

# Ausarbeitung alter Prüfungsfragen aus Verteilte Systeme

26.11.2007, Christoph Redl

Quelle der Fragen:

<http://www.informatik-forum.at/showthread.php?t=58328>

<http://www.informatik-forum.at/showthread.php?t=55891>

<http://www.informatik-forum.at/showthread.php?t=54729>

<http://www.informatik-forum.at/showthread.php?t=47318>

122 Fragen, 5 unbeantwortet

## Was ist Offenheit

Damit meint man dass das verteilte System an Standards orientiert sein soll. Das bezieht sich auf Protokolle auf den verschiedensten Ebenen, z.B. für die Datenübertragung auf Ebene des Transportlayers selbst (TCP, UDP), aber auch auf höherer Ebene (z.B. SMTP, FTP, etc.) oder im Bereich Security. Vorteil davon ist, dass Komponenten unterschiedlicher Hersteller zusammenarbeiten können und somit auch austauschbar sind.

## Was ist Skalierbarkeit

Die Fähigkeit des Systems, dass es auch bei wachsenden Anforderungen noch einsatzfähig ist. Man kann unterscheiden zwischen Skalierbarkeit mit der Anzahl der User und Ressourcen, geografische Ausdehnung und Erstreckung über mehrere Organisationen als ursprünglich geplant.

## Für was braucht man http

Es ist ein allgemeines Client-Server-Protokoll, das für viele dokumentenbasierte Übertragungen im Internet verwendet wird. Es ist nicht für eine bestimmte Anwendung spezialisiert. Am bekanntesten ist die Übertragung gewöhnlicher Webseiten, es können aber auch ganze Anwendungen auf http aufgebaut werden. Obwohl http selbst zustandslos ist kann man mit Hilfe von Cookies ein zustandsbehaftetes Protokoll simulieren, was für moderne Webabwendungen notwendig ist.

Insgesamt wird http verwendet um von Clients aus bestimmte Services in Anspruch nehmen zu können.

## Objektserver aufzeichnen + Erklärung

Ein Objectserver leitet die Requests zuerst in einen Multiplexer. Dieser gibt sie an den zuständigen Object Adapter weiter (ein Object Adapter ist für die Verwaltung ein oder mehrerer Objekte mit gleichen Activation Policies – siehe unten – zuständig). Der Object Adapter ruft dann den Server-Stub

(Skeleton) des jeweiligen Objekts auf, der wiederum den eigentlichen Methodenaufwurf durchführt. Die Antwort geht genau in die gegengesetzte Richtung durch die einzelnen Komponenten.

## Unterschied RPC und verteilten Objekten in Hinsicht auf Marshalling

RPC übergibt grundsätzlich *by-value*. Sollten doch Referenzen übergeben werden müssen, so werden sie durch Kopien und anschließendes Rückkopieren (also zuerst *by-value*, dann Rückkopieren, genannt *copy-and-restore*) übergeben.

Bei verteilten Objekten ist tatsächlich eine Übergabe als Referenz möglich (solange die Referenz auf ein verteiltes Objekt zeigt, lokale Objekte werden nach wie vor kopiert).

Abgesehen davon ist es bei RMI einfacher einen objektspezifischen Client- oder Server-Stub zu schreiben (zum Beispiel indem als Objektreferenz einfach ein Implementation Handle verwendet wird) als bei RPC, bei dem nur allgemeine Stubs verwendet werden.

## CORBA Object Map

Die Active Object Map wird vom Portable Object Server, bei dem CORBA-Objekte registriert werden, verwendet, um einen Object Identifier aufzulösen und den Skeleton des jeweiligen Servants aufrufen zu können. Alternativ zum Skeleton kann in der Active Object Map für einen bestimmten Identifier auch ein Activator eingetragen sein, der den Servant erst initialisiert (für die Activation Policy, bei der Objekte erst on-demand erzeugt werden).

Alternativ können auch alle Object Identifier auf den selben Servant geleitet werden, und der Servant trifft selbst die Unterscheidung welches konkrete Objekt referenziert wird.

## Jini erklären

Jini ist ein Generative Communication Koordinationsmodell für in Java entwickelte koordinationsbasierte verteilte Systeme. Es basiert auf JavaSpaces als shared Datastore. Data Items werden darin von Prozessen in Form von Tupeln, bestehend aus serialisierten Java-Objekten, abgelegt. Andere (zeitlich und referentiell entkoppelte) Prozesse können auf diese Data Items zugreifen, indem sie einer read-Operation für den Datastore ein Template übergeben. Dieses Template definiert einige (oder alle) Eigenschaften, die das abgefragte Data Item haben soll. Durch dieses Verfahren können einzeln rechnende Prozesse in einem solchen System koordiniert werden. Es ist eine Spezialform von Publish/Subscribe-Systemen.

## Skizzieren sie eine flache und eine hierarchische Prozessgruppe und beschreiben Sie diese kurz

Gruppen von Prozessen werden verwendet um fehlerhafte Prozesse maskieren zu können.

Bei hierarchischen Gruppen gibt es einen Koordinator, der die Arbeit auf alle anderen Prozesse in der Gruppe verteilt, von jedem eine Antwort berechnen lässt und schließlich eine Entscheidung trifft. Normalerweise entscheidet die Mehrheit. Problem dabei ist, dass der Koordinator den Single Point of Failure darstellt. Fällt er aus, so wird eine Election gestartet um einen neuen Koordinator zu wählen.

Bei flachen Gruppen werden spezielle Protokolle benötigt um in der Gruppe eine Mehrheitsentscheidung zu treffen. Eine Möglichkeit ist es, dass jeder Prozess sein Ergebnis an alle anderen schickt. Jeder Prozess erhält somit einen Vektor von Ergebnissen, den er ebenfalls wieder an alle ausschickt. Damit hat jeder der  $n$  Prozesse  $n - 1$  Vektoren, die er elementweise vergleichen und somit eine Mehrheitsentscheidung treffen kann. Die nicht fehlerhaften Prozesse einigen sich somit auf ein Ergebnis, falls maximal  $k$  von  $3k + 1$  Prozessen fehlerhaft sind.

## 4 mögliche Angriffe auf verteilte Systeme

Interception: Daten oder Nachrichten werden unberechtigt ausgelesen bzw. mitgehört

Interruption: Ein Service wird gestört, z.B. durch physische Zerstörung oder DOS-Attacken

Modification: Daten oder Nachrichten werden unberechtigterweise geändert

Fabrication: Ein Unberechtigter schleust Nachrichten oder Daten ein

## Vertraulichkeit, Verfügbarkeit und Integrität erklären

Vertraulichkeit: Es dürften nur berechtigte Benutzer oder Prozesse Zugriff auf ein System und die darin gespeicherten Daten haben

Verfügbarkeit: Die Wahrscheinlichkeit, dass ein Service zu einem beliebigen Zeitpunkt verfügbar ist

Integrität: Meint allgemein dass ein System keine ungültigen Zustände annehmen darf. Im Bereich Security ist vor allem gemeint dass unberechtigte User keine Änderungen durchführen dürfen

## Welche Probleme löst Naming

Die Location-, Relocation- und Migrationstransparenz wird gefördert, indem nicht mehr die physische Adresse einer Entität bekannt sein muss um darauf zuzugreifen.

Es werden für den Menschen leichter merkbare Namen eingeführt, als das bei direkter Verwendung von Adressen der Fall ist.

Man kann auch auf Ressourcen zugreifen, wenn man nur deren Eigenschaften, nicht aber deren genaue Bezeichnung kennt (Attribute-based Naming).

## Activation Policy

Das ist eine Menge von Regeln wie und wann ein Objekt in einem Object Server angelegt werden soll und wie es aufzurufen ist.

Es wird unterschieden zwischen transienten und persistenten Objekten, die entweder gleich beim Start des Object Servers oder beim ersten Aufruf geladen werden. Wann die Objekte zerstört werden ist ebenfalls konfigurierbar. Beim Aufruf von Methoden ist zu unterscheiden ob ein Thread für den ganzen Object Server, pro Objekt, pro Methode oder pro Request gestartet wird.

All diese unterschiedlichen Umgangsweisen mit Objekten werden in Activation Policies zusammengefasst und von einem Object Adapter implementiert. Objekte mit gleichen Activation

Policies werden gemeinsam von einem Object Adapter verwaltet, der die Aufrufe von Methoden dieser Objekte zuerst entgegennimmt, die notwendigen Activation Policies umsetzt, und sie dann an den Server-Stub des Objekts weiterleitet.

## RSA- Algorithmus erklären

Das ist ein asymmetrischer Kryptoalgorithmus. Es werden 2 sehr große Zufallszahlen  $p$  und  $q$  erzeugt, und deren Produkt  $n=p*q$  sowie  $z=(p-1)*(q-1)$  berechnet. Anschließend wird ein Schlüssel (z.B. der öffentliche)  $e$  so gewählt dass, dass  $e$  relativ prim zu  $z$  ist, also  $\text{ggT}(e, z) = 1$ . Der andere Schlüssel  $d$  wird so gewählt dass  $e*d \bmod z = 1$  ist. Der gesamte öffentliche Schlüssel ist dann  $(e, n)$  und der private  $(d, n)$ . Die beiden Schlüssel sind aber gleichwertig, d.h. man kann privaten und öffentlichen auch vertauschen (natürlich nur solange man keinen davon veröffentlicht hat).

Die Verschlüsselung passiert durch  $E(m) = m^e \bmod n$ , die Entschlüsselung durch  $m = D(E(m)) = E(m)^d \bmod n$ .  $m$  ist dabei ein Teil der Nachricht, der als Integer interpretiert im Bereich  $0 - (n-1)$  liegen muss (dadurch kann der Rest bei Division durch  $n$  maximal  $n-1$  sein).

## Stateful und Stateless Server erklären + Beispiel

Ein Stateful Server hält Informationen über die Clients, die mit ihm verbunden sind, ein Stateless Server macht das nicht (zumindest nicht aus funktionalen Gründen, höchstens für Optimierungen). Für einen Stateless Server ist es also, falls er überhaupt Informationen hält, nicht schwerwiegend wenn diese verloren gehen.

Beispiel für einen Stateless Server: ein gewöhnlicher http-Server

Beispiel für einen Stateful Server: ein Server mit einer laufenden JSP-Anwendung, in der ein Warenkorb für den Benutzer gespeichert ist. Obwohl http grundsätzlich zustandslos ist kann durch Cookies und Sessions am Server ein zustandsbehaftetes Verhalten simuliert werden.

