Schichtname	Einheit	Regelt
Anwendungsschicht (Anwendungsschicht)	Nachrichten	Funktionen für Anwendungen
Darstellungsschicht	Nachrichten	Zeichensätze, Zahlendarstellung
Kommunikations-steuerungsschicht	Nachrichten	Login&Logout
Transportschicht (Transportschicht)	Nachrichten	Übertragungsgarantie, Reihenfolge, Fluss/Überlastkontrolle, Prozessadres-sierung, beliebige Nachrichtengröße
Vermittlungsschicht (Internetschicht)	Pakete	Routing
Sicherungsschicht (Netzwerkschicht)	Frames	Fehlererkennung/-korrektur, Medienzugriff, Framing, Zuverlässige Zustellung
Bitübertragungsschicht	Bits	Frequenzen, Kabel, Kodierung

LLC - Link Logical Control	siehe Schicht 2 OSI
MAC - Media Access Control	Medienzugriff)
PHY - Physical Layer	Bitübertragung

TCP/IP-Prüfsumme:

1. Aufsummieren aller 16-Bit-Blöcken in Einerkomplementdarstellung
   Einerrücklauf(Wert / 65536)
   Tatsächlicher Wert: mod 65536; 65535 = 0;
2. invertieren aller Prüfsummenbits (Erg = 65535−Wert)

=> auf Empfängerseite muss 00…0 herauskommen

CRC (Cyclic Redundancy Check)

Bitkette ⇔ Polynom,
z.B. 1011 ⇔ x^3+x^1+1
+ = − =Xor

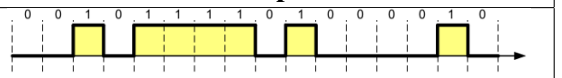
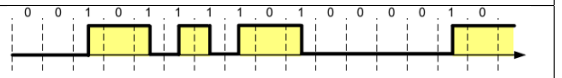
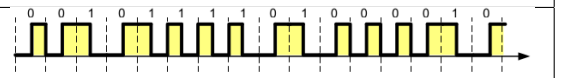
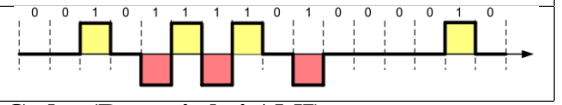
I(x) = Nutzdaten Informationspolynom
G(x) = vorgegeben Generatorpolynom
k = grad von G(x)
C(x) = Frame Codepolynom

Berechnen: R(x) = I(x) · x^k mod G(x);

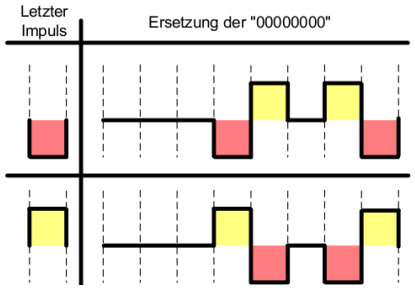
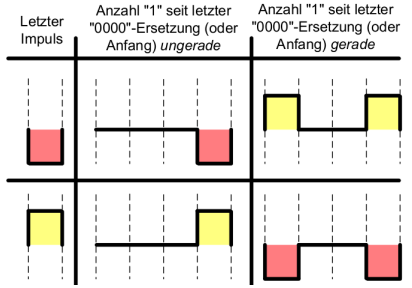
C(x) = I(x) ·x^k + R(x);
=> Empfänger prüft, ob C(x)/G(x) = 0;

I= 100010, G(x)=$x^3+x^1+1 \widehat{=} 1011$
100010000:1011=101100
1011
001110
1011
01010
1011
000100 <= R(x)
=> C(x) = 100010100

Kodierungen

	0	1	Beispiel
NRZ	0	1	
NRZI	halten	ändern	
Manchester	0 → 1	1 → 0	
AMI	0	±1	

Code-Ersetzungen bei Ternären Codes (Rest wie bei AMI):

Code:	B8ZS	HDB3
zu Ersetzen:	8 Nullen	4 Nullen
Regel:		

4B/5B mit maximal 3 Nullen in folge

4-Bit-Daten	5-Bit-Code
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
0111	01111
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

