1 Verteilte Systeme

Früher: Mainframe: ein Rechner bietet Dienste an. Viele Benutzer haben gleichzeitig Zugang

Heute: PCs: ein einzelner Rechner bietet nicht alle Dienste an. Rechner werden nur von einem oder wenigen Benutzern genutzt.

Häufige Dienste: VPN-Dienst, Datei- und Druckerfreigabe (Fileserver-Dienst), ping, Druckserver, Datenbank-Server, Mailserver, Webserver, Internetdienste (Chat, Video-Streaming

Vorteile VS

Preis / Leistung Standard-PCs

Gemeinsame Nutzung Peripherie

(Zugriff gemeinsame Daten und Programme)

Inkrementelle Erweiterbarkeit

→ Komplettausfälle des Systems werden vermieden

Nachteile VS

→ Pflege und Client-Rechnern ist aufwendig

Server bilden einzelne Ausfallpunkte für die jeweiligen Dienste

Ziel von verteilten Systemen

Ziel ist Transparenz

Ortstransparenz:

Zugriffstransparenz:

Nebenläufigkeitstransparenz: parallele Zugriffe möglich, beeinflussen sich nicht

Skalierungstransparenz:

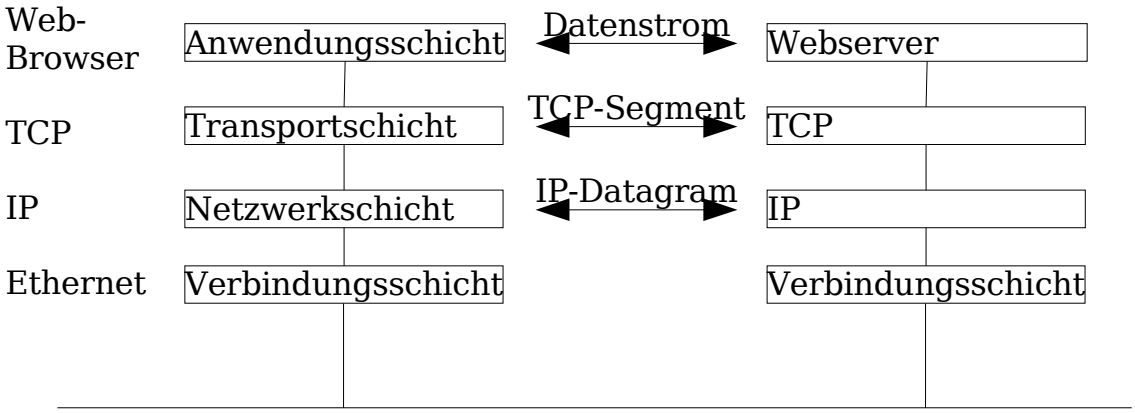
Migrationstransparenz:

Arten von Netzen

- Lokales Netz (LAN)
- Stadtnetz (MAN)
- Fernnetz (WAN)
- Internet

Metcalf's Law:

TCP/IP



TCP (Transmission Control Protocol)

Überprüft fehlerfreie Übertragung der Pakete

Fehlende Pakete werden neu angefordert / gesandt

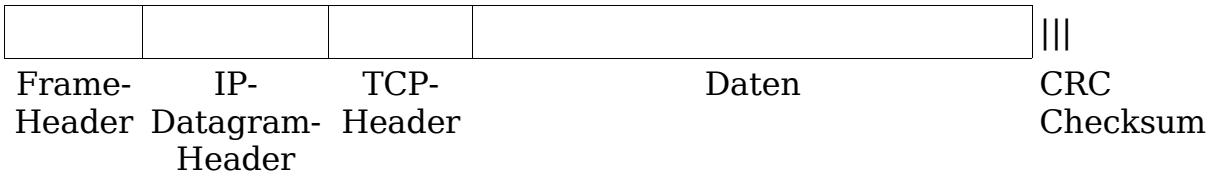
Pakete in falscher Reihenfolge werden geordnet

UDP (User Datagram Protocol)

Schneller und unzuverlässiger als TCP

Zustellung von Paketen wird nicht überprüft

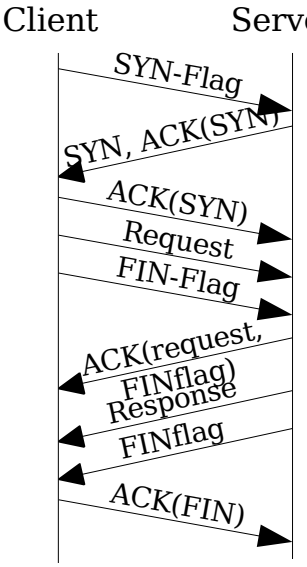
Beispiele: Multimedia-Anwendungen, die Verluste haben dürfen; DHCP; viele Online-Spiele wegen Geschwindigkeit



Adressierung:

TCP:	Port	2 Bytes	bekannt 0 ~ 1024, max 65535 (netstat prüft auf Listen von Ports); 80 HTTP; 110 POP3; 25 SMTP
IP:	IP-Adresse	4 Bytes	
Ethernet:	MAC-Adresse	6 Bytes	ARP → Tabelle, die Verknüpfung von IP-Adresse zu MAC-Adresse herstellt

ICMP (Internet Control Message Protocol)



primitiver als UDP und TCP, läuft aber auch über IP (z.B. ping)

Namensauflösung

DNS: Domain Name System (Tool zum Testen der Namensauflösung: nslookup)

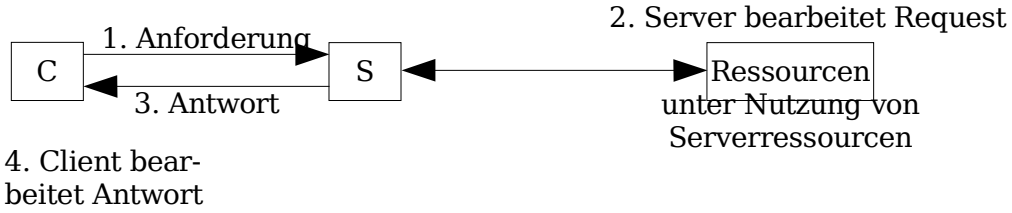
$$194.94.240.186 \approx \underbrace{\text{www}}_{\text{Rechner}}.\underbrace{\text{fh}-\text{ingolstadt}}_{\text{Domäne}}.\underbrace{\text{de}}_{\text{TOP-Level-Domain}}$$

DHCP: DHCP-Client (z.B. in Windows fest implementiert und aktiv, wenn keine feste IP-Adresse eingestellt; schickt bei der Anforderung (z.B. Start des Rechners oder ipconfig /Renew) einen Broadcast mit der Anfrage eines DHCP-Server, worauf dieser mit einer Client-TCP-Adresse antwortet (+ DNS-Server-IP, GW-Adresse, Subnet, ...)

2 Architekturen für verteilte Systeme

Server: Stellt Dienst bereit, entweder Knoten (Rechner) oder einfach nur ein Prozess

Client: greift auf Dienste eines Servers zu, besitzt n:1 Beziehung zu Server. Initiative einer Interaktion geht stets vom Client aus



Prüfung:

Proxy: $C^+S_PS^+$

- cached Internetseiten zwischen und versucht Client selbst zu bedienen
- vertritt mehrere Server
- reduziert Netzlast, schnelle Antworten, erhöhte Sicherheit

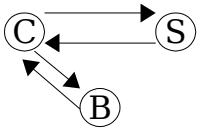
Broker: $C^+S_BS^+$

- vermittelt Dienste von anderen Servern
- Dienste werden dynamisch oder statisch dem Broker mitgeteilt, damit dieser diese Dienste dem Client zuteilen kann
- Vorteile: Migrations- und Replikationstransparenz

<u>Arten von Brokern:</u>

intermediate Broker

forwarding Broker



hybride Broker benutzen beide Arten

Trader: $C^+S_TS^+$

- sucht besten Service für Client aus mehreren Servern aus

Balancer: $C^+S_BS^+$

- für große Serverleistungen
- optimale Verteilung der Clients auf mehrere Server

Filter: $C^+S_FS^+$

Vorwärts-Filter

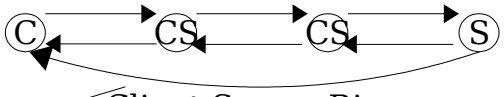
Rückwärts-Filter
Einsatz: Verschlüsseln / Entschlüsseln / Komprimieren von Daten

Koordinator:
wenn Client-Anfrage aus mehreren Services besteht, dann setzt er die Antwort selbstständig aus mehreren Einzelserviceanfragen zusammen.