Daneben gibt es auch noch hybride Ansätze, bei denen die Informationen über einen Client nur eine bestimmte Zeit lang gehalten werden (soft state).

## FIFO und kausale Konsistenz

Die Bedingung von FIFO-Multicast ist nur, dass die von einem Prozess gesendeten Nachrichten in der Reihenfolge ankommen, in der er sie geschickt hat.

Bei kausaler Konsistenz müssen alle potentiell kausal abhängigen Nachrichten in der Reihenfolge empfangen werden, so dass diese Abhängigkeit gewahrt bleibt. Diese Reihung kann mit Vector Timestamps erzielt werden. 2 Nachrichten sind dann voneinander potentiell kausal abhängig, wenn eine davon gesendet wurde nachdem eine andere empfangen wurde. In diesem Fall muss davon ausgegangen werden, dass zum Erzeugen der 2. Nachricht Informationen aus der 1. verwendet wurden.

Die beiden Begriffe beziehen sich darauf in welcher Reihenfolge die einzelnen Nachrichten bei einem Multicast an die Applikation weitergegeben werden. Beide Varianten lassen aber noch einen gewissen Spielraum frei, weshalb es keine eindeutige Zuordnung gibt. Je nachdem ob sich um einen totally-ordered Multicast handelt, ist die Reihenfolge bei allen Prozessen gleich oder auch nicht.

## Quorumbasierte Protokolle

Das sind Replikationsprotokolle (technische Umsetzungen von Replikationsmodellen), die die sequentielle Konsistenz realisieren. Sie sind den Replicated-Write-Protokollen zuzuordnen, da sie das Schreiben auf ein beliebiges Replikat ermöglichen.

Jedes Datenelement hat eine Versionsnummer. Global ist ein Lese- und ein Schreibquorum ( $Q_r$  und  $Q_w$ ) definiert, so dass bei  $n$  Replikaten gilt:

- $Q_r + Q_w > n$  um Lese-/Schreibkonflikte zu vermeiden
- $Q_w > n/2$  um Schreib-/Schreibkonflikte zu vermeiden

Bei einem Lesezugriff werden mindestens  $Q_r$  Replikate gelesen und das Ergebnis mit der aktuellsten Versionsnummer als das richtige zurückgegeben. Bei einem Schreibzugriff werden mindestens  $Q_w$  Replikate geschrieben (die zuerst alle auf die aktuellste Version unter diesen  $Q_w$  Replikaten gebracht werden).

## 4 Anforderungen für Fault Tolerance laut Kopetz

Availability: Die Wahrscheinlichkeit, dass ein System zu einem beliebigen Zeitpunkt verfügbar ist.

Reliability: Bezieht sich auf Zeitspannen und sagt aus, wie viel Zeit zwischen 2 Failures durchschnittlich vergeht.

Safety: Failures eines Systems dürfen keine Katastrophen zur Folge haben. Dazu zählt die Gefährdung von Gesundheit und Leben sowie schwerwiegende wirtschaftliche Folgen.

Maintainability: Failures sollten schnell und leicht (im Idealfall automatisch) behoben werden können.

## Berkeley Algorithmus

Das ist ein Algorithmus zur Synchronisation von Uhren untereinander, jedoch nicht mit einer externen Zeitquelle.

Ein Koordinator schickt dazu seine lokale Zeit an alle anderen aus. Diese senden ihre (positive oder negative) Abweichung an den Koordinator zurück. Er berechnet daraufhin eine durchschnittliche Abweichung und schickt diese wieder an alle aus (auch an sich selbst). Schließlich können alle ihre Uhren um diese Abweichung vor- oder zurückstellen (wobei das Zurückstellen durch Verlangsamung der Uhr über einen längeren Zeitraum geschieht).

## Vervollständigen Sie das Koordinationsmodell (referential/temporal) und nennen Sie jeweils ein Beispiel

Je nachdem ob die Prozesse einander namentlich kennen und/oder zur gleichen Zeit exekutieren müssen verschiedene Modelle angewendet werden um ihre Aktivitäten zu koordinieren.

Zeitlich und referentiell gekoppelt: Direkte Koordination über eine gewöhnliche Verbindung zwischen den Prozessen.

Nur referentiell gekoppelt: Mailboxen, in denen Nachrichten für einen Prozess abgelegt werden können, der sie von dort lesen kann.

Nur zeitlich gekoppelt: Publish/Subscribe-Systeme, bei denen Nachrichten mit einem gewissen Typ versehen werden. Diese werden an alle Prozesse zugestellt, die sich dafür interessieren (indem sie sich vorher dafür registriert haben). Dieser Ansatz wird als Meeting-based bezeichnet.

Beides entkoppelt: Generative Communication durch einen Shared Dataspace. Dort können von Prozessen Daten in Form von Tupeln eingetragen werden. Andere Prozesse können von dort für sie interessante Tupel durch ein Pattern Matching-System abfragen. Das System ist eine Spezialform von Publish/Subscribe-Systemen.

## **Erklären Sie mit Skizze die static und dynamic Invocation im Zusammenhang mit RMI**

Ein statischer Aufruf sieht im Sourcecode wie ein lokaler Methodenaufruf aus:

```
ret_value = method1(arg1, arg2, ...);
```

Ein dynamischer Aufruf wird hingegen über eine spezielle Call-Methode realisiert, der die Parameter sowie der Identifier/Name der aufzurufenden Methode übergeben wird:

```
ret_value = callRemoteMethod(getId(„method1“), arg1, arg2, ...);
```

Vorteil der statischen Variante ist, dass es einem lokalen Aufruf aus Programmiersicht näher kommt und dass der Compiler diverse Überprüfungen durchführen kann. Beim dynamischen Aufruf kann es leichter zu Laufzeitfehlern kommen, dafür muss das Clientprogramm nicht neu übersetzt werden, wenn sich die Interfaces geändert haben (mit gewissen Einschränkungen: wenn verwendete Methoden gelöscht wurden, dann natürlich schon). Außerdem ist ein dynamischer Aufruf günstig, wenn man zur Übersetzungszeit nicht weiß welche Methoden aufgerufen werden sollen, zum Beispiel bei einem Object Browser.

## **Nennen Sie 2 Aufgaben von Mutual Exclusion**

Es soll der exklusive Zugriff auf eine Resource ermöglicht werden, so dass sich 2 Clients nicht gegenseitig stören während sie diese Resource verwenden.

Neben dem gegenseitigen Ausschluss ist es auch wichtig dass jeder, der sich für die Resource interessiert, diese auch irgendwann verwenden kann (Vermeidung von Starvation) und dass die Zuteilung fair (z.B. FIFO oder prioritätsgesteuert) funktioniert.

## **Erklären Sie einen zentralistischen Algorithmus für Mutual Exclusion**

Es gibt einen Koordinator der den Zugriff auf die Resource regelt. Jeder der die Resource verwenden will teilt dies dem Koordinator mit. Falls die Resource gerade frei ist antwortet er mit einem Ack. Falls sie nicht frei ist, dann antwortet er vorerst nicht und trägt die Anfrage in eine Queue ein. Sobald ein Client eine Resource nicht mehr benötigt gibt er sie durch eine Nachricht an den Koordinator wieder

frei . Dieser kann dann dem nächsten Prozess in der Queue (falls sie nicht leer ist) das Zugriffsrecht erteilen.

## **Skizzieren Sie den Aufbau eines multi-threaded Servers, welche Vorteile bringt ein solcher**

Es gibt einen Dispatcher-Thread der auf neue Clientanfragen wartet. Für jede Anfrage wird ein Worker-Thread gestartet, der sich von nun an um diese Verbindung kümmert (vom Client liest und zu ihm schreibt). Der Dispatcher kann inzwischen auf weitere Anfragen warten.

Vorteil: Es können mehrere Clients gleichzeitig abgefertigt werden, was insgesamt zu einem höheren Durchsatz führt (z.B. falls im Worker-Thread blockierende Systemaufrufe stattfinden). Insgesamt ist so ein paralleler Server (der grundsätzlich auch in einem Thread implementiert werden kann) als multithreaded Server leichter zu implementieren.

## **Erklären Sie die Begriffe POA, ORB und Servant im Zusammenhang mit CORBA**

**ORB: Object Request Broker: Ist der Kern von CORBA und ist für das Multiplexing der eingehenden Request auf die verschiedenen Object Adapter zuständig.**

**POA: Portable Object Adapter: Ein portabler (d.h. ist an andere ORB-Instanzen übertragbar) Object Adapter. Ein Object Adapter ist für die Verwaltung ein oder mehrerer Servants (das sind Laufzeitobjekte) zuständig, die die gleiche Activation Policy (Regeln, wann und wie ein Objekt zu instanzieren und aufzurufen ist) verwenden.**

**Servant: Ein Laufzeitobjekt. Zur Unterscheidung von Compiletime-Objects (Objekte, die sich direkt durch das Instanzieren von Klassen der Programmiersprache ergeben) werden beliebige Kapselungen von Daten und Operationen, die sich dem Benutzer gegenüber wie ein Objekt präsentieren (was durch Object Adapter erreicht wird) als Servants bezeichnet.**

**TODO: Richtig?**

## **Erklären Sie Callback und Polling beim asynchronen Aufruf unter CORBA, was ist es und Skizze**

Durch diese beiden Varianten kann ein verteiltes Objekt mit CORBA serverseitig genauso implementiert werden wie bei einem synchronen Aufruf. Nur der Client-Stub arbeitet anders.

Bei Callbacks wird jeder Aufruf durch 2 Aufrufe ersetzt:

- Registrierung eines Callback-Handlers, der den Rückgabewert der Methode als Parameter erhält
- Nicht-blockierender Aufruf der Remote Methode ohne Rückgabewert

Der Stub kümmert sich darum, dass beim Eintreffen einer Antwort der Callback-Handler aufgerufen wird.

Beim Pollig gibt es ebenfalls 2 Methoden. Es wird dabei im Unterschied zu Callbacks kein Handler installiert, sondern die Applikation fragt mit einer einen Methode regelmäßig bei der Middleware nach, ob bereits eine Antwort eingetroffen ist. Diese Methode liefert in diesem Fall den Rückgabewert des Servers.

## **Erklären Sie die Begriffe Prozess und Thread und grenzen Sie die beiden Begriffe ab**

Ein Prozess ist eine Einheit für Ressourcenzuteilung und unabhängige Ausführung. Ein Thread ist dagegen nur eine Einheit für unabhängige Ausführung. Mehrere Threads in einem Prozess teilen sich aber die Ressourcen (z.B. den Speicher). Damit ist eine Kommunikation zwischen Threads ohne Mode-Switch zum Kernel und Context-Switch zum anderen Prozess möglich, was zu wesentlich höherer Performance führt.