$A = \log_2(B) \Leftrightarrow B = 2^A$

Übertragung

$v_{max} = 2B;$ $D_{max} = 2B\log_2(L)$ $D_{max} = B\log_2(1+\frac{S}{N})$ $\frac{S}{N} = 10^{SNR[db]/10}$

v_{max} = Schrittgeschwindigkeit; D_{max} = Datenrate; B = Bandbreite;
L = Signalstufen; SNR = Signal-Rausch-Abstand [db]

Link-State-Routing

Dijkstra

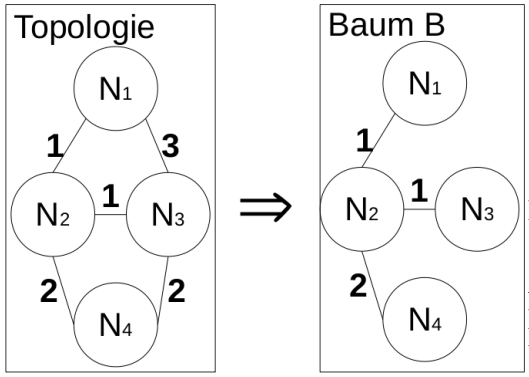
Menge M:=schon abgearbeitete Knoten;
Baum B:= gesuchter Quellbaum
di:= berechneter Abstand zu Ni

initialisiere:
Startknoten in M und B einfügen,
di=Wert der Kante Ni zu Startknoten, sonst ∞
Bis alle Knoten in M sind:

1. suche das kleinste di,
2. füge diesen Knoten Ni zu M hinzu,
3. füge die kürzeste Kante von B zu Ni in B ein.
4. passe die d der Nachbarn von Ni an, wenn di + Kante von Ni kürzer ist

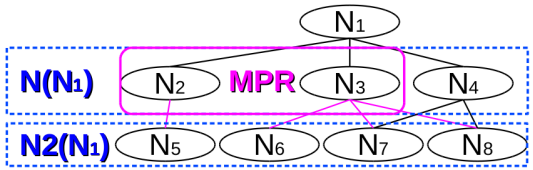
Bsp: Startknoten: N_1

Runde	M	d_2	d_3	d_4
1	N_1	1	3	∞
2	N_1, N_2		2	3
3	N_1, N_2, N_3			3



OLSR - MPR

Fluten nur an ausgewählte Nachbarn.
N(K):= Nachbarn von K
N2(K):= Nachbarsnachbarn von K
MPR(K):= Weiterflutende knoten in N(K)



1. MPR = Knoten in N(K), die einzige Verbindung eines Knotens in N2(K) sind (Bsp: N_2)
2. Solange noch nicht alle Knoten in N2 erreicht werden:
   Suche den Knoten aus N(K), der die meisten fehlenden Knoten in N2(K) erreicht und füge ihn zu MPR hinzu (N_3)
   (bei mehreren Kandidaten denjenigen mit meisten Nachbarn)
3. Entferne Knoten, wenn danach immer noch alle Knoten in N2(K) erreicht werden

Link-Reversal-Routing

Weg zu einem Ziel: Directed Acyclic Graph (DAG)

1. Full Reversal: hat ein Knoten nur eingehende Kanten: drehe alle um
2. Partial Reversal (listenbasiert): Knoten führen Liste über von Nachbarn gedrehte Kanten; sind alle Kanten drin lösche die Liste
   gibt es nur eingehende Kanten drehe alle Kanten um, die nicht in der Liste stehen;

DBF:

Jeder Knoten hat eine Tabelle mit Ziel, Hop zum Ziel, Metrik(Kosten);
Initialisierung: Loopback mit Metrik 0 eintragen;
Verbindung zu den Nachbarn mit der gemessenen Metrik eintragen
Alle Anderen Verbindungen: Hop = ? Metrik= ∞
Abgleich mit Nachbartabelle: In jeder Zeile prüfen:

- Berechne Eintrag der Nachbartabelle + Verbindungsmetrik
- Ist der Nachbar der Hop der Zeile: Übernimm veränderten Wert
- Ist die Route über den Nachbarn besser → Nachbarn als Hop mit Metrik eintragen

