Digitale Kommunikation und Internetdienste 1

Wintersemester 2004/2005 – Teil 11

Belegnummer Vorlesung: 39 30 02

Übungen: 39 30 05

Jan E. Hennig

AG Rechnernetze und Verteilte Systeme (RVS)
Technische Fakultät
Universität Bielefeld

jhennig@rvs.uni-bielefeld.de

basierend auf den Arbeiten von Michael Blume, Heiko Holtkamp, Marcel Holtmann und I Made Wiryana

Organisatorisches

- Termine für Übungen verlegt:
  - Übung 5 statt am 23./24.12.2004 nun am 06./07.01.2005
  - Übung 6 statt am 06./07.01.2005 nun am 13./14.01.2005

- Klausur am 20.01.2005:
  - es wird eine zusätzliche Frage auf der Klausur angeboten (evtl. auch zwei)
  - wenn passend beantwortet zeigen diese, daß man zusätzliche 15 Stunden über die Veranstaltung hinaus dem Thema gewidmet hat
  - diese 15h bedeuten 0,5 LP und damit könnte man als MGSler 5 LP statt 4,5 LP für diese Veranstaltung bekommen
  - für MIGler gilt dieses Angebot nicht, da sie fixierte 4 LP bekommen und weniger Leistung beweisen müssen, um die Klausur zu bestehen

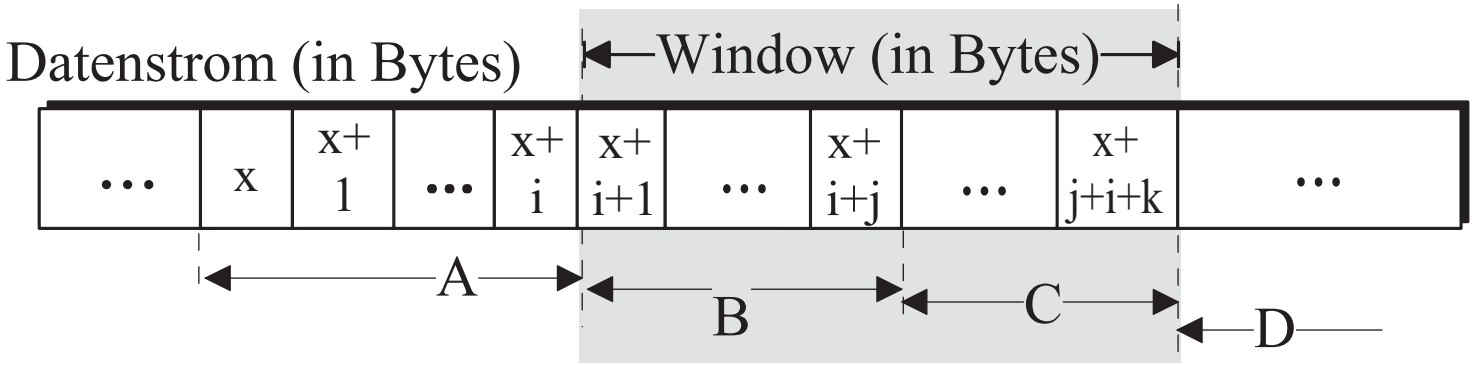
Übersicht

- TCP (Teil 2)
- TCP-Erweiterungen und -Verbesserungen
- T/TCP
- Domain Name Service (DNS)
- Dynamic DNS
- IP-over-DNS

Erinnerung (1)

- TCP sorgt für zuverlässigen Transfer von Datenströmen
- es benutzt einen modifizierten Sliding-Window-Algorithmus
- darüber wird die Zuverlässigkeit durch Fehlerkontrolle hergestellt (ARQ)
- außerdem dient er zur Datenflußkontrolle
- im Betrieb kann TCP verschiedene Betriebsparameter automatisch anpassen

Erinnerung (2)



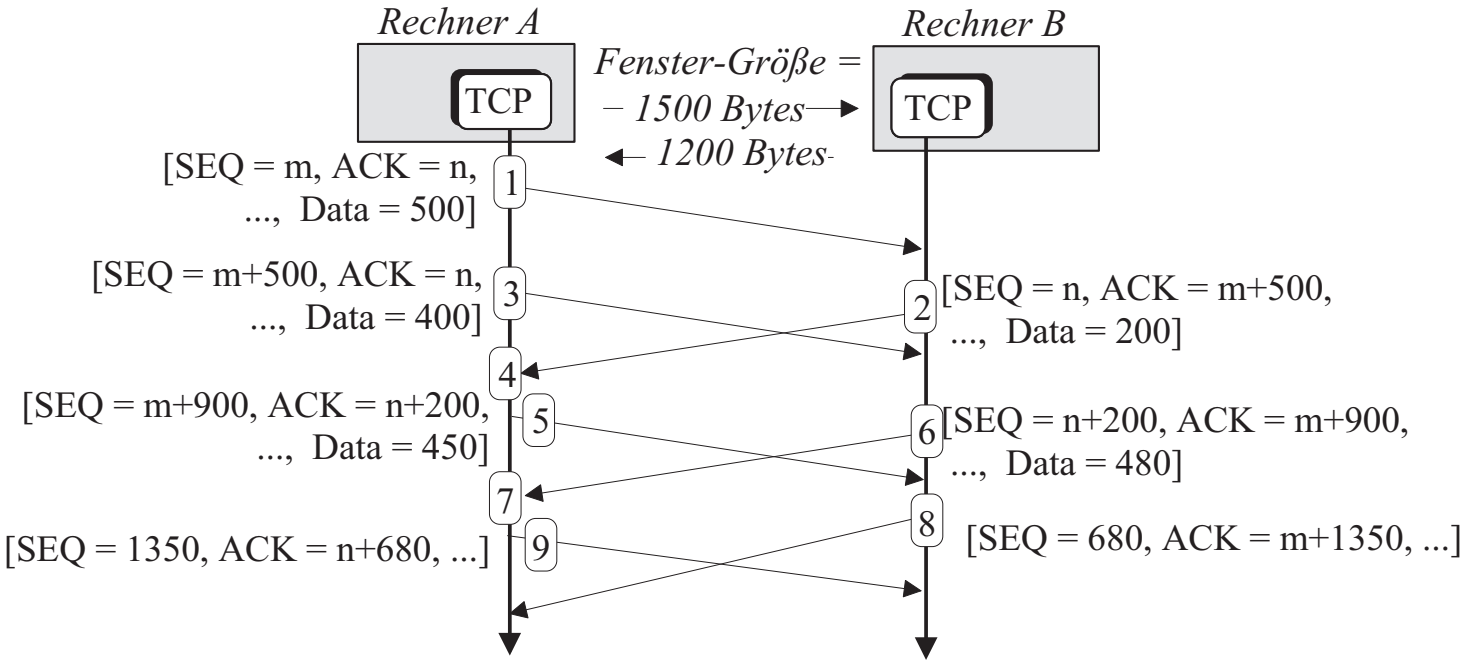
x = Initial Sequence Number (ISN)