## **Beschreiben Sie 3 HTTP-Requests**

Header: Nur die Metadaten zu einem Dokument abfragen

Get: Abfrage eines Dokuments, die keine Änderungen an der Resource vornimmt

Post: Daten zu einem Dokument hinzufügen

Put: Ein neues Dokument anlegen

Delete: Eine Resource löschen

## **Erklären Sie den Unterschied zwischen einer persistent und non-persistent HTTP-Connection**

Non-persistent: Eine TCP-Verbindung wird aufgebaut, der Request wird übertragen, der Response wird übertragen und die TCP-Verbindung wird wieder abgebaut.

Persistent: Die gleiche TCP-Verbindung kann für mehrere Request/Response-Paare verwendet werden (z.B. um eine HTML-Seite inklusive Bilder zu übertragen). Außerdem ist es möglich, gleich mehrere Requests hintereinander abzuschicken, ohne zuerst auf die Antwort auf den ersten Request warten zu müssen.

## **Was ist SSL/TSL, ordnen Sie SSL in den Internet Protocol Stack ein**

SSL steht für Secure Socket Layer und verschlüsselt eine TCP-Verbindung. SSL befindet sich direkt über dem Transportlayer.

Der Ablauf ist wie folgt:

- Der Client teilt dem Server seine Verschlüsselungs- und Kompressionsmöglichkeiten mit
- Der Server wählt welche davon aus und teilt dies dem Client mit
- Der Server authentifiziert sich mittels Zertifikat
- Eventuell authentifiziert sich auch der Client mittels Zertifikat
- Der Client erzeugt eine Zufallszahl, signiert sie, verschlüsselt sie mit dem Public Key des Servers und sendet sie diesem
- Beide leiten aus dieser Zufallszahl einen Session Key ab, der ab nun verwendet wird

## Was ist eine digitale Signatur und wozu dient sie?

Damit wird dem Empfänger einer Nachricht garantiert, dass der Sender diese Nachricht wirklich in der Form abgeschickt hat, in der er sie empfangen hat. Der Sender verschlüsselt dazu den Hash-Wert der Nachricht (oder die Nachricht selbst – das ist jedoch unnötig aufwendig) mit seinem privaten Schlüssel. Der Empfänger rechnet den Hash-Wert der Nachricht nach, entschlüsselt den mitgeschickten Hash-Wert mit dem öffentlichen Schlüssel des Senders, und vergleicht die Ergebnisse. Bei Gleichheit wurde die Nachricht wirklich vom Sender abgeschickt.

## Skizzieren sie ein konkretes Beispiel, wie eine Signatur erzeugt und verifiziert wird

Sender:

1. Erzeuge Nachricht  $m$
2. Berechne Hashwert  $h = H(m)$
3. Verschlüsse  $h$  mit dem eigenen privaten Schlüssel:  $E(h, K_{s-})$
4. Übertrage  $m$  und  $E(h, K_{s-})$

Empfänger:

1. Empfange  $m$  und  $E(h, K_{s-})$
2. Rechne Hashwert nach:  $h_2 = H(m)$
3. Entschlüsse Signatur mit öffentlichem Schlüssel des Senders:  $h' = D(E(h, K_{s-}), K_{s+})$
4. Vergleiche  $h'$  mit  $h_2$
5. Bei Gleichheit war die Verifizierung erfolgreich

## Beschreiben Sie die primary-based Replikationsprotokolle im Zusammenhang mit Konsistenz

Es gibt dabei immer nur ein Replikat, auf dem eine Änderung durchgeführt werden darf. Dieses kann aber für jedes Datenitem ein anderes Replikat sein.

Der Client, der eine Änderung durchführen will, wendet sich dabei an den Primary (oder im Falle von Remote-Write-Protokollen an einen beliebigen Server, den den Request an den Primary weiterleitet). Am Primary werden die Änderungen durchgeführt und an alle anderen Replikate propagiert. Anschließend wird dem Client ein Ack zurückgeschickt (im Falle der asynchronen Variante kann das Ack auch sofort zurückgeschickt werden).

Durch dieses Protokoll wird die sequentielle Konsistenz garantiert.

Eine weitere Alternative ist es, dass ein Datenitem seinen Primary Server wechseln kann. Der Client wendet sich dabei immer einen beliebigen Server, der dann der neue Primary für dieses Item wird. Er schickt ein Ack zurück, ändert das Item und propagiert die Änderungen. Die synchrone Variante ist in diesem Fall nicht sinnvoll, weil sie keinerlei Vorteile zur Remote-Write-Methode hat.

## Nennen Sie 4 client-centric Konsistenzmodelle und beschreiben Sie diese

Monotonic Read: Wenn ein Client ein Datum gelesen hat, dann kann er später nur noch die gleiche oder eine aktuellere Version lesen (egal bei welchem Replikat)

Monotonic Write: Wenn ein Client schreibt, dann überschreiben zukünftige weitere Schreiboperationen nur die zuvor geschriebene, oder eine noch aktuellere Version, aber niemals eine ältere.

Read your Writes: Wenn ein Client geschrieben hat, dann liest er später nur die zuvor geschriebene oder eine noch aktuellere Version, niemals eine ältere

Writes follow Reads: Wenn ein Client ein Datum gelesen hat, dann überschreiben alle zukünftigen Schreibzugriffe das Datum in der zuvor gelesenen oder einer noch aktuelleren Version.

## **Bully-Algorithmus erklären und wozu wird er verwendet?**

Damit kann ein Koordinator (ein Prozess, der irgendeine spezielle Aufgabe erfüllen soll) gewählt werden.

Ein Prozess, der feststellt dass der aktuelle Koordinator nicht mehr verfügbar ist, startet eine Election. Dazu schickt er Nachrichten an alle Prozesse mit einer höheren Nummer als er selbst. Diese Prozesse schicken die Nachrichten ebenfalls an alle Prozesse mit einer höheren Nummer weiter.

Falls ein Prozess A eine Nachricht von einem Prozess B mit einer niedrigeren Nummer erhält antwortet er diesem und lässt damit B als neuen Koordinator ausscheiden. Der Prozess, der keine Antworten erhält, wird neuer Koordinator. Er teilt dies allen über einen Broadcast mit.

## **IDL**

Das ist eine programmiersprachenunabhängige Definitionssprache für Interfaces, die von Remote Objekten oder Prozeduren implementiert werden. Aus diesen Definitionen können mit speziellen IDL-Compiler automatisch die Server- und Client-Stubs für verschiedene Programmiersprachen erzeugt werden, die dann mit dem Client- und Servercode gelinkt werden können. Man erspart sich damit die manuelle Implementierung der Stubs, die ohnehin immer sehr ähnlich aussehen. Die unterscheiden sich im wesentlichen nur durch Methodennamen und Parameteranzahl und -typen.

## **Was ist Stream-oriented Communication + Bsp**

Darunter versteht man Datenübertragungen bei denen nicht ganze Einheiten von Informationen geblockt gesendet werden, sondern eine Folge von sehr vielen und sehr kleinen zusammenhängenden Einheiten. Neben Obergrenzen für die End-to-End-Delays sind hier in der Regel auch Untergrenzen gefordert, da die Zeit eine wesentliche Rolle für die Nützlichkeit spielt. Bei gewöhnlichen Daten-orientierten Übertragungen ist dagegen die Korrektheit durch schlechte Performance nicht gefährdet.

Beispiel: Eine Live-Video-Übertragung über das Internet.

## **Was heißt QoS und wie realisiert?**

QoS steht für Quality of Service und bezeichnet die nicht-funktionalen Anforderungen an ein verteiltes System. Dazu zählen z.B. Security Policies, maximales End-to-End-Delay, Übertragungsgeschwindigkeit (Bits/Sekunde), Untergrenzen für die Verzögerung (z.B. zeitabhängigen Übertragungen).

Realisiert wird QoS durch Einstellungsmöglichkeiten in der Middleware, mit denen Parameter entsprechend gesetzt werden können. Die Middleware realisiert diese Services ihrerseits durch Verwendung der Features von darunter liegenden Protokollen, die entsprechend genutzt und kombiniert werden können (z.B. das Prioritätsflag von IP-Paketen).

Neben Ausnutzung von Services der unteren Layer kann die Middleware auch selbst Mechanismen zur Sicherung von QoS anbieten. Dazu zählt zum Beispiel ein Buffer, mit dem Jitter (unregelmäßige Verzögerungen der Pakete) ausgeglichen werden kann.

## TokenBucket

TODO

## Corba Event-Service erklären und Pull/Push Modelle dazu.

TODO

## Diffie-Hellman im Detail erklären.

Der Algorithmus wird zum Schlüsseltausch (eines symmetrischen Schlüssels) über einen unsicheren Kanal verwendet.

Ein Partner erzeugt 2 große Zahlen  $g$  und  $n$  sowie eine geheime Zufallszahl  $x$ . Er berechnet  $g^x \bmod n$  und schickt  $(g, n, g^x \bmod n)$  zum anderen Partner. Dieser erzeugt sich auch eine geheime Zufallszahl  $y$  und berechnet  $g^y \bmod n$ , was er zum ersten Partner zurückschickt.

Es kann nun jeder den symmetrischen Schlüssel berechnen, indem er das vom Partner empfangene Ergebnis mit seiner eigenen Zufallszahl potenziert und  $\bmod n$  rechnet. Das ist auf die Tatsache zurückzuführen, dass  $(g^x \bmod n)^y \bmod n = (g^y \bmod n)^x \bmod n = g^{(xy)} \bmod n$  ist. Die Sicherheit beruht darauf, dass der Schlüssel selbst nie übertragen wurde und aus  $g^x \bmod n$  bzw.  $g^y \bmod n$  bei ausreichend großen Zahlen  $x$  und  $y$  auch praktisch nicht errechenbar ist.

## Event-Mechanismus bei JINI und Beispiel mit 2 Clients.

Jini verwendet JavaSpaces als Shared Dataspace. Darin werden Data Items, das sind Tupel bestehend aus serialisierten Java-Objekten, abgelegt, wenn ein Client eine write-Operation durchführt. Andere Prozesse können aus diesem JavaSpace Data Items abfragen, indem sie ein Template zusammenstellen. Jini sorgt dafür, dass diese Prozesse dann die passenden Tupel aus dem JavaSpace geliefert bekommen.

