JOHANNES GUTENBERG UNIVERSITÄT MAINZ

NA62 P326

# Design und Entwicklung der Online-PC-Farm des NA62-Experiments am CERN

**Diplomarbeit**
am Fachbereich Physik, Mathematik und Informatik der
Johannes Gutenberg-Universität Mainz
Institut für Physik

vorgelegt von
**Jonas Kunze**
geboren in Worms

Mainz, den 17. April 2012

| | |
|---|---|
| Betreuer: | Dr. Rainer Wanke |
| 1. Korrektor: | Prof. Dr. Volker Büscher |
| 2. Korrektor: | Prof. Dr. Josef Pochodzalla |

Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it? - Brian W. Kernighan [1]

# Vorwort

Ziel des NA62-Experiments am CERN ist es, das Matrixelement $|V_{td}|$ der CKM-Matrix mit einer Genauigkeit von ca. 10% zu bestimmen. Dies ermöglicht die Suche nach Physik jenseits des Standardmodells der Elementarteilchenphysik. Innerhalb der Jahre 2014 und 2015 sollen dazu ca. 100 der äußerst seltenen Zerfälle $K^+ \to \pi^+ \nu\bar{\nu}$ gemessen werden. Das Standardmodell der Elementarteilchenphysik sagt für diesen Zerfall ein Verzweigungsverhältnis von ca. $8 \cdot 10^{-11}$ vorher, so dass bei einer Signalakzeptanz von etwa 10% insgesamt $\mathcal{O}(10^{13})$ Kaon-Zerfälle untersucht werden müssen. Dabei wird ein Signal-zu-Untergrund-Verhältnis von 10:1 angestrebt.

Das Experiment ist so konzipiert, dass durch Kaon-Zerfälle und Untergrund mit einer Rate von ca. 10 MHz Trigger-Signale ausgelöst werden. Die verschiedenen Detektoren des Experiments produzieren bei jedem Trigger-Signal ca. 227 Kilobyte an Rohdaten, so dass insgesamt durchschnittlich 2,3 Terabyte Daten pro Sekunde verarbeitet werden müssen. Obwohl man einen möglichst großen Anteil dieser Daten speichern möchte, ist die maximale Rate aus finanziellen Gründen begrenzt. Die benötigte Reduktion der Daten beträgt einen Faktor der Größenordnung $\mathcal{O}(10^{-4})$ und wird durch ein dreistufiges Triggersystem realisiert. Die erste Stufe ist in der Ausleseelektronik des Experiments implementiert und reduziert den Datenstrom um eine Größenordnung. Die restlichen beiden Stufen sind in Form von Software auf der Online-PC-Farm realisiert.

Innerhalb dieser Arbeit wurde ein Konzept für die Online-PC-Farm entwickelt und das benötigte Framework zur Datenverarbeitung implementiert. Dabei wurden verschiedene Aspekte unterschiedlicher Netzwerk-Topologien verglichen und eine Methode entwickelt, mit der mehr als 20% der Hardware eingespart wird.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Kapitel 1.

# Einführung

## 1.1. Motivation für das NA62-Experiment

### 1.1.1. Das Standardmodell der Teilchenphysik

Die Teilchenphysik beschäftigt sich mit der Frage „Woraus ist unsere Welt gemacht?“. Noch bis zu Beginn des 20. Jahrhunderts galten Atome als unteilbare Grundbausteine der Materie. Im Jahre 1911 entdeckte Ernest Rutherford die Substruktur des Atoms [2] und bis heute hat sich das Bild von Materie weiter verschärft. Als eines der wichtigsten Modelle der Physik beschreibt das **S**tandard**m**odell der Teilchenphysik (SM) alle heute bekannten Elementarteilchen, aus deren Kombination die bekannte Materie besteht, sowie die für den Mikrokosmos wichtigen Wechselwirkungen [3].

Die Elementarteilchen des SMs gruppieren sich zu den drittelzahlig geladenen Quarks und den ganzzahlig geladenen Leptonen mit halbzahligem Spin, sowie den Eichbosonen mit ganzzahligem Spin. Dabei kommen Quarks in der Natur nur in gebundenen Zuständen vor und bilden entweder Baryonen (drei (Anti-)Quarks) oder Mesonen (ein Quark und ein Antiquark) mit jeweils ganzzahliger Ladung.

