Kommunikationsprotokolle

Allgemeines über Kommunikationsprotokolle

Protokolle sind formale Regeln zum Handeln (Funktionsvorschriften).

Protokolle koordinieren die Kommunikation zwischen Kommunikationspartnern.

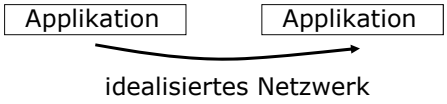
Protokolle liefern

- Adressierung des Kommunikationsendpunkts
- Kontrolle des Datenflusses
- Verlässlichkeit des Services (Datentransfer)

Kommunikation kennt zwei Arten an Protokollen

1. verbindungsorientierte Leitungsvermittlung
2. verbindungslose Paketvermittlung

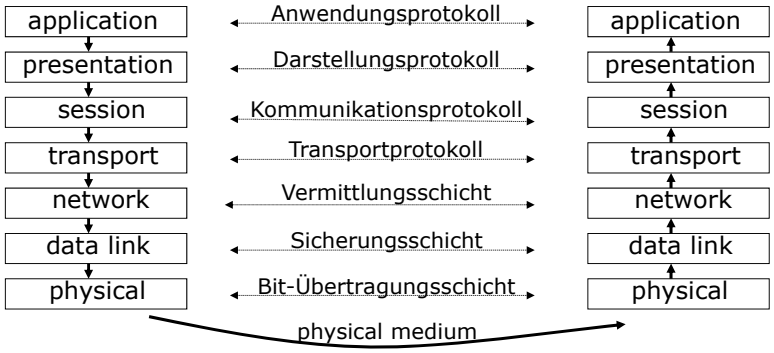
Protokolle



- verbindungsorientierte Leitungsvermittlung , z.B. öffentliches Telefonnetz:
  - Teilnehmer wird ein Übertragungskanal und dessen Bandbreite zur alleinigen Nutzung zur Verfügung gestellt
  - Verbindungsaufbau, Datenübertragung, Verbindungsabbruch
  - Merkmale: kurze Verweilzeiten der Nachrichten im Netz, ungenutzte Übertragungskapazitäten
- verbindungslose Paketvermittlung, z.B. Internet:
  - Nachrichtenzerlegung in individuell adressierte Pakete
  - Datenpakete werden in Netzknoten zwischen gespeichert (store&forward) ⇒ Verzögerungen möglich, aber eine bessere Ausnützung der Übertragungskanäle/ Netzzugänge
  - Quittung an den Sender für jedes korrekt empfangene Paket

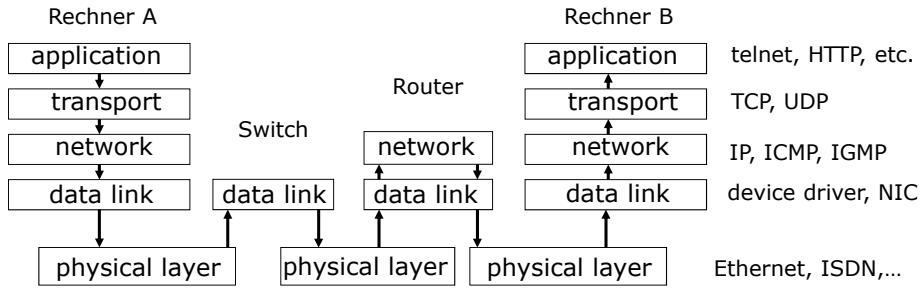
Netzwerkmodelle

OSI Referenzmodell



- im OSI-Modell ist ein Netz in sieben Schichten aufgeteilt
- jede Schicht kommuniziert nur mit seiner oben und unten benachbarten Schicht

Internet Referenzmodell



- UDP
  - verbindungsloser, ungesicherter Datentransport
  - Keine Garantie ob Daten in der richtigen Reihenfolge oder überhaupt ankommen
- TCP
  - verbindungsorientierter, gesicherter Datentransport
  - Verbindungsaufbau, Sortierung und automatische Wiederholung bei fehlerhafter Übertragung

TCP/IP-Protokoll

- TCP/IP ist eine Menge an Protokollen die kooperierenden Computern netzwerkweit existierende Ressourcen gemeinsam nutzbar machen
- Gesamtmenge an Protokollen wird als „Internet Protocol Suite" bezeichnet
- TCP (Transmission Control Protocol) und IP (Internet Protocol) sind die bekanntesten Protokolle dieser Suite
- TCP/IP wurde durch eine Gruppe Wissenschaftlern für das ARPAnet entwickelt (1973)

- IP mit Packetgrößen < 1500 Byte bzw. < 9000 Byte (Jumbo Frames, limitiert durch 32-bit CRC)

