Aufgaben eines Betriebssystems	- Starten/Beenden von Programmen - Kommunikation von I/O geräten - Sicherheit - Management vom Memory - Logging - Userprozesse dürfen nicht mehr direkt auf Hardwaresysteme zugreifen. Nur noch über Syscalls erlaubt
Syscalls	Programme können Syscalls erstellen und damit Anfragen an das OS geben (verbindung von Userprogrammen und OS) Syscalls sind synchrone Interrupts Syscall-Standard = POSIX - heute halten sich die OS bedingt an diesen Standart Leiten einen Contextswitch ein (wechsel von CPU context (PSW und PC) und Ring privilegien)

Resourcenmanagment des OS	Das OS Abstrahiert Physikalische Ressourcen um dem User die Komplexität abzunehmen
	Virtuellen Resourcen:
	  - Prozesse (Prozessoren)
	  - Virtuell Memory (Mainmemory)
	  - Ports (Network kommunikation)
	  - Files (Harddrive storage)

Prozesse	- Die Anzahl der Prozesse steht nicht in Verbindung mit der Anzahl der Prozessoren - Ermöglicht das Vortäuschen von Gleichzeitigkeit durch intelligentes Scheduling der verschiedenen Prozesse - Prozesse können verschiedene Zustände besitzen: Siehe Prozesszustände - Jede Instanz eines Programms ist ihr eigener Prozess - Ein Prozess kann mehrere Threads besitzen
Prozesszustände	Es Existieren folgenede Prozesszustände: - New -> Der Prozess wurde gerade erstellt - Ready -> Der Prozess läuft gerade nicht, könnte aber laufen - Running -> Der Prozess belegt gerade die CPU - Blocked -> Der Prozess wartete auf etwas (BSP: Belegtes I/O) - Blocked Suspendet -> Der Prozess wartet gerade wurde aber aus dem Mainmemory in den Harddrive ausgelagert - Ready Suspendet -> Der Prozess könnte laufen befindet sich aber gerade nicht im Mainmemory, sondern im Harddrive - Exit -> Prozess wurde beendet
Virtual Memory	- Management vom Virtuellen Speicherraum pro Prozess - Ermöglicht das abstrahieren der realen Adressen für die Userprozesse
Files	- Managemnt von der Langzeitspeicherung in einem System aufgebaut als Dateien - Abstrahiert die physikalische Organisation des Harddrives
Ports	- Managed Network Kommunikation - Ermöglicht es viele verschiedene Netzwerkverbindungen über eine einzige Leitung zu führen
Protective Rings	Jede CPU besitzt die möglichkeit zwischen verschiedenen Protective Rings zu wechseln. Jeder Protective Ring besitzt unterschiedliche Hardwareberechtigungen. Ring 0 = höchsten Hardwarezugriff Ring 1-3 immer weniger Zugriffsrechte OS Kernal und möglicherweise teile des OS in Ring 0. Userprozesse in Ring 3 Bestimmte Aktionen, die nicht in Ring 3 erlaubt sind müssen über Syscalls angefragt werden
OS Kernal	Der OS Kernal übernimmt die Basisfunktionen des Systems Min. muss er Context Switches können (zwischen Prozessen und Ringen wechseln)
Monolythische Kernal	Der Monolytische Kernal enthält viele der Aufgaben im Ring 0, die auch ausgelagert werden könnten. Der Kernal selbst ist zwar dardurch größer und massiver, doch er benötigt viel weniger Kommunikation und viel weniger Context Switches als der Micro Kernal. Sicherheitstechnisch ist der Microkernal besser, da er weniger Funktionalität im Ring 0 hat
Micro Kernal	Der Micro Kernal enthält nur die absoluten Basisfunktionalitäten im Ring 0 und lagert jedliche weitere Funktion als Userprozess in äußere Ringe aus. Dadurch verbessert sich zum einen die Sicherheit, da weniger Prozesse Ring 0 Privilegien besitzen, doch benötigt viel mehr Kommunikation: Bsp: Kernal Anfragt an das Filesystem wird erst an den Kernal geschickt und von dort aus an das Filesystem, welches sich im Ring 3 befindet (2 Context Switches)
Mode Switch	- Temporörer Wechsel vom Protextive Ring - PSW muss nicht umbedingt gesichert werden -> verbesserte Performance, da weniger gesichert werden muss
Process Switch	- Speichern des gesamten PSW - Stoppt den bisherigen Prozess komplett