Die bisher experimentell nachgewiesenen Elementarteilchen innerhalb des SMs sind in Abbildung 1.1 zusammen mit den wichtigsten Eigenschaften aufgelistet. Die nicht aufgezeigten sechs Anti-Quarks und sechs Anti-Leptonen tragen dabei die jeweils zu ihren Partnern umgekehrte Ladung und haben sonst die gleichen Eigenschaften.

Neben den Teilchen als solchen werden innerhalb des SMs auch die möglichen Wechselwirkungen zwischen diesen beschrieben. Diese sind die elektromagnetische, die schwache, sowie die starke Wechselwirkung. Die Gravitation wird nicht berücksichtigt. Dabei dienen fünf Eichbosonen, auch Austauschteilchen genannt, der Übermittlung dieser Wechselwirkungen.

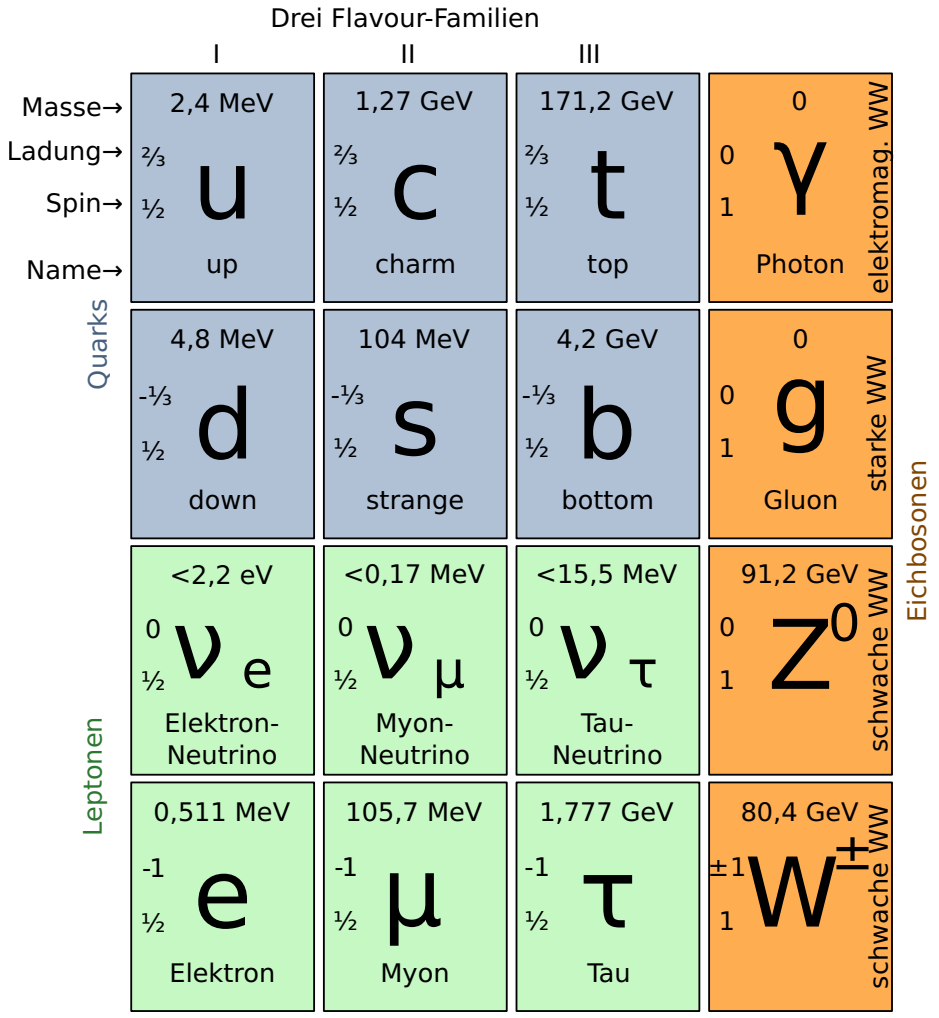


**Abbildung 1.1.:** Das Standardmodell der Teilchenphysik: Die bisher experimentell nachgewiesenen Elementarteilchen. Die jeweiligen Antiteilchen wurden nicht explizit aufgelistet. Die Daten stammen von [4].

