Fragenkatalog zur Fachprüfung 1678 „Verteilte Systeme“ BSc. Informatik

Stand Mai 2006

Dieser Fragenkatalog ist eine Zusammenfassung der Prüfungsprotokolle von StudienkollegInnen aus den Prüfungen bei Prof. Icking vom 24.2.06, 12.5.05, 15.2.05, 4.10.05, 16.3.04, 19.12.01 und bei Prof. Haake vom 16.2.05. Dabei handelt es sich um Bachelor Fachprüfungen und Diplomprüfungen.
Jene Fragen, die nur von Prof. Haake sind, sind mit [Haa] gekennzeichnet.

Keine Gewähr auf Richtigkeit.

Fehlermeldungen an: wzeindl@gmail.com

Ressource für Abbildungen aus dem Buch „Verteilte Systeme – Konzepte u. Design“ 3.Auflage von Coulouris, Dollimore u. Kindberg: http://www.cdk3.net/ig/

Inhaltsverzeichnis

1. Definition verteiltes System: [4x]

Ein verteiltes System ist ein System, in dem sich HW- od. SW-Komponenten auf vernetzen Computern befinden und nur über den Austausch von Nachrichten kommunizieren und ihre Aktionen koordinieren. Aus dieser Def. Ergeben sich folgende Konsequenzen:

- *Nebenläufigkeit der Programmausführung*.
- *Keine globale Uhr* -> es gibt kein globales Konzept einer genauen Uhr *(da Kommunikation ausschließlich über das Senden von Nachrichten erfolgt).*
- *Unabhängige Ausfälle* -> jede Komponente des Systems kann unabhängig von den anderen ausfallen, während die anderen weiterhin funktionieren *(und womöglich lange od. überhaupt nie etwas davon merken).*

Ziel ist die gemeinsame Nutzung von Ressourcen (HW u. SW).

2. Welche Alternative zu verteilten Systemen gibt es?: [2x]

Der zentrale Großrechner der bspw. Terminals bedient.

3. Was sind denn die Eigenschaften verteilter Systeme und was deren Vor- und Nachteile?: [3x]

Eigenschaften siehe 1.

Vorteile (gegenüber Großrechner und unabhängigen PCs):

- Wirtschaftlichkeit (gegenüber Großrechner)
- Geschwindigkeit (gegenüber PC)
- Verteiltheit - dadurch sind u.U. natürlichere Problemlösungen möglich
- Zuverlässigkeit durch Redundanz
- Einfache Skalierbarkeit

Nachteile:

- Komplexere Software
- Kommunikationsprobleme sind möglich - dadurch schlechtere Performance, Verlust von Nachrichten
- Nur schwache Schutzvorkehrungen - Kompromiss zwischen Sicherheit und Einfachheit des Zugriffs auf Ressourcen
- Keine zentrale Wartbarkeit,

3. Skalierbarkeit, was ist das?: [1x]

Ein System das skalierbar ist, bleibt auch dann effektiv, wenn die Anzahl der Ressourcen und die Anzahl der Benutzer wesentlich steigt. Beim Entwurf skalierbarer verteilter System ergeben sich folgende Problemstellungen:

- *Kostenkontrolle für die physischen Ressourcen* -> ein System sollte zu vertretbaren Kosten erweiterbar sein, um steigendem Bedarf nachzukommen.
- *Kontrolle des Leistungsverlustes*
- *Leistungsengpässe vermeiden* -> Dezentralisieren von Algorithmen, z.B. Aufteilung der Namenstabellen des DNS auf verschieden Server.

4. Wie sieht es mit der Sicherheit aus?: [1x]

Verteilte Systeme bieten mehr Angriffspunkte, da es Schnittstellen nach außen zur Verfügung stellen muss. Die Sicherheit für Informationsressourcen kann dabei durch sichere Kommunikationsverbindungen (Verschlüsselung -> siehe Kurs 1866) gewährleistet werden, wobei diese Sicherheit 3 Bereiche betrifft:

- *Vertraulichkeit* – Schutz gegen Offenlegung gegenüber nicht berechtigten Personen
- *Integrität* – Schutz gegen Veränderung oder Beschädigung
- *Verfügbarkeit* – Schutz gegen Störungen der Methoden zum Ressourcenzugriff

Folgende Sicherheitsaspekte werden dabei aber nicht vollständig berücksichtigt:

- *DoS Angriffe*
- *Sicherheit mobilen Codes* (Virenproblematik)

5. Transparenz, was ist das? Was ist Orts- und Fehlertransparenz?: [1x]

Ein System, dass seine Verteiltheit vollständig vor dem Benutzer verbirgt und wie ein Einprozessorsystem erscheint, heißt transparent da die Implementierung für den Benutzer durchsichtig - im Sinne von unsichtbar - ist. Man unterscheidet (nach *ANSA – Advanced Network Systems Architecture*):

- *Ortstransparenz* – erlaubt den Zugriff auf die Ressourcen, ohne dass man ihre Position/Ort kennt.
- *Fehlertransparenz* – erlaubt das Verbergen von Fehlern, sodass Benutzer ihre Aufgaben erledigen können, auch wenn HW- od. SW-Komponenten ausgefallen sind.
- *Zugriffstransparenz* – ermöglicht den Zugriff auf lokale und entfernte Ressourcen unter Verwendung identischer Operatoren.
- Nebenläufigkeitstransparenz – erlaubt, dass mehrere Prozesse gleichzeitig mit den selben gemeinsam genutzten Ressourcen arbeiten, ohne sich gegenseitig zu stören.
- *Replikationstransparenz* – erlaubt, dass mehrere Instanzen von Ressourcen verwendet werden, um die Zuverlässigkeit und die Leistung zu verbessern, ohne dass die Benutzer wissen, dass Repliken verwendet werden.
- *Mobilitätstransparenz* – erlaubt das Verschieben von Ressourcen und Clients innerhalb eines Systems, ohne dass die Arbeit von Benutzern beeinträchtigt wird.
- *Leistungstransparenz* – erlaubt, dass das System neu konfiguriert wird, um die Leistung zu verbessern wenn die Last variiert.
- *Skalierungstransparenz* – erlaubt, dass sich das System und Applikationen vergrößern, ohne dass die Systemstruktur oder die Applikationsalgorithmen geändert werden müssen.

6. Wie erfolgt die Kommunikation in verteilten Systemen? Was wurde dazu definiert?: [1x]

Die Kommunikation erfolgt über Netzwerk/Internet, und zwar über Nachrichtenaustausch. Es werden dabei vorhandene standardisierte Netzwerkprotokolle verwendet, die auf dem OSI Schichtenmodell basieren.

7. Wie sieht das ISO/OSI Schichtenmodell aus?: [5x]

1. Bitübertragungsschicht – physical layer:

Bsp.: Ethernet-Basisband-Signale, ISDN

Stellt phys. Übertragungskanäle zur Verfügung, die es gestatten, beliebige Bitfolgen zu übertragen. Von der anfordernden Schicht können Anforderungen an den Übertragungskanal wie Fehlerrate, Verfügbarkeit, Übertragungsgeschwindigkeit und –verzögerung vorgegeben werden.

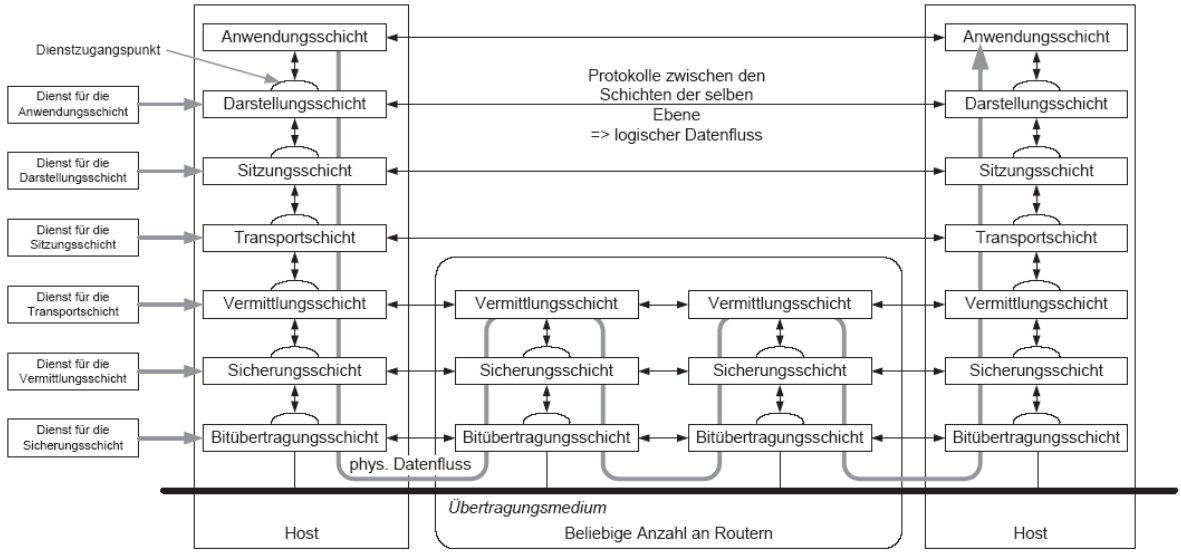
Netzwerkschnittstellenschicht fasst Sicherungs- u. Bitübertragungsschicht zusammen, wird auch *Host-an-Netz-Schicht* genannt.

2. Sicherungsschicht – data link layer:
Bsp.: Ethernet MAC, ATM-Zellenübertragung, PPP
Stellt weitgehende sichere Übertragungskanäle für die Übertragung von Datenblöcken zur Verfügung. Dafür werden Fehlererkennungs-/korrektur-Verfahren eingesetzt.
3. Vermittlungsschicht – network layer:
Bsp.: IP, ATM, virtuelle Kanäle
Stellt logische Übertragungskanäle zw. Endsystemen zur Verfügung. Bsp. IP. Bietet der Transportschicht Funktionen zur Flusskontrolle an.
4. Transportschicht – transport layer:
Bsp.: TCP, UDP
Erweitert die Dienste der Vermittlungsschicht um Wiederanordnung, Güteanforderungen *(Verzögerungszeiten, Fehlerrate, Durchsatz, Sicherheit)* u. Priorität. Stellt logische Übertragungskanäle zwischen aktiven, kommunizierenden Prozessen auf Endsystemen zur Verfügung. Bsp. sind TCP und UDP.
5. Anwendungsschicht:
Unterstützt Netzwerkanwendungen wie z.B. HTTP, FTP, SMTP für e-Mail usw.

Im ***ISO/OSI-Referenzmodell*** sind die unteren 4 Schichten ident zum Internet-Schichtenmodell. Die Anwendungsschicht ist aber noch mal detaillierter aufgegliedert:

- ***5. Kommunikationssteuerungs- od. Sitzungsschicht (session layer)*** – Stellt Dienste zur Verfügung, die es kommunizierenden Prozessen erlaubt, ihren Dialog zu kontrollieren und zu synchronisieren.
- ***6. Darstellungsschicht (presentation layer)*** – Stellt Dienste zur Verfügung, die die Darstellung von Daten in unterschiedlichen Repräsentationen umwandeln. (*Bsp.: SSL, CORBA*)
- ***7. Anwendungsschicht (application layer)*** implementiert die eigentliche Anwendungsfunktionalität. (*Bsp.: HTTP, FTP, SMTP, CORBA IIOP*)

Das ISO/OSI-Modell bildet ein *Rahmenwerk zur Implementierung von Protokollen*.



8. Wie kommunizieren die Schichten miteinander?: [2x]

Der logische Datenfluss passiert zwischen den Schichten der selben Ebene, für den physikalischen Datenfluss verwenden alle Schichten (außer der Bitübertragungsschicht) die Dienste der jeweils darunter liegenden Schicht.

9. Netzwerkschicht und Transportschicht: welche verbindungslose und welche verbindungsorientierte Protokolle gibt es? [1x]

Netzwerkschicht: besteht aus 3 Komponenten:

- IP-Protokoll (IPv4) – definiert Adressierung, Felder eines Datagramms (3-PDU) und Aktionen auf Datagramm
- Pfadbestimmungskomponente – bestimmt Pfad eines Datagramms
- ICMP-Protokoll (Internet Control Message Protokoll) – erlaubt Fehler in Datagrammen anzuzeigen und Infos über Vermittlungsschicht abzufragen

Dienstmodell = *verbindungsloser Best-Effort-Dienst*:

- Pakete können verloren gehen
- Paketreihenfolge nicht garantiert
- zeitlicher Abstand zw. Paketversand u. –empfang nicht definiert
- keine Zusagen über Bandbreite
- keine Information bzgl. Netz-Überlast an Empfänger-/Senderprozess

(In einem VC-Netz wird ein verbindungsorientierter Dienst realisiert.)

