welcher Stelle im Netzwerk werden die Fragmente reassembliert?

Erst der Empfänger, hier also PC2, reassembliert die Fragmente wieder. Tatsächlich kann i. A. kein anderer Knoten die Reassemblierung durchführen, da die Fragmente jeweils einzelne und voneinander unabhängige Pakete darstellen. Dies bedeutet insbesondere, dass sie unabhängig voneinander geroutet werden und daher u. U. verschiedene Wege zum Ziel nehmen können – das sieht man aus dem einfachen Beispiel in Abbildung 1 natürlich nicht, da es hier nur einen Pfad zwischen PC1 und PC2 gibt.

g) Skizzieren Sie ein einfaches Weg-Zeit-Diagramm, welches **alle Rahmen** berücksichtigt, die auf den jeweiligen Verbindungen übertragen werden müssen. **Nennen Sie die Art der ausgetauschten Rahmen und geben Sie den Rahmen Nummern (1,2,3,...).** (Das Diagramm muss nicht maßstabsgetreu sein. Serialisierungszeiten und Ausbreitungsverzögerungen sind zu vernachlässigen.)

Gehen Sie davon aus, dass derzeit keinerlei Mappings zwischen IP- und MAC-Adressen gecached sind.

Nummerieren Sie die einzelnen Pakete Spaltenweise (Spalte $\triangleq$ Bereich z. B. zwischen R1 und R2).



Paket (4) ist eine ICMP-Fehlermeldung “Packet too big”, welche insbesondere die MTU auf dem nachfolgenden Linkabschnitt enthält. PC1 kann daraufhin die Fragmentierung lokal durchführen.

h) Bestimmen Sie die Destination-MAC-Adresse des ersten übertragenen Rahmens.

Es handelt sich um die Solicited-Node Address, welche laut Vorlesung das Präfix `ff02::1:ff00:0/104` hat. Die letzten 24 bit werden durch die letzten drei Oktette der angefragten IP-Adresse ersetzt, welche in diesem Fall die Link-Local Adresse von R1 ist. Demnach ist lautet die Solicited Node Adress `ff02::1:ffaf:fe21`. Von dieser wiederum lässt sich nach Vorlesung die zugehörige Multicast MAC-Adresse `33:33:ff:af:fe:21` ableiten.

Am Ende dieses Übungsblatts finden Sie Vordrucke für Ethernet-Header, ICMPv6 und IP-Header (mehr als benötigt). Es ist nicht notwendig, den Header binär auszufüllen. Achten Sie lediglich darauf, dass Sie die Zahlenbasis deutlich Kennzeichnen, z. B. `0x10` für hexadezimal oder $63_{(10)}$ für dezimal.

i) Füllen Sie für die ersten beiden Rahmen aus Teilaufgabe (g) jeweils einen Ethernet- und einen IP-Header sowie die passende Payload aus. Beschriften Sie die gestrichelte Box neben dem jeweiligen Header/Paket mit der jeweiligen Rahmennummer.
Hinweis: Nutzen Sie den Cheatsheet zum bestimmen der Werte (z. B. Next Header). Sollte ein Wert nicht eindeutig bestimmt sein, treffen Sie eine sinnvolle Wahl.

j) Füllen Sie pro Pfadabschnitt (z. B. zwischen R1 und R2) für das jeweils erste fragmentierte Paket jeweils einen Ethernet- und einen IP-Header aus. Beschriften Sie die gestrichelte Box neben dem jeweiligen Header/Paket mit der jeweiligen Rahmennummer.
Hinweis: Nutzen Sie den Cheatsheet zum bestimmen der Werte (z. B. Next Header). Sollte ein Wert nicht eindeutig bestimmt sein, treffen Sie eine sinnvolle Wahl.

Aufgabe 2 Distanz-Vektor-Routing

Gegeben sei die in Abbildung 2 dargestellte Topologie mit den vier Routern A bis D. Die Linkkosten sind jeweils an den Kanten angegeben. Wir notieren die Routingtabellen in Kurzform als Vektor $[(x_A, y_A), \ldots, (x_D, y_d)]$. Die Tupel (x, y) geben dabei die Kosten sowie den Next-Hop zum Ziel an.