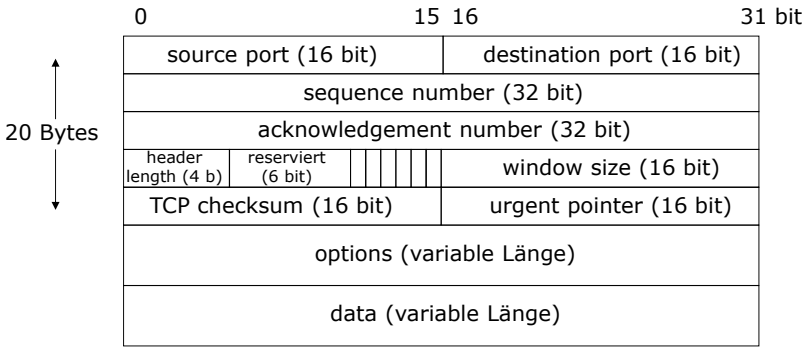
TCP Grundprinzip

- Byte-Stream-Transport
  - Anwendung übergibt Byte-Strom an TCP-Protokollmaschine
- Segmentierung
  - TCP-Protokollmaschine zerlegt Byte-Strom in nummerierte Segmente
- Sendefenster (*Congestion Window*)
  - Segmente innerhalb des Congestion Windows werden an IP-Schicht übergeben

1 2 3 4 5 6 7 8 9 10

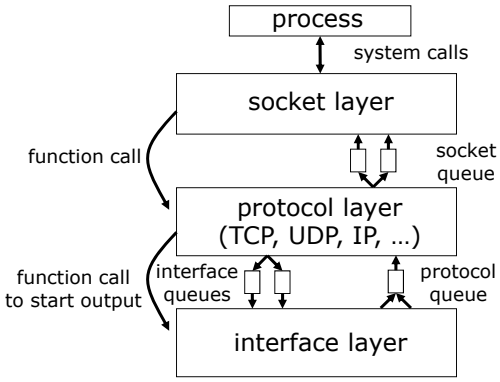
- Acknowledgements
  - Empfänger schickt Empfangsbestätigungen für jedes korrekt empfangene Segment
- Sliding Window
  - Für jedes bestätigte Segment wird das Congestion Window soweit nach rechts verschoben, dass sich links vom Sendefenster nur bestätigte Segmente befinden
- Retransmission
  - Unbestätigte Segmente werden erneut übertragen

Aufbau eines TCP-Segments



UNIX Netzwerk-Stack

Struktur des BSD Netzwerk-Stacks



IP-Packete

Preamble	Destination	Source	Frame Type	Data	Checksum
8 Bytes	6 Bytes	6 Bytes	2 Bytes	46-1500 Bytes	4 Bytes

Erweiterungen

- Gigabit-Ethernet Jumbo-Frames:
  - Packetgrößen bis zu 9000 Bytes (limitiert durch 32-Bit CRC)
- IPv6
  - Destination-, Source-Adressen: je 16 Bytes
    (Beispiel: 2001:0db8:85a3:08d3:1319:8a2e:0370:7344)

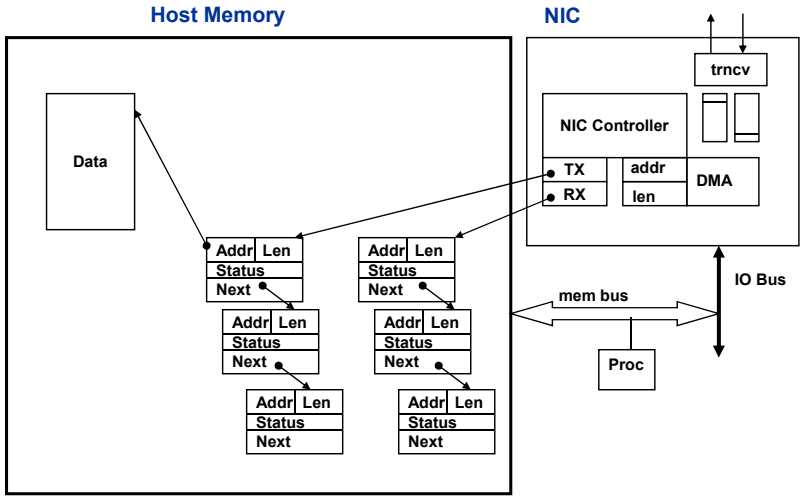
Aufbau eines Ethernet-Frames IPv4/TCP

<table>
<tr><td>OSI-Schicht</td><td>TCP/IP-Schicht</td><td colspan="12">Struktur</td></tr>
<tr><td>4</td><td>Transport</td><td colspan="8"></td><td>TCP-Header</td><td>Nutzlast (1460 bytes)</td><td colspan="2"></td></tr>
<tr><td>3</td><td>Internet</td><td colspan="7"></td><td>IP-Header</td><td colspan="3">Nutzlast (1480 bytes)</td><td colspan="2"></td></tr>
<tr><td rowspan="2">2</td><td rowspan="2">Netzzugang</td><td colspan="2"></td><td>MAC-Empfänger</td><td>MAC-Absender</td><td>802.1Q-Tag (opt.)</td><td>EtherType</td><td colspan="4">Nutzlast (1500 bytes)</td><td>Frame Check Sequence</td><td></td></tr>
<tr><td>1</td><td>Präambel</td><td>Start of Frame</td><td colspan="8">Nutzlast (1518/1522 bytes)</td><td>Interframe Gap</td></tr>
<tr><td colspan="2">Oktette</td><td>7</td><td>1</td><td>6</td><td>6</td><td>(4)</td><td>2</td><td>20</td><td>20</td><td>≤1460</td><td>4</td><td>12</td></tr>
</table>