Process Controll Block (PCB)	- Das OS legt für jeden Proozess einen eigenen PCB an
	- Im PCB werden Informationen zum Prozess gespeichert, wenn er nicht gerade läuft
	  - BSP: PSW und CPU status
	  - Weitere Informationen des Prozesses

Prozess Scheduling	Ziele:
	  - Prozessoren möglichst effizien Prozesse zuweisen
	  - Hoher Throughput
	  - Prozessor am effizientesten auslasten
	  - Hohe Responsiveness
	
	  Ziele stehen sich im weg (Bsp. hohe effizienz und hohe Responsiveness)
	
	Prozesszustände können eingeteilt werden in jenachdem wie schnell sie wieder benötigt/angefragt werden können:
	- Short Term (Blocked, Ready, Running) gut für Prozesse die demnächst benötigt werden
	  - Scheduler entscheidet, welche Prozess laufen soll
	  - User Orientiert (Responsiveness)
	  - System Orientiert (Effektivestes nutzen der CPU)
	- Medium Term (Blocked Suspendet und Ready Suspendet (wobei das letztere blöd ist)) gut für Prozesse die länger nicht mehr laufen müssen/können
	  - OS entscheidet, welcher Prozess ausgelagert/eingelagert werden soll
	- Long Term (New und Exist) Prozesse die frisch sind oder fertig sind
	  - meist User oder OS welcher Prozesse startet

Preemtives/Nicht Preemptives Scheduling	- Preemtives Scheduling berechnet das Scheduling nach jedem Timer-Interrupt für alle Prozesse neu
	  - OS kann laufende Prozesse absetzen (in Ready schieben)
	  - Kein Prozess kann die CPU Monopolisieren
	- Nicht Preemtive Scheduling Verfahren nehmen einen Prozess nicht von der CPU bis er fertig ist
	  - Bis er sich selbst blockiert

Priority Inversion und Inheritance	Es kann passieren, dass ein höherstufiger Prozess auf einen niedrigeren gestuften Prozess warten muss. Daher kann es passieren, dass ein Prozess, der zwischen diesen beiden Prozessen in der Priorität liegt vor dem höherstufigen erledigt wird. Das nennt man Priority Inversion Lösung: Der niedrigere Prioritäts Prozess erbt die Priorität des höhereingestuften Prozesses
Scheduling FCFS	First Come First Served Nicht Preemtive Kein Verhungern möglich Einfache queue.
Scheduling SPN	Shortest Process Next Nicht Preemtiv Verhungern möglich Der Prozess, der die niedrigste Laufzeit hat wird Priorisiert
Schedulung HRRN	Highest Response Ration Next Nicht Preemtiv Kein Verhungern möglich Der Prozess mit dem höchsten response ratio wird priorisiert. Der Response ratio ergit sich aus (cycles_todo+cylces_waited)/cycles_todo Vorteil: lange wartende Prozesse werden auch Priorisiert. Jeder Prozess kommt dran
Scheduling SRT	Shortest Remaining Time Preemtiv Verhungern möglich Der Prozess mit der geringsten todo Zeit wird priorisiert
Scheduling RR	Round Robin Preemtiv Kein Verhungern möglich Prozesse werden im Kreis abgefahren

Multipleprozessors and Real-Time Scheduling	More Resources Available to Schedule Möglichkeit 1: - Prozesse bleiben auf 1 Prozessor (weniger Probleme mit der Cache verwaltung) - Mögliche ungleichverteilung von Prozessen Möglichketi 2: - Prozessoren werden als Pool angesehen - Prozess kann auf vielen Prozessoren laufen - Cache muss gehandlet werden bei Prozessorwechsel Real-Time-Scheduling - Der Korrekte Systemstatus hängt von der Einhaltung der Deadlines an - Scheduling muss mit betrachtung und Priorität der Deadlines geschehen
OS Memory Management	Zwei Methoden: - Paging - Unterteilung der Memorysegmente in gleich große Einheiten - Segmentierung - Unterteilung der Memorysegmente in Varriable lange Einheiten
Pagin	- Feste größe der Memorysegmente - Auslagerung von Pages ist einfach, da jede Page gleich groß ist - Verteilung von Pages kann nicht sequenziell sein - Bei Pages kann interne Fragmentierung entstehen - Memory ist nicht Perfekt ausgenutzt, da vielleicht eine Page nicht komplett gefüllt ist doch selben Platz braucht