Szenarien

Drei Szenarien:

1. fehlerfreie Datenübermittlung
2. Sendeblockade bei der Datenübermittlung
3. fehlerbehaftete Datenübermittlung

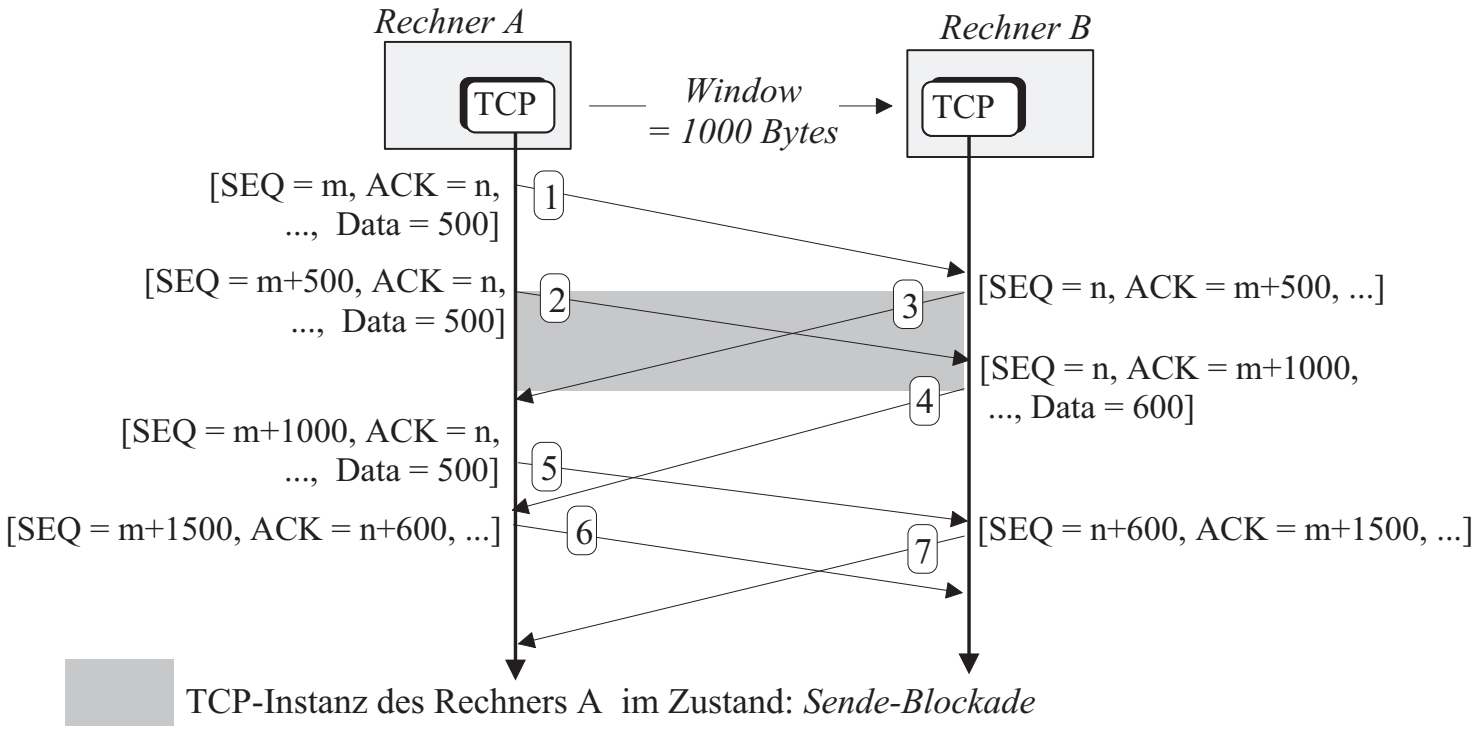
Fehlerfreie Datenübermittlung (1)



Fehlerfreie Datenübermittlung (2)

1. A→B: TCP-Segment ab Position m enthält 500 Bytes Daten
2. B→A: 200 Bytes zurück ab n, ACK für $m + 500$
3. A→B: 400 Bytes ab $m + 500$, ACK für n, Sendefenster zu B: 1500-900=600 Byte
4. Quittung erreicht A: Sendefenster zu B vergrößert auf 1500-400=1100
5. A→B: 450 Bytes ab $m + 900$, ACK für $n + 200$
6. B→A: 480 Bytes ab $n + 200$, ACK für $m + 900$
7. Quittung erreicht A: Sendefenster zu B wird entsprechend verschoben
8. B quittiert $m + 1350$
9. A quittiert $n + 680$