IP Protokoll im Cluster

- Das Internet-Protokoll (IP) ist weltweit Standard für Netzwerke (best-effort Ansatz)
- Transmission Control Protocol (TCP) meistens eingesetzt
- Internet-Protokolle haben Nachteile für Cluster:
  - Overhead durch das Betriebssystem (viele Prozesswechsel)
  - Durch die Strukturierung des TCP/IP in mehrere Schichten
    => zwangsläufig mehrfaches Kopieren im Speicher
    (Reduzierung von Applikation nutzbaren Speicherbandbreiten)
  - Durch Schichtenstruktur schlechte Programmlokalität (Cache Ineffizienz)
  - Die Flusskontrollmechanismen des TCP (viele Paketverluste bei hoher Last)

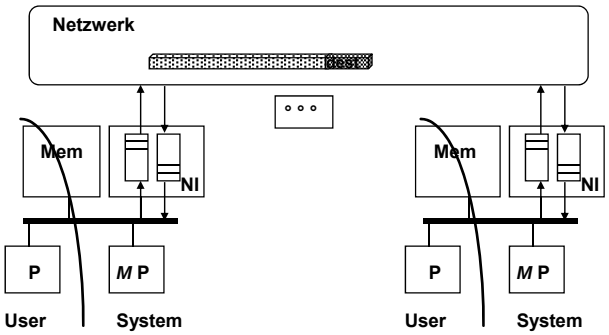
Konventionelles LAN Network Interface (NIC)



Kommunikationsprotokoll für Cluster

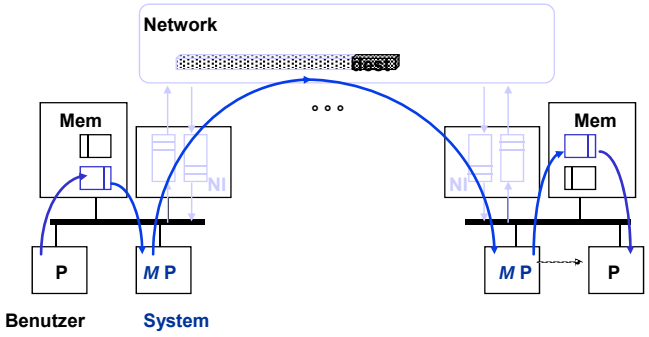
- Spezialisierung auf Message-Passing
  - Leichtgewichtige Kommunikationsprotokolle
  - Unterstützung großer Pakete
  - Schnelle Synchronisation
  - Gruppenkommunikationen
- Sichere Netzwerkübertragung
  - Fehlerbehandlungen ist absolute Ausnahme
- Einfache Switches
  - Optimierte Paketierung
  - Einfaches Routing (schnelle Wegewahl)
  - Keine Umsetzung zwischen Protokollen
- Mehr Unterstützung durch die Hardware im NIC
  - Beschleunigung der Protokollbearbeitung in den Komponenten Netzwerkinterface (NIC) und Switch

Nachrichtentransport ohne Hardwareunterstützung



- (dedizierter) Prozessor bearbeitet Nachrichtenausgabe auf Systemebene und interpretiert einkommende Nachrichten auf Systemebene
- User-Prozessor ↔ Msg-Prozessor (MP) via Shared-Memory
- Msg-Prozessor ↔ Msg-Prozessor via Netzwerktransaktion

Nachrichtenaustausch im Cluster

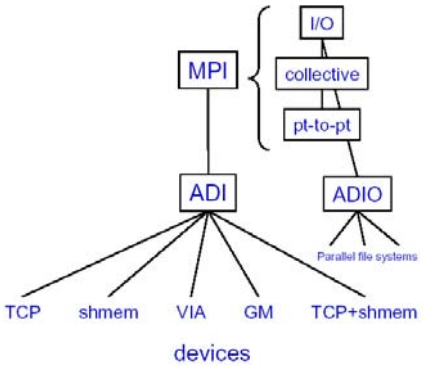


- Benutzer-Prozessor speichert Cmd/Msg/Data in Ausgabe-Queue
  - Test ob Queue bereits voll (oder „elastischer" Bereich)
- Kommunikationsassistent (Proz.) führt Transaktion aus
  - Checks, Transaktion, Scheduling, Transport, Interpretation
- Flusskontrolle auf mehreren Protokollebenen