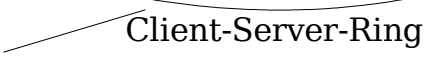
Agent:
ähnlich zu Koordinator, aber keine festgelegten Ausführungsschritte, sondern mittels z.B. KI ermittelt.

Client-Server-Ketten:

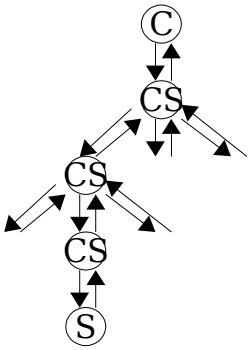
rekursiv



transitiv



Client-Server-Baum



Abarbeitung der Serviceanfragen am Server

Zwei Typen von Servern

Iterativ

```
for (;;) {
     get_request()
     process_request()
     send_response()
}
```

Parallel (Konkurrenz)

Dispatcher

```
for (;;) {
     get_request()
     create_process()
}
```

Worker

```
Process() {
     process_request()
     send_response()
}
```

Vorteil:

– Parallele (konkurrente) Server können mehrere Anfragen gleichzeitig

abarbeiten

- bessere Auslastung
- fatale Fehler führen unter Umständen nur zum Absturz des Workers

Weitere Typisierung

Dateninvariant: persistente Daten des Servers werden durch Requests der Clients nicht verändert (z.B. Name-Server)

Datenändernd: Client-Anfragen ändern persistente Daten des Servers. Reihenfolge der Abarbeitung von Bedeutung. (Fileserver)

Zustandsspeichernd: Server hat ein Gedächtnis (Session), in dem er Daten temporär speichern kann (File-Handles)

Zustandslos: Anforderungen müssen alle nötigen Parameter bereitstellen

- Mehr Kommunikation nötig
- Höhere Fehlertoleranz

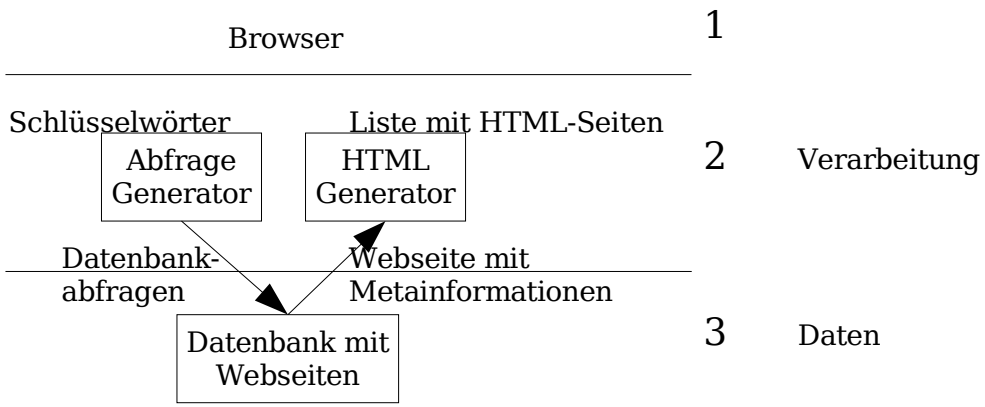
(reiner HTML-Server)

Aufteilung Client und Server

Logische Ebenen einer verteilten Anwendung:

- Ebene der Benutzeroberfläche (Frontend)
- Verarbeitungsebene (Applikation)
- Datenebene (Backend)

Beispiel Suchmaschine:



Logische Ebenen können unterschiedlich auf Client und Server verteilt werden.

Benutzer	Ben	Ben	Ben	Ben	Ben
Applikation	App	App	App	App	App
Daten	Daten	Daten	Daten	Daten	Daten
	a)	b)	c)	d)	e)

a) Null-Client (z.B. Terminals)
Client leitet Eingaben ohne Verarbeitung an den Server weiter

b) Thin-Client
Client kontrolliert gesamte Benutzereingabe (Webbrowser, der für Darstellung verantwortlich ist)

c) Applet-Client
Teile der Applikation laufen auf Client und überprüfen Konsistenz und Korrektheit der Daten (Formular komplett ausgefüllt)

d) Fat-Client (PC-Client / Fileserver)
Server ist nur noch für Datenhaltung zuständig

e) Fat Client
Teile der Datenhaltung auf Client (Server stellt Indexdaten für Suchabfrage bereit, die vom Client gemacht werden)

Vorteil Thin-Clients

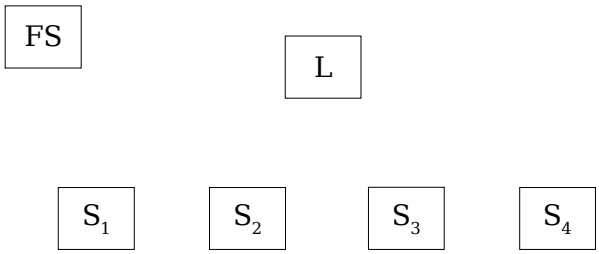
– leichte Administration

Vorteil Thick-Clients

– erfordern weniger Kommunikation

– teilweise schneller, da Clientperformance genutzt wird

Implementierung von verteilten Systemen



Einfache Steuerung:

Lehrer steuert:

- Word starten
- Abspeichern der Datei

Three- und Multitier-Architektur, Middleware

Die drei Ebenen einer Anwendung können auf mehrere Systeme verteilt sein.

Three- und Multitierarchitekturen nutzen oft Middleware. Diese läuft auf allen Rechnern der verteilten Anwendung.

Funktionen der Middleware:

- erlaubt Nutzung heterogener Rechnersysteme
- kümmert sich um Details des Datenaustausches und der Synchonisation
- stellt dem Programmierer eine proprietäre oder standardisierte API zur Verfügung
- entkoppelt Frontend und Backend
- stellt Orts-, Zugriffs- und Skalierungstransparenz her

Vorteil Middleware:

- große Anwendungen werden hiermit schneller implementiert

Nachteil Middleware:

- großer Overhead zur Socketprogrammierung und deshalb langsamer

Beispiel Middleware:

- objektorientiert: CORBA, DCOM
- RPC-basiert: DCE (Distributed Computing Environment)
- nachrichtenorientiert: MQSeries, Tibco

3

4 Verteilte Systeme: Problemstellungen und Algorithmen

4.1 Finden von Diensten

Können angesprochen werden über:

- Adressen (z.B. IP-Adressen oder benutzerfreundliche Namen)

– eindeutige ID

Beispiele:

- URL: $\underbrace{\text{https}}_{\text{Zugriffsprotokoll}}\text{://}\underbrace{\text{redhat}}_{\text{Rechner}}\text{.}\underbrace{\text{stroehmer}}_{\text{Domainname}}\text{.de:}\underbrace{\text{443}}_{\text{Port}}\underbrace{\text{/inha/abt1}}_{\text{Ressource}}$
- feste Lokationen (schnell aber unflexibel)
- variable Lokationen, z.B. Broadcast (nicht geeignet für große Netze, hier wird „Multicast“ eingesetzt)