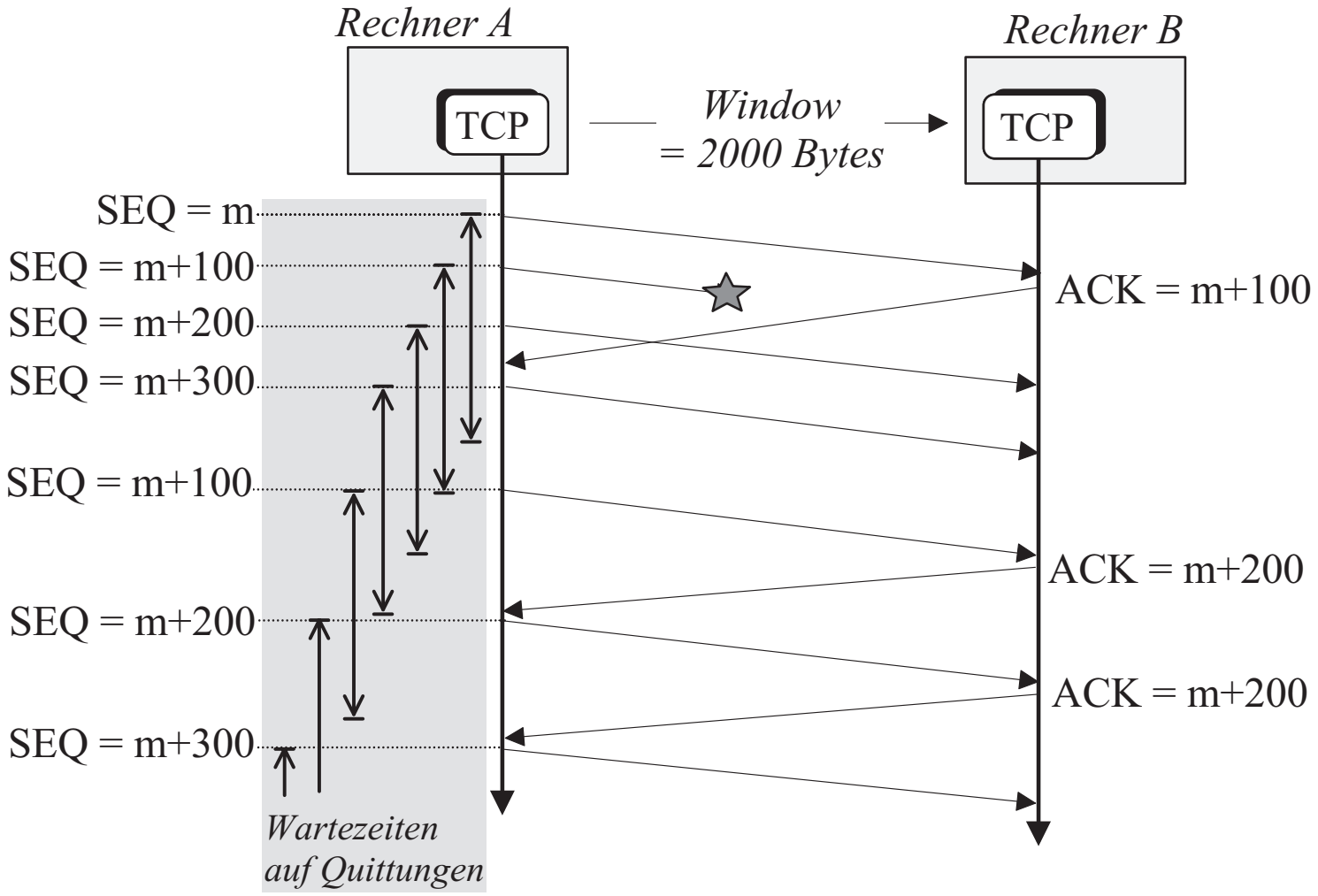
Blockierte Datenübermittlung (1)



Blockierte Datenübermittlung (2)

1. A→B: 500 Bytes ab m, ACK für n
2. A→B: 500 Bytes ab $m+500$, ACK für n, Sendeblockade, da Fenstergröße erreicht
3. Quittung erreicht A: Sendefenster verschiebt sich um 500 Byte
4. B→A: 600 Bytes ab n, ACK für $m+1000$
5. A→B: 500 Bytes ab $m+1000$, ACK für n
6. A quittiert $n+600$
7. B quittiert $m+1500$

Fehlerhafte Datenübermittlung (1)



Fehlerhafte Datenübermittlung (2)

- Empfänger kann nicht aktiv Neuübertragung auslösen (kein NAK vorgesehen)
- stattdessen muß er abwarten, bis beim Sender der Timer abläuft
- im Beispiel: Segment mit SEQ=$m + 100$ hat Empfänger nicht erreicht
- Empfänger sendet keine Bestätigungen für SEQ=$m + 200$ und SEQ=$m + 300$, da Lücke
- Bestätigung erfolgt nur, wenn keine Lücken vorhanden
- dies kann dazu führen, daß auch fehlerfrei empfangene Segmente wiederholt gesendet werden

Wartezeit

- die maximale Wartezeit ist ein wichtiger Parameter im TCP-Betrieb
- sie hängt u.a. von der Verzögerung ab (Messung der RTT)
- die Wartezeit selbst variiert im Laufe der Zeit
- daher wird sie nicht fest eingestellt, sondern dynamisch geführt
- an dieser Stelle setzen verschiedene Verfahren zur Optimierung an

Entwicklung

- TCP wurde im Laufe der Zeit weiterentwickelt
- nicht nur das Protokoll selbst, sondern auch dessen Implementierungen
- dabei wurden verschiedene Algorithmen vorgeschlagen und probiert
- diese wurden daraufhin weiter verbessert (mit der Netzentwicklung)
- Ziel: maximale Performance und Übertragungssicherheit in heutigen Netzen (siehe RFC 1122, 1323, 2001, 2018)

Nagle-Algorithmus

- Problem, daß die TCP-Instanz auf Anforderung der Anwendungsschicht sehr kleine Segmente sendet

- zur Reduzierung der Netzlast sollten die pro Verbindung von der Anwendungsschicht ankommenden Daten möglichst konkatiniert (d.h. in einem Segment zusammen) gesendet werden

- in der Folge braucht auch der Empfänger nicht jedes (kleine) Segment bestätigen

- Nagle-Algorithmus berücksichtigt 4 Faktoren:
  - TCP-Haltezeit für das Zusammenführen von Applikationsdaten in Segmente
  - verfügbarer TCP-Pufferbereich für Applikationsdaten
  - Verzögerungszeiten im unterliegenden Netz: Round Trip Time
  - Applikationstyp: besonders kritisch sind interaktive Protokolle, da jedes einzelne Zeichen (Tastatureingabe, Mausklick) für die Bildschirmanzeige „geechot“ wird

Silly Window Syndrome

- das *Silly Window Syndrome (SWS)* kennzeichnet den Zustand, wenn ein TCP-Empfänger sukzessive mit der Erhöhung des zunächst kleinen internen TCP-Puffers dies der sendenden TCP-Instanz durch ein weiteres ACK-Segment mit der neuen Windowsize umgehend mitteilt

- bei Übertragung großer Datenmengen kann es vorkommen, daß sich Sender und Empfänger hinsichtlich der Windowsize nicht mehr vernünftig abstimmen und der Sender nur noch sehr kleine Datensegmente übermittelt

- dieser Fehler ist dadurch zu vermeiden, daß der Empfänger mit der Sendung des ACK wartet, bis er hinlänglich TCP-Puffer allokieren kann

- dies ist ein zum Nagle Algorithmus komplementärer Effekt

Zero Window Probe

- eine TCP-Instanz kann der anderen durch Setzen von Windowsize=0 mitteilen, daß sie ihren TCP-Empfangspuffer auf null reduziert hat
- dies kann z.B. eine Folge davon sein, daß die TCP-Instanz die bereits anstehenden Daten nicht mehr an die Applikation weiterreichen kann
- z.B. wenn sich in einem Netzwerk-Drucker kein Papier mehr befindet
- in Anschluß daran ist es nach Ablauf des Timeouts Aufgabe des Senders, mit einer Zero Window Probe festzustellen, ob der Empfänger wieder aufnahmebereit ist
- Hintergrund hierfür ist, daß ACK-Segmente ohne Daten nicht verläßlich übertragen werden
- sollte der Empfänger hierauf nicht antworten, wird der Retransmission-Algorithmus in Gang gesetzt