Transportschicht:

- TCP – verbindungsorientiert
- UDP – verbindungslos

10. Unterschied zwischen TCP und UDP? [2x]

UDP	TCP
unzuverlässige Übertragung	zuverlässige, bidirektionale Übertragung
verbindungslos => keine initiale Verzögerung	verbindungsorientiert => initiale Verzögerung
speichert keinen Verbindungszustand => einfacher Implementierung, Serverprozess kann mehr Clients bedienen	speichert Verbindungszustand => Server kann weniger Clients als UDP bedienen
Header nur 8 Bytes => pro Segment können 12 Bytes mehr als bei TCP gesendet werden	20 Byte Header => verbraucht mehr Bandbreite
keine Fluss- oder Überlastkontrolle => Anwendung kann so viele Daten/Zeiteinheit übertragen wie sie generieren kann und die Bandbreite der Verbindung zulässt	Flusskontrolle kann Sender am verschicken der Daten hindern, ebenso Überlastkontrolle => schlecht für Echtzeitanwendungen
Multicast-fähig	nicht Multicast-fähig
unterstützt Vollduplex ??	unterstützt Vollduplex
Anwendungen auf UDP:	*Anwendungen auf TCP:*
Remote File Server (NFS)	E-Mail (SMTP)
Streaming Multimedia	Remote Terminal Access (Telnet)
Internet Telephony	WWW (HTTP)
Network Management Anwendungen (SNMP: Simple Network Management Protocol)	File Transfer (FTP)
Routing Protocol (RIP: Routing Internet Protocol)	
Domain Name System (DNS)	

11. Was macht TCP? [2x]

TCP (Transmission Control Protocol, RFC 793) realisiert einen zuverlässigen Datentransfer zw. 2 Anwendungsprozessen. TCP

- garantiert *Einhaltung* der versendeten *Bytereihenfolge* für Empfänger => *Datenstromvermittlung*
- unterstützt *Vollduplex-Datenübertragung* zwischen genau einem Sender u. Empfänger

- realisiert eine zuverlässige *Punkt-zu-Punkt-Verbindung* zw. genau 2 Anwendungsprozessen
- ist ein *verbindungsorientiertes Protokoll* => vor Datenübertragung muss Verbindung mit Empfänger aufgebaut werden

Verbindungsmanagement

Mit der IP-Adresse u. Portnummer des Destination-Prozesses kann Client-Prozess eine TCP-Verbindung zu Server-Prozess über Sockets aufbauen. Vorgangsweise bei *Verbindungsaufbau* über 3-Weg-Handshake:

1. *Client-TCP* sendet *SYN*-Segment an Server-TCP => keine Anwendungsdaten, aber in Header ist *SYN*-Bit gesetzt und *initiale Sequenznummer (client-isn)* enthalten
2. *Server-TCP* erhält *SYN*-Segment => erzeugt neue Verbindung und weist ihr entsprechende Puffer u. Variable zu => sendet *SYNACK*-Segment an Client-TCP => keine Anwendungsdaten, *Acknowledge-Feld* auf *client-isn+1* und *Sequenznummer-Feld* auf *initiale server-isn* gesetzt
3. *Client-TCP* erhält *SYNACK*-Segment => erzeugt ebenfals Verbindung inkl. Puffer u. Variable => sende *connection-ACK* mit *SYN*-Bit=0 u. *server-isn+1* an Server-TCP

Abbau der Verbindung durch Client-TCP:

1. *Client-TCP* sendet *FIN*-Segment an Server-TCP
2. *Server-TCP* bestätigt mit *ACK*-Segment
3. Dann schickt *Server-TCP* eigenes *FIN*-Segment an Client-TCP
4. *Client-TCP* empfängt *FIN*-Segment => sendet *ACK*-Segment an Server-TCP => wartet Timeout ab *(damit ACK wiederholt werden kann falls es verloren ging)* löscht Verbindung

Datentransfer in TCP

In IPv4 können in einem Segment max. 64kB übertragen werden. Übliche Werte sind aber 1500, 536 od. 512 Byte. Daten inkl. TCP-Header werden über Vermittlungsschicht (IP) übertragen. Sobald Daten im Sendepuffer sind, werden diese zu einem implemtationsabhängigen Zeitpunkt versendet.

Sequenz- u. Bestätigunsnummern

Da TCP einen Datenstrom realisiert, kann jedes Byte im Bytestrom mit einer Bytestromnummer nummeriert werden. Die Sequenznummer eines Segments ist dann die Bytestromnummer des ersten Bytes im Segment. Das Server-TCP verwendet nun als Sequenznummer für die Bestätigung die Bytestromnummer des nächsten erwarteten Bytes. TCP verwendet kumulative Bestätigung der bisher korrekt empfangen Bytes. Daten die nach einer Lücke „zu früh“ eintreffen werden entweder gepuffert oder verworfen u. dann neu angefordert.

12. Wie funktioniert das Fehlermodell bei TCP? [1x]

Fehlerkorrektur

TCP fügt zu jedem Segment eine Checksumme hinzu und überträgt verfälschte Daten neu.

Zuverlässiger Datentransfer in TCP

Dafür werden viele der Prinzipien aus 3.4.4 eingesetzt. Der Senderprozess behandelt 3 Fälle:

1. Für jedes Segment wird beim versenden ein Timer gesetzt.
2. Wenn ein Timer abläuft, wird die Übertragung des Segments wiederholt.
3. Korrekt erhaltenes ACK stoppt Timer. Wird ein selbes ACK zweimal empfangen entspricht dies einem NACK. Nach 3 ACK für das selbe Segment wird das Segment neu übertragen = *Fast Retransmit*, auch wenn der Timer noch nicht abgelaufen ist.

Flusskontrolle

Damit kann Empfänger die Senderate des Senders so verringern, das der Empfangspuffer nicht überläuft. Dazu hat jeder TCP-Prozess eine Empfangsfenster-Variable die anzeigt, wie

viel freier Pufferplatz beim Empfänger vorhanden ist. Beim Senden zieht der Sender die versendete Segmentgröße von dieser Variablen ab. Das aktuelle Empfangsfenster wird mit jedem Segment, das Empfängers an den Sender sendet, mitübertragen (im WINDOW-Feld). Falls Empfänger keine Segmente zu senden hat, muss er ein leeres Segment an den Sender verschicken.

Überlastkontrolle

Bei Netzwerküberlastung wird die Senderate solange verringert, bis keine Kollisionen mehr auftreten. Dann wird die Senderate wieder kontinuierlich solange erhöht.

13. Warum braucht man dann überhaupt UDP? Für welche Art von Anwendungen? [2x]

UDP (User Datagram Protocol, RFC 768) realisiert einen unzuverlässigen Transportdienst zw. Prozessen. Senderprozess versendet Datagramme ohne vorher Verbindung mit Empfänger aufzunehmen:

- *Nachrichten* könne *verloren* gehen
- *Versandreihenfolge* von Nachrichten ist i.A. nicht gleich der Empfangsreihenfolge *(Nachrichtenvermittlung u. nicht Datenstrom !)*
- UDP macht keine Aussage über die *Übertragungsverzögerung*
- Dienstzugangspunkt über *Datagramm-Sockets*
- Simple *Fehlererkennung* über 2 Byte *Checksumme*, berechnet als 1er-Komplement der Summe aller 16bit-Wörter des Segments

speichert keinen Verbindungszustand => einfachere Implementierung, Serverprozess kann mehr Clients bedienen
Header nur 8 Bytes => weniger Overhead, pro Segment können 12 Bytes mehr als bei TCP gesendet werden
keine Fluss- oder Überlastkontrolle => Anwendung kann so viele Daten/Zeiteinheit übertragen wie sie generieren kann und die Bandbreite der Verbindung zulässt -> besser geeignet für Echtzeitanwendungen wie Audio/Video
Multicast-fähig

Anwendungen siehe Frage 10.

14. Wenn Sie ein Internetradio mit vielen Clients betreiben wollen, welches Protokoll würden sie dafür nehmen? [1x] [Haa]

UDP.

Begründung:

- Kein großes Problem wenn Pakete ausgelassen werden.
- Rest siehe Frage 13.

15. Was passiert bei UDP, wenn das Paket nicht ankommt? [1x]

UDP realisiert eine unzuverlässige Verbindung, d.h. es gibt keine Garantie, dass eine versendetes Paket auch wirklich ankommt. Das Protokoll auf der darüber liegenden Schicht muss selbst eine entsprechende Fehlerbehandlung realisieren.

16. Können Pakete auch unterschiedliche Wege nehmen? Kommen sie dann korrekt an? [1x]

Durch adaptives Routing können die Pakete über verschiedene Verbindungen zum Empfänger gelangen und damit i.A. auch out-of-order. Wird TCP als Transportschichtprotokoll

verwendet, so wird durch TCP sichergestellt, dass die empfangenen Pakete an die darüber liegende Schicht wieder in der richtigen Reihenfolge übergeben werden.

17. Wie funktioniert das Sliding Window Protokoll? [1x]

SWP wird von den meisten verbindungsorientierten Protokollen wie *PPP* od. *TCP* eingesetzt. Es beschreibt einen Kommunikationsfluss von Sender zu Empfänger, bei dem beide Seiten mit einem Puffer, einem sog. Sende- und Empfangsfenster ausgestattet sind. Jeder Eintrag in den „Fenstern" ist mit einer Sequenznummer versehen. Der Sender merkt sich die Nummern der Pakete, die noch nicht vom Empfänger bestätigt wurden, wobei das älteste unbestätigte Paket die Untergrenze des Fensters bildet und das zuletzt gesendete Paket die Obergrenze -> Fenstergröße = Obergrenze – Untergrenze +1 darf nicht größer als der Puffer sein, der die gesendeten u. noch nicht bestätigten Pakete enthält.
Bei sinnvoller Wahl der Fenstergrößen, kann ein Datenfluss erreicht werden, der den Kanal möglichst effizient nutzt und trotzdem den Empfänger nicht überlastet -> *Flusskontrolle*.
Ablauf:
Senderfenster enthält Paketnummern, die zum Senden benutzt werden dürfen => Empfangsfenster enthält Paketnummern die Empfänger aktuell empfangen darf. Empfänger bestätigt normalerweise nicht einzelne Pakete sondern Paketfolge = *kumulative Bestätigung*.
Bei Fenstergröße > 1 zusätzliche Eingriffmöglichkeiten bei Fehlern durch:

- Aufforderung des Senders, das fehlerhafte Paket und alle danach bereits gesendeten n-1 Folgepakete zu wiederholen => *go-back-n*
- Aufforderung des Senders, nur das fehlerhafte Paket erneut zu senden (z.B. über NACK mit zugehöriger Sequenznummer) => *selectiv-reject/repeat*

Exkurs:
Zuverlässige Datenübertragung über einen verlustbehafteten Kanal mit Bitfehlern:
Zwei Aufgaben:

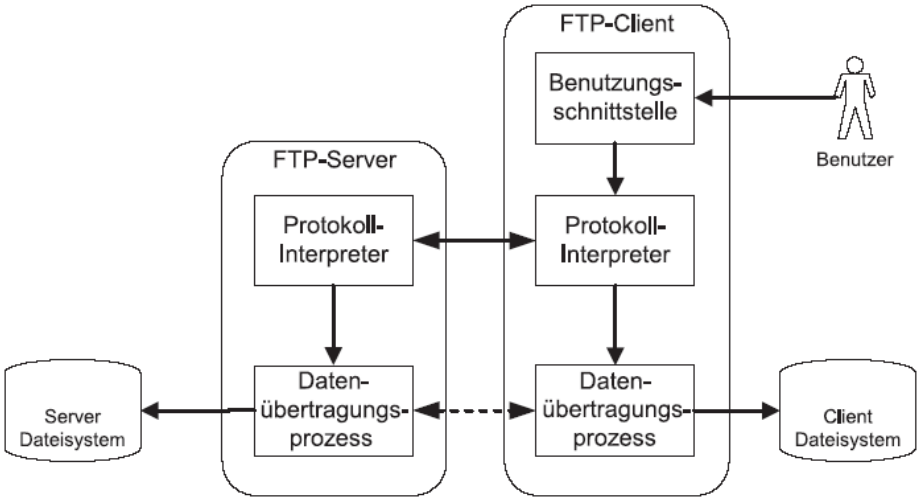
- Erkennung von Paketverlusten
- Behandlung von Paketverlusten *(mit Checksummen, Wiederholung, ACK usw.)*

Erkennung von Paketverlusten erfolgt Senderseitig => trifft eine Empfangsbestätigung nicht innerhalb eines best. Zeitintervalls beim Sender ein, wird ein Paketverlust angenommen => autom. Wiederholung des letzten Paketes => eventuelle Duplikate erkennt Empfänger an Sequenznummer.
Der Sender benötigt Timer und muss in der Lage sein:

- bei jedem gesendeten Paket den Timer zu setzen
- bei Ablauf des Timers das verlorengegangene Paket zu Wiederholen
- bei Erhalt des ACK den Timer zu stoppen

Um festzustellen, zu welchem Paket eine Bestätigung gehört, muss der Empfänger die Sequenznummer des empfangenen Paketes in ein Feld der zugehörigen Bestätigung schreiben.