Lokalisieren über Broker CSS+

Lokalisieren über Verzeichnisdienste. Hier werden die Informationen in einer Datenbank gespeichert (Bsp.: Novell NDS, Microsoft Active Directory)

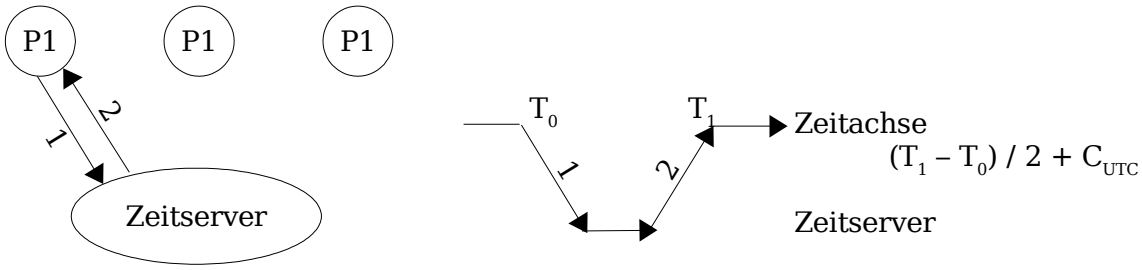
↳ nach X.500

/C=DB /O=FH Ingolstadt / OV=IN

4.2 Synchronisation

- Globale Zeitfindung
- Ermittlung globaler Zustand über alle Prozesse bzw. Knoten / Rechner

Physikalische Uhr



Bei kleinerem Unterschied zur Realzeit (UTC)

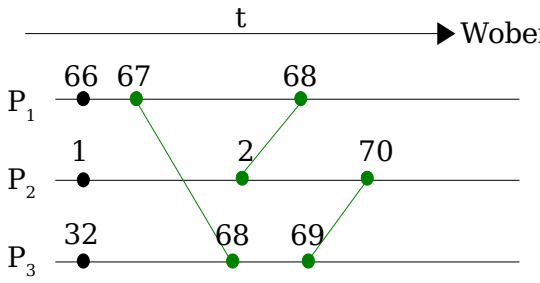
Logische Uhr

bei zwei Ereignissen a, b kann Relation $\overset{15}{\underset{p_1\,\text{Sender}}{a}} \overset{\text{(liegt vor)}}{\rightarrow} \overset{16}{\underset{p_2\,\text{Empfänger}}{b}}$ gesetzt wurde.

Logische Uhr nach Lampert:

- Ordne jedem Prozeß eine logische Uhr C_i zu, der bei jedem Ereignis inkrementiert wird und diesem einen Zeitstempel zuordnet.
  Konsistenzbedingung 1: $C_i(a) < C_i(b)$
  Konsistenzbedingung 2: P_1 sendet (Ereignis a) mit Zeitstempel $C_1(a)$ Nachricht
  P_2 empfängt (Ereignis b), wobei
  $C_2(b) = C_1(a) + 1$

Zusätzliche Forderung: Einführung einer totalen Ordnung, damit keine zwei



Kommunikation t_1 Nachricht von P_1 nach P_3
Kommunikation t_2 Nachricht von P_2 nach P_1
Kommunikation t_3 Nachricht von P_3 nach P_2

Zeiten zur selben Zeit auftreten können.
Ist a ein Ereignis in P_i und b ein Ereignis in P_j, so definiert man, daß $a \rightarrow b$ genau dann:

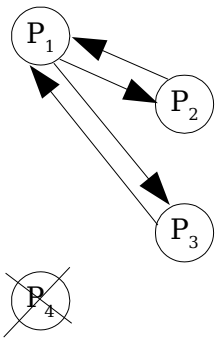
- $C_i(a) < C_j(b)$ oder
  $C_i(a) = C_j(b)$ und $i < j$

4.2.1 Auswahlalgorithmen

Zur Bestimmung eines neuen Prozesses eines replizierten Systems beim Ausfall des Masterprozesses.

Bully-Algorithmus

Strategie: der aktive Prozess mit dem höchsten Index wird neuer Master



1 Einer der replizierten Prozesse bemerkt den Ausfall.

2 Der Checker sendet an alle höheren Prozesse eine Nachricht und wartet bis zum Timeout. Falls er eine Rückantwort eines höheren Prozesses erhält, beendet er seine Aufgabe.

3 Der höchste Prozess (der also keine Antwort von einem höheren Prozess bekommen hat) benachrichtigt seine darunterliegenden Prozesse, daß er nun der Master ist.

Kommunikationsaufwand sehr hoch $O(n^2)$

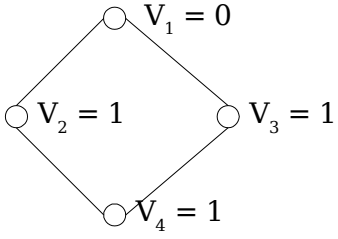
Verbesserung durch Ringtopologie → Jeder Prozeß kommuniziert nur mit seinem Nachfolger.

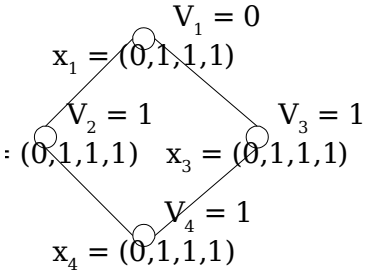
4.2.2 Übereinstimmungsalgorithmus

Problem: mehrere Prozesse sollen sich auf einen gemeinsamen Wert einigen:

– Fehlerhaftes Kommunikationsmedium

– Fehlerhaftes Verhalten eines oder mehrerer Prozesse (byzantinische Prozesse)

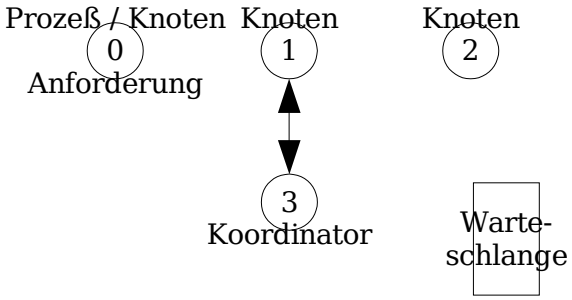




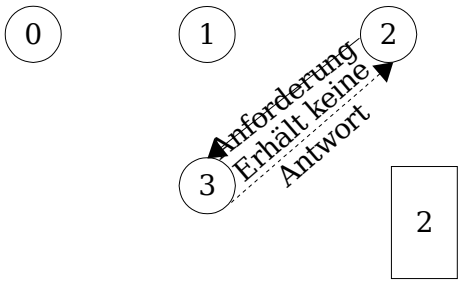
4.2.3 Wechselseitiger Ausschluß

Betreten eines kritischen Bereichs über Koordinator:

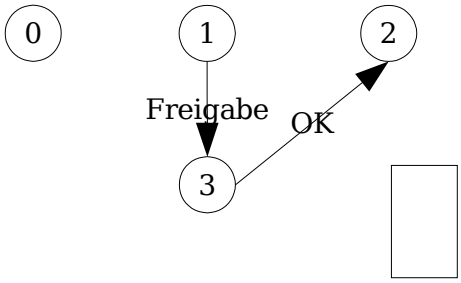
Will ein Prozeß einen kritischen Prozeß betreten, fordert er eine Berechtigung vom Server an. Der Koordinator erteilt diese, falls / sobald der kritische Bereich frei ist.