TCP Keep-Alives

- TCP verzichtet i.d.R. auf ein Keep-Alive-Verfahren
- es wird aber auch nicht ausgeschlossen
- TCP-Keep-Alive-Informationen werden in ACK-Segmenten mit einem bedeutungsfreien Datenbyte oder völlig ohne Daten eingeschlossen
- sie dürfen nur versendet werden, wenn keine anderen regulären Daten ausgetauscht werden
- entscheidend ist, daß die generierte Sequenz-Nummer dem obersten Wert des Sendefensters abzüglich einem Byte entspricht
- dieser Wert liegt außerhalb des ausgehandelten Sendefensters, was den TCP-Partner veranlaßt, mit einem ACK zu antworten

Anforderungsänderungen

- das Einsatzgebiet von TCP hat sich mit der Erweiterung des Internets und den Entwicklungen der lokalen Netze stark erweitert:
- schnelle, d.h. durchsatzstarke und latenzzeitarme LANs, wie z.B. Gigabit-Ethernet bzw. geswitchte Fast-Ethernet-Netze
- ausgedehnte, große Netzstrukturen (WAN) mit unterschiedlichen Trägernetzen wie ATM oder Frame-Relay mit z.T. signifikanten Verzögerungszeiten bei der Übertragung (sog. Long Fat Networks (LFN))
- zu übertragende Datenvolumen, die im Bereich von Gigabyte liegen, d.h. den Bereich der einfach-adressierbaren TCP-Sequenznummern überschreiten

Abschätzung der RTT

- Abschätzung sollte möglichst präzise erfolgen, da dies speziell für LFNs den maximalen Durchsatz bestimmt
- drei unterschiedliche Methoden sind sowohl in Endstationen wie in Routern gebräuchlich:
- *Originalimplementierung*: sie sieht vor, die RTT für jedes einzelne TCP-Paket zu ermitteln und hieraus ein gewichtetes Mittel rekursiv zu berechnen
- *Karn/Partridge-Implementierung*
- *Jacobsen/Karel-Implementierung*

Originalimplementierung (1)

- $RTT'_{est} := a \times RTT_{est} + (1 - a) \times RTT_{samp}$
- RTT_{samp} ist der für das ACK benötigte aktuell gemessene Wert
- RTT_{est} startet bei $2s$
- Abschätzung für den Timeout: $Timeout' := b \times RTT_{est}$

Originalimplementierung (2)

- übliche Werte sind $a = 0,9$ und $b = 2$
- Probleme mit diesem Ansatz:
- bereits ein verlorengegangenes Paket führt zur Unterschätzung der RTT
- außerdem keine Korrelation mit den ACKs des Empfängers gegeben

Karn/Partridge-Implementierung

- umgeht die letzte Einschränkung:
- erst das ACK bestätigt den Datenempfang
- wiederholte TCP-Pakete werden nicht in den Algorithmus einbezogen
- zudem wird die Timeout-Zeit nach jedem Empfang angehoben:
- $Timeout' := 2 \times Timeout$

Jacobsen/Karel-Implementierung

- verfeinert den Karn/Partridge-Algorithmus durch Einbeziehen der Varianz in RTT_{samp}:
- $\delta(RTT) := RTT_{samp} - RTT_{est}$
- $RTT'_{est} := RTT_{est} + g_0 \times \delta(RTT)$ mit $g_0 = 0,125$
- $Abweichung' := Abweichung + g_1 \times \delta(RTT)$ mit $g_1 = 0,25$
- $Timeout' := p \times RTT_{est} + q * Abweichung$ mit Erfahrungswerten $p = 1$ und $q = 4$

Congestion control (1)

- TCP beinhaltet verschiedene Mechanismen zur *Verstopfungskontrolle (congestion control)*
- *TCP Slow Start*: Datenübermittlung beginnt mit einem kleinen *congestion window* cwnd, i.d.R. 1 minimales Segment (536 Byte)
- anschließend wird cwnd mit jedem empfangenen ACK-Segment exponentiell vergrößert
- d.h. $cwnd' := cwnd \times cwnd$
- die Anzahl der übertragenen TCP-Segmente wird entsprechend erhöht
- bis der Empfänger bzw. ein zwischengeschalteter Router Paketverluste signalisiert
- daraufhin wird cwnd reduziert (da die Kapazität des Netzwerks bzw. des Empfängers überschritten wurde)

Congestion control (2)

- *Congestion Avoidance*: Annahme, daß Datenpakete in heutigen Netzen kaum mehr verlorengehen

- sondern bei Timeouts und doppelt empfangener ACKs (dACKs) eine Überlast im Netzwerk aufgetreten ist

- neben cwnd wird eine Variable *Slow Start Threshold* ssthresh benutzt

1. Initialisierung: $cwnd$:= 1 $Segment$, $ssthresh$ = max. Windowsize (64 kByte)
2. maximale zu sendende Datenmenge: $min(cwnd, advWin)$
3. beim Empfang von dACKs wird ssthresh neu berechnet:
   $ssthresh' := max(2, min(cwnd/2, advWin))$
4. falls Timeouts registriert werden, gilt: $cwnd' := 1$
5. beim Empfang neuer ACKs wird cwnd nach Slow Start oder nach Congestion Avoidance wieder erhöht

Congestion control (3)

- *Fast Retransmit/Fast Recovery*: geht davon aus, daß mehrere empfangene dACKs den Verlust lediglich eines TCP-Segment bedeuten

- wenn Anzahl der dACKs Schwellwert überschreitet (z.B. 3 dACKs), tritt ein Fast Retransmit in Kraft, indem nicht der TCP-Timeout abgewartet wird

- anschließend Fast Recovery: Annahme, daß ein (noch) anhaltender Datenfluß vorliegt

Congestion control (4)

- *Selective Acknowledgements*: begegnet dem Problem, daß mit hoher Wahrscheinlichkeit beim Registrieren von kumulativen dACKs nur ein Paket neu übertragen werden muß
- Fast Retransmit würde hingegen mehrere TCP-Segmente wiederholen
- Empfänger teilt Sender den Beginn und das Ende derjenigen Datenblöcke mit, die er als letzte zusammenhängend in seinem Empfangsfenster (Datenpuffer) verarbeitet hat
- SACK-Option erweitert das Optionsfeld um bis zu maximal 40 Bytes

Transaction TCP (T/TCP) (1)