Zu Beginn seien die Routingtabellen noch leer, d. h. die Router kennen noch nicht einmal ihre direkten Nachbarn. Dies wird durch die Schreibweise `(/,/)` angedeutet. Sich selbst erreichen die Router natürlich mit Kosten 0.

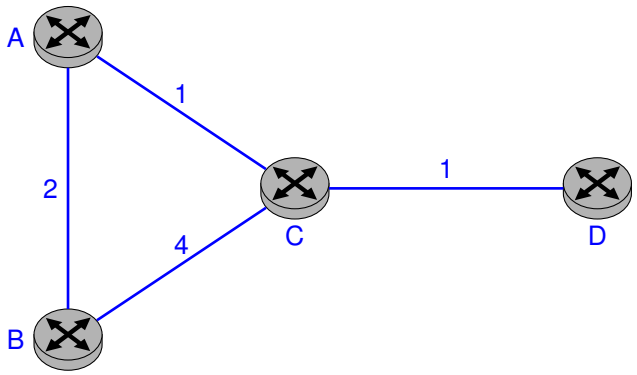


Abbildung 2: Netztopologie (Aufgabe 3)

Die Router beginnen nun damit, in periodischen Zeitabständen ihre Distanz-Vektoren mit ihren direkten Nachbarn auszutauschen. Dabei schickt beispielsweise Router B ein Update an Router C, welches lediglich die Distanz zum jeweiligen Ziel enthält (nicht aber den Next-Hop). Wenn nun Router A ein solches Update von B erhält und darin eine Route zu D finden würde, so wüsste A, dass er D über B erreicht. Die Kosten zu D entsprechen dann den Kosten zu B zuzüglich der Kosten, mit denen B das Ziel erreichen kann.

Im Folgenden wollen wir dieses Verhalten untersuchen. Da das Ergebnis allerdings davon abhängt, in welcher Reihenfolge Updates ausgetauscht werden, treffen wir die idealisierte Annahme, dass alle Router exakt zeitgleich ihre Updates verschicken.

a)* Geben Sie gemäß obiger Tabelle die Routingtabellen aller vier Router in den folgenden Schritten an. Brechen Sie ab, sobald ein konvergenter Zustand erreicht ist.

Schritt	Router A	Router B	Router C	Router D
0	`[ (0,A) (/,/) (/,/) (/,/) ]`	`[ (/,/) (0,B) (/,/) (/,/) ]`	`[ (/,/) (/,/) (0,C) (/,/) ]`	`[ (/,/) (/,/) (/,/) (0,D) ]`
1	`[ (0,A) (2,B) (1,C) (/,/) ]`	`[ (2,A) (0,B) (4,C) (/,/) ]`	`[ (1,A) (4,B) (0,C) (1,D) ]`	`[ (/,/) (/,/) (1,C) (0,D) ]`
2	`[ (0,A) (2,B) (1,C) (2,C) ]`	`[ (2,A) (0,B) (3,A) (5,C) ]`	`[ (1,A) (3,A) (0,C) (1,D) ]`	`[ (2,C) (5,C) (1,C) (0,D) ]`
3	`[ (0,A) (2,B) (1,C) (2,C) ]`	`[ (2,A) (0,B) (3,A) (4,A) ]`	`[ (1,A) (3,A) (0,C) (1,D) ]`	`[ (2,C) (4,C) (1,C) (0,D) ]`

b) Welcher (Graph-)Algorithmus findet hier Verwendung?

Es kommt eine verteilte (dezentrale) Implementierung des Algorithmus von Bellman-Ford zum Einsatz.

Nun fällt die Verbindung zwischen den Knoten C und D aus. Die Knoten C und D bemerken dies und setzen die entsprechenden Pfadkosten auf unendlich.

c) Was passiert in den folgenden Schritten, in denen die aktiven Knoten weiter ihre Distanzvektoren austauschen? Geben Sie nach jedem Schritt die Distanztabellen an, bis das weitere Ergebnis klar ist.