Zur selben Zeit



1 hat kritischen Bereich verlassen



Alternativen:

Verteilter Altorithmus ohen Koordinator (langsam, komplex)

Token Ring: Nur der Prozess mit dem Token darf kritischen Pfad betreten (Fair, Token kann verloren gehen)

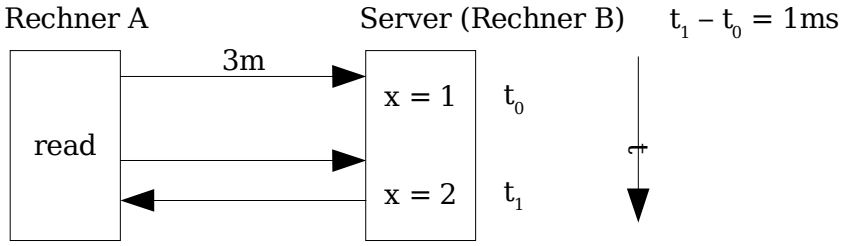
4.3 Replikation und Konsistenz

Replikation für Zuverlässigkeit und Leistungssteigerung (DNS) (nur wenn vorwiegend Lesezugriffe)

4.3.1 Lösung Konsistenzprobleme bei verschiedenen Replikaten

- Strenge Konsistenz
  - Jede Leseoperation gibt Wert zurück, der zuletzt geschrieben wurde (globale Zeit wird
  - Schreiboperationen müssen atomar durchgeführt werden

Strenge Konsistenz in VS unmöglich aus diesen und aus dem weiteren Grund, das die Kommunikation mit endlicher Geschwindigkeit erfolgt.



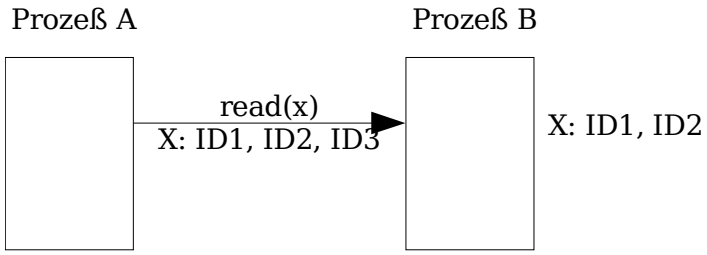
Durch die physikalischen Grenzen der Kommunikation kann nicht sichergestellt werden, daß zu einer Zeit t_x der aktuelle Wert gelesen werden kann.

Daher: Abschwächung der Lesekonsistenz

Beispiel: Streng monotone Lese-Konsistenz

– wenn Clients mit unterschiedlichen Replikaten arbeiten, liest ein Prozeßt ein Datum, so geben nachfolgende Leseoperationen den gleichen oder aktuelleren Wert zurück.

Implementierung: Den Schreiboperationen werden IDs zugeordnet. Diese werden beim Lesen übermittelt.



Prozeß bemerkt anhand des veralteten Zeitstempels ID, daß das Datum x nicht mehr aktuell ist und holt sich den aktuellen Wert von x.

Nachteil: hoher Kommunikationsaufwand

4.4 Datenformate

Little Endian (Motorola): niedrigstes Byte hat die niedrigere Speicheradresse

Big Endian (8086): umgekehrt

Solche und andere Unterschiede heterogener Rechnersysteme müssen bei der Kommunikation beachtet werden.

5 Kommunikation (über Netzwerke)

Bytestream: message based

Vollduplex, Halbduplex

connection, datagram

point-to-point, broadcast

5.1.1 Interaktionssemantik

Blockierend / synchron: Client wartet nach Absenden der Anforderung auf Rückantwort des Servers

- einfach zu programmieren
- ineffizient

Implementierung: accept ...

nicht blockierend / asynchron: Client arbeitet nach Absenden der Anforderung weiter. Irgendwann später nimmt er Antwort des Servers entgegen.

- Effizienter
- erhöhter Kontrollaufwand nötig
- (Variante: Ein-Weg-Kommunikation [UDP])

Implementierung:

- über Warteschlangen
- Calllocks, ähnlich Interrupt Request Routine (Funktionen oder Ereignisse, die beim Vorlegen der Antwort aufgerufen / ausgelöst werden)

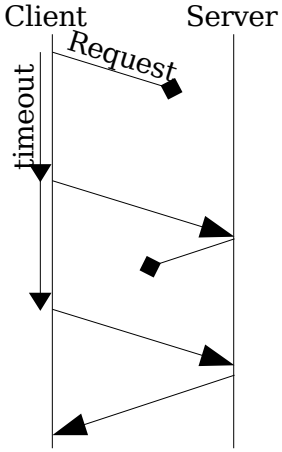
5.1.2 Ablaufsemantik (Umgang mit Interaktionsfehlern)

(unzuverlässiges Netzwerk (Medium), Verzögerungen, Ausfälle)

may be: Nachricht kommt nicht oder genau einmal beim Empfänger an (UDP)
(z.B. DHCP, Spiele)

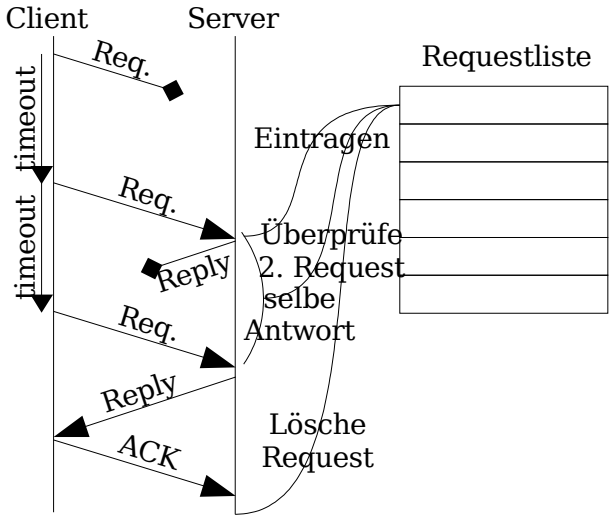
at least once: Nachricht kommt mindestens 1x an

- arbeitet mit Quittierungen
- Senden muß blockiert werden, bis Quittierung eintrifft
- nach timeout (warten auf Quittierung) erneutes Senden
- Nachricht kann öfter beim Empfänger ankommen



at most once: Nachricht kommt höchstens 1x an (außer Serverausfall)

- bekommt Empfänger eine Anforderung zum 2. mal, so weiß er, daß seine Antwort auf die 1. verloren ging und sendet die Antwort nochmals



exactly once:

- Nachricht kommt genau 1x an
- Nachrichten müssen in persistentem Speicher gehalten werden

5.2 Sockets

- Schnittstelle zur rechnerübergreifenden Interprozesskommunikation
- kapselt die unteren 3 Schichten, d.h. man muß sich bei der Programmierung nicht darum kümmern
- Endpunkte der Kommunikation, die mit Ports auf dem Rechner assoziiert sind

- Server erwartet Anfragen der Clients am Server-Socket
- Clients stellen Anfragen über Client-Socket
- es gibt TCP- und UDP-Sockets

5.2.1 Client-Sockets in Java

Ein Client-Socket (java.net.Socket) hat folgende Data Members

✗ Ferne IP-Adresse, Ferner Port

✗ lokale IP-Adresse, lokaler Port

✗ OutputStream zum Schreiben

✗ InputStream zum Lesen

folgende Methoden:
Konstruktor: erzeugt Socket + stellt Verbindung her
Close: zum Schließen der Verbindung und der Sockets

Kommunikationsablauf Client:

1. Socket erzeugen (mit IP / Port)
2. Daten schreiben
3. Daten lesen
4. Socket schließen

ServerSockets in Java (java.net.ServerSocket)

– lokale IP-Adresse, lokaler Port

– Backlog, Warteschlange für Verbindungsanforderungen

Methoden hierzu:
Konstruktor: Erzeugen eines Sockets
Accept: wartet, bis Client-Verbindung eintrifft. Dann wird ein Socket erzeugt, das mit dem anfordernden Socket verbunden ist
Close: Schließen des Serversockets