18. FTP als Protokoll: wie funktioniert es, welcher Verbindungsaufbau findet statt? [1x]



FTP nutzt TCP – *Steuerverbindung* über eigenen Port (21) während gesamter Sitzung aufrecht, Steuerinformationen „*out-of-Band*" – *Datenverbindung* (default Port 20) nur während aktiver Übertragung – Protokoll zustandsbasierend *(merkt sich Zustand seiner Sitzungen)* .

Auf Client und Server existieren jeweils 2 Prozesse:

- *Datentransferprozess*: zur Übertragung von Dateien und Verzeichnisinformationen – wird vom Server bei jeder Übertragung auf- und danach abgebaut.
- *Protokoll-Interpreter*: für die Übertragung (im Klartext!) von Kommandos und Kontrollinformationen, bleibt während der gesamten Sitzung geöffnet. Damit können auch noch während der Dateiübertragung Steuerkommandos (z.B. Abbruch) absetzen.

FTP kennt auf dem Server Dateien und Verzeichnisse, zus. Informationen (Dateirechte) werden vom Protokoll zwar nicht abgedeckt, können aber im Server implementiert werden.

Verbindungsaufbau:
Ein Client meldet sich beim Server an, indem er die Kontrollverbindung zum Server öffnet. Danach wird USER (u. ev. PASSWORD) an den Server übertragen und damit das „Login" abgeschlossen. Bei USER=anonymuous kann ein beliebiges PASSWORD angegeben werden (z.B.: e-mail Adr.) Nun kann der Client sich durch das Dateisystem am Server bewegen, Verzeichnisse/Dateien löschen/anlegen bzw. up-/downloaden usw. Vor der Datenübertragung kann der Client auch bei Bedarf das Datenverbindungsport und die Übertragungscodierung der Daten festlegen.
Alle Befehle werden vom FTP-Server mit einer Antwortnachricht quittiert, die einen 3-stelligen Zahlencode für die Ergebnisklassen und eine Klartextbeschreibung enthält.

19. FTP: welche Authentifizierung gibt es, warum ist FTP unsicher, welche Gründe sprechen dafür es trotzdem zu verwenden? Wie funktioniert anonymuous FTP? [1x]

Authentifizierung über Username und Password, allerdings unverschlüsselt! Trotz der mangelnden Sicherheit ist es neben historischen Gründen auch wegen seiner Plattformunabhängigkeit und der Verwendung von Klartextnachrichten sehr weit verbreitet.
Anonymuous FTP siehe Frage 18.

20. HTTP als Protokoll: welche Möglichkeiten hat man, welche Daten kann man sich damit anschauen? [1x]

HTTP nutzt TCP - Objektadressierung über URL (Uniform Resource Locator). Namensschema **<protocol-id><protocol specific address>** - nicht-persistente (HTTP 1.0) und persistente Verbindungen (HTTP 1.1) möglich – Timout zum Schließen unbenutzter Verbindungen – persistente Verbindungen mit/ohne Pipelining – sendet Steuerinformationen „*in-Band*“ *(im Header der Daten)* – zustandsloses „*pull*“-Protokoll – jedes Objekt einer Website wird einzeln angefordert.
HTTP wird benutzt, um Dateiinhalte von einem Webserver auf den lokalen Rechner zu übertragen. HTTP-Requests bestehen aus einer Kommandozeile, Nachrichtenheader und Nachrichtenkörper. Es bietet nur sehr eingeschränkte Möglichkeiten die sich in der Praxis auf das Herunterladen von Objekten von einem Server beschränken, da die meisten Webserver Befehle wie PUT od. DELETE nicht unterstützen. Es bietet also keine Möglichkeit

- Verzeichnisse aufzulisten
- Dateien am Server zu verändern
- für eine explizite Modellierung von Besitzern von Dateien

-> kein echtes verteiltes Dateisystem!
Es sind mehrere Caching-Strategien vorgesehen, und zwar

- ein Server-Cache, wenn sehr viele Clients immer wieder die gleichen Daten anfordern
- ein Cache in Form eines Proxy in Intranets, um den externen Verkehr zu reduzieren
- ein Client-Cache der vom Browser verwaltet wird

Zur Unterstützung der Konsistenzwahrung wird in den letzten beiden Varianten von HTTP ein *Expires*-Feld unterstützt, in dem der Server einträgt, ab wann die Server-Antwort veraltet ist.

21. Welche Middleware haben sie im Kurs kennen gelernt? [1x] [Haa]

CORBA, Java RMI, SOAP und RPC.
Middleware ist eine SW-Schicht, die eine Programmierabstraktion darstellt und die Heterogenität der zugrunde liegenden HW, Netzwerke, OS und Programmiersprachen verbirgt. Die meiste Middleware wird über Internet-Protokolle implementiert.

22. Was ist SOAP? [3x]

Simple Object Access Protocol. Definiert ein Datenaustauschformat auf XML-Basis und die Anbindung an verschiedene, darunter liegende Kommunikationsprotokolle wie HTTP. Damit sollen folgende Probleme gelöst werden:

- Abbildung der Dateirepräsentation auf ein plattformunabhängiges Zwischenformat (Marshalling).
- Unterstützung der Adressierung der Kommunikationspartner (Port und Host) u. ev. Verbergen vor Programmierer
- Zugangspunkte für die Kommunikation auch bei Systemen mit Firewall

Wird SOAP über HTTP implementiert, so werden SOAP-Anfragen als besondere HTTP-Anfragen mit einem zusätzlichen Header-Feld abgesetzt und der nachfolgenden SOAP-XML-Nutzlast (SOAP-Envelope wieder bestehend aus Header und Body). Die Verwendung von HTTP hat den Vorteil das damit das Port 80 adressiert wird, das von den meisten Firewalls freigeschaltet ist. Der Anfrageninhalt in Form von XML-Daten hat den Vorteil der Plattformunabhängigkeit und dass XML-Daten üblicherweise auch von Firewalls durchgelassen werden.

23. Warum ist Marshalling nötig? [1x] [Haa]

Informationen werden innerhalb von Programmen als Datenstrukturen dargestellt. Um diese übertragen zu können müssen sie in ein geeignetes, plattformunabhängiges Format für die Netzwerkübertragung zu bringen. Dabei ist zu beachten, dass die kommunizierenden Systeme eventuell unterschiedliche Datenstrukturen besitzen (z. B. wegen unterschiedlicher Darstellung von Gleitkommazahlen, little/big-endian-Problemen, Zeichensätzen). Es gibt zwei Möglichkeiten dieses Problem zu beheben:

- Die Systeme die Daten werden in ein externes Format gewandelt – übertragen und dann beim Empfänger in ein von ihm benutztes Format zurückgewandelt. Ein aktuelles standardisiertes Datenformat ist *XML (eXtensible Markup Language)*. Welche Form die folgenden Daten haben teilen sie Prozesse über *DTD (Document Type Definition)* mit.
- Die Werte werden im Format des Senders übertragen, zusammen mit einer Angabe, über die Art der Daten.

CORBA und die Objektserialisierung in Java führen ein Marshalling in binäres Format durch. Hier wird das Marshalling von einer Middleware ausgeführt wodurch sich der Programmierer nicht mehr darum zu kümmern braucht. HTTP ist ein Beispiel für ein Marshalling in ASCII Text.

24. Wie macht man das bei Objekten, die zusätzlich noch Zeiger auf andere Inhalte enthalten (z.B. bei Java RMI)? [1x] [Haa]

Wird ein Objekt serialisiert, das Verweise auf andere Objekte hat, werden alle entsprechenden Objekte ebenfalls serialisiert. Jedem serialisiertem Objekt wird ein eindeutiger *Handle* zugewiesen. Hat ein weiteres Objekt ebenfalls einen Verweis auf das gleiche Objekt wird nur noch der Handle mit übergeben. Das Objekt wird also nur einmal serialisiert.

25. Firewalls allgemein: Welche Architekturen gibt es? [1x]

setzt sich aus mehreren Prozessen zusammen, die auf unterschiedlichen Protokollebenen arbeiten; überwacht Kommunikation zwischen Intranet und Außenwelt: Dienstkontrolle (erlaubt nur Zugriff auf definierte Dienste und weist alles andere zurück), Verhaltenskontrolle (Aussortieren von Spam, Reglementieren von 4 Nutzern), Benutzerkontrolle (verschiedene Benutzerrechte, VPN);
Strategien dafür:
IP-Paketfilterung (entscheidet anhand von Quell- und Zieladresse, Servicefeld, Portnummer...), TCP-Gateway (TCP-Segmente werden geprüft), Gateway auf Applikationsebene (agiert als Proxy für Telnet, FTP...)

Architekturen:

Packet filtering router:

Ist i.d.R. ein speziell eingerichteter Router, der für jedes IP-Paket entscheidet, ob es passieren darf oder nicht. Dafür wird anhand der Informationen aus dem IP-Paket-Header (z.B. IP-Adresse od. Port-Nr. des Senders/Empfängers) aus einer Tabelle mit Regeln eine Regel ausgewählt. Damit kann die Firewall bestimmte Dienste oder Adressen komplett sperren. Damit die Filterung schnell arbeitet, wird die Regeltabelle sequentiell abgearbeitet und die erste passende Regel angewandt. Deshalb sollten die spezielleren Regeln am Anfang der Liste stehen und dann erst die allgemeineren.

Nachteile:

- Mit der Anzahl der Filterregeln sinkt damit auch der Durchsatz des Routers.

- Ein Paketfilter erstellt i.d.R. keine Protokolle, Angriffe um den Filter zu überlisten sind also nur schwer erkennbar.
- Es können keine benutzerbezogenen Regeln erstellt werden, da im IP-Header keine Informationen über Benutzer enthalten sind.
- Die Filterentscheidung erfolgt auf Basis einzelner IP-Pakete, es kann also z.B. für einer ftp-Sitzung nicht einstellen, dass nur bestimmte Verzeichnisse erlaubt sind.
- Eingehende Bestätigungspakete (SYN-ACK, ACK) können nicht geprüft werden, ob sie tatsächlich Antworten auf einen Verbindungs-Request sind oder von einem Angreifer generiert wurden.
- Es kann auch keine Adressumsetzung durchgeführt werden, da der Verbindungszustand vom Packet-Filter nicht gespeichert wird.

Vorteile:

- Transparent für den Anwender.
- Keine zusätzliche SW auf den internen Rechnern.
- Moderne Router besitzen bereits eingebaute Packet-Filter.

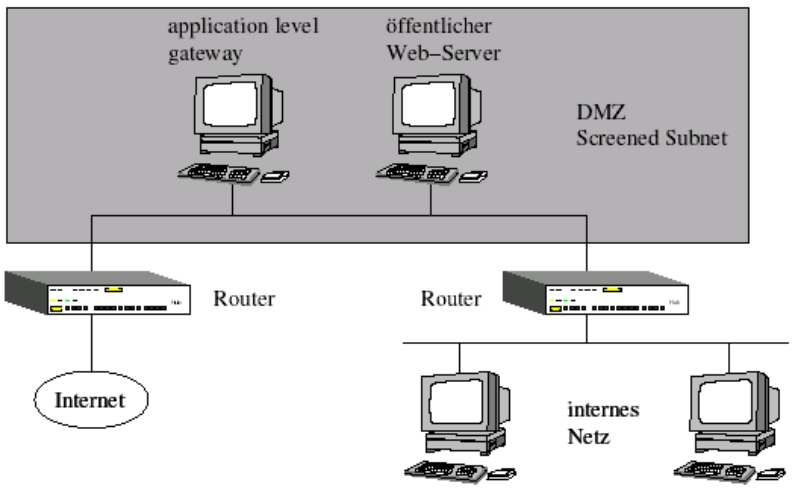
Stateful inspection filter:
ist ein erweiterter Packet-Filter mit einem lokalen Gedächtnis über den aktuellen Verbindungszustand.

Application level gateway:
arbeitet auf der Ebene der Anwendungsprotokolle. Ein *Proxy*-Server läuft auf dem Gateway und fungiert als Stellvertreter zwischen Client und dem Server.

- Er baut die Verbindung nach innen auf und ist der Kommunikationspartner für Benutzer von außen.
- Er übernimmt auch Identifikation/Authentifikation der Benutzer die eine Verbindung aufbauen wollen.
- Der *Proxy* kann somit für eine Filterung der Kommunikation auf Anwendungsebene eingesetzt werden.
- Ein HTTP-*Proxy* cached auch Web-Sites und reduziert damit die Netzlast.
- Ein HTTP-*Proxy* anonymisiert den internen Benutzer gegenüber dem Internet.