Das SM wurde aufgestellt als nur drei der heute sechs bekannten Quarks experimentell nachgewiesen waren. So wurde das charm-Quark bereits 1970 vorhergesagt, jedoch erst 1974 das erste künstlich erzeugt[1]. Es folgten der Nachweis des bottom-Quarks im Jahre

[1]Das entdeckte Teilchen in dem das charm-Quark gebunden ist wird heute J/Ψ genannt. Grund dafür ist, dass es zeitnahe von zwei Arbeitsgruppen am Stanford Linear Accelerator Center und am Brookhaven National Laboratory entdeckt wurde, wobei jede Gruppe ihren Teil des heutigen Doppelnamens beitrug.

1977 und des wesentlich schwereren top-Quarks im Jahre 1995, jeweils am Fermi National Accelerator Laboratory.

Trotz des großen Erfolgs des SMs existieren noch einige Bereiche innerhalb der Teilchenphysik, die durch dieses Modell nicht beschrieben werden. So gibt es überzeugende Hinweise dafür, dass die Masse des Universums vier- bis fünfmal so hoch ist wie die tatsächlich sichtbare und durch das SM beschriebene Masse [5]. Der Schweizer Physiker und Astronom Fritz Zwicky führte 1933 zu dieser Erklärung den Begriff der „dunklen Materie“ ein, eine hypothetische Materie die nicht durch das SM beschrieben wird.

Ein weiteres Phänomen, welches innerhalb des SMs nicht erklärt werden kann, ist die Baryonenasymmetrie, auch als Materie-Antimaterie-Asymmetrie bezeichnet: Analysen der kosmischen Hintergrundstrahlung zeigen, dass deutlich mehr Baryonen als Antibaryonen im Universum existieren. Mit den Annahmen der Urknalltheorie widerspricht dies den Aussagen des SMs.

Der bisherige Erfolg des SMs in Verbindung mit dessen Lücken motiviert die Forschung nach Physik jenseits des Standardmodells. Das NA62-Experiment trägt einen Teil zu diesem Forschungsbereich bei.

### 1.1.2. Die Cabibbo-Kobayashi-Maskawa-Matrix

Wie in Abbildung 1.1 dargestellt, werden die bekannten Quarks und Leptonen in drei Kategorien, genannt Flavour-Familien, unterteilt. Die Flavour-Quantenzahl beschreibt innerhalb der Quantenchromodynamik eine globale Symmetrie wodurch sie bei starken Wechselwirkungen erhalten bleibt. Innerhalb der Theorie der elektroschwachen Wechselwirkungen ist diese Symmetrie jedoch gebrochen. Es existieren somit flavourändernde Prozesse, welche im Falle der Quarks durch die **C**abibbo-**K**obayashi-**M**askawa (CKM)-Matrix beschrieben werden.

Die CKM-Matrix V beschreibt die Mischung der Flavoureigenzustände ($|d\rangle$, $|s\rangle$, $|b\rangle$) zu den Eigenzuständen der schwachen Wechselwirkung ($|d'\rangle$, $|s'\rangle$, $|b'\rangle$) und wird in Gleichung 1.1 dargestellt. Demnach ist die Wahrscheinlichkeit für einen Prozess, in dem ein u-artiges Quark i ($i \in \{u, c, t\}$) in ein d-artiges Quark j ($j \in \{d, s, b\}$) umgewandelt wird, proportional zu $|V_{ij}|^2$.

$$\begin{pmatrix} |d'\rangle \\ |s'\rangle \\ |b'\rangle \end{pmatrix} = \begin{pmatrix} V_{ud} & V_{us} & V_{ub} \\ V_{cd} & V_{cs} & V_{cb} \\ V_{td} & V_{ts} & V_{tb} \end{pmatrix} \cdot \begin{pmatrix} |d\rangle \\ |s\rangle \\ |b\rangle \end{pmatrix} =: V \cdot \begin{pmatrix} |d\rangle \\ |s\rangle \\ |b\rangle \end{pmatrix} \tag{1.1}$$

Die CKM-Matrix $V$ ist innerhalb des SMs unitär ($V^\dagger = V^{-1}$). Die Überprüfung der Unitarität ist Bestandteil aktueller Forschungen. Sollte sich diese Eigenschaft der CKM-Matrix nicht bestätigen, so ist dies ein Zeichen für neue Physik jenseits des Standardmodells[2]. Dazu müssen alle Matrixelemente mit möglichst hoher Genauigkeit gemessen werden. Die Bestimmung des Matrixelements $|V_{td}|$ ist das Hauptziel des NA62-Experiments.