Segmente	- Segmentgröße Varriert nach benötigtem Platz - Blöcke müssen aber sequenziell geschrieben werden - Externe Fragmentierung entsteht -> Beim auslagern von Segmenten können Lücken entstehen, die von weiteren Segmenten nicht Perfekt gefüllt werden können
Probleme mit Virtueller Memory	- Wann muss eine neue Page/Segment geladen werden - Zeitliche- und Örtliche-Lokalität - Wo werden Daten hingepackt die geladen werden? - Bei Pages einfach, da jede Lücke groß genug ist - Bei Segmenten muss erst benötigter Platz gefunden werden
Dynamische Plazierungs Algorithmen	- First Fit - erster Platz, der groß genug ist für das Segment, wird belegt - Next Fit - erster Platz, der groß genug ist für das Segment, wird belegt. Untschied ist nur, dass die Suche nach neuen Plätzen immer dort beginnt wo das vorheriger abgelegt wurde - Best Fit - Platziere das Segment in die Stelle, die am besten Passt
Beispiele für Swapping Algorithmen	- FIFO - LIFO - LRU (Least recently used) - LFU (Least frequently used) - LRD (Least reference density) - Verbindung von LRU und LFU - Ratio zwischen wann es reingeladen wurde und wann es zuletzt benutzt wurde
Page Table	Der Page Table wird vom OS angelegt und behält die Position im Memory für jede Pages eines Prozesses. Will jetzt ein Prozess auf eine bestimmte Adresse zugreifen muss zunächst im Page Table nachgeschaut werden, ob die Page in der Memory ist. - Wenn ja -> Nehme die Position der Page und Addiere den Offset der Adresse - Wenn nein -> Lade Page ins memory und wiederhole "Wenn ja"
Umrechnung der Virtuellen Adresse	Virtuelle Adresse besteht aus: - Pagenumber - Offset Wie in "Page Table" erklärt: Wenn die Page sich im Pagetable befindet wird zunächst der Basepointer zu dieser Adresse geholt. Daraufhin wird der Offset Addiert, um die korrekte Adresse zu bekommen
Hierarchische Page Table	Page Table können eine Gewisse größe übersteigen, sodass sie nicht mehr in einer Page gehalten werden können. Um dies zu unterstützen kann man Hierarchien von Page Tables erstellen. Im ersten Page Table (root table) steht dann nur die Adresse der Page, die entweder direkt die Position der Page mit dem Datum enthält oder die Position des nächsten Page Tables. Dies wird solange gemacht, bis die Page mit dem Datum gefunden wurde. Im schlimmsten fall müssen immer n-1 Pages geladen werden (alle bis auf den root table, da auf diesen bei jedem Zugriff zugegriffen werden muss)
Translation Lookaside Buffer	Der Translation Lookaside Buffer (TLB) enthält die direkte Position des Datum innerhalb des Memory, wenn sie sich dort befindet und bereit angefragt wurde. Bei einer Anfrage wir zunächst im TLB geschaut, ob die Virtuelle Adresse dieses Prozesses schon einmal angefordert wurde: Ja -> Die logische Adresse befindet sich im TLB und kann direkt ausgelesen werden Nein -> Es wird geschaut, ob die Page sich im Hauptspeicher befindet: Ja -> Die Adresse wird umgerechnet, geladen, und die übersetzte Adresse im TLB gespeichert Nein -> Die Page wird aus dem Hintergrundspeicher geladen und dann wird die Adresse im TLB gespeichert
I/O Managment	- Programmed I/O - Die CPU beschäftigt sich aktiv mit dem Auslesen der I/O daten ins Memory und blockiert somit alle weiteren Prozesse - Interrupt I/O - I/O Gerät läd die Daten in den geteilten Speicher und meldet sich bei der CPU zurück, sobald alle Daten geladen wurden - Direct Memory Access (DMA) - Extra Stück Hardware, welches sich mit dem Auslesen der Daten des I/O's beschäftigt
Direct Memory Access	- I/O Device meldet sich bei der CPU, dass es Daten besitzt, mittels eines Interrrupts - CPU stößt das Laden der Daten über den DMA an und behandelt daraufhin weitere Prozesse - DMA läd Daten Blockweise oder als Stream (je nach implementation) in den Memory - Nach beendigung des ladens eines Blocks meldet sich der DMA Controller bei der CPU das er fertig ist.