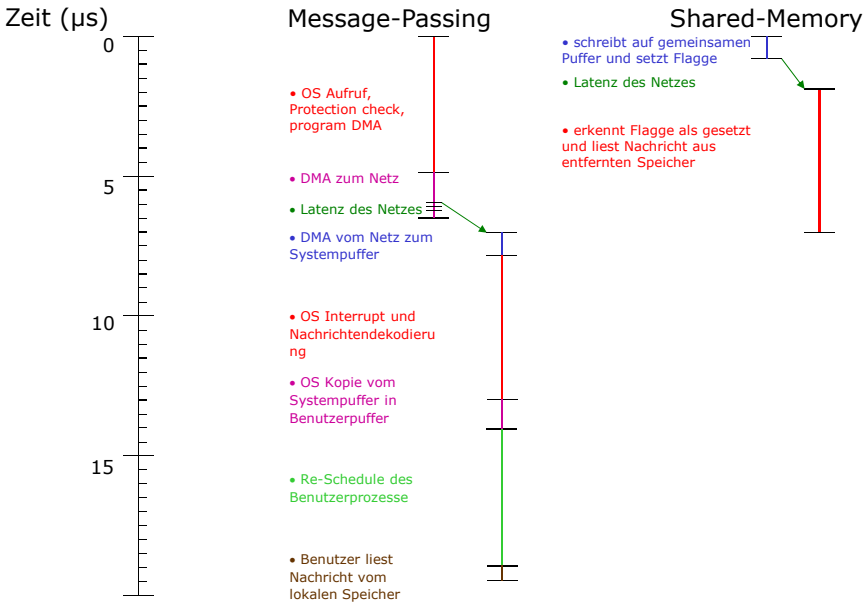
Message Passing Interface: MPIch Design

- MPI - Message Passing Interface
- Standardisiert (MPI-1, MPI-2)
- Open Source Implementierung MPIch

MPIch
- Argonne National Lab
- Aufteilung in Kommunikations-netzwerk unabhängigen und abhängigen (ADI) Teil
- Optimierung für verschiedene Kommunikationsprotokolle



Message-Passing vs. Shared-Memory



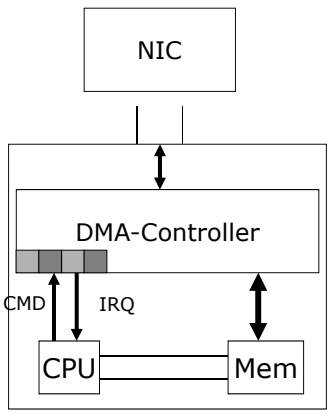
Quelle: Lenosky/Weber95

Techniken zur Beschleunigung

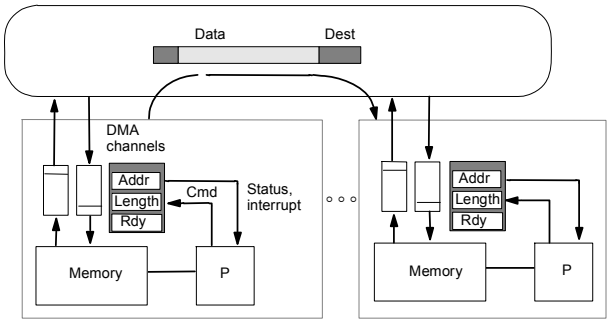
- Direct Memory Access (DMA)
  - Hardware unterstütztes Kopieren im lokalen Speicher
- OS-Bypass
  - Reduzierung der Anzahl an Prozesswechsel (Anwenderprozess, Systemprozess, Anwenderprozess)
  - Verzicht auf teure Interrupt-Behandlung
- Zero-Copy Protokolle
  - Trotz Hardware-Unterstützung (DMA) teuer
  - Speicherbandbreite ist Engpass
- Remote DMA
  - Hardware unterstütztes Kopieren in entfernten Speicher

Direct-Memory-Access (DMA)

- DMA-Controller auf Knoten
- DMA-Transfer
  - Transferiert nur Speicherblöcke
  - Kommando besteht aus Quell-, Zieladresse und Blocklänge
  - IRQ bei Nachrichten Aus-/Eingang
- Führt zur Entlastung der Applikationsprozessoren
- Auch Listen an Blöcke per DMA übertragbar



Netzwerk-Transaktionen: Hardware DMA



- DMA ist durch Register kontrolliert, erzeugt Interrupts
- DMA meist „blind physikal" → Betriebssystem initiiert Transfers
  - Sende-Seite: konstruiert systemseitig innerhalb des Kernels „Umschlag" für Benutzerdaten
  - Empfänger-Seite: muss im System-Buffer empfangen, da keine Interpretation durch CA
- Logik befindet sich meistens direkt auf dem NIC