Schritt	Router A	Router B	Router C	Router D
4	[(0,A) (2,B) (1,C) (2,C)]	[(2,A) (0,B) (3,A) (4,A)]	[(1,A) (3,A) (0,C) (/,/)]	[(/,/) (/,/) (/,/) (0,D)]
5	[(0,A) (2,B) (1,C) (6,B)]	[(2,A) (0,B) (3,A) (4,A)]	[(1,A) (3,A) (0,C) (3,A)]	[(/,/) (/,/) (/,/) (0,D)]
6	[(0,A) (2,B) (1,C) (4,C)]	[(2,A) (0,B) (3,A) (7,C)]	[(1,A) (3,A) (0,C) (7,A)]	[(/,/) (/,/) (/,/) (0,D)]
7	[(0,A) (2,B) (1,C) (8,C)]	[(2,A) (0,B) (3,A) (6,A)]	[(1,A) (3,A) (0,C) (5,A)]	[(/,/) (/,/) (/,/) (0,D)]
8	[(0,A) (2,B) (1,C) (6,C)]	[(2,A) (0,B) (3,A) (9,C)]	[(1,A) (3,A) (0,C) (9,A)]	[(/,/) (/,/) (/,/) (0,D)]
9	[(0,A) (2,B) (1,C) (10,C)]	[(2,A) (0,B) (3,A) (8,A)]	[(1,A) (3,A) (0,C) (7,A)]	[(/,/) (/,/) (/,/) (0,D)]
10	[(0,A) (2,B) (1,C) (8,C)]	[(2,A) (0,B) (3,A) (11,C)]	[(1,A) (3,A) (0,C) (11,A)]	[(/,/) (/,/) (/,/) (0,D)]
⋮	⋮	⋮	⋮	⋮

d)* In der Vorlesung wurden **Split Horizon**, **Triggered Updates** und **Path Vector** als mögliche Gegenmaßnahmen für das Count-to-Infinity-Problem genannt. Erläutern Sie in der Gruppe die Funktionsweise dieser Verfahren.

- **Split Horizon**
  „Bewerbe eine Route nicht über das Interface, über das die Route ursprünglich gelernt wurde.“ Im konkreten Fall würden also Router A und B keine Route zu D an C schicken. Durch die ringförmige Topologie wird das Problem dadurch jedoch noch nicht behoben.
- **Triggered Update**
  Anstelle Routing-Updates nur in periodischen Zeitabständen zu versenden (bei RIP standardmäßig 30 s), werden Updates sofort geschickt, wenn Topologieänderungen erkannt werden. Dies löst das Count-to-Infinity-Problem zwar nicht, beschleunigt den Vorgang aber. Das Problem besteht darin, dass kurzzeitig viel Datenverkehr durch Routingupdates verursacht wird. Triggered Updates werden von den meisten Routingprotokollen unterstützt (RIP erst ab Version 2).
- **Path Vector**
  Es wäre möglich, dass sich jeder Router bei einem Update merkt, woher das Update kam und diese Information bei Routing-Updates mit einschließt. Auf diese Weise könnte jeder Router den vollständigen Pfad nachvollziehen, den ein Paket zum jeweiligen Ziel nehmen wird. Entdeckt ein Router sich selbst in diesem Pfad, so weiß er, dass eine Schleife vorhanden ist. Er kann das Update in diesem Fall verwerfen. Dieses Verfahren wird von BGP eingesetzt.

Aufgabe 3 IPv6 & Supernetting (Hausaufgabe)

Der GRNVS AG wurden nun die IPv6 Adressebereiche `2001:0db8:0001:000d:0000:0000:0000:0000/64` (*NET1*) und `2001:0db8:0001:000e:0000:0000:0000:0000/64` (*NET2*) zugeteilt.

a)* Geben Sie die in *NET1* enthalten IPv6 Adresse `2001:0db8:0001:000d:0000:00f0:0000:0000` in kompakter Schreibweise an.

- führende Nullen werden weg gelassen: `2001:db8:1:d:0:f0:0:0`
- der größte konsekutive Block von mindesten 2 „Nuller“-Blöcken kann durch `::` abgekürzt werden: `2001:db8:1:d:0:f0::`

b)* Wieviele Adressen enthält jedes Präfix?