- Weiterentwicklung und Ergänzung von TCP (RFC 1644)
- TCP arbeitet symmetrisch (zwei gegenläufige Verbindungen)
- viele Anwendungen kommunizieren jedoch asymmetrisch
- z.B. HTTP/Webseiten: kleine Anfrage, große Antwort
- Frage/Antwort-Schema wird auch als *Transaktion* bezeichnet
- mit TCP möglich: unabhängige Verbindungen je Anfrage oder Wiederverwenden einer Verbindung

Transaction TCP (T/TCP) (2)

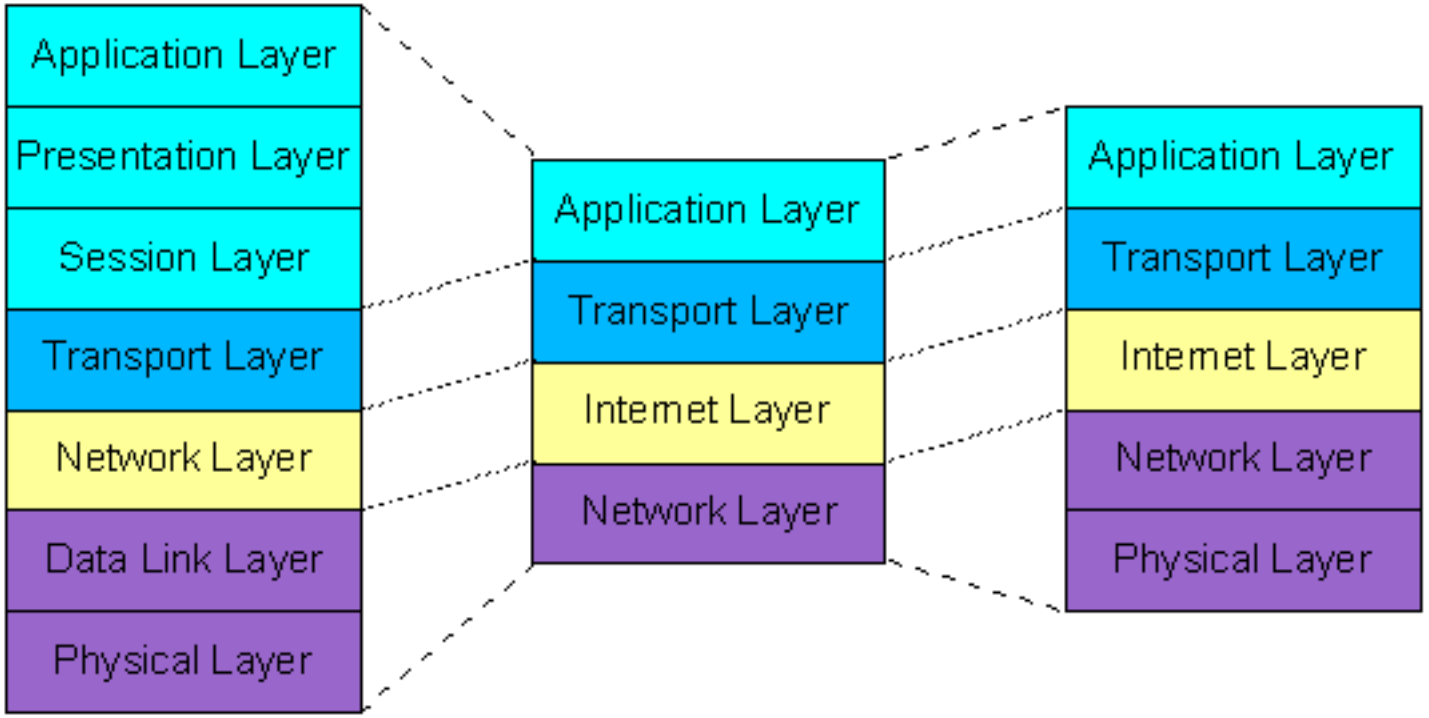
- T/TCP: *Connection Count (CC)* wird als Transaktionszähler mitgesendet
- außerdem: TCP Accelerated Open (TAO): Möglichkeit, den 3-Wege-Handshake zu umgehen, indem der letzte Wert des empfangenen und gesendeten CC mitgeführt wird
- Bei Empfang von SYN mit CC-Wert höher als im Cache: Datentransferphase
- da sicher, der TCP-Sender sich auf die vorige Transaktion bezieht
- falls CC-Wert nicht mehr im Cache wird normaler Verbindungsaufbau durchgeführt
- nötig für T/TCP ist jedoch ein sehr viel umfangreicheres Zustandsmodell

Name Service (1)

- bislang nur vorgestellt: Identifizieren von Knoten über *Adressen*
- Adressen sind sehr gut geeignet für die Verarbeitung in Routern
- jedoch sind sie nicht benutzerfreundlich
- Menschen jedoch können mit *Namen* sehr viel besser umgehen
- daher wünschenswert: Dienst, der Namen in Adressen übersetzt (auch umgekehrt)

Name Service (2)

Einordnung von Name-Services: Sitzungsschicht (session layer)



Name Service (3)

- Unterschiede zwischen Hostnamen und Hostadressen:
- Namen besitzen i.d.R. variable Länge
- Namen bestehen i.d.R. aus alphanumerischen Zeichen
- Namen beinhalten i.d.R. keine Routing-Informationen
- einige Adressformen hingegen enthalten Routing-Informationen
- bei Adressen sind sog. *flache Adressen (flat addresses)* die Ausnahme
- d.h. Adressen, die sich nicht in weitere Bestandteile aufteilen lassen

Name Service (4)

Terminologie:

- *Namensraum (name space)*: Menge möglicher Namen
- *flacher Namensraum*: Namen lassen sich nicht in weitere Bestandteile aufteilen
- z.B. Dateinamen bei CP/M
- *hierarchischer Namensraum*: Namen lassen sich in hierarchisch geordnete Bestandteile aufteilen
- z.B. Unix-Verzeichnis- und -Dateinamen

Name Service (5)

Terminologie (cont'd):

- *Verknüpfungen (bindings)* zwischen Namen und Werten (z.B. Adressen)
- *Bestimmungs- oder Auflösungsmechanismus (resolution mechanism)*: ein Prozedere, das bei gegebenem Namen den zugehörigen Wert ermittelt
- *Name Server*: spezielle Implementation eines Auflösungsmechanismus, die über ein Netzwerk das Abfragen durch Zusenden von Nachrichten erlaubt

Historie: `hosts.txt` (1)

- das erste verbreitete System zur Namensauflösung: `hosts.txt`
- eingeführt, als das Internet noch aus wenigen hundert Hosts bestand
- Datei wurde vom *Network Information Center (NIC)* zentral verwaltet
- enthalten in der Datei: Zuordnungen Name–Adresse (je eine pro Zeile)
- immer, wenn ein neuer Host aufgenommen werden sollte, wurde eine Email ans NIC geschickt