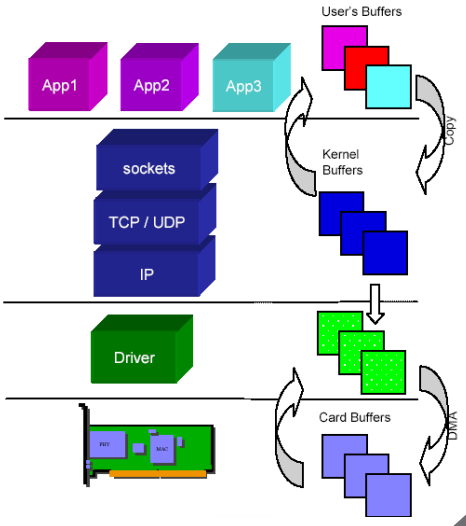
OS-/ Application-Bypass

- OS-Bypass
  - Daten fließen direkt in den Adressbereich der Anwendung
  - Keine Interrupts ins Betriebssystem beim Empfangen von Nachrichten
  - Adressvalidierung geschieht durch das Betriebssystem (Initialisierungsphase) bevor Nachrichten übertragen werden
- Application-Bypass
  - Anwenderprogramm ist nicht beim Empfangen der Nachrichten involviert
    - Gut bei langen Nachrichten
    - Acknowledgements notwendig
  - Beispiel: nicht-blockierendes MPI_receive während die Anwendung numerisch intensive Berechnung durchführt
    - Kommunikation wird nebenläufig ausgeführt
    - Verdeckung der Übertragungszeit

Zero-Copy

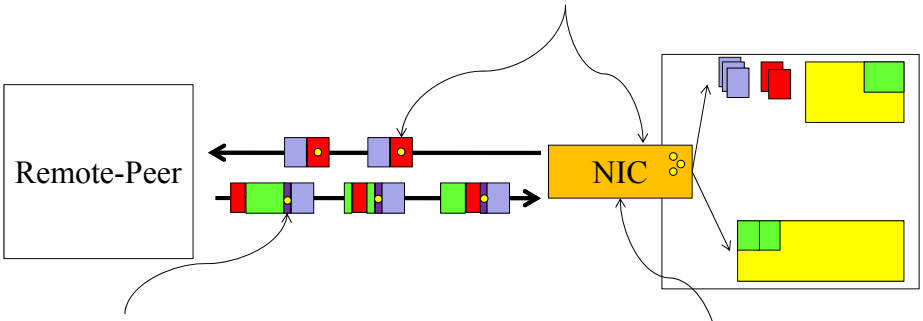
Einsparung überflüssiger Kopieroperationen

- Ziel:
  - Reduzierung der Latenzzeit der Kommunikation
  - Reduzierung des Bandbreitenbedarfs im Hauptspeicher
- Methoden:
  - Optimale Platzierung der Kommunikationsbuffer (Direct Memory Placement)
  - Kommunmikationsbuffer im Adressbereich der Anwendung
  - Remote DMA (RDMA) Direkt Memory Access über Knotengrenzen hinweg



Remote DMA (RDMA)

Registrierung von Puffern für Steering-Tags im NIC; Weiterleitung an Remote-Peer.

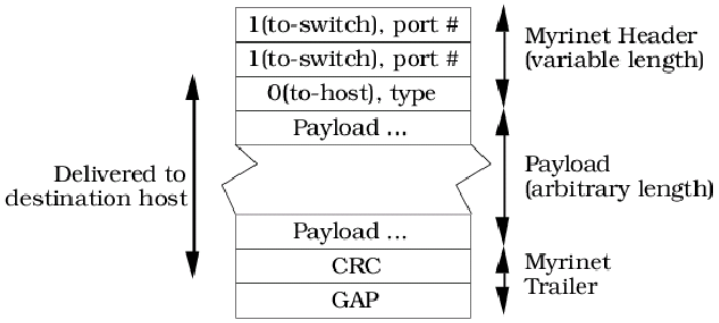


RDMA-Rahmen transportieren Direktiven und Steering-Tags im Datenstrom.

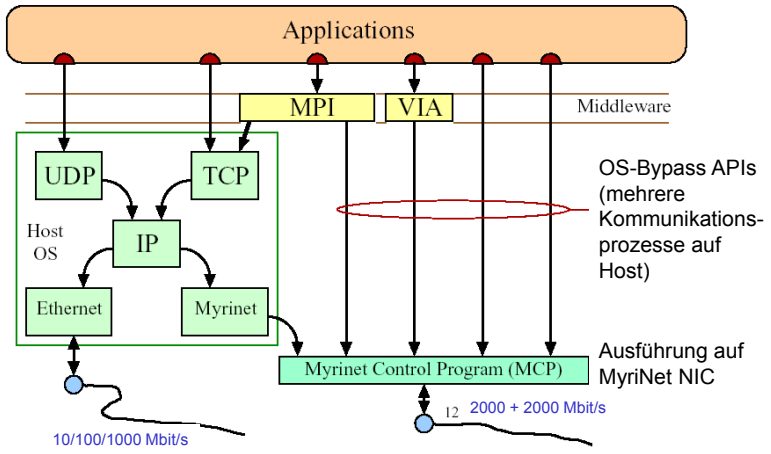
Direktiven und Steering-Tags steuern NIC Datenplatzierung.