Count-To-Infinity-Problem: nach Verbindungsabbruch evtl. Feedback-Schleife

- definiere kleinen Wert als Unendlich
- Split Horizon: Zeilen werden nicht dem eingetragenen Hop weitergereicht
- Versionsnummern der Einträge (DSDV)

DSDV:

DBF Tabelle um Spalte für Sequenznummer erweitern (mit -1 initialisiert)
Abgleich mit Nachbartabelle: höhere Sequenznummer gewinnt (Gleiche: siehe DBF)
Nach Abgleich mit allen Nachbarn : *Eigene* Sequenznummer +2 (Metrik=0);
Verbindungsabbruch zu Nachbarn Nx: in alle Zeilen mit Hop = Nx:
Metrik = ∞ Hop = ? Sequenznummer++

IPv6

Adressen: 8 4er-Blöcke(Hex) z.B. 47cd::22:1234:a456:12
64 Bit Präfix(Netzwerkanteil) 64 Bit Interface Identifier

Adressen	Funktion
::1	Local Loopback
fe80::/10	Link Local Unicast (Autokonfiguration)
fc00::/7	Unique Local Unicast (Private Adressen)
ff00::/8	Multicast (Broadcast mit verschiedenen Reichweiten)
weitere:	Anycast (Route an irgendeinen Host einer speziellen Gruppe)

Autoconfiguration:

1. Berechne Identifier (aus MAC oder zufällig)
2. => frage bei Router mit fe801::/64 + Identifyer (Link Local Unicast Adresse)
3. Router teilt mögliche Präfixe mit
4. Host prüft ob Adresse schon vorhanden (Double Address Detection)

DHCP:

1. Client: DHCPDISCOVER (per Broadcast)
2. Server: DHCPOFFER
3. Client: DHCPREQUEST
4. Server: DHCPACK (Client hat nun Adresse geleast)

IPv4 Hostanteil 1...1= Broadcast 0...0= Netzadresse

Klasse	Von	bis	
A	0.0.0.0	127.255.255.255	16.777.214 Hosts
B	128.0.0.0	191.255.255.255	65.534 Hosts
C	192.0.0.0	223.255.255.255	254 Hosts
D	224.0.0.0	239.255.255.255	Multicast
E	240.0.0.0	255.255.255.255	Reserviert
	127.0.0.0	127.255.255.255	Loopback
A	10.0.0.0	10.255.255.255	privat
B	172.16.0.1	172.31.255.255	privat
C	192.168.0.1	192.168.255.255	privat

Oktett	Bits
255	1111 1111
254	1111 1110
252	1111 1100
248	1111 1000
240	1111 0000
224	1110 0000
192	1100 0000
128	1000 0000

Fragmentieren

Ident(16Bit) ID eines Pakets, Offset(13Bit): Position des Fragments **/ 8**

Fragmentierungsbit: 1=verboten, More: 1=weitere , 0=letztes Fragment

Vorgehen

Größtmögliche Pakete herausschneiden, evtl ein kleinerer Rest im letzten Paket.

Beachten: Fragmentieren auf Vielfache von 8 z.B. 512,520... u.U. IP-Header 20Byte

Router "defragmentieren"**nicht**;

TCP

Nagle: Sende wenn MTU erreicht oder alle bisher gesendeten Segmente bestätigt

Sliding Window

Empfänger antwortet mit: (**Acknowledge**,**Advertised Window**)

Acknowledge: nextByteExpected -1

Advertised Window= maxRecieveBuffer - (lastByteRecieved - lastByteRead)

Sender kann Senden:

Effective Window = **Advertised Window**- (LastByteSent - LastByteAcked)

Bei Effective Window = 0: Sender sendet periodisch 1 Byte (bis Effective Window steigt)

Timeouts

RTT_{Last}: gemessener RTT

$RTT = a \cdot RTT + (1-a) \cdot RTT_{Last}$

- Timeout = 2* RTT
- Deviation = Deviation · a + $|RTT_{Last} - RTT| \cdot (1-a)$

  Timeout = RTT + 4·Deviation

Überlastkontrolle

Additive-Increase/Multiplicative-Decrease:

beginne mit Congestionwindow = 1; bei ACK ++ ; bei Timeout /2;

Slow Start:

bei Start, nachdem Datenrate auf 0 gefallen: erhöhe CongestionWindow durch *2

DNS

Namensraum:bis 255 Zeichen aus verketteten Labels;

Labels: 1-63 Zeichen aus a-z,0-9 und -; Sonderzeichen mit Punycode

Records: Zuordnugstabellen Typen: A (IPv4) AAAA (IPv6) PTR (Reverse Lookup) ...