Historie: `hosts.txt` (2)

- beim NIC wurde die Information manuell eingetragen und verwaltet
- die modifizierte Tabelle wurde dann alle paar Tage an alle Systemadministratoren verschickt
- diese sorgten dafür, daß die neue Version auf den ihnen unterstellten Hosts installiert wurde
- die Namensauflösung geschah auf jedem Rechner lokal durch Nachschlagen in der Datei
- prinzipiell kann das System noch heute benutzt werden

Auszug /etc/hosts

- unter Unix-Systemen: Datei /etc/hosts:

```
127.0.0.1       localhost

129.70.123.67   sadewa.rvs.uni-bielefeld.de   sadewa
129.70.123.40   matrix   matrix.rvs.uni-bielefeld.de
129.70.123.20   mailhost
129.70.123.40   timehost  loghost

129.70.123.31   sunpizz1
129.70.123.32   sunpizz2
129.70.123.33   sunpizz3

# special IPv6 addresses
::1             localhost ipv6-localhost ipv6-loopback

fe00::0         ipv6-localnet

ff00::0         ipv6-mcastprefix
ff02::1         ipv6-allnodes
ff02::2         ipv6-allrouters
ff02::3         ipv6-allhosts
```

Auszug `lmhosts`

- unter Windows: `C:\WINDOWS\lmhosts` oder `C:\WINNT\System32\Drivers\Etc`

- inkl. Zusatzfunktionen (z.B. #PRE: sofort in Nameservercache laden)

```
129.70.123.12  studs012  #PRE  #DOM:RVS_CIP
129.70.123.13  studs013  #PRE  #DOM:RVS_CIP
129.70.123.14  studs014  #PRE  #DOM:RVS_CIP
129.70.123.15  studs015  #PRE
129.70.123.16  studs016  #PRE
```

DNS (1)

- zwar kann der `hosts.txt`-Ansatz heute immer noch verwendet werden
- jedoch ist die Anwendung in größeren Netzen zu aufwendig
- im Internet wurde dieser Ansatz daher zugunsten eines neuen verworfen
- Mitte der 1980er wurde das *Domain Naming System (DNS)* eingesetzt
- heute lassen sich beide Konzepte auch gemischt einsetzen
- dabei besitzt die `hosts.txt` nur lokale Gültigkeit

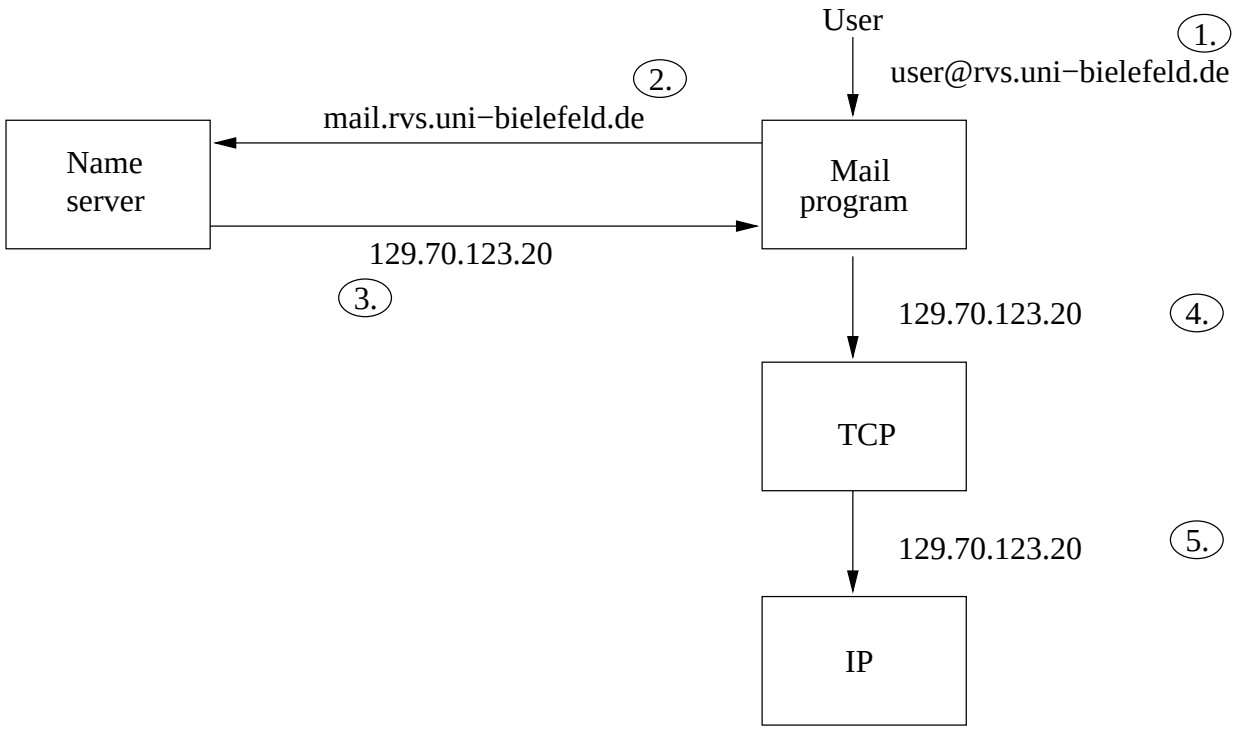
DNS (2)

- DNS benutzt einen hierarchischen Namensraum
- die „Tabelle“ der Verknüpfungen liegt nicht zentral vor
- sondern ist in disjunkte Teile (Untertabellen) geteilt über das Internet verteilt
- diese Untertabellen werden von *Nameservern* zwecks Abfrage zur Verfügung gestellt
- jeder Nameserver ist nur für seinen Anteil zuständig
- es ist möglich, daß ein Nameserver Anfragen über andere Bereiche direkt an andere, dafür zuständige Nameserver weiterreicht

DNS (3)

- ein Anwender braucht seiner Anwendung nur einen Hostnamen geben
- die Anwendung kann dann das DNS benutzen, um diesen in die Hostadresse zu übersetzen
- mit diesen Informationen kann die Anwendung anschließend eine Kommunikation starten
- dem Transportprotokoll gibt sie die nachgeschlagene Hostadresse mit
- dieses reicht die Information an das Internetprotokoll weiter. . .