Beispiel: MyriNet

- MyriNet ist auf Data-Link Ebene, (Ebene 2) des ISO Referenzmodells, definiert
- MyriNet beschreibt Paketformat und Flusskontrolle
- Einfaches Packetvermittlungsverfahren
- Mehrere Implementierungen der physikalischen Ebene
- Source-Routing mit Header-Reduzierung



Beispiel: MyriNet

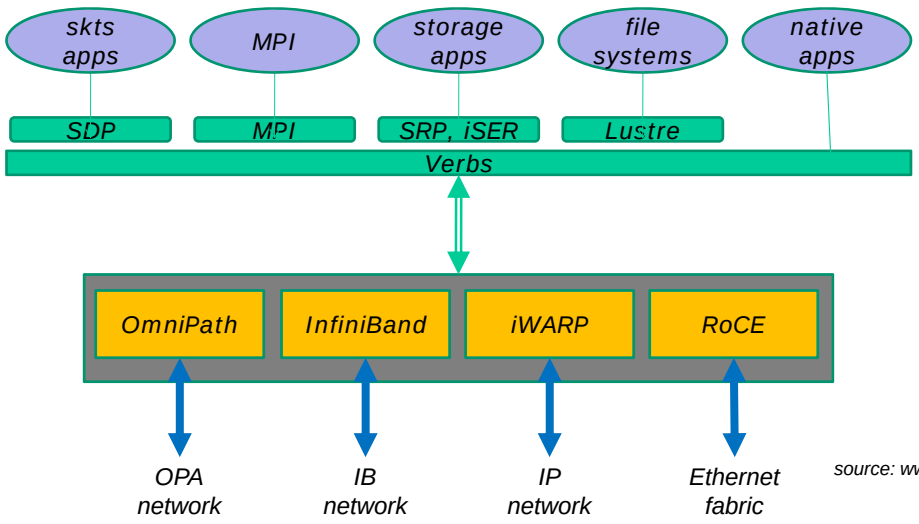


- Aufteilung der Software-Schicht auf Rechenknoten und NIC
- MPI von MyriNet setzt auf eine Zwischenschicht (MX) ein
  - Schutz der Kommunikationshardware vor unbefugten Zugriff
  - Koordinierung gleichzeitiger Zugriffe

MX Message-Passing System

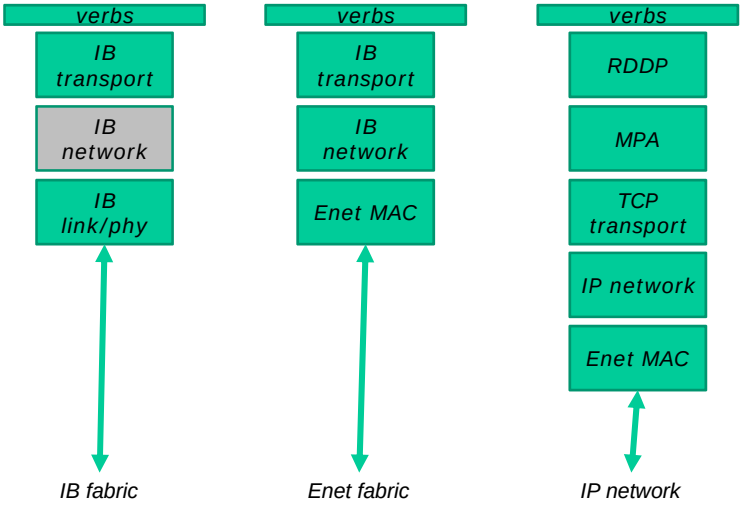
- MPI von MyriNet setzt auf eine Zwischenschicht (MX) auf
- Schnittstelle zur Anwenderebene / MPI
  - Schutz der Kommunikationshardware vor unbefugten Zugriff
  - Koordinierung gleichzeitiger Zugriffe
- Paketierung
  - Segmentierung und Zusammensetzen großer Pakete
- High-Level Flusskontrolle
  - Sicherer, geordneter Nachrichtentransport
  - Toleranz gegen Netzwerkfehler

Example: OpenFabrics



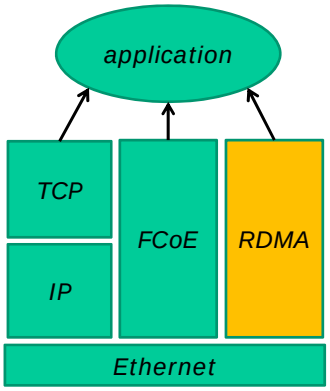
source: www.openfabrics.org

3 Transport-Protokolle



Quelle: www.openfabrics.org

Why RoCE?