Kommunikationsablauf eines Servers

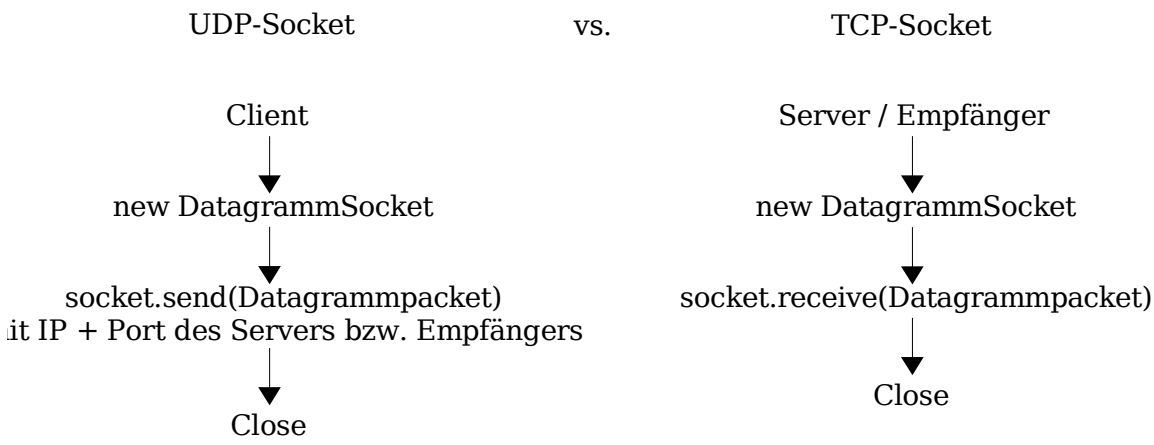
1. Server-Socket erzeugen (lokale IP und Port)
2. Warten auf Client-Anforderung mit accept (Akzeptiert diese hiermit)
3. Durch das accept erzeugte socket-Verbindung nutzen für: Daten lesen, Daten schreiben, schließen

Mehrfaches Akzeptieren von Verbindungsanforderungen

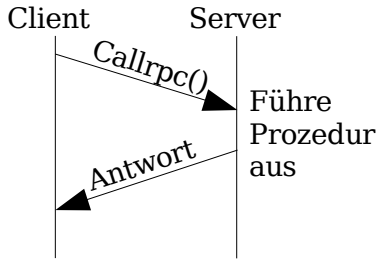
- Ein Serversocket kann ganze Warteschlangen von Anforderungen verarbeiten
- Es kann neue Verbindungsanforderung annehmen, bis alte Verbindung geschlossen wurde

Es existieren mehrere Socket-Verbindungen, bei denen die IP / Port-Kombinationen des Clients unterschiedlich sind (Beispiel: mehrere Browserfenster zum selben Webserver hört auf dem Client verschiedene Ports bei gleicher IP-Adresse)

UDP.Java-Beispiel

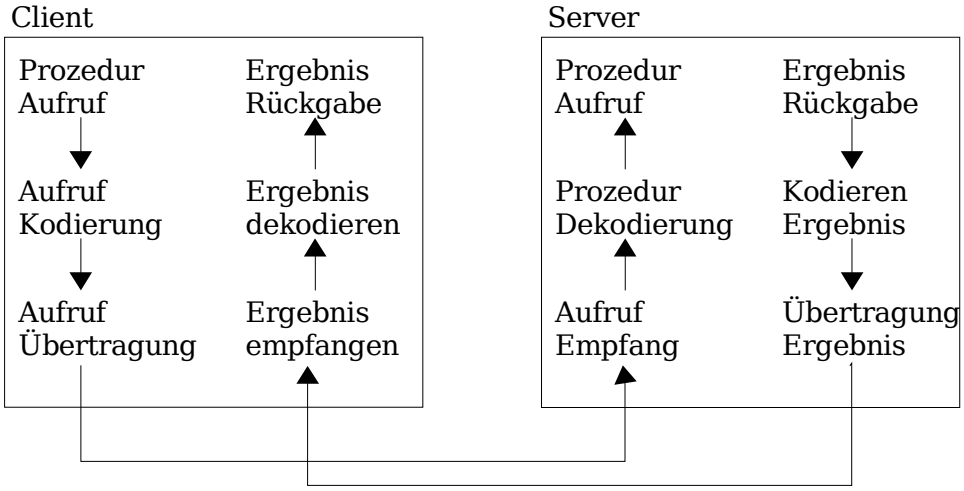


5.3 RPC



RPC übernimmt folgendes:

- Kodierung (z.B. ASN.1 oder XDR)
- Übertragung des Aufrufs
- Rückübertragung der Ergebnisse
- Dekodierung der Ergebnisse



- Werteparameter nicht problematisch
- Referenzparameter hingegen schon fremder Adreßraum
  - Verbieten
  - Call-by-Copy / restore
    - f(&a, &a)
      void f(int *x, *y) {
      *x++;
      *y++;
      }

Lokalisieren Server:

statisches Binden

direkte Ausgabe der Server-Adresse

dynamisches Binden

logischer Name, der vom Broker oder über Broadcast in physikalische Adresse umgewandelt wird

Beispiel für RPC-Systeme

- XEROX Courier RPC
- SVN RPC
- DCE RPC

Zum eigentlichen Code müßten noch folgende erstellt werden:

- Stubs samt Header

- Datenkonvertierungsfilter

Diese können über das RPC-System generiert werden.

Beispiel RPC von Sun / Linux

1. Interface erstellen math.x - ADD
   - Multiply
   - CUBE

2. rpcgen -a math.x
   erzeugt math.h
   math_clnt.c mit Client Stubs
   math_svc.c mit low-level Funktionen am Server (Registrierung beim rpcbind-Daemon, Serverstub)
   math_xdr.c mit XDR-Konvertierungsroutinen

   wegen Option -a wird zusätzlich erzeugt:
   math_server.c ein Beispiel-Server, in den man die gewünschte Funktionalität einbaut
   math_client.c der Beispiel-Client, den man anpassen kann
   Makefile

5.4 Objekt / Komponentenorientierte Modelle

- Prozedurale Programmierung: Aufruf von Prozeduren über Netzwerk
- Objektorientierte Programmierung: Aufruf von Methoden über Netzwerk

Der Client meint wieder, die Methode eines lokalen Objekts aufzurufen. In Wirklichkeit befindet sich das Objekt auf entferntem Rechner.
(C++ Überladen von „->“ und „.“)

Beispiele: CORBA, Java RMI, um Middleware-Features angereicherte: EJB (Enterprise Java Beans)

Objektklassen:

- persistente Objekte: existieren auch dann, wenn sie nicht im Adreßraum des Servers gehalten werden.
- Transiente Objekte: existieren nur solange, wie es sich im Adreßraum des Servers befindet.

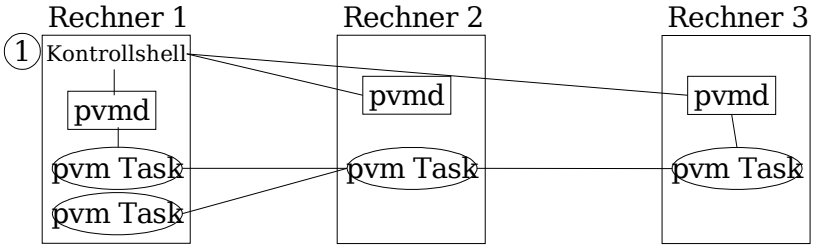
5.5 PVM (Parallel Virtual Machine)

Parallelisierung für wirtschaftliche und technische (rechenintensive) Programme

Speedup = Faktor, um den sich beim Rechnen die Ausführungsgeschwindigkeit beschleunigt. (optimal n-fache Beschleunigung)

- Public Domain (Programmbibliothek)
- Zusammenschluß von Workstations zu virtuellen Parallelrechnern
- Parallelität auf Basis von Unix-Prozessen
- Auf jedem Rechner läuft ein Dämon (pvmd), der die anderen Rechner kennt.