Beispiel:

Client 1:

```
Tupel t = new Tupel(){
    public Integer value1 = new Integer(10);
    public Integer value2 = new Integer(20);
}
JavaSpace.write(t);
```

Client 2:

```
Tupel template = new Tupel(){
    public Integer value1 = null;
    public Integer value2 = new Integer(20);
}
```

```
// Diese Zeile liefert das Tupel, das von Client 1 eingetragen wurde
Tupel matchingtupel = JavaSpace.read(template);
```

## Active Replication bei Replication Write.

Es kann auf ein beliebiges Replikat geschrieben werden. Dieses Replikat schickt die Änderungen dann an alle anderen weiter, und zwar in Form von Operationen (nicht das Ergebnis der Operationen, sondern diese selbst werden weitergereicht). Die anderen Replikate berechnen die Operationen ebenfalls und aktualisieren ihre Datenbasis.

Falls dieses Verfahren nicht anwendbar ist (z.B. weil nicht deterministische Operationen, wie etwa `time()` oder `rnd()` verwendet werden) kann es auch mit der Übertragung des Operationsergebnisses ab diesem Zeitpunkt kombiniert werden.

## Cristian Algorithmus

Dies ist ein Algorithmus zur Synchronisation von Uhren mit einer externen Zeitquelle (z.B. einer Atomuhr). Er ist eine Implementierung des NTP (Network Timing Protocol).

Ein Rechner A, der sich synchronisieren will, schickt dazu eine Anfrage an einen Rechner B mit genauerer Zeit. A merkt sich dabei die Zeit beim Senden ( $t_1$ ). B merkt sich 2 Timestamps:  $t_2$  wann er die Nachricht empfangen hat und  $t_3$  kurz vor dem Absenden der Antwort. Beim Eintreffen der Antwort erhält auch A einen zusätzlichen Timestamp  $t_4$ .

Es kann durch  $((t_4 - t_1) - (t_3 - t_2)) / 2$  auf die Übertragungszeit  $t_u$  geschlossen werden. Die positive oder negative Abweichung von den Uhren bei A und B kann somit durch  $t_2 - t_1 - t_u$  berechnet werden.

## Was ist Access Transparency

Das meint dass ein Prozess auf Ressourcen zugreifen kann ohne genau wissen zu müssen wie diese gespeichert sind. Ein Beispiel dafür ist der Zugriff auf Dateien mittels URLs, wodurch es nicht mehr notwendig ist das lokale Dateisystem des Servers zu kennen. Ein weiteres Beispiel ist die Abstraktion von Big- und Little-Endian durch eine gemeinsame Repräsentation in der Middleware.

## Wie könnte Replication Transparency in Remote Object System realisiert werden mit Bsp.

Eine Menge von Replikaten wird zu einer Gruppe zusammengefasst, wobei ein Replikat der Koordinator dieser Gruppe ist. Aufgerufen wird eine Methode immer bei allen Replikaten um sicherzustellen, dass alle den aktuellen Status haben (etwa über Reliable Multicast).

Falls dieses replizierte Objekt jedoch selbst wieder eine Methode eines anderen distributes Objects aufruft, dann wird das nur vom Koordinator der Gruppe gemacht. Andernfalls würde die Methode zu oft aufgerufen werden. Auch die Antwort eines Aufrufs wird nur vom Koordinator zurückgeschickt.

Beispiel:

A	B1	C1
	B2	C2
	B3	

Falls von A aus ein Aufruf in B erfolgt, so ruft A alle Replikate in B (B1, B2, B3) auf. Wenn B nun seinerseits wieder eine Methode von C aufruft, so wird das nur vom Koordinator von B (z.B. B1) gemacht (und zwar bei C1 und C2). Andernfalls (wenn alle 3 Replikate von B den Aufruf durchführen würden) wäre die Methode von C zu oft aufgerufen worden. Die Antworten werden nur vom Koordinator zurückgegeben, also von z.B. C1 zu allen 3 Replikaten von B, und von B1 zu A.

## Asymetrische und symetrische Verschlüsselung, Vor- Nachteile und Beispiel

Bei symetrischer Verschlüsselung verwenden beide Kommunikationspartner den gleichen Schlüssel zur Ver- und Entschlüsselung. Vorteil: viel schneller, Nachteil: sehr viele Schlüssel benötigt.

Bei asymetrischer Verschlüsselung hat jeder Partner einen Schlüssel zum Ver- und einen zum Entschlüsseln. Der zum Verschlüsseln ist öffentlich, der zum Entschlüsseln privat. Die beiden Schlüssel sind unterschiedlich. Vorteil: nur  $2n$  Schlüssel für  $n$  Kommunikationspartner, Nachteil: viel langsamer.

Um die Vorteile beider Verfahren zu nutzen gibt es hybride Verfahren, bei denen mittels asymetrischer Verschlüsselung ein symetrischer Schlüssel, ein sogenannter Session Key, ausgetauscht wird.

## Was ist Middleware und 3 Services?

Das ist eine Softwarekomponente die viele oft von verteilten Systemen benötigte Services zusammenfasst und in parametrisierbarer Form anbietet. Man muss daher gleiche Funktionalitäten nicht immer neu implementieren. Die Schicht ist logisch zwischen Application- und Transportlayer einzuordnen. Services von Middleware sind z.B. Verschlüsselung (oder allgemein Security), Replikation, Access-Transparent, Naming, etc.

Beispiele für Services:

- Verschlüsselung
- Replikationsverwaltung und Konsistenz

- Naming-Services

**Möglichkeiten für update notification (da waren wir auch nicht sicher was sie genau wollen, ich habe geschrieben: just update notification, send updated data, active replication also send update operation)**

TODO

## Cooperative Caching

Dies ist eine Form von Caching, bei der mehrere Caches in einem Netzwerk existieren, die aber nicht periodisch abgeglichen wurden. Stattdessen fragt ein Cache, der eine Anfrage nicht selbst beantworten kann, bei seinen Nachbarchaches nach, ob diese das Ergebnis bereits kennen. Ist dem so, so übernimmt er das Ergebnis des Nachbarchaches in seinen eigenen Cache und beantwortet die Anfrage. Nur wenn auch seine Nachbarn kein Ergebnis liefern können muss der Originalserver kontaktiert werden.

Eine Spezialform davon ist ein hierarchisches Modell, bei dem die Caches im Falle eines Cache Miss den übergeordneten Cache befragen können. Das Problem dabei ist aber, dass Caches auf höherer Ebene mit sehr großen Datenmengen umgehen können müssen.

## Home-Based Approach

Das ist eine Möglichkeit wie Naming für mobile Entitäten umgesetzt werden kann und die Location- und Relocation-Transparent zu gewährleisten. Jede Entität hat dabei einen Home-Server, der eine mehr oder weniger fixe Adresse hat (der Home-Server kann von einem Client über ein anderes Naming-Server gefunden werden).

Immer wenn eine mobile Entität eine neue Adresse bekommt meldet sie diese an den Home Server. Clients können somit beim Home Server immer die aktuelle Adresse der mobilen Entität erfragen und diese dann kontaktieren.

## Persistent und Transient Objects

Persistente Objekte werden von einem Object Server erzeugt und überleben dessen Verfügbarkeit. Das bedeutet, nachdem der Object Server ausgefallen oder heruntergefahren ist, sind die Objekte weiterhin existent (z.B. als Files auf einer Platte) und können von dort von einem anderen Object Server gelesen und wieder angeboten werden.

Transiente Objekte existieren dagegen nur solange der Server läuft, der sie ursprünglich erzeugt hat.

## Zone Transfer

Darunter versteht man das Übertragen einer DNS-Tabelle für eine Zone zu einem anderen Nameserver. Benötigt wird das, wenn der primäre Nameserver einer Zone aktualisiert wird und die Replikate (wovon es mindestens eines für jede Zone geben muss) die Änderungen übernehmen

müssen. Sekundäre Nameserver fragen für einkommende Request beim primären nach den neuseten Daten über diese Zone.

## Token Ring Algorithmus

Das ist ein Algorithmus zur Gewährleistung der Mutual Exclusion. Ein Token (eine spezielle Nachricht) wird in einem logischen Kreis von Prozess zu Prozess weitergereicht. Wer den Token gerade inne hat ist berechtigt auf die Resource zuzugreifen. Wer die Resource nicht (oder nicht mehr) benötigt reicht den Token weiter.

Probleme dabei sind die Erkennung wann ein Token verloren gegangen ist sowie die Vermeidung von doppelten Token.

## Was ist ein MIME-Type?

Multipurpose Internet Main Exchanged: Der Typ eines embedded Documents (ein Dokument, das mittels Verweis in ein anderes Dokument eingebunden wird). Damit wird dem Browser mitgeteilt wie das Dokument zu interpretieren ist (entweder durch den Browser selbst oder über ein externes Programm). Es wird unterschieden zwischen Ober- und Untertypen. Beispiele sind text/html, text/xml, img/jpeg.

## 3 Skalierungsprobleme

- Es werden mehr User und Ressourcen in das System inkludiert als das ursprünglich geplant war: Broadcasts überlasten das Netz
- Das System dehnt sich geografisch aus: die Latenzzeiten werden höher und synchrone Kommunikation ist u.U. nicht mehr erträglich
- Es werden mehr Organisationen eingebunden: die Policies widersprechen sich

## 3 Skalierungsmethoden mit Beispiel

- Replikation: Ressourcen werden zur Performance-Steigerung vervielfältigt, z.B. über Content Delivery Networks
- Verwendung von asynchronen Methoden um Latenzzeiten zu verbergen, z.B. Asynchronous RPC
- Caching als Spezialform von Replikation: Daten werden am Client zwischengespeichert, z.B. die Caches von Web-Browsern

## Vektor Timestamps, wie funktionieren sie? Welches Problem von Lamport Timestamps korrigieren sie?

Jeder Prozess hat einen Vektor der Länge  $n$ , wenn es  $n$  Prozesse gibt. Jedes Element  $j$  des Vektors bei Prozess  $i$  sagt aus, wie viele Ereignisse von Prozess  $j$  er schon gesehen hat.

Bei jedem Ereignis in Prozess  $i$  zählt er sein eigenes Element  $i$  hoch. Sendet er eine Nachricht, so hängt er den Vektor an die Nachricht an. Der Empfänger kann dann elementweise mit seinem eigenen Vektor vergleichen und jeweils den höheren Wert übernehmen, um so seinen eigenen Vektor zu aktualisieren. Falls sich der empfangene Vektor in mehr als einem Element von dem eigenen unterscheidet, so muss die Nachricht zuerst zurückgestellt werden, da dazwischen noch andere Nachrichten fehlen. So ist es garantiert, dass Nachrichten so zur Applikation geliefert werden, wie sie potentiell voneinander abhängen.