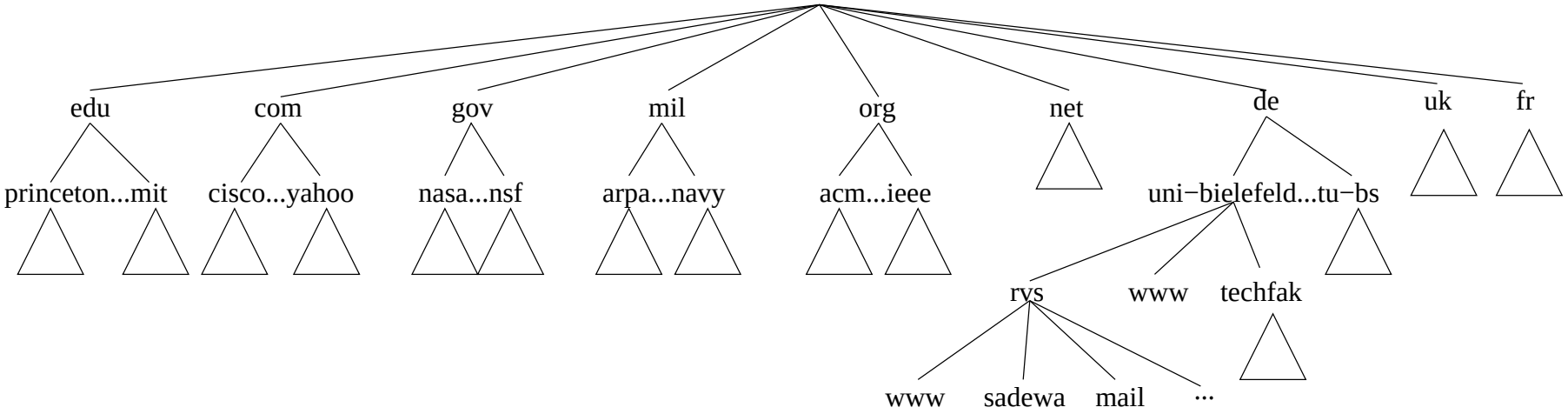
DNS (4)


User
1.
user@rvs.uni−bielefeld.de
2.
mail.rvs.uni−bielefeld.de
Name server
Mail program
129.70.123.20
3.
129.70.123.20
4.
TCP
129.70.123.20
5.
IP

DNS (5)

- DNS-Namen bilden eine Hierarchie, Trennung geschieht durch Punkte
- DNS-Namen werden von rechts nach links verarbeitet
- jedoch meist von links nach rechts gelesen
- Beispiel: `sadewa.rvs.uni-bielefeld.de`
- `sadwea` ist ein Hostname der Domäne `rvs.uni-bielefeld.de`
- `rvs` ist eine Domäne im Domänenbereich `uni-bielefeld.de`
- `uni-bielefeld` ist eine Domäne im Domänenbereich `de`

DNS (5)


edu
com
gov
mil
org
net
de
uk
fr
princeton...mit
cisco...yahoo
nasa...nsf
arpa...navy
acm...ieee
uni−bielefeld...tu−bs
rvs
www
techfak
www
sadewa
mail
...

DNS (6)

- was ist nun eine Domäne?
- ein Kontext in dem weitere Namen definiert werden können
- Domänen können direkt Hosts aber auch Unterdomänen enthalten
- die oberste DNS-Domänenebene ist historisch bedingt
- die enthaltenen Domänen werden *Top Level Domains* genannt
- sie enthält neben Länderdomänen die Domänen `edu`, `com`, `gov`, `mil`, `org`, `net`
- derzeit werden weitere Top-Level-Domains in Betrieb genommen (z.B. `info`, `name`)

DNS (7)

- diese ganze Domänenhierarchie existiert nur im Abstrakten
- tatsächlich werden jeweils einzelne *Zonen (zones)* separat verwaltet
- für jede Zone gibt es eine zuständige Administration
- diese verwaltet jedoch nur ihren eigenen Ausschnitt
- die oberste Ebene wird weiterhin vom NIC verwaltet
- in der obersten Ebene sind die sogenannten *Root-Server* angesiedelt

DNS-Funktion (1)

- jede Zone besitzt mindestens einen Nameserver, der für die Zone zuständig ist
- üblich sind zwei oder mehr Nameserver für Redundanz (zur Ausfallsicherung)
- ein Nameserver kann dabei auch mehr als eine Zone verwalten, falls nötig
- die Zoneninformation ist als Sammlung aus einzelnen Einträgen (*resource records*) zusammengesetzt
- jeder Eintrag besteht aus einem 5er-Tupel:
- *Name, Wert, Typ, Klasse, TTL (name, value, type, class, ttl)*

DNS-Funktion (2)

- zu *Name* wird der *Wert* zugeordnet
- *Typ* gibt an, wie der Wert zu interpretieren ist
- *Klasse* sollte dazu dienen, daß auch andere Organisationen als das NIC Einträge verfassen können
- bis heute ist jedoch nur eine Klasse wirklich relevant, die Internet-Klasse: *IN*
- *TTL (time to live)* gibt an, wie lange ein Eintrag gültig ist

DNS-Funktion (3)

Wichtige Typen sind:

- *A*: Wert ist eine IP-Adresse
- *NS*: Wert gibt zu Hostnamen zu einem Domainnamen an, auf dem ein Nameserver läuft, der Namen dieses Domainnamens auflösen kann
- *CNAME*: Wert gibt den kanonischen Namen zu einem Host an – dient dazu, Aliase zu definieren
- *MX*: Wert gibt den Namen eines Hosts an, der zur gegebenen Domäne adressierte Mails akzeptiert

DNS-Funktion (4)

Beispieleinträge (hier ohne TTL)

- < rvs.uni-bielefeld.de, NS1.uni-bielefeld.de, NS, IN >
- < rvs.uni-bielefeld.de, mail.rvs.uni-bielefeld.de, MX, IN >
- < NS1.uni-bielefeld.de, 129.70.182.7, A, IN >
- < sadewa.rvs.uni-bielefeld.de, 129.70.123.67, A, IN >
- < www.rvs.uni-bielefeld.de, 129.70.123.10, A, IN >
- < mail.rvs.uni-bielefeld.de, 129.70.123.20, A, IN >

DNS-Funktion (5)

- Anfrage nach sadewa.rvs.uni-bielefeld.de an Root-Server
- dieser liefert „best match“ zurück: NS-Eintrag für uni-bielefeld.de: `ws-mue1.win-ip.dfn.de`/193.174.75.166
- Anfrage nach sadewa.rvs.uni-bielefeld.de an 193.174.75.166
- dieser liefert „best match“ zurück: NS-Eintrag für rvs.uni-bielefeld.de: `NS1.uni-bielefeld.de`/129.70.182.7
- Anfrage nach sadewa.rvs.uni-bielefeld.de an 129.70.182.7
- dieser liefert A-Eintrag zurück: `sadewa.rvs.uni-bielefeld.de`/129.70.123.67