Für jeden Dienst wird ein eigener *Proxy* eingerichtet, womit die Kommunikation nicht nur gefiltert sondern auch protokolliert werden kann. Dafür muss aber das IP-Routing auf dem Gateway deaktiviert werden.
Single homed: eine Netzwerkkarte, nur sinnvoll in Verbindung mit einem packet filter router.
Dual homed: zwei Netzwerkkarten.

Screened subnet:
Bezeichnet ein eigenes Teilnetz, das zwischen dem internen Netz und dem Internet liegt und an beiden Enden durch einen Filter geschützt ist: **Demilitarized Zone – DMZ**.



Die Router sind so konfiguriert, dass alle Paket über die Rechner der *DMZ* laufen müssen.
Vorteile:

- Es erfolgt eine Filterung auf Paketebene und auf Anwendungsebene
- Die Architektur ist recht gut skalierbar (Verteilung des *Application Level Gateway* auf mehrer Rechner).
- Es können auch neue Dienste installiert werden, für die noch kein *Proxy* zur Verfügung steht. Die Router werden dann so konfiguriert, dass Pakete dieser Dienste direkt durch die *DMZ* durchgereicht werden.

Mit einem *dual homed Application Level Gateway* kann das *DMZ* nochmals aufgesplittet werden und damit die Sicherheit erhöht werden. (Web-Server ist nicht mehr so leicht angreifbar)
Siehe auch Zusammenfassung Kurs 1866!

26. Was ist ein Prozess? [1x]

Ein Programm, das sich gerade in Ausführung befindet, heißt *Prozess*. Zum Prozess gehört aber nicht nur ein Programmstück, sondern auch der *Prozesskontext*
bestehend aus

- den Registerinhalten, insbesondere
- Befehlszähler und
- Grenzen des Adressraums, sowie der
- Prozessnummer
- und anderen Informationen.

27. Was ist ein Thread? [1x]

Threads teilen sich *Programm, Adressraum* und dieselben *Dateien* wie ihr *Erzeuger*. Jeder Thread hat aber seinen *eigenen Stack und Befehlszähler*. Eine solche Gruppe zusammengehöriger Threads nennt man auch *Task*.

- **Vorteil**: schneller Kontextwechsel, Datensharing
- **Nachteil**: fehlender Speicherschutz, blockierende I/O's blockieren u.U. ganzen Task *(außer Thread ist im Kernel implementiert)*

Ist LWP im Kernel implementiert, so wird er wie ein schwergewichtiger Prozess behandelt insbesondere Prozesswechsel und Scheduling wird vom Kernel ausgeführt (so in Linux) => Erzeugung von LWP's mit *clone*.
Anwendung bei Servern und in Gerätetreiber.

28. Haben Threads denn auch einen eigenen Bereich für Daten im Speicher? [1x]

Ja, den Stack aber auch den Heap für statisch lokale Vars.

29. Wie können Threads eines Adressraums miteinander kommunizieren? [1x]

Sie können über gemeinsame Speicherbereiche innerhalb des gemeinsamen Adressraumes (z.B. globale Vars des umgebenden Programms od. shared memory) kommunizieren oder andere Mechanismen wie Pipes oder Queues nutzen.

30. Wer weist Prozesse zu? [1x]

Bei Einprozessorsystemen ist immer genau 1 Prozess rechnend. Alle anderen existierenden Prozesse sind entweder blockiert od. bereit. Welcher der bereiten Prozesse als nächster rechnend wird entscheidet der CPU-Scheduler. Scheduling-Strategien:

- ***FCFS (first come first served)*** – leicht mittels Queue zu implementieren (FIFO) – Nachteil: ein früh eintreffender Prozess mit hoher Rechenzeit od. Endlosschleife hält alle auf.
- ***SJF (shortest job first)*** – bereite Prozesse werden in Reihenfolge aufsteigenden Rechenzeitbedarfs bearbeitet. Vorausetzung: voraussichtliche Rechenzeit ist bekannt – dann aber sehr effizient und gut geeignet für *Stapel-* od. *Batch*-Betrieb mit Prozessen ohne User-Interaktion. Problem: wenn einige lange Prozesse laufen und immer wieder kurze Prozesse gestartet werden „*verhungern*" lange Prozesse => Abhilfe durch „*aging*".

Time-Sharing-Betrieb moderner Rechner vermittelt eine scheinbar gleichzeitige Abarbeitung mehrerer Prozesse auf einem Einprozessorsystem. Dies wird durch ein ***Zeitscheiben***-Scheduling erreicht:

- Scheduler weist jedem Prozess ein Zeitscheibe zu, deren Dicke (also Dauer der Rechenzeit) z.B. von der Prozesspriorität oder von der bereits insgesamt verbrauchten Rechenzeit abhängen kann. Ein Zeitgeber überwacht die zugewiesen Zeitdauer.
- Verschiedene Verfahren für Verteilungsstrategie von Zeitscheiben – eins davon ist ***round-robin*** (Linux) – Prozesse werden *reihum* bedient – werden in der Reihenfolge ihres Eintreffens in ringförmiger Queue gespeichert – Effizienz des Verfahrens hängt von Dicke der Zeitscheiben ab: *zu dicke* Zeitscheiben für z.B. I/O-Prozesse (oft blockiert) sind *ineffizient*, *zu dünne* Zeitscheiben bedeuten CPU-Belastung durch *häufigem Context-Switch*.

Context-Switch wird von *Dispatcher* durchgeführt – da sehr häufig muß dieser sehr schnell arbeiten.
Ein rechnender Prozess kann durch *system-call* dem OS mitteilen, dass er auf ein Ereignis warten will – er wird dann blockiert u. kommt in eine Warteschlange.

31. Wie sieht es mit der Migration (von Prozessen) aus? [1x]

non-preemptiv: der Zielknoten für den Prozess wird bei der Erzeugung zugeteilt
preemptiv: während der Ausführung;

1. Einfrieren auf dem Quellrechner,
2. Erzeugen auf dem Zielrechner,
3. Übertragung des Prozesszustandes,
4. Übertragung des Arbeitsspeichers,
5. Aktivierung auf Zielrechner.

Schwierigkeiten bei nicht abgeschlossenen I/O-Operationen. Einfaches warten auf Ende aller I/O-Operationen ineffizient. Lösung bei

- geöffneten Dateien
  - entweder Schließen abwarten (kann ewig dauern)
  - oder über Dateiserver auf Quellrechner zugreifen (Prozess noch abhängig)
  - oder Dateien Replizieren (Probleme mit nebenläufigen Zugriffen)
- Netzwerkverbindungen (Prozess kann nicht transparent umziehen, also ohne dass die verbundenen Prozesse etwas merken -> mögliche Lösung ist dass der Quellrechner als Proxy verwendet wird, d. h. aber dann wieder Abhängigkeit und nicht vollständig migriert).

Übertragung des Arbeitsspeichers zwei Strategien:

- Nach dem Einfrieren des Prozesses: Bei Nutzung eines großen Arbeitsspeichers durch den Prozess können bei der Übertragung des gesamten Arbeitsspeicher in einem Durchgang relativ lange Zeiten entstehen, in denen der Prozess nicht arbeiten kann -> nicht od. nur schwer möglich bei Prozessen mit zyklischen Aktionen (Time-Triggered).
- Vor dem Einfrieren des Prozesses: Alle Seiten des Speicher werden nacheinander übertragen. Alle Seiten die nach dieser Übertragung vom Prozess noch einmal benutzt werden, werden in einem neuen Durchlauf wiederholt übertragen, solange bis die Anzahl der zu kopierenden Seiten stabil bleibt oder 0 wird. Dann kann der Prozess eingefroren werden und es müssen nur mehr die restlichen Seiten übertragen werden -> ist i.A. effizienter.

32. Was wird alles übertragen? [1x]

Siehe Frage 31.

33. Wie kann man Kommunikation sichern/schützen? [3x]

Durch verschlüsselte Übertragung der Daten.

34. Warum macht man das? Warum Verschlüsselung? [2x]

Um sich vor Angriffen auf

- Vertraulichkeit – Abhören der Kommunikation
- Integrität – Verändern der Daten die kommuniziert werden
- Authentizität – Originalität der Daten verletzen
- Verfügbarkeit – z.B. DoS-Angriffe

zu schützen.

35. Welche Methoden zur Verschlüsselung existieren? [5x]

- *Symmetrische* bzw. *private key* Verfahren – beide Teilnehmer haben den selben geheimen Schlüssel für Ver- und Entschlüsselung.
- *Asymmetrische* bzw. *public key* Verfahren – der Sender besitzt öffentlichen Schlüssel des Empfängers zum Verschlüsseln und der Empfänger verwendet seinen geheimen Schlüssel zum Entschlüsseln. Es werden also nur öffentliche Schlüssel ausgetauscht.

36. Welche klassischen Verschlüsselungsverfahren kennen Sie? Angriffsmöglichkeiten? [3x]

- *Verschiebechiffre* (Cäsar, ROT13): Verschiebung um festen Wert, kann durch statistische Analyse erkannt werden;
- *Substitutionschiffre*: (Ersetzen jedes Buchstaben durch anderen): durch Häufigkeitsanalyse knackbar;
- *Vigenère* (Verschiebung anhand eines code-Wortes): durch statistische Analyse knackbar, wenn Schlüssellänge deutlich kleiner als Textlänge ist;

37. Warum ist ein Verschiebechiffre nicht sicher? [1x]

Da hier mit festen Werten für die Verschiebung gearbeitet wird und somit durch Analyse der Häufigkeitsverteilung des Cipher-Textes bei bekannter Sprache und somit bekannter

Normverteilung der Zeichenhäufigkeit recht einfach der Verschiebewert ermittelt werden kann.

38. Wie kann man bei Vigenére den Cipher-Text brechen? [3x]

Es wird versucht, aus dem Cipher-Text die Größe des verwendeten Alphabets zu ermitteln. Danach wird für verschiedene Schlüssellängen der Koinzidenzindex berechnet (gibt an, mit welcher Wahrscheinlichkeit 2 zufällige Zeichen aus einem Text gleich sind). Wenn bekannt ist, aus welcher Sprache der Cipher-Text erzeugt wurde, kann über bekannte Koinzidenzindizes für die jeweilige Sprache und den errechneten Werten für die verschiedenen vermuteten Schlüssellängen auf eine wahrscheinliche Schlüssellänge geschlossen werden. Danach wird für alle Teiltexte der Teilschlüssel mit Hilfe der Analyse der Häufigkeitsverteilung gesucht und so der verwendet Schlüssel zusammengestellt.

39. Wie kann man die Vigenére-Verschlüsselung noch sicherer machen? [2x]

In dem man die Schlüssellänge annähernd gleich der Textlänge wählt. Somit steht bereits für die Bestimmung der Schlüssellänge zu wenig statistisches Material zur Verfügung und auch die Teiltexte sind dann für Rückschlüsse aus der Häufigkeitsuntersuchung zu kurz. Generell sollte die Schlüssellänge eine untere Grenze nicht unterschreiten, um ein einfaches Durchsuchen des Schlüsselraumes (*brute force*) von vornherein abzuwehren (Schlüsselraum $K=m^l$, mit m=Größe des Alphabets und l=Schlüssellänge).