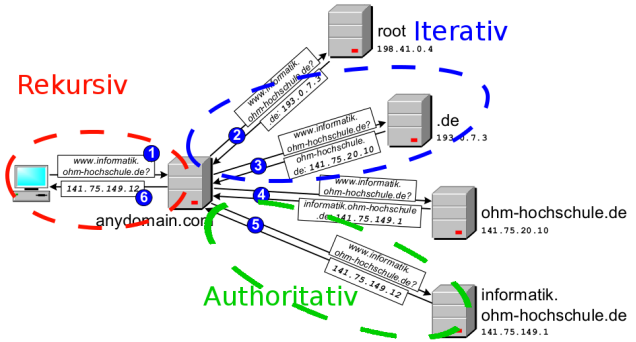
organisatorische Struktur:

aufteilung in Zonen (Domain und darunterliegende Hosts);

jede Zone hat eigene Nameserver, Verantworlichen und Namenskonventionen

Nameserver kennen: direkt untergeordnete Hosts und Nameserver; Root-Server

Auflösungsmechanismus:



Reverse-Lookup:

Umwandeln der IP-Adresse in spezielle Domain, anfrage in Record PTR

TLDs: in-addr.arpa, ip6.arpa

z.B. 141.75.201.12 → 12.201.75.141.in-addr.arpa

DNSsec

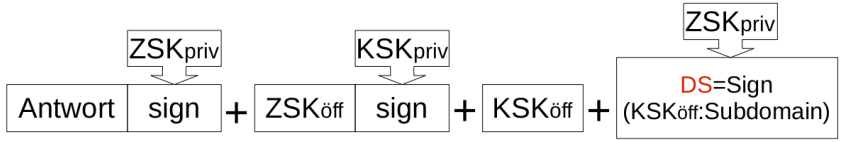
Antwort wird signiert, Schlüssel durch nächsthöheren DNS bestätigt

2 Schlüsselpaare (Erleichtert Schlüsseltausch, kurze Schlüssel möglich):

ZSK (Zone Signing Key): Signieren der Antworten (kurz, Änderung Tage-Wochen), Tausch hat nur lokale Auswirkungen

KSK (Key Signing Key): Signieren der Schlüssel (lang, Änderung Jahre), Bei Tausch absprache mit übergeordneten Domain

DS (Delegation Signer): Mit ZSK signierter öffentlicher KSK der **Subdomain**



URL

Protokoll :// Benutzer : Passwort @ Domain : Port / Pfad ? Anfrage # Abschnitt

Chord

Fingertabelle für einen Host A:

k	start	end	Node
1	$A_{end}+1$	nächster Start-1	Host mit start in Hashtable
2	$\rightsquigarrow$ +1	"	suche start in Hashtable
3	$\rightsquigarrow$ +2	"	
4	$\rightsquigarrow$ +4	"	
k	$\cdots$	$A_{end}-1$	

z.B. Fingertabelle für A mit DHT (Hashlänge = 5 => rechnen mod 32)

Knoten	Start	Ende
B	5	14
A	15	20
C	21	4

$\Longrightarrow$

k	start	end	Node
1	21	21	C
2	22	23	C
3	24	27	C
4	28	3	C
5	4	19	B

Join: Vorraussetzung: Bekannter Rechner n';

1. zuständigkeitsbereich ausrechnen (Hashen der Adresse) und zuständigen Host suchen
2. Fingertabelle anlegen (lookup durch n')
3. Anpassen aller Fingertabellen, die auf den neuen Rechner zeigen (ist abhängig von der größe des übernommenen Hashbereichs)
4. Übernehmen der relevanten Einträge vom bisher zuständigen Host