Im Unterschied zu Lamport Timestamps kann durch Vergleichen der Vektoren festgestellt werden, ob 2 Nachrichten potentiell voneinander abhängen. Bei Lamport Timestamps gilt nur:  $a < b \rightarrow T(a) < T(b)$ , bei Vektor Timestamps gilt die Implikation dagegen in beide Richtungen.

## **Sie implementieren einen Webserver, warum verwendet man persistent Connections?**

Um zu vermeiden dass für jedes Request/Response-Paar eine eigene TCP-Verbindung erzeugt werden muss. Man spart damit also Overhead ein.

## **Was sind Intermittend Faults mit Beispiel?**

Das sind Faults die abwechseln an- und abwesend sind, z.B. ein Wackelkontakt.

## **Erklären Sie Authorisation und Authentication, was ist Unterschied?**

Authentication ist die Überprüfung der Identität eines Users oder Prozesses (ist er der, für den er sich ausgibt?).

Authorisation ist die Überprüfung, ob jemand, dessen Identität bereits überprüft wurde, berechtigt ist ein Service in Anspruch zu nehmen.

## **Was sind Session Keys/Verbindungsschlüssel?**

Das sind symmetrische Schlüssel die nur für die Dauer einer Session gültig sind. Bei der nächsten Session wird ein neuer Schlüssel verwendet.

## **Ihr Chef schlägt vor, timestamps als Session keys zu verwenden. Ist das eine gute Idee?**

Nein, Timestamps können leicht erraten werden.

## **Wie implementieren sie einen performanten Fileserver?**

Mittels mehreren Threads. Dadurch können mehrere Anfragen gleichzeitig abgefertigt werden, man muss nicht warten bis die blockierenden Zugriffe auf die Dateien fertiggestellt sind, sondern kann in der Zwischenzeit die nächste Anfrage entgegennehmen.

8)a) Unterschied zwischen normalem und asynchronem RPC mit Skizzen erklären

Synchron: Der Sender muss warten bis der RPC abgeschlossen ist, also bis der Server die Anfrage bearbeitet hat und das Ergebnis eingetroffen ist.

Asynchron: Der Sender kann inzwischen weiterarbeiten und wird beim Eintreffen der Antwort über einen Callback-Handler darüber informiert (wird also unterbrochen).

Es gibt noch Hybridformen, bei denen der Client zumindest den Erhalt der Nachricht durch den Server abwartet.

## Unterschied zwischen RPC und DOS (distributed object system)

Bei RPC werden nur einzelne Prozeduren aufgerufen wie das bei prozeduralen Programmiersprachen gemacht wurde. Die einzigen Daten, auf denen solche Prozeduren operieren, kommen aus den Parametern. Bei DOS werden hingegen Methoden von Objekten aufgerufen, die auch einen Status kapseln.

Beim Marshalling besteht insofern ein Unterschied als dass RPC nur call-by-value und copy-and-restore unterstützt, während DOS tatsächlich globale Referenzen (auf distributed Objects) ermöglichen.

## 2 Möglichkeiten, Client und Object zu binden bei remote object invocation

Explizit oder implizit.

Explizit: Der Client fragt eine globale Objektreferenz ab, sucht die Referenz auf den zugehörigen Client Stub und ruft über diesen schließlich Methoden auf.

Implizit: Der Client ruft Methoden (scheinbar) direkt über die globale Objektreferenz ab. Das Binden zum Client Stub geschieht im Hintergrund automatisch, ist aber nicht in allen Programmiersprachen notwendig. Es geht nur, wenn eine Form von Operatorenüberladung, etwa die Überladung des ->-Operators in C++, unterstützt wird.

## Welche zwei Mechanismen benötigt man, um Namensauflösung durchführen zu können?

- Resolution: Während der Auflösung muss es eine Möglichkeit geben ein Verzeichnis zu durchsuchen, den Identifier des nächsten zuständigen Namensservers zu finden und so den Namensraum abzusteigen.
- Closure Mechanismus: Ein Punkt, bei dem die Namensauflösung starten kann. Dies ist bei DNS zum Beispiel die well-known-Adresse der 13 Root-Server. Bei Unix-Filesystemen die Tatsache, dass das Root-Verzeichnis immer über inode 0 zugreifbar ist.

## **Unterschied zwischen rekursiver und iterativer Namensauflösung erklären**

Rekursiv: Die Anfrage wird an den Top-Level-Nameserver gestellt. Dieser löst die Adresse so weit auf wie es ihm möglich ist (das Ergebnis ist die Adresse eines anderen Nameservers für eine Subdomain) und übergibt die restliche Anfrage automatisch an den nächsttiefer gelegenen Nameserver. Dieser setzt in der gleichen Form fort, bis die Adresse ganz aufgelöst ist. Das Ergebnis wird wieder nach oben gereicht, bis der Top-Level-Server das Ergebnis an den Client schicken kann.

Iterativ: Der Top-Level-Server löst seinen Teil auf und schickt das Ergebnis (Adresse des Nameservers einer Subdomain und verbleibende Anfrage) sofort an den Client zurück. Dieser muss sich nun darum kümmern, dass auch der Rest aufgelöst wird.

## **Effekt von Replication und Caching auf Name Service**

Replication: Zur Erhöhung der Availability muss in DNS jeder Namensserver zumindest doppelt vorhanden sein.

Caching: Wird massiv verwendet, da sonst die Top-Level-Server komplett überlastet wären. Bei rekursiven Anfragen ist Caching effektiver einsetzbar, jedoch wird es auf den oberen Ebenen in DNS trotzdem nicht eingesetzt, da die Server auf dieser Ebene den höheren Aufwand nicht abfertigen könnten.

Insgesamt führen diese Methoden dazu, dass nicht jede Anfrage immer bis zum Schluss entlang der Hierarchie weitergereicht werden müssen. Oft können Namen bereits lokale vollständig aufgelöst werden.

## **CORBA Zeugs: unter anderem das Event-Service beschreiben und noch einiges mehr**

TODO

## **Weak und strong mobility erklären, Unterschied zwischen sender-initiated und receiver-initiated migration**

Weak: Nur nicht laufender Code wird übertragen, so wie z.B. bei einem Java-Applet. Es kann dann nur von Anfang an, oder von einem von wenigen Startpunkten aus gestartet werden.

Strong: Auch laufender Code kann übertragen werden, wofür aber auch die Datensegmente und sonstige Ressourcen mitgegeben werden müssen (der Aufwand ist viel größer).

Sender/Receiver-initiated: je nachdem von wem die Migration angefordert wird. Receiver z.B. bei Downloads von Java-Applets, Sender z.B. beim Herunterfahren eines Servers, der seine laufenden Services zuerst auf einen sekundären Server übergibt.

## **Mailserver löscht Mails größer 10mb sowie markiert er html mails als gefährlich. Welche Art von Firewall ist es?**

Eine FW auf Ebene des Application-Layer.

## **Welche 2 Methoden gibt es zum Implementieren von verteilten Transaktionen (mit Erklärung)**

TODO

## **JavaSpaces**

Das ist der Shared Dataspace von Jini, einem Java-basierten Coordination-based Distributed System. Darin werden Datenelemente in Form von Tupeln abgelegt, deren Elemente serialisierte Java-Objekte sind. Abgefragt können die Objekte von dort werden, indem der abfragende Prozess ein anderes (nicht notwendigerweise vollständig ausgefülltes) Tupel als Template anbietet. Jini wird dann ein Matching des Templates und der abgelegten Objekte durchführen und, wenn ein passendes Tupel vorhanden ist, dieses zurückliefern.

## **Begriffserklärung: Distributed System, Transparenz, Skalierbarkeit**

Distributed System: Ein aus mehreren unabhängigen Rechnern (die über ein Netzwerk verbunden sind) bestehendes System, das sich den Usern gegenüber so verhält als wäre es nur ein einzelnes kohärentes System.

Transparenz: Eine wichtige Eigenschaft von verteilten Systemen. Die Verteilung soll vom Benutzer verborgen werden (siehe Definition von VS). Dazu zählt unter anderem dass die geografische Position von Ressourcen verborgen wird (Location Transparency), dass Fehler verborgen werden (Failure Transparency), dass unabhängig von den technischen Details einheitlich auf Ressourcen zugegriffen werden kann (Access Transparency), und ähnliches.

Skalierbarkeit: Die Fähigkeit eines Systems mit wachsenden Anforderungen (Anzahl von Benutzern oder Ressourcen, geografische Ausdehnung oder Einfluss mehrerer Organisationen) umgehen zu können.

## **Was ist Marshalling, was ist ein Stub? Wo wird eingesetzt?**

Marshalling: Die Verpackung von Parametern an eine remote aufgerufene Prozedur in eine Nachricht.

Stub: Die am Client befindliche Repräsentation des Servers, bzw. die am Server befindliche Repräsentation des Clients. Er ist für Marshalling/Demmarshalling zuständig und sorgt für die Verteilungstransparenz. Seine Schnittstellen sind die gleichen wie die des richtigen Servers (bzw beim Server-Stub: wie die des richtigen Clients).

Eingesetzt werden sie bei RPC und RMI.

## **Vorteile eines Single-threaded Prozesses gg. über einer Multithreading-Implementierung**

- Leichter anzulegen
- Kein Overhead durch Thread Scheduling
- Es kann keine Race Conditions geben
- Keine Synchronisation notwendig
- Leichter zu implementieren

## **Wie kann mit Hilfe von Cookies ein stateful server realisiert werden? Was passiert auf Client-, was auf Serverseite?**

Der Server schickt dem Client mittels Cookie (eine Textdatei mit einem Schlüssel/Wert-Paar) zurück, anhand dessen er den Client später wieder identifizieren kann. Bei jeder zukünftigen Anfrage gibt der Client das Cookie wieder an den Server, der dann die am Server über den Client gespeicherten Daten aus einer speziell für diesen Client angelegten Session wieder abfragen kann. Das Cookie dient also der Identifikation des Clients, die notwendig ist, da Request in http ohne weitere Maßnahmen unabhängig von einander sind (stateless).