2^{128-64} = 18 446 744 073 709 551 616 = 18,4 Trillionen

c) Wie oft kann der gesamte IPv4 Adressbereich (`0.0.0.0/0`) in *NET1* abgebildet werden?

$2^{(128-64)-32} = 2^{32}$ = 4 294 967 296 = 4,2 Milliarden.

d)* Welche Bedingungen müssen erfüllt sein, damit 2 Subnetze aggregiert werden können?

- gleich groß, d. h. selbe Präfixlänge n
- benachbart (auf die letzte Adresse im ersten Netz muss direkt das nächste Netz folgen)
- Es muss eine valide Präfixmaske mit Länge $n - 1$ existieren, d. h. die beiden Netze dürfen sich nur genau im letzten Bit ihres Präfix unterscheiden.

e)* Können die beiden Subnetze *NET1* und *NET2* in ein `/63` Subnetz aggregiert werden?

Obwohl die Netze von gleicher Größe sind und nebeneinander liegen, können sie nicht aggregiert werden, da sie nicht im gleichen `/63` Präfix liegen. Für die Bits 61 bis 64: $\texttt{d}_{16} = 11\underline{0}1_2$, $\texttt{e}_{16} = 11\underline{1}0_2$.

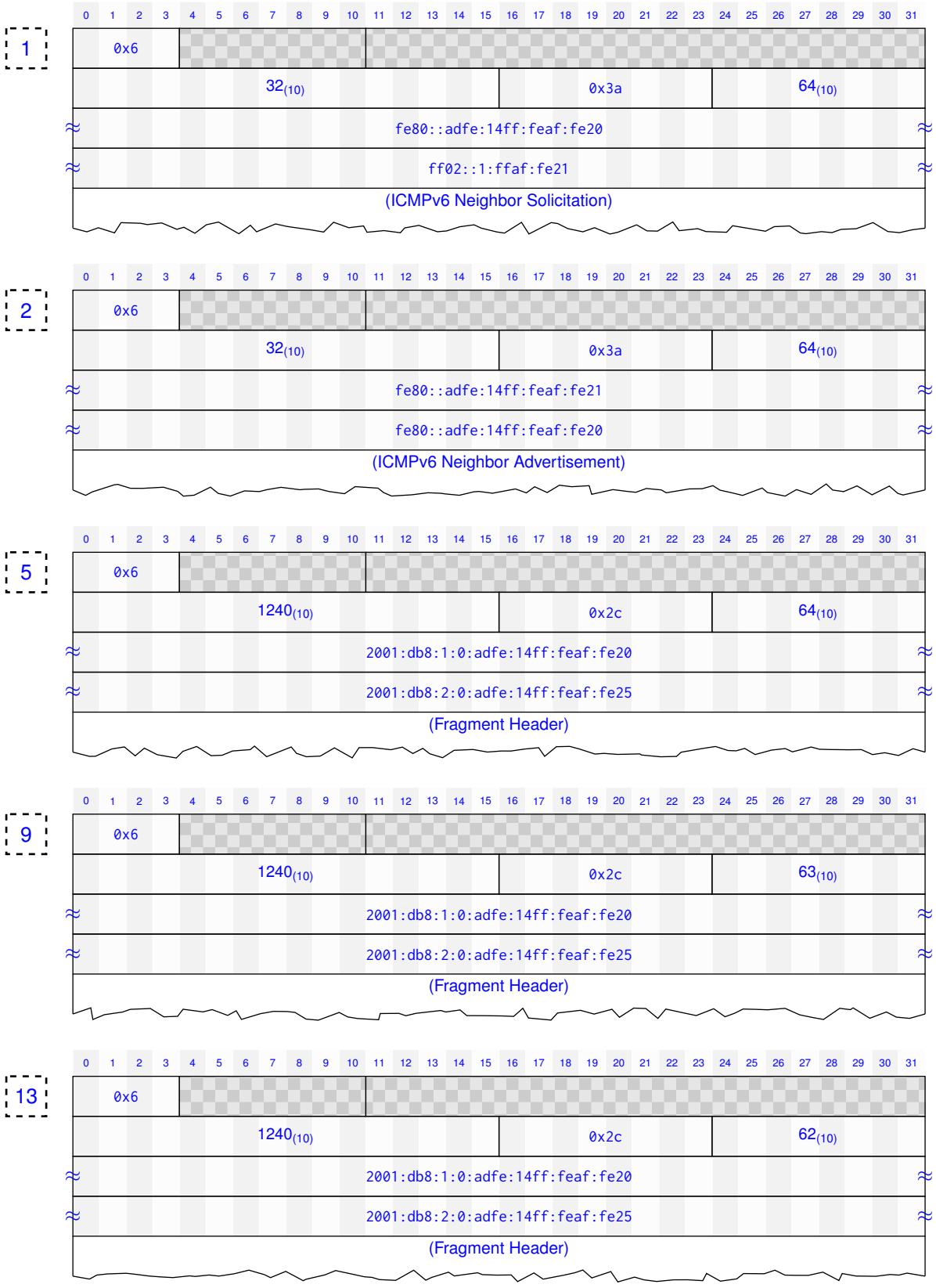
`2001:db8:1:c::/62` würde die beiden Netze umfassen, aber zusätzlich auch `2001:db8:1:c::/64` und `2001:db8:1:f::/64` enthalten.