Direct Memory Access Implementationen	Single Bus - Deatached DMA - Ein einziger Bus über den alles verbunden ist. - Sehr einfach aber Sehr uneffizient Single Bus - Integrated DMA - Alles läuft über einen Bus jedoch jede I/O (group) besitzt einen eigenen DMA - Sehr effizient doch kostspielig I/O bus - Eigener Bus über den alle I/O's angelegt sind der über einen DMA an den Hauptbus verbunden ist
I/O Buffering	Dadurch, dass I/O eindeutig langsamer ist als die CPU bietet sich hier das Prinzipt des Buffering an: - Die I/O läd mehrere Daten vor währenddessen die CPU noch alte Daten verarbeitet Blockorientiert: - Daten werden in festen Blocks eingelesen und ausgelesen Streamorientiert: - Datenblöcke habe keine feste größe und können beliebig ein/ausgelesen werden
I/O Scheduling (Writing into Memory)	- FIFO - SSTF - Der Block wird als nächstes geschrieben wofür sich der Schreibkopf an wenigsten bewegen muss - Scan - Der Schreibkopf bewegt sich stehts von einer Seite zur nächsten und bearbeitet Anfragen immer nur in eine Richtung
RAID	RAID wird genutzt um ein mögliches Ausfallen einer Festplatte abzufangen. Hierzu werden z.B in RAID Level 5 die Blöcke verteilt auf verschiedene Disks geschreiben. Pro Festplatte gibt es eine Disk die für die Parität zuständig ist. Fällt eine Festplatte aus kann die Disk über die Parität wiederhergestellt werden. Die Parität entsteht darduch, dass man alle Blöcke XOR nimmt. Fällt eine Festplatte aus kann durch das XOR über alle und der Parität alle wieder hergestellt werden.
Inodes	Files bzw. jegliche im Filesystem dargestellten Objekte sind in Inodes Codiert. Diese Inodes enthalten Metadaten wie: - Attribute wit z.B. Zugangszeit - Besitzer Informationen (Read/Write Rechte) - Position in der Logischen Struktur des Mediums (z.B. wie es auf dem Harddrive abgespeichert ist) Eine Inode kann sowohl eine Datei als auch ein Ordner (Symlink, Socket, etc.) sein. Eine Datei enthält keine weiteren Links zu weiteren Inodes währendessen ein Directory eine Liste von einem Tupel mit Inodenamen und Inode ID enthält
Inode Struktur	Eine Inodedatenstruktur kann sowohl ein Baum sein ohne loopbacks als auch ein Graph mit Loops zwischen Inodes. Zwei verschiedene Referenzen auf eine Inode können auch existerien
File Allocation	Eine Datei wird im Hintergrundspeicher (Harddrive) als Blöcke abgespeichert. Hierzu gibt es 3 Varrianten: - Sequilized Allocation - Eine Datei wird immer hintereinander abgespeichert. Blöcke müssen Sequenziell geschreiben werden -> Easy Random Access (Basepointer + Offset) aber zu unflexible Struktur - Chained Allocation - Ein Block wird egal wo im Hintergrundspeicher abgelegt und verweißt auf den nächsten Block -> Sehr flexible Struktur doch kein random Access möglich (im schlimmstfall müssen n Blöcke durchsucht werden bevor der richtige Block gefunden wurde) - Indexed Allocation - Eine Datei wird in mehrere Blöcke unterteilt die teils sequienziell abgespeichert werden. Jede Datei braucht zudem einen Indexblock -> Relativ flexible Struktur und guter Random Access. Indexfile darf nur nicht zu groß werden