DNS-Funktion (6)

- wodurch ist Root-Server bekannt?
- es gibt eine Liste von *well known addresses* von Root-Servern
- diese ändert sich (fast) nie
- meist gibt es einen lokalen Nameserver, der die Anfragen im Auftrag stellt
- und dann gleich das Endergebnis an die anfragende Anwendung zurückliefert
- diesen wird die Liste bekannt- bzw. mitgegeben
- die Anwendung braucht nur den lokalen Nameserver kennen

Beispiel BIND-Konfiguration


```
options {
        directory "/var/cache/bind";
        forwarders { 194.158.230.53; 145.253.2.203; 145.253.2.19; 194.25.2.129; 212.185.248.180; };
        auth-nxdomain no;    # conform to RFC1035
}

// prime the server with knowledge of the root servers
zone "."                { type hint;   file "/etc/bind/db.root";  };

// be authoritative for the localhost forward and reverse zones, and for broadcast zones as per RFC 1912
zone "localhost"        { type master; file "/etc/bind/db.local"; };
zone "127.in-addr.arpa" { type master; file "/etc/bind/db.127";   };
zone "0.in-addr.arpa"   { type master; file "/etc/bind/db.0";     };
zone "255.in-addr.arpa" { type master; file "/etc/bind/db.255";   };

// add your zones below here
zone "mynet.local"         { type master; file "/etc/bind/my.zone";  };
zone "17.172.in-addr.arpa" { type master; file "/etc/bind/my.rev";   };
```

Beispiel my.zone


```
$TTL    604800
@       IN      SOA     dns.mynet.local. root.dns.mynet.local. (
                     2000101301         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      dns.mynet.local.
@       IN      MX      10      mail.mynet.local.

; Server
isengard        IN      A       172.17.1.1
isengard        IN      HINFO   "AMD K5" "Linux"
gimli           IN      A       172.17.1.10
gimli           IN      HINFO   "Cyrix 6x86" "NetBSD"

; Aliase
dns             IN      CNAME   isengard.mynet.local.
mail            IN      CNAME   isengard.mynet.local.
time            IN      CNAME   isengard.mynet.local.
www             IN      CNAME   gimli.mynet.local.

; normale Rechner
feodor          IN      A       172.17.2.1
faramir         IN      A       172.17.2.2
```

Beispiel `my.rev`


```
$TTL    604800
@       IN      SOA     dns.mynet.local. root.dns.mynet.local. (
                     2000101301         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      dns.mynet.local.

; Server
1.1     IN      PTR     isengard.mynet.local.
10.1    IN      PTR     gimli.mynet.local.

; normale Rechner
1.2     IN      PTR     feodor.mynet.local.
2.2     IN      PTR     faramir.mynet.local.
```

Dynamic DNS

- DynDNS-Dienst gedacht für Hosts, die öfters ihre IP-Adresse wechseln
- z.B. DSL-Zwangstrennung durch Provider mit anschließend neuer IP-Adresse
- Nameserver muß außerhalb stehen (ständig erreichbar)
- Host teilt bei jeder IP-Adreßänderung diese dem Nameserver mit
- dazu dient i.d.R. ein spezielles Programm (nicht standardisiert)
- TTL-Wert wird sehr niedrig angesetzt (üblicherweise sonst erheblich größer)
- Ergebnis: sogar Server mit nur kurzen Aussetzern betreibbar

DNS-Sicherheit

- DNS ist von sich aus nicht besonders abgesichert
- in der Vergangenheit hat es viele Sicherheitslücken bei Nameservern gegeben
- Nameservereinträge selbst werden aber üblicherweise nicht von außen getätigt
- sondern vom jeweils zuständigen Administrator
- üblicher Zugriff ist lesend
- prinzipiell aber sind Nameserver Sicherheitsrisiken

IP-over-DNS (1)

- viele Nameserver sind öffentlich und frei zugänglich
- oft noch bevor sich ein Host authentifiziert hat
- z.B. an Flughäfen, öffentlichen Hot-Spots
- findige Hacker haben einen Tunnelmechanismus entworfen
- sie tunneln IP über DNS

IP-over-DNS (2)

- benötigt wird dafür eine eigene Domain mit eigenem Nameserver
- dieser kann sich irgendwo im Internet befinden (z.B. zuhause)
- er muß spezielle Antworten generieren können auf Anfragen
- ist man z.B. mit seinem Notebook an einem Flughafen mit offenem DNS
- so startet man dort den Client-Teil des Tunnels

IP-over-DNS (3)

- es werden nun die IP-Pakete in DNS-Anfragen umgesetzt
- der Flughafen-Nameserver fragt sich bis zum modifizierten Nameserver durch
- die IP-Pakete in Gegenrichtung werden vom modifizierten Nameserver als DNS-Antwort verschickt
- der Flughafen-Nameserver stellt die Antworten zu
- dadurch Umgehung der Sicherheitskonzepte möglich
- Weiteres siehe `http://slashdot.org/articles/00/09/10/2230242.shtml`

Zusammenfassung (1)

- wir haben drei verschiedene Ebenen von Identifikatoren betrachtet
- Domänennamen, IP-Adressen, physische Adressen
- außerdem die verschiedenen Funktionen zum Verknüpfen einer Ebene mit einer anderen
- diese Verknüpfungen geschehen an verschiedenen Punktenin der Netzwerkarchitektur

Zusammenfassung (2)

1. Benutzer nennen Domänennamen, wenn sie mit einer Anwendung interagieren

2. die Anwendung benutzt DNS, um Namen in IP-Adressen umzuwandeln. Die IP-Adresse wird in jedes Datengramm eingefügt, nicht der Domänenname

3. IP sorgt für das Weiterleiten an jedem Router, wobei evtl. IP-Adressen in andere IP-Adressen übersetzt werden

4. IP benutzt ARP, um die IP-Adresse für den nächsten Sprungpunkt in die physische Adresse (z.B. MAC-Adresse) umzuwandeln. Der nächste Sprungpunkt kann das endgültige Ziel oder nur ein weiterer Router sein. Die Rahmen im physischen Netzwerk beinhalten die physische Adresse im Header.

Aussicht

Themenübersicht für die kommende Vorlesung:

- Anwendungsprotokolle und -mechanismen

Ende Teil 11. Danke für die Aufmerksamkeit.

Ich wünsche ein frohes Weihnachtsfest und einen guten Rutsch ins neue Jahr!