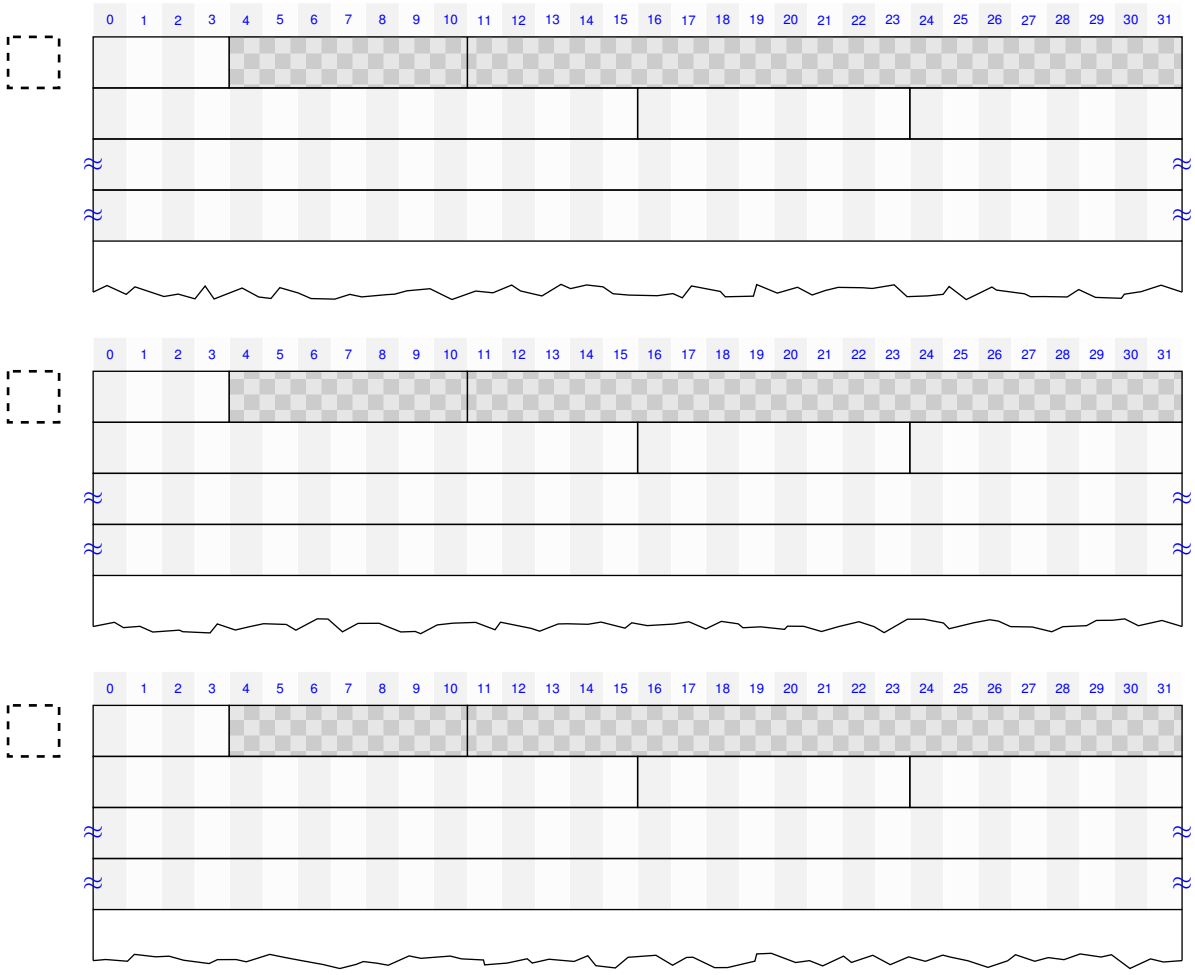
Vordrucke für Protokoll-Header:

Ethernet-Frames

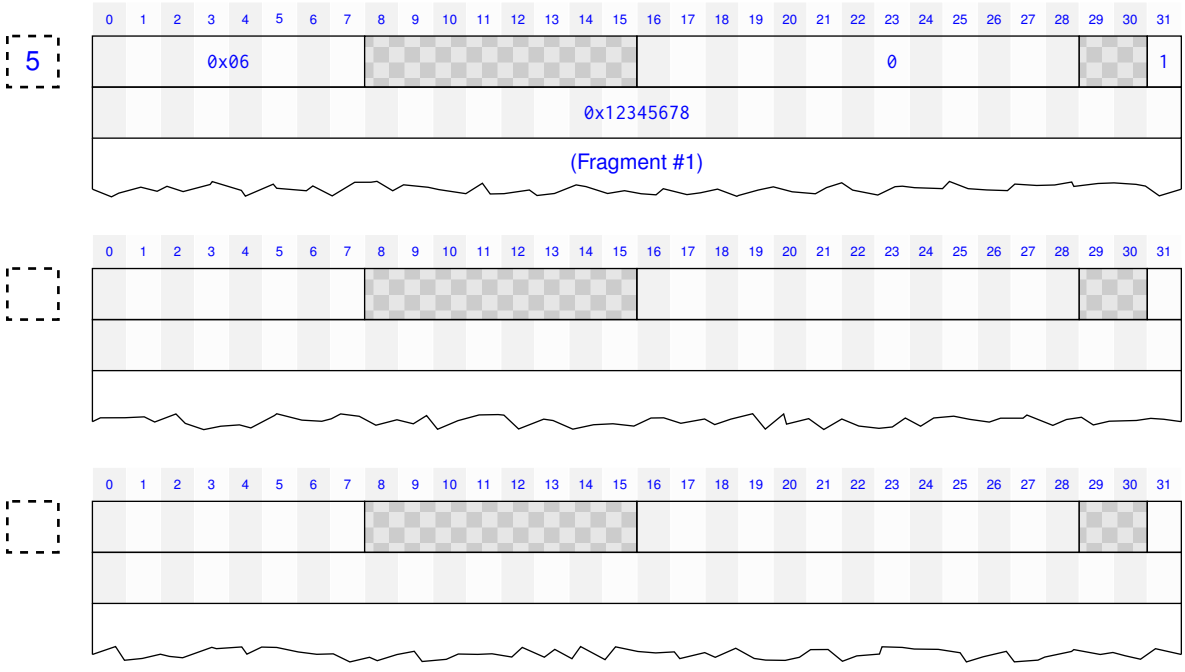
1	33:33:ff:af:fe:21	af:fe:14:af:fe:20	0x86dd	Payload	FCS
2	af:fe:14:af:fe:20	af:fe:14:af:fe:21	0x86dd	Payload	FCS
5	af:fe:14:af:fe:21	af:fe:14:af:fe:20	0x86dd	Payload	FCS
9	af:fe:14:af:fe:23	af:fe:14:af:fe:22	0x86dd	Payload	FCS
13	af:fe:14:af:fe:25	af:fe:14:af:fe:24	0x86dd	Payload	FCS
				Payload	FCS
				Payload	FCS
				Payload	FCS
				Payload	FCS
				Payload	FCS
				Payload	FCS
				Payload	FCS
				Payload	FCS
				Payload	FCS