## **Hierarchische Lösung für reliables Multicasting durch Graphik erklären**

Die Menge der Empfänger einer reliable Nachricht (eine Nachricht, die garantiert bei allen ankommt) wird unterteilt in Gruppen, wobei jede Gruppe einen Koordinator hat. Die Gruppen werden anschließend hierarchisch angeordnet. Wenn nun eine Nachricht an alle Empfänger gesendet wird, so wird jeder Koordinator die Nachricht cachen für den Fall dass sie jemand nicht empfängt und daher noch einmal übertragen werden muss und anschließend an seine direkt angeschlossenen Clients sowie die Sub-Koordinatoren weitersenden.

Jeder Empfänger bestätigt den Erhalt der Nachricht seinem Koordinator. Sobald dieser eine Bestätigung von allen direkt angeschlossenen Clients sowie seinen Sub-Koordinatoren erhalten hat, kann er die Nachricht aus dem Cache wieder löschen. Sollte er innerhalb einer gewissen Zeit keine Antwort erhalten, überträgt er die Nachricht noch einmal.

## **Sie arbeiten als Informatiker bei einer Bank. Welche Sicherheitsmechanismen würden Sie verwirklichen? Erklären Sie ihre Antwort.**

Auhtifizierung: Es muss sichergestellt werden, dass Personen, die auf das System zugreifen, wirklich die sind, für die sie sich ausgeben. Es darf z.B. niemand im Namen eines anderen Überweisungen durchführen.

Authorisierung: Es darf niemand Daten einsehen oder verändern, zu denen er keine Zugriffsrechte hat.

Encryption: Die Daten sollten in verschlüsselter Form abgelegt und übertragen werden, damit unberechtigte Personen diese nicht mithören können.

Auditing: Alle Zugriffe sollten protokolliert werden, damit im Falle von Angriffen oder Fahrlässigkeit die Haftung geklärt werden kann.

Signaturen: Es muss nachweisbar sein, dass Aufträge tatsächlich erteilt wurden. Es muss vermieden werden dass es später abgestritten werden kann.

### 3 Voraussetzungen für die Lamport Time Stamps

- Wenn 2 Ereignisse a und b auf einem Prozess in der Reihenfolge  $a \rightarrow b$  stattgefunden haben, dann muss gelten  $T(a) < T(b)$
- Wenn ein Prozess eine Nachricht empfängt (a) und dann eine andere Nachricht schickt (b), dann muss gelten  $T(a) < T(b)$
- Die Relation ist transitiv abgeschlossen

### Wie kann bei replizierten Objekten sichergestellt werden, dass concurrent invocations überall in der gleichen Reihenfolge erfolgen. 2 Methoden und je ein Vorteil.

- Die Methodenaufrufe werden als Totally-Ordered-Multicast-Nachrichten verschickt. Somit kommen sie überall in der gleichen Reihenfolge an. Zusätzlich müssen sich noch die Thread Scheduler untereinander abgleichen, falls auf den Replikaten mehrere Threads Aufrufe durchführen. Andernfalls könnte durch ungünstiges Scheduling die Reihenfolge der Aufrufe trotz Totally-Ordered-Multicast wieder zerstört werden.  
Vorteil: Keine Überlastung eines Primary Objekts und kein Single Point of Failure
- Aufrufe werden nur auf einem Objekt durchgeführt (Primary-based), das die Aufrufe per Active Replication verteilt.  
Vorteil: Leichter zu implementieren

TODO: Richtig?

### Zuordnung: (a)synchrone Kommunikation zu E-Mail, udp, tcp, http

Asynchron: E-Mail, UDP, persistent http

Synchron: TCP, non-persistent http

Replicated-Write-Protokolle

Bei all diesen Protokollen können Änderungen an einem beliebigen Replikat durchgeführt werden. Das Protokoll sorgt dafür, dass die Änderungen entsprechend propagiert werden.

Active Replication: Die Operationen selbst werden an alle andern Replikate weitergeschickt, damit diese nachziehen können. Das ist aber nur bei deterministischen Operationen möglich. Das passiert meist über totally-ordered-Multicasting. Eine Alternative wäre ein Sequencer, was aber dann eher in Richtung Primary-based-Protokolle geht.

Quorum-basierte Methode: Die Änderungen und Lesezugriffe werden an einer bestimmten Anzahl von Replikaten durchgeführt um Konsistenz zu garantieren (Details: siehe separate Prüfungsfrage).

## **Ein Ziel von Sicherheitsmechanismen ist die Einfachheit. Warum ist dem so? Kann Einfachheit immer erreicht werden?**

Einerseits weil es für die Entwickler leichter ist die Richtigkeit ihrer Implementierung zu überprüfen, andererseits auch weil das Vertrauen der User in ein einfaches System normalerweise größer ist, weil die Vorgänge leichter nachvollziehbar sind.

Es ist nicht immer erreichbar, weil einfachere Systeme natürlich auch weniger flexibel sind. Für größere Anforderungen ist der Implementierungsaufwand naturgemäß höher. Z.B. ist eine Firewall auf Applikationsebene schwieriger zu implementieren, aber auch mächtiger als eine auf TCP-Ebene.

## **E-Mail an xyz@infosys.tuwien.ac.at wie findet man die Adresse des Mailservers?**

Man löst die Adresse infosys.tuwien.ac.at so weit auf, bis ein MX-Eintrag für die Domain infosys.tuwien.ac.at gefunden wird. In diesem sind die zuständigen Mailservers für alle Mails in diese Domäne inklusive Prioritätsflag aufgelistet.

Hat man einen Mailservers ausgewählt, so löst man dessen DNS-Namen auf und bekommt schließlich seine IP-Adresse.

Wie weit obige Adresse aufzulösen ist hängt davon ab, ob infosys über einen eigenen Nameserver verfügt (also eine eigene Zone ist) oder ob vielleicht direkt im Nameserver der TU-Wien ein entsprechender MX-Eintrag vorhanden ist. Es kommt also darauf an ob infosys eine eigene Zone ist, oder ob tuwien.ac.at eine Zone als ganzes ist.

## **Die 3 Arten wie man einen Thread implementieren kann beschreiben**

- User Level Threads: Es wird eine Thread Library verwendet, mit der Threads erzeugt und gescheduled werden können. Alles passiert hier im User Level.
- Systemaufrufe: Der Kernel kümmert sich um das Erzeugen und Scheduling von Threads
- Lightweight Processes (LWP): Eine Hybridform, bei der auf Kernelebene mehrere LWPs (ähnlich Kernel Level Threads) angelegt werden, die dann auf User Level Threads gemappt werden. Jeder LWP ist also damit beauftragt einen User Level Thread auszuführen. Durch das Scheduling kann diese Zuordnung jederzeit geändert werden. Bei diesem Ansatz müssen zur Laufzeit kaum Kernel Level Threads angelegt werden (nur hin und wieder, vor allem bei Prozessstart, die LWPs), trotzdem blockiert ein blockierender Systemaufruf nicht den ganzen Prozess, weil ein anderer LWP (mit einem anderen User Level Thread) weiterlaufen kann.

## **DNS: Was ist soa, srv, mx, cname, ptr eintrag, [zone | domain | host | node] jeweils zuordnen**

SOA: Eine Angabe wer für eine bestimmte Zone zuständig ist (z.B. E-Mail-Adresse des Administrators). Wird pro Zone angegeben.

SRV: Speichert die Adresse oder den Namen eines Hosts, der in einer Domain für ein bestimmtes Service zuständig ist (ist also eine Verallgemeinerung des MX-Eintrages). Wird pro Domain angegeben.

MX: Wie SRV, aber speziell für Mailserver (inklusive Prioritätsflag).

CNAME: Canonical Name. Ein Alias für einen Knoten, z.B. um [www.domain.com](http://www.domain.com) auf [webserver.domain.com](http://webserver.domain.com) umzuleiten. Wird pro Node angegeben.

PTR: Ein Eintrag für den Reverse Lookup, also um den DNS-Namen zu einer IP-Adresse zu finden. Wird pro Host angegeben.

## Triple Modular Redundancy

In einer Sequenz von Prozessen wird jeder 3 mal ausgeführt. Die 3 Ergebnisse werden jeweils durch einen Voter zusammengefasst (der eine Mehrheitsentscheidung trifft) und in den Folgebaustein geleitet. Die Voter sind dabei für jede der 3 Ausführungen des Folgebausteins separat vorhanden, um auch ein Fehlverhalten eines Voters ausgleichen zu können.

## 2 Schwachstellen von DES

- Heutzutage sind wegen der kurzen Schlüssel (56 Bit) Brute Force Attacken möglich
- Verwendet man andere als die ursprünglich designten S-Boxen, so verliert DES sehr schnell seine Sicherheit

## Vergleich von Strong, Sequential und Causal Consistency, Welche Voraussetzungen gibt es für Strong Consistency?

Strong: Alle Operationen müssen auf allen Prozessen in der Reihenfolge durchgeführt werden, in der sie tatsächlich abgesetzt wurden. Voraussetzung: Es existiert ein globaler Clock.

Sequential: Alle Operationen müssen auf allen Prozessen in irgendeiner Reihenfolge durchgeführt werden (aber bei allen Prozessen in der gleichen) und außerdem müssen 2 Operationen a und b, die bei einem Prozess in der Reihenfolge  $a \rightarrow b$  durchgeführt wurden, bei allen Prozessen in dieser Reihenfolge ausgeführt werden.

Causal: Nur potentiell kausal abhängige Schreiboperationen müssen bei allen Prozessen in der gleichen Reihenfolge durchgeführt werden.

## Replikation von Webservern am WWW unterscheidet zwischen Replikation im Web Server Clustern und Mirroring (Spiegeln). Erklären Sie, wie diese Konzepte funktionieren und worin sie sich unterscheiden.