40. Welche modernen Verfahren kennen Sie? [1x]

Symmetrische bzw. private key Verfahren:

- *DES* – Data Encryption Standard: Ist im wesentlichen ein Feistel-Verfahren
- *IDEA* – International Data Encryption Standard: Nachfolger von *DES*, aber mit größerer Schlüssellänge. Wird auch für *PGP* eingesetzt.
- *Blowfish*: (von Bruce Schneider) Basierend auf dem Feistel-Verfahren werden in jeder Runde beide Hälften des Wortes verändert.
- *RC5*: (von Ron Rivest) RC5 ist eine Familie von Algorithmen und beruht nicht auf dem Feistel-Verfahren.
- *CAST-128*: (von Calisle Adams u. Stafford Tavares) Es ist ein klassischer Feistel-Algorithmus.
- *AES* – Der Advanced Encryption Standard

Asymmetrische bzw. Public key Verfahren:

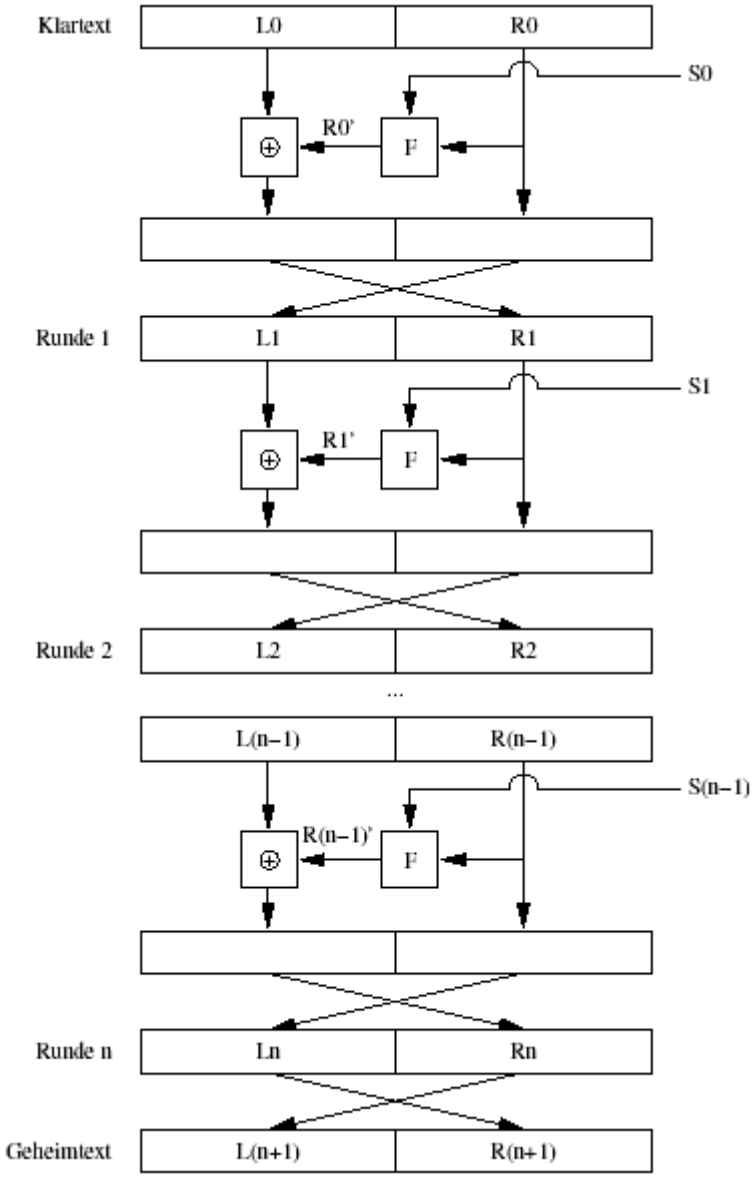
- *RSA*: (von Rivest, Shamir u. Adleman) RSA kann zur Verschlüsselung und Erzeugung digitaler Signaturen benutzt werden. Die Sicherheit von RSA beruht darauf, dass es schwer ist, große Zahlen in ihre Primfaktoren zu zerlegen (Faktorisieren).
- *El-Gamal*: Die Sicherheit des Verfahrens von El-Gamal beruht auf der Schwierigkeit der Berechnung von diskreten Logarithmen über endlichen Körpern.
- *Diffie-Hellmann*: Ähnliches Verfahren wie El-Gamal, mit dem sich 2 Teilnehmer auf einen geheimen Schlüssel verständigen können (z.B. bei SSL).

Details siehe Zusammenfassung Kurs 1866!

41. Wie funktioniert das DES Verfahren? [2x]

Basiert im wesentlichen auf dem Feistel-Verfahren:
(Nach H. Feistel) Die Grundidee ist, die Operationen Ersetzen und Vertauschen mehrfach hintereinander zu benutzen.

Prinzip: Klartextwort der Länge *g* wird in ein linkes L_0 u. rechtes Teilwort R_0 der Länge *g*/2 aufgeteilt -> aus dem Schlüssel *S* wird ein erster Teilschlüssel S_0 berechnet, der eine Funktion *F* steuert, welche auf R_0 angewandt wird -> das Ergebnis R_0' und L_0 werden über XOR verknüpft -> Ergebnis aus XOR und R_0 werden vertauscht. Dieser Vorgang (Runde) wird mehrmals hintereinander ausgeführt, wobei für jede Runde *i* ein Teilschlüssel S_{i-1} generiert wird. Nach der letzten Runde werden R_n und L_n noch vertauscht.
Die Entschlüsselung erfolgt nach dem selben Prinzip, die Teilschlüssel werden aber in der umgekehrten Reihenfolge benutzt. Die in Abb. 2.2 gezeigten Runden werden also von unten nach oben durchlaufen. Wegen der Tatsache das *F* eine Funktion ist (jede Eingabe wird eindeutig auf eine Ausgabe abgebildet) und da das XOR nach 2-maliger Anwendung wieder die Eingabe liefert, erhält man am Ende wieder den Klartext.



DES:

Funktion F:	Kombination aus XOR und festen Ersetzungsboxen (S-BOX)
Teilschlüssel:	Generierung von 48 Bit Teilschlüssel durch Shift und Permutation.
Runden:	16
Blockgröße:	64 Bit
Schlüssellänge:	56 Bit

Schwachpunkt des DES ist die rel. kleine Schlüssellänge von 56 Bit, der bereits 1999 in 22,25 Stunden (mit Parallelrechnern und einer Spezialmaschine „deep crack“) geknackt werden konnte. Eine mögliche Verbesserung ist die Verwendung mehrerer verschiedener Schlüssel um so den Suchraum für einen Angriff zu vergrößern. Heute wird meist *triple DES* (*3DES*) benutzt.

42. Auf welchem mathematischen Problem beruht RSA und wie funktioniert RSA? [3x]

Die Sicherheit von RSA beruht darauf, dass es schwer ist, Produkte großer Primzahlen zu faktorisieren.

Prinzip: Für die Erzeugung eines Schlüsselpaares werden 2 große Primzahlen *p* und *q* gewählt und $n = p \cdot q$ berechnet. Für jede Zahl $m \leq n$ gilt dann

$$m^{k(p-1)(q-1)+1} \bmod n = m$$

für ein beliebiges *k*.

Nun wählt man eine natürliche Zahl e, die teilerfremd zu *(p-1)(q-1)* ist. Der öffentliche Schlüssel besteht dann aus *e* und *n*. Der geheime Schlüssel *d* wird so berechnet, dass $e \cdot d \bmod (p-1)(q-1) = 1$ gilt, woraus folgt dass $e \cdot d = k(p-1)(q-1)+1$ für ein *k* gilt.

Verschlüsselung: Die Nachricht *m* wird mit dem öffentlichen Schlüssel verschlüsselt

$$Crypt_e(m) = m^e \bmod n$$

Entschlüsselung: Mit $c = Crypt_e(m)$ und $DeCrypt_d(c) = c^d \bmod n$ folgt insgesamt

$$\begin{aligned} DeCrypt_d(Crypt_e(m)) &= Crypt_e(m)^d \bmod n \\ &= (m^e)^d \bmod n \\ &= m^{e \cdot d} \bmod n \\ &= m^{k(p-1)(q-1)+1} \bmod n \\ &= m \end{aligned}$$

Mit RSA können nur Nachrichten verschlüsselt werden, die kleiner *n* sind. Größere Nachrichten müssen vorher in passende Blöcke zerlegt werden. Wegen der Kommutativität der Multiplikation können Nachricht auch mit dem geheimen Schlüssel verschlüsselt und mit dem öffentlichen Schlüssel wieder entschlüsselt werden (in obiger Ableitung einfach *e* mit *d* vertauschen). Damit ist RSA auch für digitale Unterschriften einsetzbar.

Die Sicherheit von RSA beruht darauf, dass aus dem öffentlichen Schlüssel nicht auf den privaten Schlüssel geschlossen werden kann. Alle bekannten Angriffsverfahren versuchen, n in seine Primfaktoren p und q zu zerlegen (Faktorisierung). Der Aufwand dafür steigt mit der Größe von n. Derzeit sind Schlüssellängen von 1024 Bit bereits die untere Grenze in Punkto Sicherheit.

Da die RSA Verschlüsselung wegen der Multiplikation sehr großer Zahlen sehr zeitaufwändig ist (ca. Faktor 100 langsamer als DES), wird sie hauptsächlich dafür verwendet, den geheimen Schlüssel für die Session (***session key***) eines symmetrischen Verfahrens zu übertragen. Damit ein Angreifer den session key beim Abhören von verschlüsselten Nachrichten nicht herausfinden kann, sollte der session key

- groß genug sein, so dass ein simpler Vergleich mit allen bereits mitgehörten Nachrichten zu aufwändig wäre
- von guten Zufallsgeneratoren mit echten Zufallsquellen zur Initialisierung abhängen.

Weiter Details siehe Zusammenfassung Kurs 1866.

43. Angriffsmethoden auf Verschlüsselung? [2x]

Mit *Kryptoanalyse*-Techniken ist es möglich, verschlüsselte Daten ohne Kenntnis des Schlüssels wieder zu entschlüsseln.
Das Unterscheidungskriterium zwischen den verschiedenen Angriffstypen ist die Menge der Informationen, die dem Angreifer zur Verfügung stehen. *(Die Auflistung der Angriffstypen beginnt mit dem Angriff mit der geringsten Menge an zur Verfügung stehender Information)*:

Ciphertext only Angriffe:
Dem Angreifer steht nur eine oder mehrere verschlüsselte Nachrichten zur Verfügung. Außerdem ist davon auszugehen, dass auch der Algorithmus selbst bekannt ist. Im wesentlichen bestehen diese Angriffe aus *brute force* Angriffen oder aus statistischen Angriffen, wo Methoden der Statistik oder Wahrscheinlichkeitsrechnung angewandt werden. Der Erfolg der statistischen Methoden hängt davon ab, dass genügend Geheimtext zur Verfügung steht.

Known Plaintext Angriffe:
Hierbei befindet sich der Angreifer im Besitz einiger bekannter Paare von Klartext und zugehörigem Geheimtext. *(Z.B. kennt der Angreifer bei Blockverschlüsselung den Inhalte einiger Blöcke, weil z.B. Teile der verschlüsselten Daten, wie z.B. Copyrights in Java Source, allg. bekannt sind.)* Damit können Klar- u. Geheimtext auf bestimmte Muster hin untersucht werden und so Rückschlüsse über den benutzen Schlüssel getroffen werden. -> siehe *Lineare Kryptoanalyse*.

Chosen Plaintext Angriffe:
Der Angreifer hat die Möglichkeit, bestimmte Klartexte verschlüsseln zu lassen. Die können so gewählt werden, dass sie Muster enthalten, die nach der Verschlüsselung Rückschlüsse auf den verwendeten Schlüssel zulassen -> siehe *Differentielle Kryptoanalyse*.
Eine Erweiterung sind die ***Adaptive Chosen Plaintext*** Angriffe, wo auf den Ergebnissen der bisherigen Analysen immer wieder neue Klartexte zur Verschlüsselung ausgewählt werden können.

Weiter Angriffstypen:

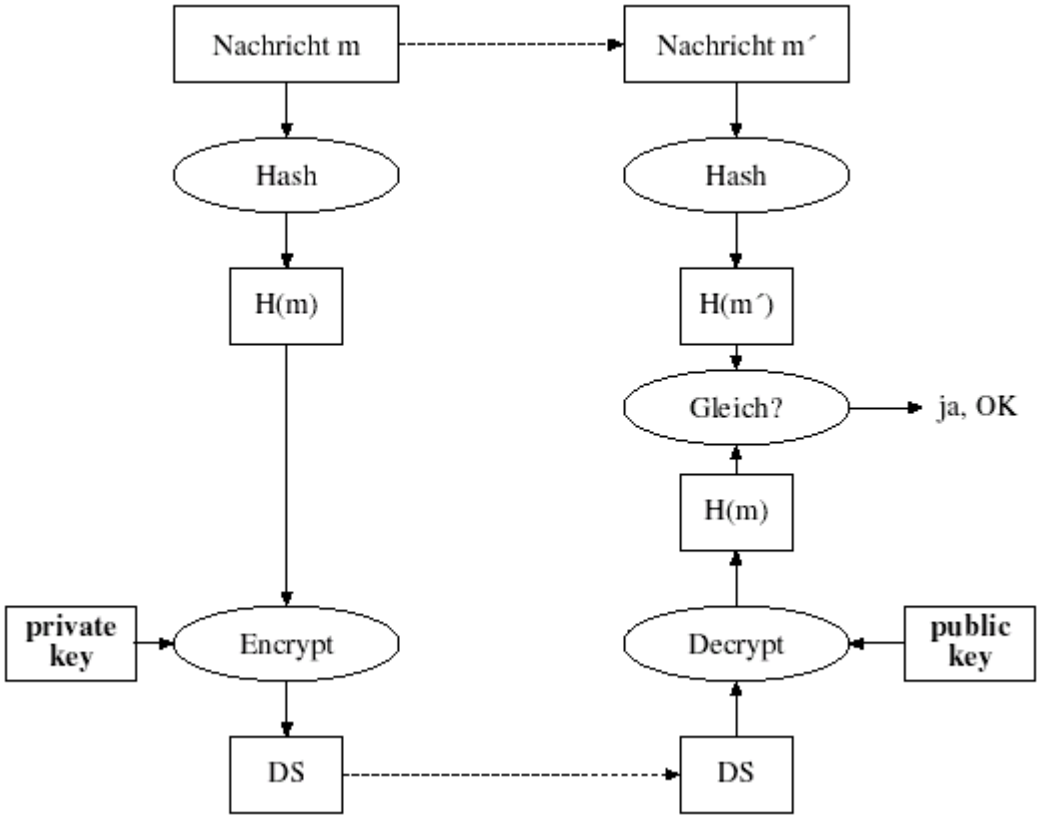
- ***Chosen Ciphertext Angriff***: Angreifer kann verschiedene Geheimtexte erstellen und die zugehörigen entschlüsselten Klartexte analysieren.
- ***Chosen Key Angriff***: Angreifer benutzt Informationen über den Schlüssel.
- ***Kryptoanalyse mit Gewalt***: Angreifer versucht von jemanden, der den Schlüssel kennt, die Herausgabe zu erzwingen.