- Expliziter Nachrichtenaustausch zwischen beteiligten Prozessen (Nachrichten nicht persistent)



① Kontrollshell:
Verwaltungsoperationen: Host hinzufügen / entfernen,
Prozesse spawnen (≙ erzeugen)

weitere parallele Implementation gibt es MPI (Message Parsing Interface)
http://www.csm.gov/pvm/pvm_home.html

5.6

5.6.1 Zwei Arten von Kryptosystemen

a) Symmetrisch, gleiche Schlüssel zum Ver- und Entschlüsseln

b) Asymmetrisch, verschiedene Schlüssel zum Ver- und Entschlüsseln

a) Gut geeignet für vertrauenswürdige Partner

z.B.

- DES – Data Encryption Standard
  Jeder 64 Bit Klartext → 64 Bit Chiffretext
  Schlüssel ist 64 Bit, wobei 56 + 8 Bit Prüfsumme = $2^{56} = 10^{17}$ mögliche Schlüssel

  Erweiterung des Chiffretextes: Komplexe Reihe von Permutationen und Substitutionen
  Ergebnis wird zusätzlich mit X-OR mit der Eingabe verknüpft und das Ganze 16x wiederholt
- IDEA – International Data Encryption Algorithm
  wie DES, jedoch mit 128 Bit Schlüssel
  werden 64 Bit Klartext verschlüsselt und 17 Wiederholungen mit sogenannter „Fleischwolf-Funktion“
- RCH – Rivest Chiper No. 4
  neben DES und IDEA besonders im Internet geläufig
  (genauere Beschreibung im Buch von Schluch)

b) Asymmetrische Kryptosysteme

$$\underbrace{W_d}_{\text{decrypt}} + \underbrace{W_e}_{\text{encrypt}}$$

W_d kennt nur Empfänger; ist schwer abzuleiten von W_e
W_e ist öffentlich
P Public Key = Öffentlicher Schlüssel
S Secret Key = Geheimer Schlüssel

Nachteil: Sehr rechenintensiv und nur für kleine Datenmengen geeignet. Deshalb wird dieses Verfahren nur zur Herstellung einer gesicherten Verbindung zwischen zwei Knoten verwendet, die anschließend ein symmetrisches Kryptosystem verwenden.

Bekanntes Beispiel: RSA

5.6.1.1 Schlüsselverteilungsproblem

2 Benutzer (Personen, Knoten, Programme) brauchen passende Schlüssel zum Ver- und Entschlüsseln von Nachrichten

→ privater Kanal

Hierzu muß jedoch ein Schlüssel von einem zum anderen über das gleiche unsichere Medium transferiert werden. Deshalb muß er verschlüsselt werden.

Zum Anlegen eines privaten Kanals mit symmetrischer Verschlüsselung kann ein

a) asymmetrisches Kryptosystem oder

b) ein Schlüssel-Server

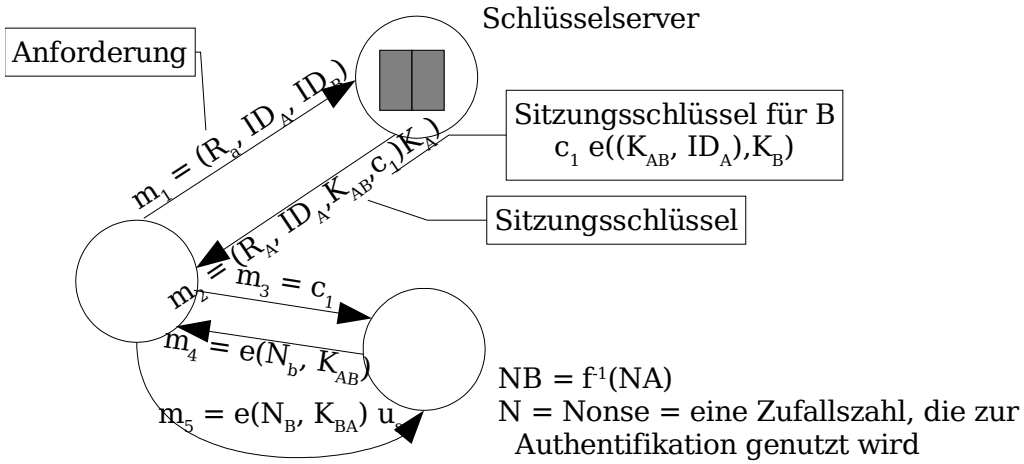
benutzt werden.

Zu a): Schlüsselübertragung mit Hilfe eines Kyrptosystems:

1) Knoten A hinterlegt öffentlich K_z für B
2) Knoten B hinterlegt öffentlich Schlüssel K_z zum Verschlüsseln des eigentlichen symmetrischen Schlüssels K $C_1 = e(K, K_e)$ und sendet diesen an A.
3) A entschlüsselt mit seinem geheimen Schlüssel K_B den verschlüsselten, symmetrischen Schlüssel K $K = d(e(K, K_e), K_d)$ und hat somit den symmetrischen Schlüssel K, um fortan die Nachrichten mit diesem zu verschlüsseln

zu b) Schlüsselverteilung über Schlüsselserver

Zum Einrichten eines sicheren Kanals zwischen A und B kann man auf ein asymmetrisches Kryptosystem verzichten, wenn man zwei zentrale vertrauenswürdige Schlüsselserver einsetzt. Der Server besitzt auch eine Tabelle mit allen gelesenen Schlüsseln der Benutzer.



5.6.2 Authentifikation

Mögliche Techniken:

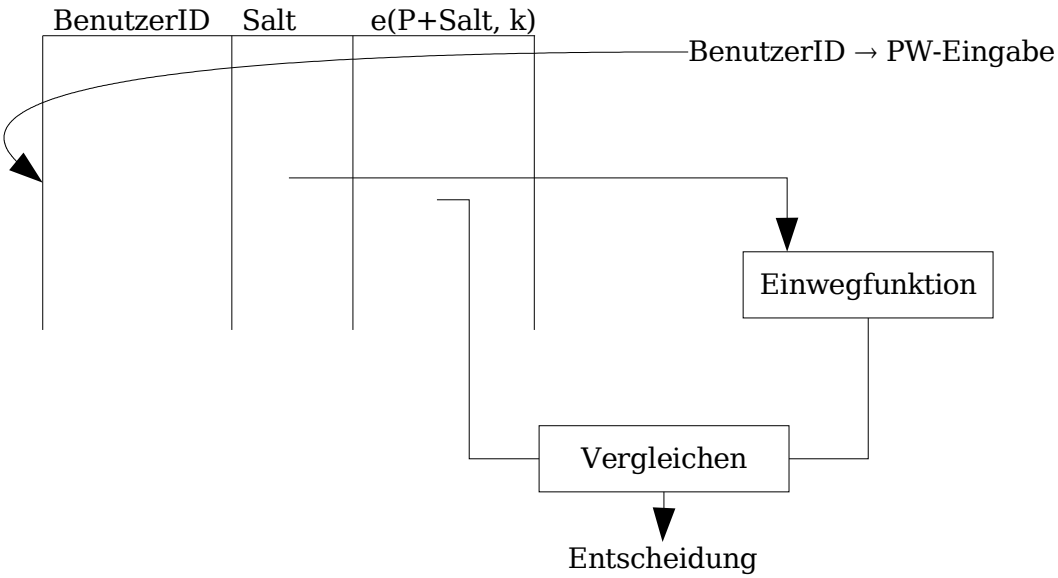
- proof by knowledge: Was der Anfordernde weiß, z.B. Paßwort
- proof by posession: Was der Anfordernde besitzt, z.B. Schlüssel
- proof by property: bestimmte Eigenschaft, z.B. Fingerabdruck