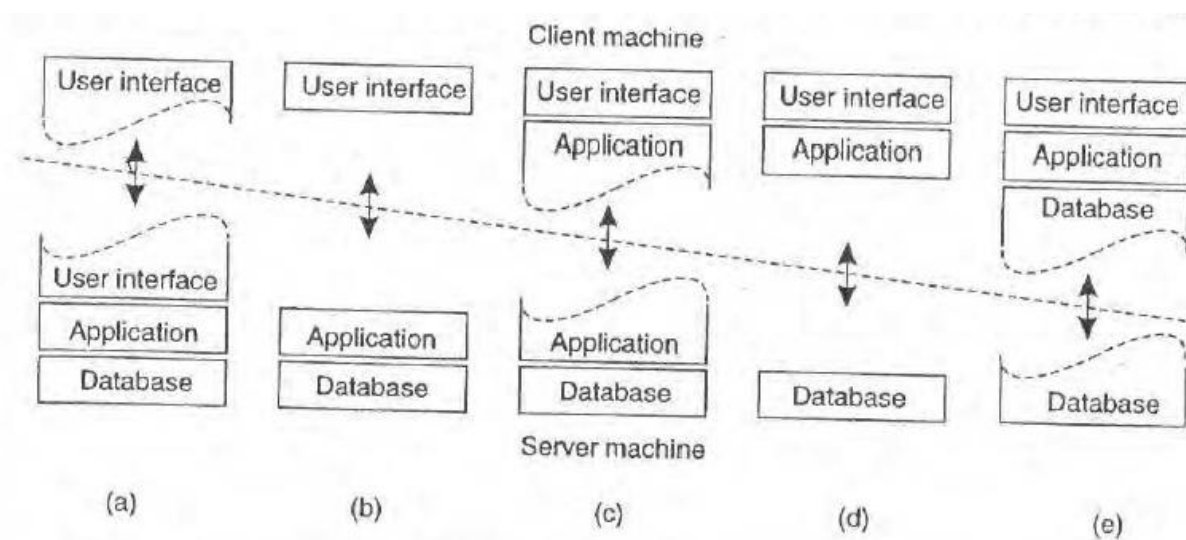
Replikation in Clustern: Es wird eine Menge von Servern aufgestellt, die jeweils die gleichen

Datenbasis haben. ein Switch leitet eingehende Request abwechselnd an die verschiedenen Server weiter (Round Robin), oder inspiziert die Anfragen auf Application-Level-Ebene und trifft danach eine Auswahl. Es handelt sich bei Server Clustern um eine permanente Replikation.

Mirrors sind ebenfalls eine permanente Replikation, jedoch wird hier die Existenz verschiedener Server nicht vom Benutzer verborgen. Statt dem Switch sind einfach die verschiedenen Replikate unter verschiedenen Adressen erreichbar, so dass der Benutzer eine Auswahl treffen kann. Typisches Beispiel sind Mirrors für FTP-Server, bei denen der Benutzer üblicherweise einen Server in seiner Nähe wählt.

## Ordnen Sie die folgenden Systeme den unten dargestellten „Client Server“ Organisationen zu:

1. Ein Browser, der auf einem Web-Server zugreift welcher die Daten aus einer Datenbank bezieht und keine Funktionalität beim Client verwendet.
2. Ein Browser, der eine JavaScript bereicherte Webseite anzeigt, die der Web-Server aus einer Datenbank generiert.
3. Ein Browser, der ein Applet anzeigt, das direkt auf einer Datenbank zugreift.



1. a
2. c
3. d

## Stellen Sie sich vor, Sie haben einen Web-Browser der als einzelner Prozess realisiert ist.

1. Welchen Schritten werden nacheinander ausgeführt, um eine HTML-Seite, die Bilder enthält, zu laden und anzuzeigen?  
Nehmen Sie dabei an, das Sie keine User-Interaktion berücksichtigen müssen. Geben Sie an, wann und wie lange der Prozess blockiert.

2. Welchen Nachteil hat diese Implementierung gegenüber Multithreading?
  1. Der Prozess fragt die HTML-Seite ab und wartet bis diese empfangen ist. Danach ruft er nach einander die embedded Documents (Bilder, Videos, etc.) ab und wartet bis diese auch empfangen sind. Anschließend kann er die Seite rendern und dem Benutzer wieder die Möglichkeit zur Navigation übergeben. Der Prozess blockiert daher von Beginn der Anfrage bis die Seite komplett geladen ist.
  2. Hier kann der Benutzer sofort navigieren, sobald Teile der Seite geladen ist. Durch die verschiedenen Threads kann ein Thread die Seite bereits rendern und ein anderer Benutzereingaben entgegennehmen, während wieder andere Threads noch die embedded Documents übertragen. Außerdem können mehrere embedded Documents gleichzeitig übertragen werden, was zum Beispiel das Ausnutzen von Replikaten des Web-Server ermöglicht.

## **Was versteht man unter dem Address Resolution Protocol (ARP)? Erklären Sie dessen Funktionsweise**

Das Suchen einer MAC-Adresse zu einer gegebenen IP-Adresse. Die IP-Adresse wird dabei als Broadcast ausgeschildt, woraufhin der Rechner, dem diese Adresse gehört, darauf antwortet.

## **Stellen Sie sich vor, in einem verteilten System erfolgt die Synchronisation der Uhren mit Hilfe des Cristian's Algorithmus.**

Ein Rechner r1 fragt um 12:34:56.123 (Zeit auf Rechner r1) einen Rechner r2 nach der Uhrzeit und erhält um 12:34:56.543 (Zeit auf Rechner r2) die Antwort 12:34:56.123. Nehmen Sie an, dass die Interrupt-Zeit r2 0 ist.

1. Wie muss die Uhrzeit von r1 korrigiert werden? Berücksichtigen Sie dabei auch die Zeit, die bis zum Empfangen der Nachricht vergeht.
2. Was ist bei der Zeitumstellung auf r1 zu beachten? Wie kann man dieses Problem umgehen?
  1. Die Gesamtzeit der Anfrage ist 420ms. Da die Interrupt-Zeit bei r2=0 ist, kann darauf geschlossen werden dass die Übertragungszeit in eine Richtung 210ms ist. Die Zeiten von r1 und r2 stimmen scheinbar überein, allerdings nur deswegen, weil 210ms vergangen sind, als r2 seinen Timestamp abgegeben hat. Zum Zeitpunkt als r1 seinen (ersten) Timestamp gemacht hat, war r2 noch um 210ms hinten, weshalb die Zeit bei r1 um 210ms zurückgestellt werden muss.
  2. Das Problem ist dass die Zeit niemals zurückgestellt werden darf. Man muss stattdessen die Uhr für eine längere Zeitspanne verlangsamen.

## **Wodurch unterscheiden sich security policy und security mechanism?**

Security policies sind eine Beschreibung der geforderten Schutzmaßnahmen. Darin steht etwa, wer auf welche Services zugreifen darf.

Security mechanisms sind eine Menge von technischen Maßnahmen zur Realisierung der Security policies, z.B. Encryption, Authentication, Authorization und Auditing.

## **Erklären Sie kurz den Ablauf des „challenge-response“ Protokolls zwischen zwei Kommunikationspartnern zu deren gegenseitiger Authentifizierung.**

Ein Partner stellt eine Anfrage an einen anderen, die dieser nur beantworten kann, wenn er wirklich der ist, für den er sich ausgibt. Das geschieht zum Beispiel mit einer Zufallszahl von A, die der Kommunikationspartner B mit dem Schlüssel  $K_{a/b}$  verschlüsseln und zurückschicken soll. A kann dies überprüfen indem er wieder entschlüsselt. Da nur A und B den Schlüssel  $K_{a/b}$  haben, kann A danach sicher sein, dass sein Partner wirklich B ist. Das wird natürlich in beide Richtungen durchgeführt um wechselseitige Authentifizierung zu ermöglichen.

## **Nenne ein Beispiel für horizontale Distribution in Bezug auf Verteilte Systeme.**

Ein replizierter Web-Server. Dabei stellt jedes Replikat die gleiche logische Softwareschicht zur Verfügung (im Unterschied zur vertikalen Distribution).

## **Was sind Message Brokers?**

Das sind spezielle Router in Message Passing Systemen, die eine Konvertierung von Nachrichten vornehmen können, um die Kommunikation zwischen 2 Partnern auch dann zu ermöglichen, wenn sie verschiedene Formate erwarten. Die Konvertierung ist mit Hilfe von Regeln aus einer Datenbank möglich.

## **Wozu dient das inhaltspezifische Anfragen bei Web Server Clusters?**

Um verschiedene Anfragen nicht nur nach dem Round Robin-Verfahren an die Replikate verteilen zu können, sondern auch in Abhängigkeit vom Inhalt des Request, also z.B. der angeforderten Seite. Das hilft die Performance zu erhöhen, da dann nicht mehr alle Replikate alle Dateien speichern müssen, was einerseits die Konsistenzerhaltung vereinfacht und andererseits effizienteres Caching ermöglicht.

## **Nenne die untersten drei Schichten des OSI-Modells und beschreiben Sie diese.**

Physical Layer: Befasst sich mit physikalischen Übertragung, also der Signalcodierung.

Data-Link-Layer: Ist für Adressierung mittels MAC-Adressen und die Erkennung von Fehlern zuständig.

Network Layer: Ermöglicht erstmals die Kommunikation über Netzwerkgrenzen hinaus, indem Routing verwendet wird. Dadurch werden End-To-End-Verbindungen ermöglicht, während auf Layer 2 eine Weiterleitung immer nur bis zum nächsten Hop möglich ist.

## **Was verstehst du unter clock synchronization? Wie funktioniert die Zeitabfrage in einem zentralen System?**

- Der Abgleich der Uhren zwischen verschiedenen Prozessen. Dies kann entweder nur untereinander oder auch in Einklang mit einem externen Zeitgeber (z.B. einer Atomuhr) geschehen. Auch rein logische Uhren (Zählen der Ereignisse) erfordern Synchronisation.
- Bei zentralen Systemen wird einfach die clock()-Funktion des Kernels aufgerufen. Synchronisation ist nicht erforderlich.

## **Was macht die RSA-Verschlüsselung so sicher?**

Dass kein effizienter Algorithmus zur Berechnung der Primfaktoren einer Zahl existiert.

## **Nenne 3 Corba-Services und erkläre sie kurz.**

CORBA Naming Service: Ermöglicht es die Interoperable Object Reference (IOR) zu einem mit seinem Namen spezifizierten Objekt zu finden.

CORBA Trading Service: Kann Objekte anhand ihrer Eigenschaften (und nicht nur anhand ihres Namens) finden.

CORBA Transaction Service: Unterstützt Folgen von Methodenaufrufen auf distributed Objects, die entweder alle oder gar nicht ausgeführt werden.

## **Soll man Threads in einem Server-Prozess limitieren?**

Ja, da eine zu große Anzahl von Threads den Overhead zu sehr erhöht. Zwar ist diese viel geringer als bei mehreren Prozessen, trotzdem nicht zu vernachlässigen. Aber selbst ohne den Overhead durch Thread-Scheduling ist eine zu hohe Anzahl nicht sinnvoll, da dann die Antwortzeiten für einzelne Anfragen so oder so sehr hoch werden. In diesem Fall ist es sinnvoller den Client die Anfrage später noch einmal stellen zu lassen.