Brute Force Angriffe:
Es werden systematisch alle möglichen Schlüssel getestet. Je größer der Schlüssel ist desto größer ist auch der potentielle Schlüsselraum der durchgetestet werden muss.
Weitere Details siehe Zusammenfassung Kurs 1867.

44. Wie funktioniert das public key Verfahren (im Mailaustausch)? [5x]

Der Empfänger der Mail sendet zuerst seinen public key an den potentiellen Sender. Dieser verschlüsselt seine Mail an den Empfänger mit dessen public key. Der Empfänger entschlüsselt dann die erhaltenen Mail mit seinem eigenen private key.

45. Wie signiere ich eine Mail digital? Wie funktioniert das mit der elektronischen Unterschrift? [5x]



Prinzip: Der Sender einer Nachricht m berechnet H(m) und verschlüsselt diesen mit seinem private key zu einer DS. Dann werden die Nachricht und die DS an den Empfänger geschickt. Dieser entschlüsselt die DS mit dem public key des Senders, berechnet H(m') und vergleicht diesen mit dem entschlüsselten H(m). Bei Gleichheit ist Authentizität gegeben.

Ein Angreifer kann zwar die Nachricht verändern, aber die passende DS nicht selbst erzeugen. Ebenso kann der Empfänger nicht selbst eine passende DS erzeugen.

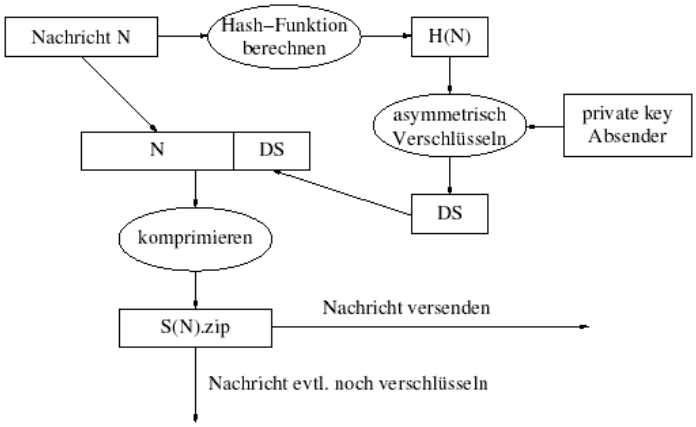
Zur Berechnung der DS kann entweder RSA oder der in den USA als Standard definierte Digital Signature Algorithm DSA (beruht auf ElGamal), der auch für PGP und GPG verwendet wird, eingesetzt werden

Detail zu Hash siehe Zusammenfassung Kurs 1866.

46. Kann man eine Mail signieren und gleichzeitig verschlüsseln? [2x]

Ja, zuerst wird eine digitale Signatur der Nachricht erstellt und an die Nachricht angehängt. Dann wird das Ganze (Nachricht + DS) noch mit dem public key des Empfängers verschlüsselt.

Ablauf beim digitalen Signieren mit PGP:



47. Warum kann der Faktor Zeit in verteilten Systemen eine Rolle spielen? Warum benötigt man Uhrzeitsynchronisation? Warum ist es wichtig, dass alle Rechner die gleiche Zeit haben?[3x]

z.B. für

- konsistente Zeitstempel für e-Commerce Transaktionen
- die Wahrung der Konsistenz verteilter Daten (Zeitstempel für die Serialisierung)
- die Prozesssynchronisation
- die Überprüfung der Authentizität einer an einen Server gesendete Anforderung (siehe Kerberos)
- den make-Prozess (es werden nur jene Dateien compiliert, die seit dem letzten Mal verändert wurden -> Datum/Zeitvergleich zwischen Sourcen und Objectdateien)

48. Welche Möglichkeiten gibt es zur Synchronisation? Wenn man ein verteiltes System abgleichen will, was macht man da? [4x]

Synchronisierung physischer Uhren:

- entweder Synchronisierung mit einer autoritativen externen UTC (coordinated universal timer) Zeitquelle
- oder Synchronisierung der Uhren untereinander mit einem bekannten Maß an Genauigkeit (interne Synchronisierung). Dabei müssen die Uhren aber nicht mit der realen Zeit übereinstimmen.
  Korrektheit von Uhren bei interner Synchronisierung:
  - wenn die Hardware-Uhr *H* eine Abweichungsgeschwindigkeit innerhalb einer bekannten Grenze besitzt
  - wenn die Monotonie-Bedingung erfüllt ist, d.h. dass die Uhr *C* immer vorwärts läuft

Synchronisierung in einem synchronen System findet zwischen zwei Prozessen statt. Da in einem synchronem System die minimale und maximale Übertragungsdauer einer Nachricht bekannt ist, kann ein Prozess seine Systemzeit an den zweiten Prozess senden. Dieser stellt nun seine Systemuhr auf $t_{Systemuhr}$ + (max + min)/2. Damit kann die Abweichung der Uhren maximal die Hälfte des Unterschiedes zwischen der maximalen und der minimalen Übertragungszeit liegen.
Verteilte System sind aber i.A. asynchrone Systeme wo gerade die max. Übertragungsdauer einer Nachricht nicht bekannt ist (speziell im Internet).

Christians Methode zur Uhrensynchronisierung:
Einsatz eines Zeitservers, der an UTC synchronisiert ist; er geht davon aus, dass es zwar keine Obergrenze für den Datenaustausch gibt, die Roundtripzeiten häufig aber ziemlich kurz sind, seine Methode erzielt ausreichende Synchronisierung dann, wenn die Roundtripzeiten im Vergleich zu der geforderten Genauigkeit ausreichend kurz sind: $t + T_{round}/2$, wobei ein Prozess T_{round} als Zeitintervall zwischen Absetzen einer Anforderungsnachricht an den Zeitserver und Erhalt der Antwort vom Server selbst misst. Die erreichbare Genauigkeit beträgt $+/-(T_{round}/2-min)$ mit min=minimale Übertragungszeit. Variabilität kann durch Verwendung der kürzesten Zeitspanne bei mehreren Anfragen kompensiert werden; Nachteil ist die Abhängigkeit von einem Server, wofür Cristian die Verwendung mehrerer Zeitserver vorschlägt, die vom Client über Multicast befragt werden, wobei nur die erste Antwort der Server vom Client verwendet wird.
Weiters kann es auch zu Problemen durch fehlerhafte oder betrügerisch modifizierte Zeitserver kommen. Dies kann im ersteren Fall durch den Berkeley-Algorithmus teilweise gelöst werden und im letzteren Fall durch Authentifizierungstechniken.

Berkeley-Algorithmus:
Hier wird der ZeitServer als Master eingetragen und die Clients als Slave. Der Master fragt in regelmäßigen Abständen die Zeit der Slaves ab, eliminiert aus dieser Menge die Ausreißer und bildet aus den anderen einen Mittelwert (fehlertoleranter Mittelwert, da nur jene Uhrenwerte zur Berechnung herangezogen werden, die sich nicht mehr als einen vorgegebenen Betrag voneinander unterscheiden). Jetzt errechnet er aus dem ebenfalls berechneten Mittelwert der RoundTripZeit (ähnl. wie bei Christian) und dem Mittelwert der erhaltenen Uhrenwerte (inkl. seines eigenen) ein Delta für jeden Client und übermittelt dies. Falls der Master ausfällt kann ein anderer Client seine Aufgabe übernehmen.

NTP – Network Time Protocol:
Im Gegensatz zu vorhergehenden Methoden definiert NTP einen Zeitdienst und Protokoll dass auch für die Übertragung im Internet geeignet ist.

- NTP ist als hierarchische Struktur (Synchronisierungs-Teilnetz in logischer Baumstruktur) ausgelegt, die über das gesamte Internet verteilt ist. Es ist
  - redundant,
  - zuverlässig,
  - auf eine große Menge von Clients ausgelegt und
  - bietet Sicherheit mittels Authentifizierung der Timing-Daten und Rückkehradressenauswertung des Servers der an ihn gesendeten Nachrichten
- Die Wurzel und Knoten des Baumes sind Zeitserver, wobei der Primär-Server an der Wurzel direkt mit einer UTC-Quelle verbunden ist und Sekundär-Server in den Knoten der darunter liegenden Ebenen jeweils mit einem Server der darüber liegenden Ebene.
- Die Server auf der untersten Ebene (Blätter) laufen auf der Maschine des Users. Die Uhrengenauigkeit sinkt von der Wurzel zu den Blättern.
- Das Synchronisierungs-Teilnetz wird neu konfiguriert wenn Server ausfallen od. Fehler auftreten. (z.B. Primär-Server verliert UTC-Quelle -> er wird zu Sekundär-Server)

NTP-Server können sich über 3 verschiedene Modi synchronisieren:

- *Multicast*-Modus: für die Verwendung in Hochgeschwindigkeitsnetzen, bietet allerdings geringere Genauigkeit (ist aber für die meisten Anwendungen ausreichend.
- *Prozeduraufruf*-Modus: ähnliche Arbeitsweise wie Christian-Algorithmus, wird verwendet wenn Genauigkeit von Multicast nicht ausreicht od. Multicast nicht unterstützt wird. (z.B. Datei-Server, die Zeitinfo für Dateizugriff benötigen, können lokalen Server im Prozeduraufruf-Modi kontaktieren)
- *Symmetrischer* Modus: für Server, die Zeitinformationen im LAN zur Verfügung stellen bzw. für Zeitserver in den höheren Ebenen eines Synchronsierungs-Teilnetzes, die höchste Genauigkeiten erzielen müssen. Die Timing-Daten, die zwischen einem Server-Paar ausgetauscht werden, werden aufbewahrt um die Synchronisierungsgenauigkeit auf längere Sicht zu wahren.

Die Nachrichten werden über UDP-Pakete ausgeliefert. Im Prozeduraufruf- und im symmetrischen Modus werden Nachtrichtenpaare (Uhrenoffset o_i und Übertragunsverzögerung d_i) ausgetauscht. NTP-Server wenden einen Datenfilter-Algorithmus auf aufeinanderfolgende Paare $<o_i, d_i>$ an, der den Offset o schätzt und die Qualität dieser Schätzung als statistisches Maß berechnet, die sog. *Filterdispersion*. Je größer die Filterdispersion desto unzuverlässiger sind die Timing-Daten.
Ein NTP-Server führt i.A. einen Nachrichtenaustausch mit mehreren Server (Peers) durch und verwendet dabei ebenfalls einen Algorithmus zur Auswahl der relativ zuverlässigeren Peers. Vereinfacht gesagt werden jene Peers bevorzugt, die näher zu einem Primär-Servers sind und

die auch die geringste *Synchronisierungsdispersion* (Summe der Filterdispersionen zw. Server und Wurzel) aufweisen.

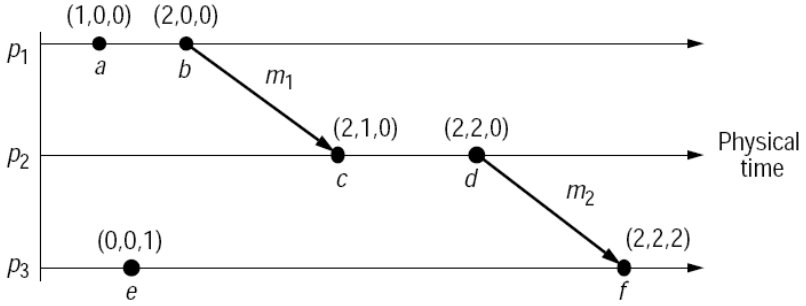
Logische Uhren:
In verteilten Systemen kann aufgrund von Synchronisierungsfehlern die physische Uhr nicht unbedingt dazu verwendet werden, um zu erkennen, in welcher Reihenfolge Ereignisse in Prozessen aufgetreten sind.
Logische Uhr von Lamport geht von 2 Voraussetzungen aus:

- wenn 2 Ereignisse im selben Prozess auftreten sind sie in der Reihenfolge aufgetreten, wie sie der Prozess wahrnimmt
- wenn eine Nachricht zwischen 2 Prozessen übertragen wird, ist das Ereignis des Sendens vor dem Ereignis des Empfangens

Diese partielle Reihenfolge wird als *Geschehen-vor*-Relation bezeichnet. Die Lamport-Uhr ist ein Softwarezähler (monoton steigender Zähler), den jeder Prozess für sich führt. Dieser Zähler inkrementiert bevor ein Ereignis im Prozess veranlasst wird. Versendet der Prozess eine Nachricht, so wird der Nachricht der aktuelle Wert der logischen Uhr beigefügt.