5.6.2.1 Passwortbasierte Authentifikation (proof by knowledge)

Paßworttabelle: geschützt, nur authentifizierter Zugriff; Namen, Paßwort als Chiffretext

Zur Verschlüsselung verwendet man nur sehr schwer umkehrbare Funktionen → Einwegfunktion

$$C = e(\underbrace{P}_{\text{PW im Klartext}}, \underbrace{K}_{\text{PW = Schlüssel}})$$

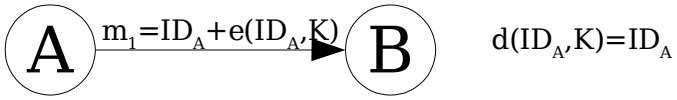


5.6.2.2 Kryptobasierte Authentifizierung (proof by possession)

Will A mit B kommunizieren, muß B die Identität von A verifizieren

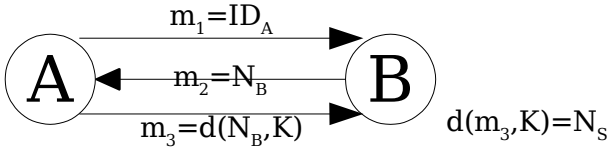
Lösung: kryptographische Operation mit Schlüssel, den beide oder nur B kennt

- symmetrisch A + B
- asymmetrisch A geheim, B öffentlich



Nachteil: Replay-Attacken

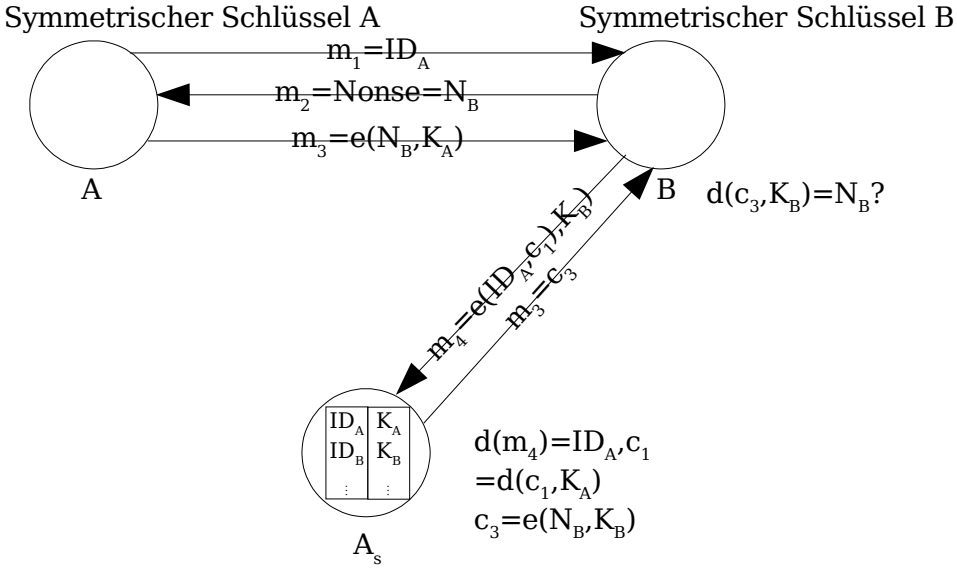
Lösung: Zeitstempel oder Nonse verwenden
Nonse basiertes Challenge Response Protocol



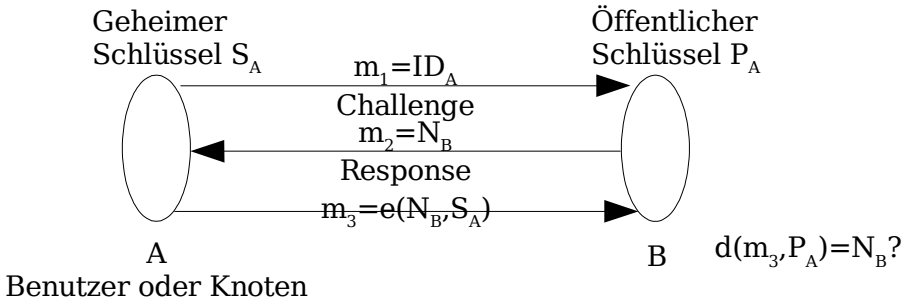
Nachteil: unpraktisch für große Systeme, da alle Benutzer geheimen Schlüssel des anderen speichern muß (Kommen und Gehen)

- unsicher: nicht bei einem vertrauenswürdigen Benutzer, sondern Schlüssel bei allen

Lösung: zentreler A_S



5.6.2.3 Challenge Response Protocol bei asymmetrischen Kryptosystemen zur Authentifikation



Permutation

JAMES BOND

EAJS...

Substitution

Ersetzung von Buchstaben durch PM Vigenere-Chiffrierung

	A	B	C	D	E	F	G	H
A	B	C	G	S	A			
B	N	H	S	D	F			
C	U	R	E					
D								
E								
D								
F								

6 Web-Programmierung

Client: Browser (IE, Mozilla, Netscape, Opera)
vearbeitet: HTML, Grafik
führt aus: JavaScript, Java Applets, ActiveX Controls, über Plugin: Acrobat Reader, Flash

Server: Webserver (Apache, IIS)
erhält Anforderung von Client URL / URI
beantwortet Anforderung (statisch / dynamisch)

Dienst: Verbindungslose Anfrage / Antwort Dienst
Protokoll: HTTP

Weblients Emulation über Telnet:

telnet www.fh-ingolstadt.de 80
GET index.html HTTP/1.0
(2x Return drücken)

6.1 HTTP-Protokoll

6.1.1 Format eines HTTP-Requests (Anfrage v. Client)

Methode URI HTTP-Version

Header Felder
Body

GET: Parameter bei URI
POST: Parameter im Body

Vorteile Parameter über POST:

– Parameter nicht sichtbar

– Parameter unterliegen nicht der URI 255-Längenrestriktion

Headerfelder

Accepted language: de
if-modified-since-Datum (nur Antwort senden wenn seit Datum modifiziert)
Referer: Falls über Links auf Seite ⇒ hier die URI der Ausgangsseite

6.1.2 Format einer HTTP-Response

HTTP-Version	Status-Code	Reason-Phrase
Header-Felder	2xx	Erfolgreich (meist 200 OK)
MessageBody	3xx	Redirection
	4xx	Fehler (404 Seite nicht gefunden)
	5xx	Serverfehler

Header-Felder	Datum als Antwort
Content-Length	Länge des Message Bodys
Content-Type	(z.B. text/html, text/xml, x-application/pdf)

Beispiel HTTP-Response:

HTTP/1.0 201 OK
Content-Type: text/html
Date: Sat, 24 Apr 2004 20:43:07
<html>
<head>
<meta ...
...
</html>

Beispiel: HTTP-Request

GET http://www.google.de/search?q=netzwerkprogrammierung, Socket
HTTP/1.0

Accept: image/gif, image/jpg, application/msword, */*

Accept Language: de

User-Agent: Mozilla/5.0

Host: www.google.de

6.2 HTML

(meist verwendete Formatierung für Web-Dokumente)

- Formatierungsanweisungen
- Einbinden von Bildern, Applets, ...
- Verlinken anderer Web-Dokumente

<html>

<head>

<title>Super Seite</title>

</head>

<body>

<h1>Hallo</h1>

Text

<img src="bild.png">

Noch tollere Seite

</body>

</html>

Der Browser kann nebem URLs auch Daten an den Server schicken.