## Ein mobiler Agent, der bei eBay von Server zu Server geht und nach Büchern meines Lieblingsauthors sucht. Wie kann der Agent geschützt werden, damit er nicht abgefangen und abgeändert wird? Wie kann eBay seine Server schützen?

Schutz des Agents: Die Änderung an sich kann nicht verhindert werden, es kann aber durch Signaturen garantiert werden, dass Änderungen später zumindest erkannt werden. Dazu berechnet jeder, der Daten in das Log schreibt, eine Signatur über diese Daten. Wenn der Agent beim Sender wieder ankommt können die Signaturen mit Hilfe der Public Keys der einzelnen Server überprüft werden.

eBay kann seine Server schützen indem der Agent in einer Sandbox ausgeführt wird, in der alle Aktivitäten des Agents genau kontrolliert und wenn notwendig geblockt werden können.

## Ein Algorithmus für Mutual Exclusion blockiert bei Ausfall eines Prozesses. Was muss man ändern, damit dem nicht so ist?

Man kann einen dezentralisierten Ansatz wählen, bei dem mehrere Koordinatoren existieren. Ein Zugriff auf eine Resource ist dann erlaubt, wenn die Mehrheit der Koordinatoren diesen gestattet hat. Ein einzelner Ausfall kann somit den Algorithmus nicht zerstören. Jedoch kann es bei diesem Ansatz (mit einer sehr geringen Wahrscheinlichkeit) passieren, dass der Algorithmus fälschlicherweise 2 gleichzeitige Zugriffe erlaubt, falls in einem kurzen Intervall zu viele Koordinatoren ausfallen und diese beim Wiederhochfahren vergessen haben wem sie bereits den Zugriff erlaubt haben.

Eine andere Alternative ist die Verwendung eines Token Ring Algorithmus. Dieser funktioniert auch nach einem Ausfalls noch einwandfrei, solange nicht der Rechner betroffen ist, der gerade den Token hält (dieser Fall ist dafür sehr schwer zu handhaben).

Wirklich verhindern kann man das Blockieren nicht ohne schwere Nebenwirkungen. Die in der Praxis gewählte Methode ist es daher, den Single Point of Failure nicht zu verhindern, da dies viele andere Probleme mit sich bringt. Stattdessen investiert man mehr Aufwand in das Verhindern eines solchen Ausfalls.

## MC

richtig	falsch	
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Ein Name Service bietet folgende Methoden an: eine Methode zum Verschlüsseln von Daten, und eine zum Entschlüsseln.
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Als Caching bezeichnet man die Synchronisierung zwischen einem Producer und einem Consumer.
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Beim Message Passing werden Nachrichten an Empfänger geschickt. Der Sender muss aber nicht wissen, wer der Empfänger ist.
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Der Microkernel implementiert Prozess-, Speicherverwaltung und die I/O

		Operationen.
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Middleware befindet sich zw. NOS und Kernel
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Bei Peer-to-Peer sind alle Prozesse/Applikationen gleichberechtigt
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Bei einem Secure Channel werden die Teilnehmer automatisch auch authentifiziert
<input type="checkbox"/>	<input checked="" type="checkbox"/>	http protocol besteht aus get, reply and acknowledge
<input checked="" type="checkbox"/>	<input type="checkbox"/>	ein Thread ist durch System-call blockiert

## Welche Arten von Namensauflösung gibt es? Vorteile + Nachteile.

Grundsätzlich lässt sich unterscheiden zwischen flachen, hierarchischen und attribut-basierten Namen.

Flach: mittels Broadcasts, Home-based, forwarding-Pointers. Vorteil: einfach zu implementieren, Nachteile: schlecht für Menschen geeignet, schlecht skalierbar.

Hierarchische: DNS, Filesystem mittels iNodes. Vorteil: Sehr gut skalierbar (z.B. durch Caching) da gut verteilbar, gut für Menschen geeignet. Nachteil: schwieriger zu implementieren.

Attribut-basiert: z.B. LDAP. Vorteil: Man muss nicht mehr genau wissen wie eine Entität heißt, sondern kann auch nicht ihren Eigenschaften suchen. Nachteil: Nicht immer einsetzbar, oft will man genau eine bestimmte Entität finden.

## Lamport-Timestamps und „Happens Before“ erklären. Welche Aussage kann man durch die „Happens Before“-Relation tätigen? Warum gilt der Umkehrschluss dieser Aussage nicht? Wie kann der Umkehrschluss doch realisiert werden?

Zwei Ereignisse a und b stehen in der happens-before-Relation  $a \rightarrow b$  zueinander wenn Ereignis b potentiell auf das andere Ereignis a aufbaut. Das ist dann der Fall, wenn beide innerhalb eines Prozesses in der Reihenfolge a, b stattgefunden haben, oder wenn ein Prozess eine Nachricht empfängt (a) und dann eine anderen Nachricht schickt (b). Aussage: Die Abhängigkeit ist nur potentiell, weil die Applikationssemantik nicht berücksichtigt wird. Man kann also nicht sagen, ob wirklich eine Abhängigkeit besteht.

Realisiert wird das, indem jeder Prozess einen Counter von Ereignissen mitführt. Für jede gesendet Nachricht wird dieser um 1 hochgezählt und mitgeschickt. Beim Empfänger wird das Maximum des lokalen Counters und des mitgeschickten Wertes übernommen um die logischen Uhren zu synchronisieren. Anschließend wird das Empfangsereignis selbst auch noch gezählt.

Der Umkehrschluss gilt nicht, weil 2 Zeitstempel auch rein zufällig eine bestimmte Ordnung haben können. Wenn z.B. A und C beide eine Nachricht an B schicken (a und c), dann kann für diese Nachrichten gelten  $T(a) < T(c)$  oder  $T(c) < T(a)$ . Man kann also aus der Ordnung keine potentielle Abhängigkeit ablesen. Vector Timestamps lösen dieses Problem, indem diese nicht für alle Paare von Vektoren eine Ordnung definieren.

## SSL erklären

Secure Socket Layer ermöglicht eine gesicherte TCP-Verbindung. Es baut auf den Transport-Layer auf. Der Client schickt dazu dem Server seine Möglichkeiten zur Verschlüsselung und Kompression, woraufhin der Server eine Auswahl trifft. Anschließend tauschen beide ihre (von einer CA ausgestellten) Zertifikate aus, womit sie ihre Public Keys einander bestätigen. Schließlich erzeugt der Client eine Zufallszahl und überträgt sie an den Server (mit dessen Public Key verschlüsselt). Aus dieser Zahl leiten beide einen (den gleichen) Session Key ab.

## Sie haben einen gemeinsamen geheimen Schlüssel (secret-shared-key) mit einem Kommunikationspartner. Wie authentifizieren Sie diesen?

Mit dem challenge-response-Protokoll. Jeder der beiden Partner schickt eine Zufallszahl an den anderen, lässt sie von dem verschlüsseln, und entschlüsselt das Ergebnis selbst wieder mit dem Shared Key. Entspricht es der ursprünglichen Zahl, so ist der Partner authentifiziert.

## Was ist ein "totally-ordered-multicast" ?

Ein Multicast, bei dem die Nachrichten bei allen Kommunikationspartnern in der gleichen Reihenfolge eintreffen, und zwar so, dass 2 Nachrichten, die von dem gleichen Prozess gesendet wurden, ihre Reihenfolge behalten.

Garantiert wird dies durch logische Zeitstempel (Lamport Timestamps) und eine Hold-back-Queue. Nachrichten werden nicht sofort an die Applikation geliefert, sondern in einer Hold-back-Queue zwischengespeichert und dort nach ihren Zeitstempeln sortiert. Jede eingehende Nachricht wird allen anderen bestätigt (Ack). Wurde eine Nachricht von allen bestätigt, und befindet sie sich ganz vorne in der Hold-back-Queue, so kann sie der Applikation übergeben werden.

## 3 Arten von Replikationen

- permanent: Von Anfang an geplant
- server-initiated: Passiert zur Laufzeit je nach Auslastung dynamisch
- client-initiated: Caches.

## Welche 2 Arten von Firewalls gibt es und wie funktionieren sie?

Auf Network-Layer: Es werden nur die Felder der IP-Pakete analysiert: Quelle und Ziel, Port, eventuell Paketgröße.

Auf Application-Layer: Es wird der Inhalt der Nachricht berücksichtigt. Dazu muss das Datenformat eine bestimmten Applikation interpretiert werden. Beispiel: Verwerfen von E-Mails mit Anhang.

## **Weisen sie die Begriffe "IP - UDP - TCP - HTTP - FTP" zu den ISO OSI Schichten zu**

IP – 3 (Network)

UDP, TCP – 4 (Transport)

FTP, http – 7 (Application)

## **Nennen sie die drei Eigenschaften des 'true identifier' bei Naming**

- Jede Entität hat genau einen Identifier
- Identifiers werden maximal einer Entität zugewiesen
- Identifier werden niemals wieder verwendet (kein Reuse)

## **Nennen sie zwei Möglichkeiten zwei simultane Zugriffe verschiedener Clients zu verhindern**

- Mutual Exclusion mittels Koordinator, der immer nur eine Berechtigung ausstellt
- Token Ring Algorithmus

## **Erklären sie die Begriffe "Transient Fault, Intermittent Fault, Permanent Fault" und geben sie jeweils ein Beispiel.**

Transient: Tritt nur einmal auf und verschwindet von alleine wieder, z.B. Störung durch Schlechtwetter

Intermittent: Tritt wiederholt auf und verschwindet dazwischen immer wieder, z.B. ein Wackelkontakt

Permanent: Verschwindet von selbst nicht mehr, z.B. defekte Hardware

## **Nennen sie zwei Möglichkeiten um die Zugriffsrechte zu verwalten (access control). Nennen sie jeweils einen Vorteil**

Matrix: Vorteil: Schnelle Überprüfung ob Berechtigung vorliegt, Nachteil: großer Speicherplatzbedarf

ACL: Vorteil: Geringerer Speicherplatzbedarf, Nachteil: lineare Suche notwendig

## **Vorteile & Nachteil von Connection oriented communication.**

Vorteil: Garantierte Ankonft der Nachricht

Nachteil: Overhead durch Verbindungsauf- und abbau

## **Wozu Zertifikate? Was ist eine CA?**

Ein Zertifikat bestätigt die Zuordnung eines Public Keys zu einer Person. Sie werden von sogenannten Certification Authorities ausgestellt. Dazu wird Identität einer Person, Public Key und ein Timestamp in ein Zertifikat zusammengefasst, das von der CA signiert wird.