Vollständig synchronisierte logische Uhren:
Paare unterschiedlicher Ereignisse in verschiedenen Prozessen können identische *Lamport*-Zeitstempel haben. Um hier trotzdem eine vollständige Reihenfolge zu erhalten werden zusätzlich noch die PID's verwendet. Es gilt dann mit T_i=lokaler Zeitstempel des Prozesses p_i mit der PID i:
für die globalen Zeitstempel $(T_i, i)<(T_j, j)$ falls $T_i<T_j$ oder $T_i=T_j$ und $i<j$
Diese Reihenfolge hat keine phys. Bedeutung (PID's sind zufällig) kann aber durchaus praktisch sein (z.B. Sortierung der Eintrittsreihenfolge von Prozessen in eine kritische Zone).

Vektoruhren:
Von Mattern und Fidge entwickelt um die Unzulänglichkeit von Lamports Uhren zu korrigieren, nämlich dass *L(e)<L(e')* nicht zwingend folgt, dass *e->e'* (z.B. wenn zwei Ereignisse nicht im selben Prozess od. dch. Nachrichten verbunden, also *e||e'*). Eine Vektoruhr für ein System mit *N* Prozessen ist ein Integer-Array mit *N* Elementen. Jeder Prozess *i* trägt seine lokale logische Uhr im Feld *i* dieses Arrays ein und verwaltet in den anderen Feldern die logischen Uhreninformationen der restlichen Prozesse, die ihm im Zuge des Nachrichtenaustausches mitgeteilt werden.



Damit kann auch aus *V(e)<V(e')* geschlossen werden, dass *e->e'* gilt und außerdem kann die Nebenläufigkeit von Ereignissen *e || f* erkannt werden wenn weder *V(e)<=V(f)* noch *V(f)<=V(e)* gilt.
Nachteil der Vektoruhren gegenüber Lamport-Uhren:

- benötigen sehr viel Speicher
- Nachrichtennutzlast steigt proportional zu Anzahl der Prozesse

49. Wenn man feststellt, dass ein Rechner nicht synchron läuft, was macht man dann, da man bei einem negativen Ergebnis die Zeit ja nicht zurückstellen darf (Monotoniebedingung)? [1x]

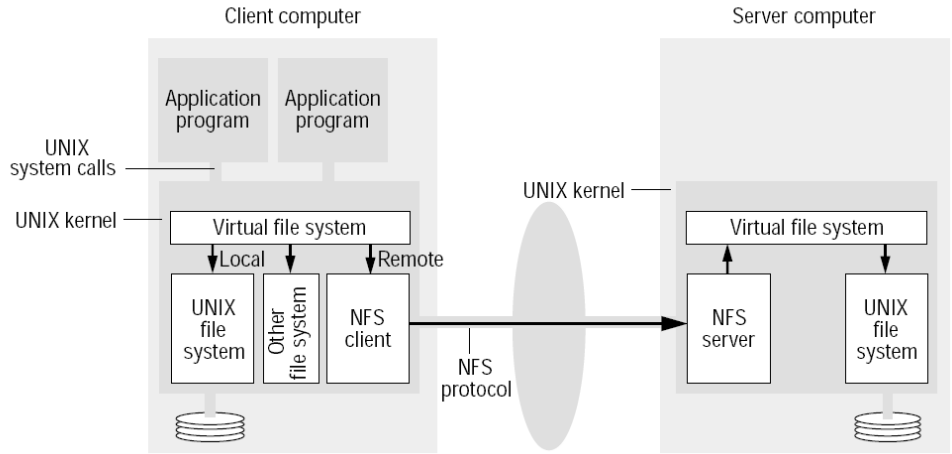
Die Zeit anhalten oder, um strenge Monotonie zu gewährleisten, die Zeit per SW langsamer laufen lassen. Dies ist möglich, da ja der Wert *H(t)* der Hardware-Uhr über den Skalierungsfaktor *a* und den Offsetwert *b* in den Wert *C(t)=aH(t)+b* der SW-Uhr umgerechnet werden. *a* und *b* können nun entsprechend modifiziert werden, so dass die SW-Uhr langsamer läuft.

50. Was ist NFS? [1x]

Sun Network File System. Dient dazu, ein entferntes Dateisystem im lokalen Dateisystem einzubinden (virtuelles Dateisystem) um für den User eine Zugriffstransparenz auf entfernte Dateien zu realisieren.

51. Wie funktioniert NFS? [2x]

Figure 8.8 NFS architecture



Die NFS Server- und Client-Module befinden sich im Kernel des OS (Unix, Linux). Anforderungen die sich auf Dateien in einem entfernten Dateisystem beziehen, werden vom Client-Modul in NFS Protokoll-Operationen und dann an das NFS Server-Modul des entsprechenden entfernten Computers weitergegeben. Die NFS-Module kommunizieren über RPCs. NFS kann entweder TCP od. UDP verwenden (konfigurieren). Ein Port-Mapper-Dienst ermöglicht den Clients sich dem Namen nach zu Diensten in einem Host zu binden. Die RPC-Schnittstelle zum Server ist offen, jeder Prozess kann Anforderungen stellen und werden bei Gültigkeit und entsprechender Benutzerberechtigung bedient. Übergabe signierter Benutzerberechtigungen und verschlüsselte Datenübertragung kann optional aktiviert werden.

NFS bietet *Zugriffstransparenz*, d.h. lokale Programme u. Benutzer können auf entfernte Dateien wie auf Dateien im lokalen Dateisystem zugreifen. Dazu wird über alle vorhandenen Dateisysteme ein virtuelles Dateisystem gestellt.

Zum Zugriff auf Dateien werden Datei-IDs verwendet, die, unter Unix, von der I-Node Nummer der Datei abgeleitet werden.

Ein *Automounter* (z.B. *autofs*) ermöglicht es, dass automatisch das entsprechende entfernte Verzeichnis gemountet wird, wenn ein Client auf einen leeren Mountpunkt verweist.

NFS nutzt das *Caching* zur Optimierung des Kommunikationsaufwandes. Beim Server wird wie beim konventionellen Dateisystem das *Read-ahead* Caching (voraus lesen) eingesetzt. Das *Delayed-write* Caching kann allerdings z.B. bei Serverabsturz

Konsistenzprobleme erzeugen. Daher gibt es seit NFS 3.0 zwei Optionen für die *write*-Operation:

- *Write-through* Caching (geschriebenen Daten werden in Cache und auf HD geschrieben)
- Daten in write-Operationen werden nur im Cache gehalten und erst durch eine *commit*-Operation des Clients auf HD übertragen. Der Client erhält erst dann eine Antwort auf das commit, wenn die Daten wirklich auf der HD sind.

Clientseitig werden die Ergebnisse von *read*, *write*, *getattr*, *lookup* und *readdir* in den Cache gestellt. Die Clients müssen aber bei den Servern die Aktualität ihrer Daten im Cache sicherstellen.

52. Was hat der Administrator vor dem Einsatz von NFS zu tun? [1x]

Er muss auf den NFS Server-Maschinen die Namen der lokalen Dateisysteme bekannt geben (bei Linux in der Datei */etc/exports*), die für das entfernte Mounten zur Verfügung stehen. Jedem Dateisystem ist dabei eine Zugriffsliste zugeordnet, in der jene Hosts angegeben sind, die berechtigt sind, das Dateisystem zu mounten.

53. Wie findet die Authentifizierung bei NFS statt? [1x]

Der NFS Server ist bis v4.0 zustandslos, d.h. es werden keine Client-Zustände verwaltet, und er behält keine Client-Dateien geöffnet. Es werden mit jeder Anfrage an den Server zur Authentifizierung die Benutzer-ID/Group-ID übertragen und mit den Zugriffsberechtigungs-Attributen der Datei verglichen. Da die Übertragung unverschlüsselt erfolgt, sind diese Authentifizierungsdaten leicht abzuhören. Diese Sicherheitslücke kann durch eine Option des RPC-Protokolls zur DES-Verschlüsselung der Authentifizierungsinformationen geschlossen werden. Außerdem ist in neueren Version von Sun NFS das Kerberos Authentifizierungssystem implementiert. Dem NFS-Mount-Server werden die vollständigen Kerberos-Authentifizierungsdaten des Benutzers übergeben, wenn seine Home- und Root-Dateisysteme gemountet werden. Die Ergebnisse dieser Authentifizierung inkl. Benutzer-ID und Client-Adresse werden vom Server mit den Mount-Informationen für jedes Dateisystem gespeichert. Bei jeder Dateizugriffsanforderung überprüft der NFS-Server die Benutzer-ID und die Adresse des Senders und erteilt nur dann Zugriff, wenn sie mit dem beim Mounten gespeicherten Daten übereinstimmen.

54. Wie genau geschieht das Mounten? Wann wird gemountet? [1x]

Separater Mount-Dienst auf Benutzerebene des NFS Server-Computers unterstützt das Mounten entfernter Dateisysteme durch Clients. Clients nutzen dabei eine abgeänderte Version des UNIX mount-Befehls. Es wird dabei

- der entfernte Host-Name,
- der Pfadname des Verzeichnisses im entfernten Dateisystem
- und der lokale Name, unter dem es gemountet werden soll

übergeben. Der modifiziert mount-Befehl kommuniziert mit dem Mount-Dienst am entfernten Host über ein Mount-Protokoll (ein RPC-Protokoll das spezielle Operationen beinhaltet). Die Position des Servers (IP-Adr., Portnummer) und der Datei-Handle für das entfernte Verzeichnis werden auf die VFS-Schicht und an den NFS Client übergeben. Entfernte Dateisysteme können auf dem Client-Computer nach 2 Varianten gemountet sein:

- *hard-gemountet*: der anfordernde Prozess wird unterbrochen, bis die Anforderung bedient werden kann (kann bei Ausfall des NFS Server auch lange dauern)

- *soft-gemountet*: das NFS Client-Modul gibt eine Fehlermeldung an den anfordernden Prozess zurück, wenn die Anforderung nach mehreren wiederholten Versuchen nicht bedient werden kann (kann Probleme erzeugen, wenn der anfordernde Prozess diese Fehlerhinweise nicht verarbeitet -> ist bei einigen Utilities und Apps. bei Dateioperationen durchaus der Fall).

Wann wird gemountet -> siehe Automounter Frage 51.

55. Was macht der NFS Client genau, um an die Dateien zu kommen? [1x]

siehe Frage 54. und Automounter Frage 51.

56. Was macht der NFS Server zum Bereitstellen der Dateien? [1x]

57. Thema Transaktionen, wie funtkioniert das im FUB? [1x] [Haa]

1. Führe Manipulationen lokal auf Datenobjekten aus und merke Änderungsinformationen im Transaktionslog.
2. Verschicke das Transaktionslog vom Client zu einem zentralen Vermittler.
3. Der Vermittler entscheidet, ob die Transaktion mit anderen (bereits zugesicherten) Transaktionen in Konflikt steht.
4. Wenn die Transaktion zu keiner anderen Transaktion im Konflikt steht, dann wird sie an alle anderen Clients weiter geleitet, die die Transaktion dann einspielen.
5. Wenn die Transaktion zu bereits zugesicherten Transaktionen im Konflikt steht, dann wird der initiierende Client über den Konflikt in Kenntnis gesetzt. Der Client muss die Transaktion dann rückgängig machen.

Die Transaktion wird also zentral synchronisiert.

58. Wie bekommen die Clients die Änderungen? [1x] [Haa]

siehe Punkt 4. Frage 57.

59. Welche Netzwerkarchitekturen haben sie im Kurs kennen gelernt? [1x] [Haa]

Client-Server Architektur, replizierende Systeme, P2P

60. Welche Probleme treten bei replizierenden Systemen auf? [1x] [Haa]

Die Daten werden dabei auf die Clients repliziert und dort lokal bearbeitet. Problematisch ist dabei neben der Transparenz der Replizierung vor allem die Aktualität und Konsistenz der Daten.