-Networking via TCP/IP

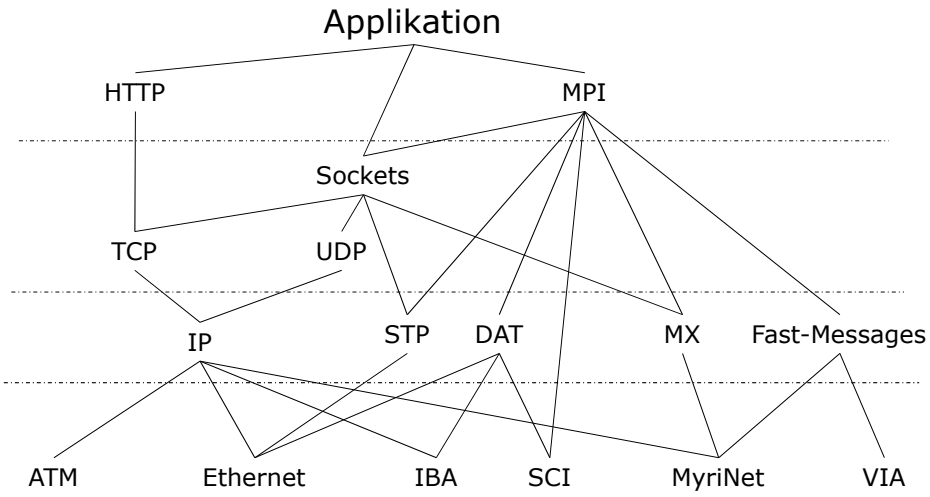
-Storage via FCoE

-Low latency via RoCE

This completes the Ethernet Trilogy – networking, storage, IPC

Quelle: www.openfabrics.org

Protokoll-Hierarchie: Beispiel



Routing

Routing-Verfahren

Übliche Packetvermittlungsverfahren

- *Wormhole-Routing*
  - Nachrichten-*Flits* bleiben bei Kollision in den Verbindungselementen stehen und blockieren andere Nachrichten
- *Virtual-Cut-Through*
  - Nachrichten werden bei Kollision in den Verbindungselementen vollständig gespeichert

Routing

- Routing-Algorithmus bestimmt
  - welche möglichen Wege als Route genutzt werden können
  - wie die Route festgelegt wird
  - R: $V \times V \rightarrow C$
    - C ist eine Kanalliste
- Fragen:
  - Routing-Mechanismus
    - Berechnungsvorschrift
    - Kanalauswahl durch Quellknoten (source-based)
    - Tabellen-getriebene Kanalauswahl im Schalter (table-driven)
  - Eigenschaften der Route (Latenzzeit, Kollisionen,…)
  - mögliche Verklemmungen (deadlock-free?)

Routing-Mechanismen

- Source-based
  - Nachrichtenkopf enthält Folge an Kanal-Adresssen
  - Kanal-Adresse verwenden und en-route von Paket entfernen
  - *CRC? Paketformat?*
  - Beispiele: Myrinet, QsNet, …

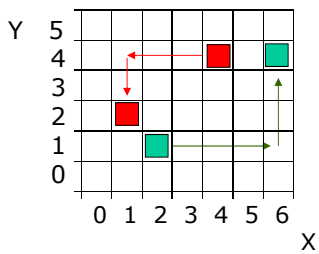
Daten	K_3	K_2	K_1	K_0

- Table-driven
  - Nachrichtenkopf enthält Index (Zielknotennummer) für Routing-Tabelle des nächsten Schalters
  - zusätzlicher Nachrichtenkopf für Auswahl der Routing-Tabelle möglich
  - Beispiele: IB, GbE, …

Daten	Ziel-ID	Tab.-ID

Routing-Mechanismen

- Wahl des Ausgangskanals für eingehende Pakete
  - Ziel: Funktion schnell zu berechnen
- einfache Arithmetik in regulären Topologien
  - z.B. X,Y-Routing in einem Gitter ($\Delta X = X_{Ziel} - X_{Quelle}$, $\Delta Y = Y_Z - Y_Q$)
    - West (-x) $\Delta X < 0$
    - Ost (+x) $\Delta X > 0$
    - Süd (-y) $\Delta X = 0$ und $\Delta Y < 0$
    - Nord (+y) $\Delta X = 0$ und $\Delta Y > 0$
    - Knoten erreicht $\Delta X = 0$ und $\Delta Y = 0$



$\Delta X = 6-2 = 4$
$\Delta Y = 4-1 = 3$

$\Rightarrow$ 4 mal Ost und 3 mal Nord

$\Delta X = 1-4 = -3$
$\Delta Y = 2-4 = -2$

$\Rightarrow$ 3 mal West und 2 mal Süd

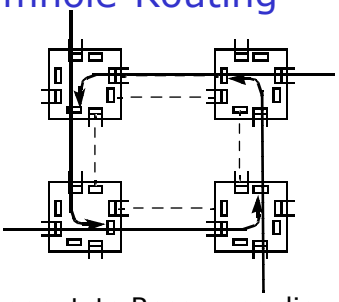
Eigenschaften von Routing-Algorithmen

- Deterministisch:
  - Route bestimmt durch (Quelle, Ziel), keine sonstige Beeinflussung, z.B. durch Verkehr im Netzwerk
- Dispersive:
  - Nachfolgende Pakete mit unterschiedlicher Wege Wahl in Source-Routed-Network
- Adaptive:
  - Route durch Verkehr auf dem Weg beeinflussbar
- Minimal:
  - nur kürzeste Wege genutzt
- Verklemmungsfrei (deadlock-frei):
  - kein Kommunikationsmuster kann zu einer Situation führen in der ein Packet nicht mehr weiter transportiert werden kann

Deadlock-Freiheit beim Wormhole-Routing

Wie entsteht eine Verklemmung i.A.?