Boot Prozess	Der Boot Prozess findet in einer Kaskade statt: - BIOS initialisierung - Boot stage 1 - Load and start Bootstage 2 - Boot stage 2 - Load Kernal - Kernal - Switch into Protected Mode - Start first Prozess (init) - Init Porzess: - Start System functionalitys
UEFI/EFI	Der BIOS wurde durch den neuen Standard UEFI/EFI ersetzt um gewisse Sicherheitlücken zu stopfen. Unteranderem kann dadurch nur noch software im nicht Protected ausgeführt werden, die eine besondere Signatur besitzt
UNIX Daemons	Daemons sind Programme die durchgängig im Hintergrund laufen ohne Interaktion des Benutzers zu benötigen
Attak Types	Passive Attak: - Belauschen von Informationen ohne Handlung auf das System Aktive Attak: - Maipulation von Daten
Password Creation	Der Passowort String wird mit einem Salt durch eine sehr langsame Hashfunktion gejagt und abgespeichert. Beim einlogen werden nur die Hashes überprüft
Backdoors	Schlupflöcher innerhalb von Programmen um sich Zugang zu verschaffen. Diese müssen vom Entwickler eingebaut worden sein
Logic Bomb	Wenn sie ausgeführt wird wird das System so manipuliert, dass es zum Absturz des Programms/Systems kommen kann
Trojanisches Pferd	Schleuse Schadsoftware mit Nützlicher Software ein
Virus	Infiziert einen Rechner und Repliziert sich selbst
Bots	Software die genutzt werden kann um Systeme fernzusteuern
ISO/OSI Modell	Besteht aus 7 layern die über Wohldefinierte Schnitstellen miteinander Kommunizieren: 7. Application 6. Presentation 5. Session 4. Transport Layer 3. Network Layer 2. Datalink Layer 1. Physical Layer
TCP/IP Modell	Ähnlich zum ISO/OSI Modell, wird jedoch, anders als das ISO/OSI Modell, in realität benutzt: 5. Application 4. Transport Layer 3. Internet Layer 2. Network Access 1. Network Access
Physical Layer	Der Physical Layer befasst sich damit das Digitale Signal in ein Physicalisches Signal Umzuwandeln. Hierzu bekommt es eine gewisse Bandbreite zugeteilt auf der es Arbeiten kann. Je höher die Bandbreit-Range ist, desto genauer kann das Signal übertragen werden. Auf dieser Ebene findet keine direkte Fehlerbehebung statt sondern nur eine kleine Minderung des Rauschens indem bei jedem Hop über ein weiteres Gerät das Signal einmal vom Physicalischen zum Digitalen zum Physicalischen übersetzt wird um kleine Anomalien zu eliminieren. Zu den Daten muss zusätzlich eine Taktung übertragen werden, damit der Reciever am ende Weiß, wann jeweils ein Hoch (1) bzw. Tief (0) ausgelesen wurde. Durch dem Clockdrift-Phänomän muss dieser ständig wieder angepasst werden. Eine Lösung hierfür ist die Manchester Codierung
Manchaster Codierung	In der Manchaster Codierung wird die Information der Nachicht und die Clockinformation gleichzeitig übertragen. Eine 0 bzw. eine 1 wird nicht mehr über ein Hoch oder Tief codiert, sondern über den Wechsel von Hoch->Tief bzw. Tief-Hoch. Dadurch entsteht selbst bei einem gleichbleibenen Signal bsp: 10000000000000000000000000000001 die Information des Clockstatus erhalten. Problem: Da mehr Information gleichzeitig übertragen wird kann nur noch die hälfte der Informationen übermittelt werden (geschwindigkeit = Bandbreite/2)