61. Wie kann man diese Probleme lösen? [1x] [Haa]

Um die Replikate aktuell zu halten, können 3 Verfahren verwendet werden:

- *Verteilung von Benachrichtigungen* über Veränderungen an alle Systeme, auf denen Replikate der betroffenen Daten gehalten werden. Es werden dann dort die gehaltenen Replikate als invalid markiert und beim nächsten Zugriff aktualisiert. -> benötigt wenig Übertragungsbandbreite, funktioniert gut bei häufigen Veränderungen aber wenig Lesezugriffen. (siehe auch *AFS* mit *call-back-promise*)

- *Verteilung der geänderten Daten*: damit werden sofort auf allen Systemen mit betroffenen Replikate diese aktualisiert -> benötigt hohe Übertragunsbandbreite, günstig wenn Lesezugriffe > Schreibzugriffe.
- *Verteilung von Update-Operationen*: Kompromiss zwischen beiden ersten Varianten. Es wird den Clients mitgeteilt dass sich ein Objekt verändert hat und wie es sich verändert hat. Client benötigt Prozess der Updates entgegen nehmen kann und die Updates sind nur auf einem definierten Ausgangszustand anwendbar und damit nur in einer definierten Reihenfolge einspielbar. -> benötigt bei komplexen Update-Operationen durchaus beträchtliche Rechenleistung auf den Clients um den neuen Zustand zu berechnen.

Wahl des Zeitpunktes zum Verschicken von Veränderungen:

- *Push*-Modell: Änderungen werden sofort nach ihrem Auftreten verschickt
- *Pull*-Modell: Clients müssen nachfragen, ob Veränderungs-Nachrichten vorliegen

Um die Konsistenz zu gewährleisten haben sich folgende Implementierungsstrategien für gleichzeitige Schreiboperatioen etabliert:

- *Änderungen nur an Master-Kopie auf dem Server*: Bei diesem Ansatz existiert für jedes replizierte Objekt eine Master-Kopie auf dem Server, die Schreibzugriffe auf sich koordiniert. Clients können in diesem Fall ihre Replikate nur zum Lesen benutzen. Möchten Sie die Daten verändern, so bitten sie den Server, die Veränderung auf den Daten vorzunehmen. -> Nachteil: bei Schreiboperationen ist Vorteil der Replikationen aufgehoben.
- *Änderung an einer Master-Kopie bei einem Client*: In diesem Fall ist zu jedem Zeitpunkt genau ein Client für jede Master-Kopie zuständig. Dieser Client darf Veränderungen auf dem Objekt vornehmen und verschickt den neuen Zustand des Objekts. Andere Clients müssen entweder beim zuständigen Client eine Veränderung beauftragen (siehe oben) oder von diesem die Verwaltung für diese Master-Kopie übernehmen. -> ähnl. wie oben aber dezentrale Haltung der Master-Kopien.
- *Änderung an jedem Objekt erlaubt – aktive Replikation*: In diesem Fall müssen die Änderungen auf jedem Replikat in der gleichen Reihenfolge ausgeführt werden. Hierzu verwendet man Zeitstempel bei den Änderungen, anhand derer die Änderungen in eine konsistente Reihenfolge gebracht werden können. Zusätzlich muss dafür gesorgt werden, dass die Änderungen miteinander ”verträglich“ sind.

62. Wie meldet man sich in einem P2P-Netzwerk an? [1x]

- *atomisches P2P*: da dieses P2P-Netz nicht zentral verwaltet wird gibt es keine wirkliche Anmeldung. Ein neu hinzugekommener Client kann entweder
  - mittels Broadcast-Anfrage prüfen, welche Clients mit welchen Diensten verfügbar sind
  - oder mit einem bekannten Host aus einer Liste potentieller Clients Verbindung aufzunehmen und von diesem Informationen über die Topologie des Netzes einzuholen.
- *benutzerzentriertes P2P*: ist ein atomisches P2P mit einem Server für die Vermittlung der Benutzer. Ein neuer Benutzer meldet sich mit seinen Diesnten bei diesem Server und wird im einfachsten Fall vom Server in ein Verzeichnis eingetragen, in dem alle Benutzer mit ihren Diensten aufgelistet sind.
- *datenzentriertes P2P*: wie benutzerzentriertes P2P, aber Server organisiert Verzeichnisse nach den Daten (Diensten), also wo welche Daten (Dienste) zu finden sind.

63. Wie kann man in einem P2P-Netzwerk suchen? [1x]

- *atomisches P2P*: siehe Frage 62.
- *benutzerzentriertes P2P*: Sucht ein Benutzer einen Dienst, so fragt er beim Server nach, welcher andere Benutzer diesen Dienst zur Verfügung stellt. Um den Dienst zu nutzen, kontaktiert der Rechner des anfragenden Benutzers den Rechner des Benutzers mit dem entsprechenden Dienst.
- *datenzentriertes P2P*: ähnl. wie benutzerzentriertes P2P, aber eben nach Diensten sortiert.

64. Was ist Groupware? [1x] [Haa]

Groupware bezeichnet die Computerunterstützung, die speziell für die Gruppenarbeit entworfen wurde. Groupware kann HW, SW, Dienste und/oder Prozessunterstützung benutzen.

65. Wie kann eine in der Welt verteilte Gruppe ihre Arbeit koordinieren? (oder) Welche Möglichkeiten kennen sie, wie Entwickler mit Hilfe eines verteilten Systems an einen gemeinsamen Projekt arbeiten können? (oder) Wie geschieht die Zusammenarbeit in Groupware [3x]

Bei der Zusammenarbeit einer verteilten Gruppe gelten die selben Einflussfaktoren wie bei nicht verteilten Gruppen:

- *gemeinsame Artefakte* wie Dokumente, Source-Code, usw. müssen in einem zentralen Repository gespeichert, versioniert und deren Entwicklungs-Historie verwaltet werden können. (z.B. RCS Revision Control System, CVS Concurrent Version System, subversion)
- für die *Kommunikation* zwischen den Gruppenmitgliedern müssen explizite Kommunikationskanäle (e-Mail, Chat, o.ä.) und/oder implizite durch Nutzung gemeinsamer Artefakte (z.B. eines gemeinsamen Whiteboards od. gemeinsamer Dokumente) zur Verfügung stehen. (z.B. CSCW-Plattformen wie CURE)
- *Koordination*: die Abstimmungsprozesse innerhalb der Gruppe sollten durch Kommunikation oder gemeinsame Artefakte (z.B. gemeinsamer Projektplan) unterstützt werden. Die Komplexität des Abstimmungsprozess hängt u.a. ab von Größe, Struktur u. Kompetenzverteilung der Gruppe sowie die örtliche u. zeitliche Verfügbarkeit der Gruppenmitglieder.

66. Wie ist die Arbeitsweise von CVS? Was genau ist CVS? [5x]

CVS (Concurrent Version System) ist ein Versionsverwaltungssystem, welches neben der Forderung nach Datenkonsistenz auch noch die Forderungen bez. Gruppenarbeit wie

- *Isolation* – Benutzer können isoliert voneinander an einer Datei arbeiten,
- *Integration* – Änderungen können wieder in die Gruppe integriert werden,
- *Aktualität* – einfach aktuelle Dateiinhalte zu erhalten,
- *Nachvollziehbarkeit* – der Änderungen und
- *Gruppenwahrnehmung* – Benutzer werden über Aktivitäten anderer Benutzer informiert

erfüllt.
Es werden dabei alle Versionen in einer Datei in einem gemeinsamen *Repository* (Datenspeicher) abgelegt. CVS unterstützt den Benutzer dabei bei der Auswahl, Erzeugung und Manipulation von Versionen. Änderungen an bereits abgelegten Versionen sind aber

nicht möglich. Änderungen müssen also in Form einer neuen Version ins Repository eingecheckt werden -> Versionliste (*Nachvollziehbarkeit*). Die Arbeitsweise beinhaltet dabei 3 Phasen:

- *check out*: (CVS *update*) Benutzer wählt zu ändernde Datei aus und kopiert sie auf seinen lokalen Arbeitsbereich (*Aktualität*)
- Die Dateien werden im lokalen Arbeitsbereich verändert (*Isolation*)
- *check in*: (CVS *commit*) Nach Fertigstellung der Änderung wird die Datei wieder ins Repository übertragen und dabei eine neue Version erzeugt (*Integration*)

67. Was passiert, wenn wir die gleiche Datei bearbeiten, kann es zu Problemen kommen? [1x]

Arbeiten mehrere Benutzer an der gleichen Datei entsteht beim einchecken i.A. ein *Versionsbaum*. Dies kann umgangen werden, indem vor dem einchecken über das *update*-Kommando ein lokaler Merge der aktuellen im Repository befindlichen Version und der lokalen Version der Datei durchgeführt wird. Sollte beim lokalen Merge ein Konflikt (semantische Abhängigkeiten, Änderungen im selben Dateibereich) auftreten, der nicht automatisch gelöst werden kann, so ist der User dafür verantwortlich den Merge-Konflikt aufzulösen und eine sinnvolle neue Version der Datei zu erzeugen die dann mit *commit* eingecheckt werden kann.

68. Wann wird es beim Merge-Schritt zu einem Konflikt kommen? [1x]

siehe Frage 67.

69. Was ist beim Merge durch das 2-Wegevergleichsverfahren nicht abgedeckt? [1x]

Es ist unmöglich, aus einer gemeinsamen Ursprungsversion eine Zeile zu löschen. Deshalb Erweiterung zum 3-Wegevergleich wo auch die Vorversion in den Vergleich mit einbezogen wird. Es werden dabei für die beiden parallelen Texte die Unterschiede (hinzugefügte, gelöschte Zeilen) zu ihren Vorversionen ermittelt.

70. In welchem Fall führen parallele Änderungen nicht zu Problemen? [1x]

Wenn es keine semantischen Abhängigkeiten gibt bzw. wenn nicht die selben Dokumentbereiche geändert wurden.

71. Wie wird die Gruppenwahrnehmung in CVS realisiert? [1x]

In CVS gibt es dafür einen allgemeinen Mechanismus, der im Anschluss an commit-Aktionen ausgeführt werden kann. Dieser lässt sich z.B. so konfigurieren, dass beim Erstellen einer neuen Version durch einen Benutzer automatisch interessierte Benutzer über e-Mail darüber informiert werden. Die Benutzer abonnieren dazu den Projektarbeitsbereich indem sie ihre e-Mail Adresse in eine Notifikationsliste eintragen. Das System verschickt dann nach jeder Änderung Informationen über die betroffene Datei und den Versionskommentar an die Abonnenten.

72. Wie sieht es mit BSCW aus, kann man dort eine ähnliche Versionsverwaltung durchführen? [1x]

BSCW (Basic Support for Cooperative Work) bietet ebenfalls eine Versionierung von Dokumenten, wobei bei paralleler Bearbeitung eines Dokuments durch mehrere Benutzer beim einchecken Versionszweige angelegt werden können.

73. Was macht BSCW bei differierenden Versionen? [1x]

siehe Frage 71. Ein Merge von 2 parallelen Versionen wird von BSCW nicht unterstützt.

74. In der letzten Kurseinheit gibt es ein Diagramm, wie man kooperative Systeme einteilen kann. Wie sieht das aus? [1x] [Haa]

Die Raum-Zeit-Matrix von Grudin zeigt Beispiele kooperativer System, die bez. Raum u. Zeit in verschiedenen Situationen zum Einsatz kommen:

Raum/Zeit	gleich (synchron)	verschieden (asynchron), vorhersehbar	verschieden (asynchron), nicht vorher-sehbar
gleicher Ort	Face-to-Face Sit-zungsraum	Organisation von Schichtar-beit	schwarzes Brett
verschiedener Ort (vorhersehbar)	Videokonferenz	E-Mail	kooperatives Schreiben via *Draft Passing*
verschiedener Ort (nicht vorhersehbar)	Mobilfunk-konferenz	asynchrone rechnergestützte Konferenz	Vorgangs-bearbeitung

75. Was verstehen sie unter dem Begriff Awareness im Zusammenhang mit solchen kooperativen Systemen? [1x] [Haa]

Bei der Awareness in gemeinsamen Arbeitsbereichen kann man 2 Arten unterscheiden:

- *Group Awareness*: bezeichnet die Anzeige von Informationen über die Präsenz und Aktivität von Gruppenmitgliedern wie z.B.
  - *Status* – online, offline, usw.
  - *Aktuelle Tätigkeit* – z.B. bearbeitet welches Dokument
  - *Aktuelle Orte* – z.B. in welchen Unterarbeitsbereichen die Person gerade aktiv ist.
- *Workspace Awareness*: bezeichnet die Anzeige von Informationen über den aktuellen Status des gemeinsamen Arbeitsbereiches. Z.B. Infos über aktuelle Strukturen und ihre Entstehung oder über die Aktivitäten.

Ein wichtiger Punkt dabei ist, dass die Awareness gerade in der Arbeit verteilter Gruppen wichtig für die Förderung des Gruppenbewusstseins ist, dass also die Mitglieder z.B. sehen, dass andere Gruppenmitglieder zeitgleich mit einem selbst an etwas arbeiten.