Notwendige Bedingungen:
- gemeinsam genutzte Ressource
- inkrementelle Ressourcenzuweisung
- Ressource nicht entziehbar (non-preemptible)



- Betrachte einen Kanal als gemeinsam genutzte Ressource die inkrementell erworben wird
  - Quell-Buffer, dann Ziel-Buffer
  - Kanalnutzung entlang der Route
- Wie kann eine Verklemmung verhindert werden?
  - Art der Zuweisung der Kanal-Ressource vorschreiben
  - Beispiel: Dimension-Order Verfahren
- Wie kann bewiesen werden, dass ein Routing deadlock-frei ist?

Beweistechnik

- Kanal entspricht einer Ressource
- Nachrichten erzeugen Abhängigkeiten zwischen Ressourcen beim Transport über eine Route
- Beschreibung möglicher Abhängigkeiten von Kanälen notwendig
- Zu zeigen ist, dass kein Zyklus in dem entstehenden Kanalabhängigkeitsgraphen existiert
  - finde eine Nummerierung der Kanalressourcen, bei der jede legale Route einer monotonen Zahlenfolge entspricht

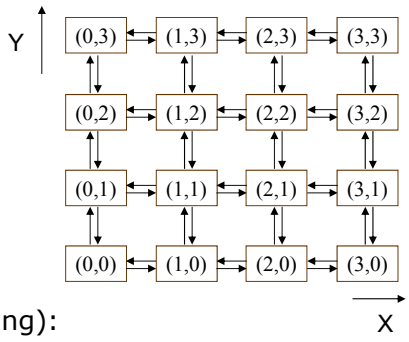
$\Rightarrow$ kein Kommunikationsmuster kann zu einem Deadlock führen

Bemerkung: Nur hinreichende Bedingung. Im Allgemeinen muss ein Channel-Dependency-Graph nicht azyklisch sein.

Beispiel: X,Y-Routing

Sei Gitter $<a_1,a_2>$

Gesucht:
Minimales und deterministisches Routing im 2-dim. Gitter mit bidir. Kanälen



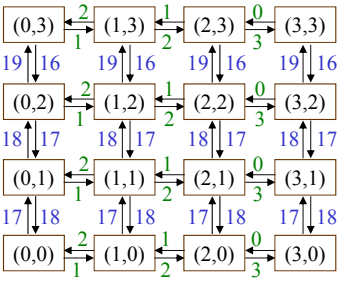
X,Y-Routing (Dimension-ordered Routing):

Sende Nachricht von Knoten *(x_Q,y_Q) nach Knoten (x_Z,y_Z)*

- *Schicke Nachricht in X-Richtung bis x_Z erreicht*
- *„Drehung“*
- *Schicke Nachricht in Y-Richtung bis y_Z erreicht*

Beispiel: X,Y-Routing im 2-d Gitter

Gitter $< a_1 \times a_2 >$ mit $N = a_1 \cdot a_2$



Bsp. Gitter <4,4>

Theorem: X,Y-Routing ist verklemmungsfrei

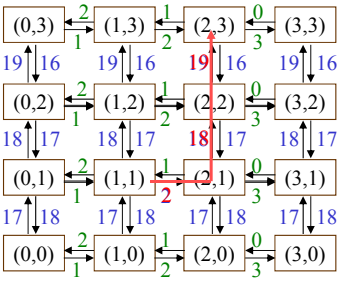
Nummerierung der gerichteten Kanten:

- x-Richtung:
  - $\forall\ 0<i<a_1$: $<i,y> \rightarrow <i+1,y>$ wird zu $i+1$
  - $\forall\ 0<i<a_1$: $<i,y> \rightarrow <i-1,y>$ wird zu a_1-i-1
- y-Richtung:
  - $\forall\ 0<j<a_2$: $<x,j> \rightarrow <x,j+1>$ wird zu $N+j+1$
  - $\forall\ 0<j<a_2$: $<x,j> \rightarrow <x,j-1>$ wird zu $N+a_2-j-1$

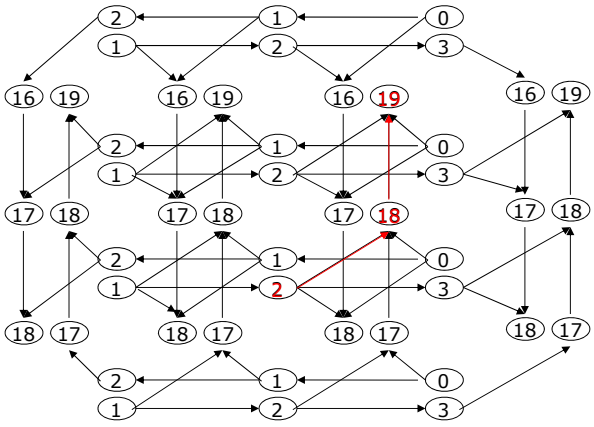
Frage: Ist jede Routing-Sequenz aus X-Richtung, Drehung, Y-Richtung aufsteigend? Beweis über Kanalabhängigkeitsgraphen.

Kanalabhängigkeitsgraph

gerichteter Graph



Konstruktion des „dualen Graphen"



keine Kante von i → j mit j ≤ i

⇒ kein Kreis enthalten