Diese werden in HTML-Formulare eingegeben

<form name="Formular" action="verarbeite.pl" method="post">
Artikelnummer <input name="artikel" size=11>
...
<input type="submit" value="Bestellen">
</form>

6.3 Trennung von Daten und Formatierung

6.3.1 CSS

Ansatzweise kann diese Trennung mittels CSS (Cascading Style Sheets) geschehen. Hier kann im HTML-Header oder in einer separaten Datei (*.css) die Formatierung bestimmter HTML-Tags definiert werden.

Alle Texte rot:

p { color: red; }

6.3.2 XML + XSLT

XML (eXtensible Markup Language) operiert wie HTML mit Tags und Text. Die Daten bilden einen Baum.

Beispiel:

```
<kunde name="Meier" knr="112">                                  Element
      <adresse typ="lieferant" strasse="Beispielweg 10" ...>    Kinder
      </adresse>
      Beschreibender Text
</kunde>
```

XML

1997 von W3C veröffentlicht

strukturierter Texte

Möglichkeit der Darstellung XML von Server auf Client:

- Browser, der darstellen kann
- Umwandlung auf Server von XML → HTML (mit XSLT)
- (DTD == Datei mit Beschreibung von XML-Datei)

DTD (Datatype Definition)

beschreibt strukturellen Aufbau und die logischen Elemente einer Klasse von Dokumenten (Dokumententyp)

XML-Editor verwendet diese als Vorlage bzw. Muster, um eine gültige XML-Instanz zu erzeugen.

Beispiel: Email DTD

```
<!- E-Mail-DTD-Version ->
<!RLRMRNT email(empfaenger,absender,betreff,nachricht>
<!RLRMRNT empfaenger(#PCDATA)>                          parsed character Data
<!RLRMRNT absender(#PCDATA)>
<!RLRMRNT betreff(#PCDATA)>
<!RLRMRNT nachricht(#PCDATA)>
```

Daraus generierte XML-Instanz:

```
<!xml version="1.0" encoding="ISO-8859-1"!>
<!DOCTYPE email SYSTRM „email.dtd">
<email>
      <empfaenger>Herbert Mustermensch</empfaenger>
      <absender>Jemand Anders</absender>
      <betreff>Ganz wichtige Mail</betreff>
      <nachricht>
            Hallo Herbert,
            ich muß Dir nachher unbedingt was ...
      </nachricht>
</email>
```

XSLT (extensible Stylesheet Language Transformation)

z.B. zur serverseitigen XML-Umwandlung nach XHTML (XML-konformes HTML)

```
<?XML version="1.0" encoding="ISO-8859-1"?>
<?XSL:stylesheet XMLns:xsl=http://www.w3.org/1999/XSL/Transform version 1.0>
<XSL:template match="email">
```

```
<html>
<head><title>E-Mail aus XML</title></head>
<body>
<table border=1>
<XSL:apply-templates/>  //hier werden folgend beschriebene Templates eingefügt
</table>
</body>
</html>
</XSL:template>

<xsl:template match="absender">
      <tr><th>Absender</th>
            <td><xsl:apply templates/= c/td>
      </tr>
</xsl:template>
.
.
.
</xsl:stylesheet>
```

6.4 Aktive Inhalte

1. Javascript

clientseitig für Validierung Benutzereingaben

- dynamische Anzeigen
- Zeit
- Maus (Mouseover)
- Ticker

Entweder in HTML Seite eingebettet oder als externe .js-Datei

```
<script language="JavaScript">
.
.
.
</script>
```

2. Applets

(Gegenstück zu Servlets, die auf Webserver ausgeführt werden)

Plugins (z.B. Mediaplayer, Flashanimation werden in <object>-Tag in HTML eingebunden) und Applets (lädt vom Server eine Java-Byte-Code-Datei und führt sie am Client aus: Einbindung in HTML: <APPLET Code = „HelloWorld.class" width = 150 height = 25></applet>) sehr ähnlich.

In Klasse java.applet.Applet sind wichtige Methoden enthalten:

init() Applet ist geladen
start()
stop()
destroy()

Applets können Netzverbindungen nur zu dem Server, von dem Applet geladen wurde, herstellen, jedoch kann neben HTTP auch andere Protokolle

implementiert werden wie TCP-Sockets, RMI oder RCP (Kontakt zu CORBA-Anwendung).

6.5 Serverseitige dynamische Webseiten

1. CGI (Common Gateway Interface)

Aufruf über URL mit möglichen Parametern

```
www.server.org/cgi-bin/ein.cgi?name="fred"
```

oder über FORM-Tag in HTML

```
<FORM name="Formular1" action="../cgi-bin/email.cgi method="POST">
```

bei jedem CGI-Aufruf wird ein Prozeß gestartet

2. Serverseitige Skriptsprachen

PHP, PEARL, ASP, ... laufen im Prozeß des Webservers

Beispiel PHP eingebettet in HTML_Seite

```
<?PHP
```

3. Java-Servlets

Bei Java-Servlets gibt der Server den Request an ein Java-Programm (Servlet) der den Response (HTML) generiert (über HTTP), Beispiel Tomcat

Aufruf:

a) <SRMVLRT>-Tag in HTML-File

b) Client fordert in einer URL ein Servlet-File an, dessen Server das Servlet zur Ausführung bringt.

c) die Seite bildet selbst URLs auf Servlets ab, die über die Klasse javax.servlet.* insbesondere HTTP Servlet Zugriff auf Request und Response

4. Java Server Page (JSP)

Während Servlets HTML Zeile für Zeile über print Statements generiert, wird bei JSP Java in das HTML integriert.

Hierfür gibt es verschiedene Auszeichnungen

Definitionen: <%! ... %>

Ausdrücke: <%= ... %>

Anweisungen:
```
<% if (cond1) {%>
        html1
<%} else {%>
        html2
<%} %>
```

<% out println („Hello World“); %>
2+2=<% 2+2 %>

JSP-Gegenstück von MS ist ASP

Dynamische Serverseiten können auch JavaScript enthalten (server- und clientseitige Ausführung).

7 Sicherheit in VS

Extrem wichtig:

Gefahr:

- beobachten (passiver Angreifer)
- aktiver Angreifer: modifizieren, stören (DOS), erzeugen

Sicherheitsmechanismen dagegen:

1. Verschlüsselung
2. Authentifizierung (Überprüfen der Identität)
3. Authorisierung (Berechtigung für Aktion gegeben?)
4. Auditing
   (Aufzeichnung, Mitteilung von verdächtigen Aktionen)

7.1 Verschlüsselung (Kryptographie)

Daten bleiben auch bei Abhören (passiver Angreifer) vertraulich durch Ver- und Entschlüsselung der Daten

Beide Knoten müssen sich um einen gemeinsamen Schlüssel einigen, um verschlüsselte Nachrichten austauschen zu können.

Notwendig:

- Schlüsselverteilungssystem (SVS)
- Authentifikation des Kommunikationspartners (Echtheit und Identität)
  Beispiel: Kerberos (verwendet von Microsoft)

Nachricht = Klartext = message = m
verschlüsselter Text = Chiffretext = ciphertext = c
Verschlüsselungsverfahren = Chiffre
verschlüsseln = encrypt = e
entschlüsseln = dedrypt = e
Schlüssel zur Verschlüsselung = encryption key = K_e
Schlüssel zur Entschlüsselung = decryption key = K_d
Verschlüsselung (Chiffrierung) = $e(m, K_e)=c$
Entschlüsseln (Dechiffrierung) = $d(c, K_d)=m$

Attacken von Angreifern, die nur die verschlüsselte Nachricht lesen

- Falls Klartext nicht bekannt: Ciphertext-Only-Attacke
- Falls Klartext bekannt: Known-Plaintext-Attacke

– kann Angreifer selbst Klartext wählen: Chosen-Plaintext-Attacke