Analoges Signal	AM = Amplituden Modeliert FM = Frequenz Modeliert
Datalink Layer	Der Datalink Layer kümmert sich um: - die Einteilung der übergeben Bits in Frames - Error Erkennung/Behebung - Acknowledgement Protocol - Flow Control - Mitteilung and Sender, in welcher Geschwindigkeit er Packete schicken kann/darf
Datalink Layer - Framing	- Unterteile Bits in Frames - Frames haben eine Gewisse Bitfolge als Start/End - Das ende ist oft eine CRC Prüfsumme - Header besteht aus Flagbits - Um sicher zugehen, dass die Payload keine Headerconfiguration besitz wird Bitstuffing betrieben - Bsp: Header = 11111 Message 1011111001 => 10 111101001 (setze 0 nach 4 aufeinander folgenden 1sen)
Error Control:	Beim versenden der Nachicht wird eine CRC Prüfsumme gebildet und als Trailer der Payload hinzugefügt. Stimmt der an der Destination übertragener CRC Wert nicht mit dem Übertragenen Wert überein kann versucht werden das Package zu recovern, ansonsten wird es gedroppt
Medium Access - Multiplexing	Um über das selbe Medium mehrere Signale zu übermitteln muss man diese nach einem Speziefischen verfahren versenden um sicher zu stellen, dass jedes Signal unique ist Hierzu gibt es verschiedene Varianten: - Time Division Multiplexing: - Man verschickt Datenpackete Zeitverschoben über das selbe Medium um Eindeutigkeit zu erhalten - ALOHA und Slotted ALOHA sind zwei Varrianten dafür - Frequenzy Division Multiplexing: - Man nutzt verschiedene Frequenzen um verschiedene Signale zu Codieren - Code Division Multiplexing: - - Space Division Multiplexing: -
ALOHA	Das System verschicken dann Daten wann sie welche haben; Es existiert keine Coordinierung. Überlappen sich zwei Signale (auch nur um ein Bit) wird das gesamte Packet verworfen und als Ungültig gekenzeichnet.
Slotted ALOHA	Das System verschickt Daten wenn es welche hat, hält aber dabei ein Zeitrahmen ein. Daraus folgt, dass sich Packete entweder ganz oder garnicht überschneiden und nicht mehr Halbüberschneidungen stattfinden. Dies benötigt jedoch eine Timerabstimmung unter den verschiedenen Systemen
Repeater/Hub	Nimmt Physicalische Signale, übersetzt sie einmal von Physikalisch -> Digital -> Physicalisch und schickt sie wieder verstärkt weiter
Switch	Genutz um mehrere Terminals untereinander zu verbinden ohne ein Router Netzwerk zu benutzen. Hierbei Identifizieren sich die verschiedenen Geräter mit ihrer MAC
Switch - Backwardslearning	Zu begin weiß ein Switch nicht, welche MAC Adresse an welchem Port verbunden ist. Dies wird erst zugeteilt, wenn dieses Gerät eine Information schickt. Schickt jetzt z.B. A eine Nachicht nach B, weiß der Switch jedoch nicht auf welchen Port es diese Nachicht schicken soll. Also wird zunächst die Nachicht an alle Terminals verschickt. Meldet sich B nun zurück kann diesem Port die Adresse von B zugeteilt werdne und bei der nächsten Nachicht für B wird nur noch an diesen Port geschickt.
Router	Router werden genutzt zur Kommunikation innerhalb eines Netzwerk bzw. zwischen Netzwerken (Diese Router werden Gateways genannat). Innerhalb eines Netzwerkes werden Intra-Domain-Routing-Methoden benutzt. Zwischen Netzen Inter-Domain-Routing-methoden.