IPv6 Header

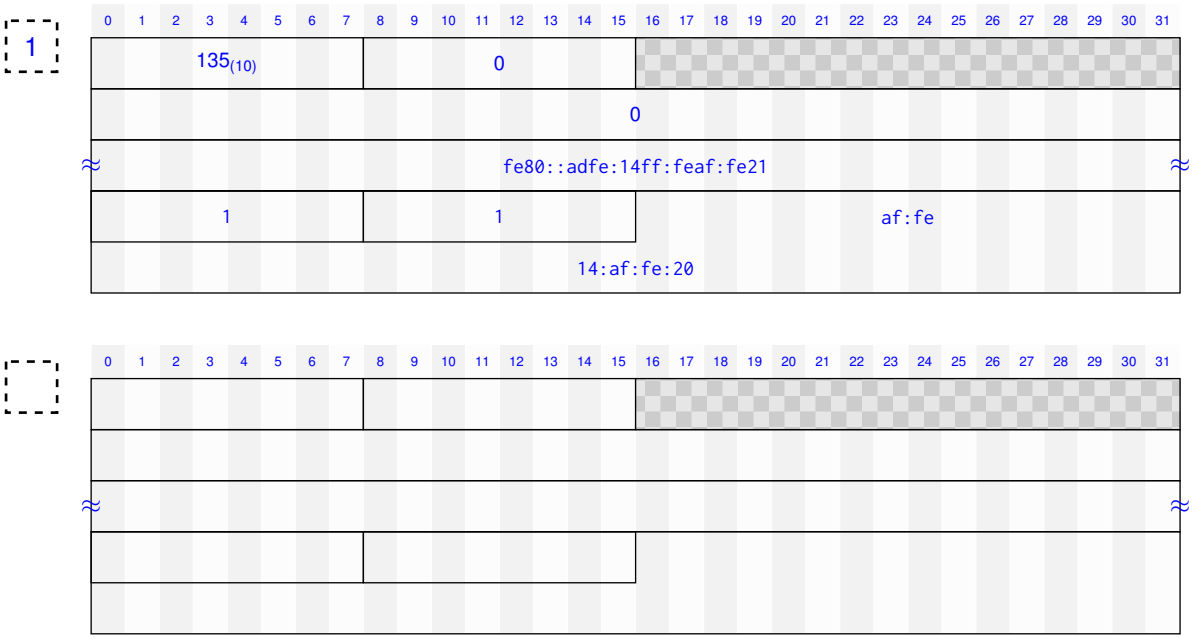

1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0x6
32(10)
0x3a
64(10)
fe80::adfe:14ff:feaf:fe20
ff02::1:ffaf:fe21
(ICMPv6 Neighbor Solicitation)
2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0x6
32(10)
0x3a
64(10)
fe80::adfe:14ff:feaf:fe21
fe80::adfe:14ff:feaf:fe20
(ICMPv6 Neighbor Advertisement)
5
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0x6
1240(10)
0x2c
64(10)
2001:db8:1:0:adfe:14ff:feaf:fe20
2001:db8:2:0:adfe:14ff:feaf:fe25
(Fragment Header)
9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0x6
1240(10)
0x2c
63(10)
2001:db8:1:0:adfe:14ff:feaf:fe20
2001:db8:2:0:adfe:14ff:feaf:fe25
(Fragment Header)
13
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0x6
1240(10)
0x2c
62(10)
2001:db8:1:0:adfe:14ff:feaf:fe20
2001:db8:2:0:adfe:14ff:feaf:fe25
(Fragment Header)



IPv6 Fragment Header



ICMPv6 Neighbor Solicitation



ICMPv6 Neighbor Advertisement