CAN

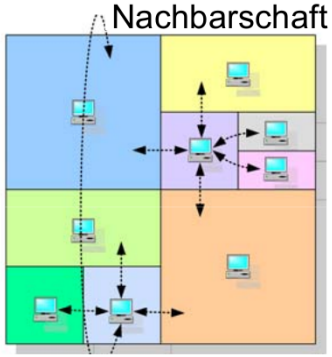
Abbilden der Hashes auf d Dimensionen

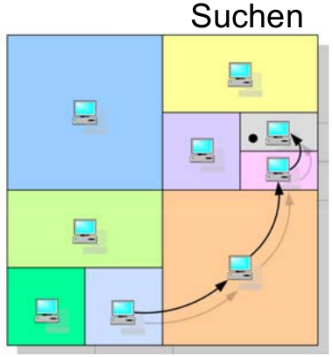
Knoten verwaltet Hashes in einem d-Dimensionalen Quader und seine direkten Nachbarn.

Suche: Anfrage an den dem Ziel nähesten Nachbarn (Distanz vom Mittelpunkt)

Join: wähle zufälligen Punkt, halbiere Bereich (wenn möglich zu Quadraten)

Leave: Übergabe des Quaders an einen Nachbarn





Sicherheit

Sicherheitsziele: Vertraulichkeit, Authentizität, Integrität, Anonymität, ...

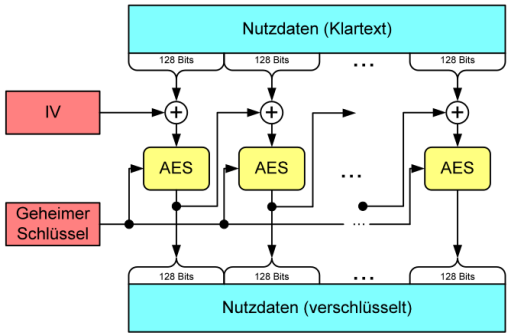
Betriebsmodi:

Codebook Mode: Jeder Block einzeln verschlüsselt

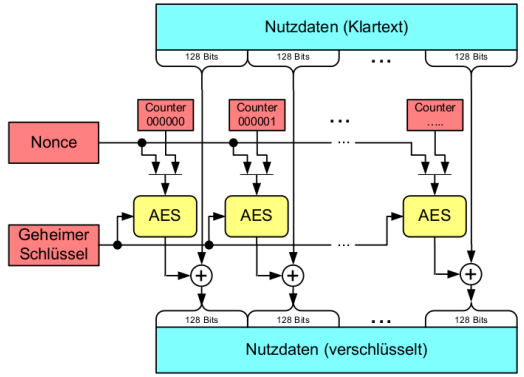
CBC: Änderung der Nachricht => keine Mustererkennung

Counter mode: Simuliert Stromchiffre mit Pseudozufallsgenerator (aus Nonce)

Cypher Block Chaining



Counter Mode



Rainbow Table:

Erstellen einer Zeile: beginne bei Startwert, wiederhole Hash- und Reduktionsfunktion bis zum k-ten Hashwert (k=Kettenlänge), speichere Start und Endwert (letzter Hash);

suchen: wiederhole Hash und Reduktion bis ein bekannter Endwert, dann gehe vom Startwert mit Hash und Reduktion bis gesuchter Wert erreicht wird

Salzen: erzeugen und Speichern von Zufallszahl, hashen von Passwort und Salz

http

Persistente Verbindung: eine TCP-Verbindung für mehrere HTTP-REQUESTS

Pipelining: Client stellt mehrere Anfragen ohne warten auf Antwort

Long Polling: Server antwortet erst nach Ereignis oder Timeout (asynchronität)

zustandslos => keine Sitzungen => SitzungsID in Cookie oder URL

Webservices

SOAP

Definition von Operationen (mit Parametern, Rückgabe ...)

Schnittstellenbeschreibung mit WSDL => Prüfen auf Aktuelle Version, Stuberzeugung möglich

REST

Zugriff auf Ressourcen über HTTP-Kommandos mit URIs

z.B. GET http://abookstore/cart/12345