Router - Intra-Domain-Routing - OSPF	Für Intra-Domain-Routing kann die Methode Open-Shortes-Path-First (OSPF) genutzt werden. Vorraussetzung ist, dass jeder Router das gesamte Routing Netz als Information vorhanden hat. Dies geschicht dadurch, dass ein Router, sobald er einen neuen Nachbarn hat, oder ein Nachbar wegfällt, diese Information an alle seiner Nachbarn weiterleitet die dann durch das gesamte Netz Propagieren. Die "länge" einer Verbindung ergibt sich durch das Messen eines Round-Trip signals. Wird nun eine Anfrage geschickt um innerhalb dieses Autonomen-Netzes zu kommunizieren wird der Dijkstra Algorithmus genutzt um den schnellsten Weg zu finden.
Router - Inter-Domain-Routing	Für die Kommunikation zwischen verschiedenen Netzwerken wird an den verschiedenen Gateways ein Boarder-Gateway-Protocol verwendet. Jeder der Router besitzt eine Tabelle mit Präfixen und dazugehörigen Bitmasken. Unser Packet wird an den Router weitergeleitet, der die beste Metrik für unser IP besitzt.
Internet Protocol (IP)	Das IP wird genutzt um jedem Terminal in einem Netwerk eine eindeutige Adresse zuzuweisen. Hierbei gibt es entweder IPv4 oder IPv6 Im Netzwerklayer werden zwei wertere Protokolle verwendet um Zusatzinformation zu bekommen: ARP und ICMP Das IP ist Hierarchisch aufgebaut. Das heißt verschiedene Netwerke werden in verschiedene Gruppen zusammengefasst. Je höher die Gruppe desto weniger IPs besitzt diese. Die höchste ist die A Gruppe und es steigt auf bis D (E ist reserviert für die zukünftige Nutzung)
ARP	Das ARP Protokoll wird verwendet um die MAC Adresse eines gewissen Terminals herrauszufinden. Hierzu wird an alle Teilnehmer des Systems gerufen: "Wer von euch hat diese IP?!?" und derjenige der als erstes Antwortet wird genommen. Falls keiner in diesem System die IP besitzt meldet sich der Router mit seiner MAC
ICMP	Das ICMP Protokoll wird verwendet um eine Rückmeldung an den versender zu schicken, falls das Packet zwischendurch verloren geht/die Time-To-Live abläuft. Dies ist ein kann und kein muss!
TCP	Das TCP Protokol ist Teil des Transportlayers. Es ist ein verbindungsorientiertes und sichers Protokol. D.h., dass eine zweiseitige Verbindung besteht durch die sichergestellt wird, dass alle Pakete in der richtigen Reihenfolge ankommen. Hierfür setzt jeder Empfänger einen Buffer fest, den er mit erhaltenden Packeten füllt. Um sicherzustellen, dass die Verbindung sowohl auf beiden Seiten richtig eingerichtet bzw. abgebaut wird benötigt man einen Timed 3 Part Handshake. Connection esteblishment: - SYN, SYN ACK, ACK Connection finish: - FIN, FIN ACK, ACK Ob alle Pakete angekommen sind wird durch ein dementspechendem ACK Signal verifiziert. Hierfür hat das jeweilige ACK die Sequenznummer+1 des vorherigen ACK Signals. SYN und FIN entspechen genau einem seq Wert
UDP	Das UDP Protokol ist ein Packetorientiertes einseitiges Verbindungsprotokol. Ob Daten in richtiger Reihenfolge bzw. Anzahl ankommen ist nicht garantiert -> Verlorende Packete werden nicht berücksichtigt.

Flowcontrol in TCP	Das TCP Protokol sieht vor, dass der Reciever nicht mit Packeten überflutet wird. Um die Sendegeschwindigkeit des Senders zu Kompensieren setzt der Empfänger einen gewissen Buffer fest, der linear gefüllt wird. Erhält der Empfänger Packages out-of-order so Speichert er diese in den Buffer und reicht die Packete erst an die Application weiter, wenn alle Packgages in der richtigen Reihenfolge vorhanden sind. Es gibt zwei Methoden um sicher zu gehen, dass dieser Buffer nicht geflutet wird: Der Sender fragt Speicherplatz an oder der Empfänger sendet keine Acknowledgements mehr.