### 1.1.3. Der Zerfall $K^+ \to \pi^+\nu\bar{\nu}$

Zur Bestimmung des CKM-Matrix-Elements $|V_{td}|$ ist der Zerfall $K^+ \to \pi^+\nu\bar{\nu}$ besonders gut geeignet. Dabei zerfällt das strange-Quark des Kaons in ein down-Quark und bildet mit dem up-Quark des Kaons ein Pion. Dieser Prozess kann innerhalb des SMs nicht direkt, also durch Austausch eines einzelnen $Z^0$-Bosons, stattfinden. Stattdessen findet der Zerfall über eine Schleife weiterer Quarks (u, c oder t) in Form eines Pinguin- oder Box-Diagramms statt (siehe Abbildung 1.2).

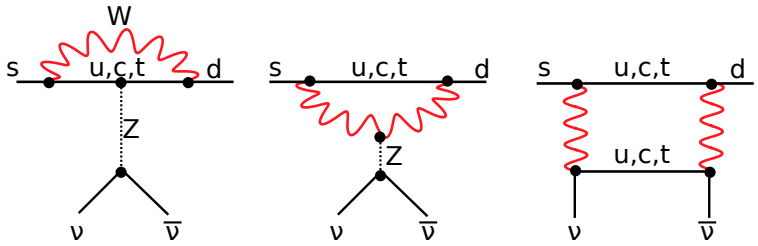


**Abbildung 1.2.:** Feynman-Diagramme des Zerfalls $K^+ \to \pi^+\nu\bar{\nu}$. Links und Mitte: Pinguin-Diagramme. Rechts: Box-Diagramm [6].

Für den Zerfall $K^+ \to \pi^+\nu\bar{\nu}$ setzt der quadratische GIM-Mechanismus ein. Das bedeutet, dass die Vertices proportional zum Massenquadrat des virtuellen Quarks ankoppeln, wie in der Amplitude 1.2 ersichtlich wird [7].

$$A_q \propto \frac{m_q^2}{m_W^2} V_{qs}^* V_{qd} \qquad q = u, c, t. \tag{1.2}$$

Somit ist der dominante Term durch das top-Quark, dem schwersten aller bekannten Quarks, gegeben. Das up-Quark kann vernachlässigt werden, und der Term des charm-Quarks geht als kleine Korrektur ein.

[2]So ist die Existenz einer vierten Fermionen-Generation, also unter anderem ein siebtes und achtes Quark, denkbar.

Durch die hohe Masse des top-Quarks hat dieser Prozess eine extrem kurze Reichweite und kann somit als Punktwechselwirkung betrachtet werden. Daher wird der Zerfall durch eine Fermi-Kopplung gut beschrieben [7]:

$$\mathcal{H}_{eff} = \sum_{l=e,\mu,\tau} \frac{G_l}{\sqrt{2}} (\bar{s}d)_{V-A} (\bar{\nu}_l \nu_l)_{V-A}. \tag{1.3}$$

Dabei ist $G_l$ die effektive Kopplungskonstante[3].

Mit 1.3 kann das Verzweigungsverhältnis (engl. branching ratio, kurz BR) des Zerfalls $K^+ \to \pi^+ \nu\bar{\nu}$ direkt mit dem BR des Zerfalls $K^+ \to \pi^0 e^+ \nu$ (kurz $K^+_{e3}$) verknüpft werden:

$$\mathrm{BR}(K^+ \to \pi^+ \nu\bar{\nu}) = 6 \cdot r_{K^+} \mathrm{BR}(K^+ \to \pi^0 e^+ \nu) \frac{|G_l|^2}{G_F^2 |V_{us}|^2}. \tag{1.4}$$

Dabei stellt $r_{K^+} = 0.901$ eine Isospin-Korrektur dar, welche Phasenraum- und QED-Effekte berücksichtigt [8]. Neben der bekannten Fermi-Kopplungskonstante $G_F$ sind das Matrixelement $|V_{us}|$ und das Verzweigungsverhältnis $\mathrm{BR}(K^+_{e3})$ experimentell bestimmt [9]:

$$\begin{aligned} |V_{us}| &= 0,2253 \pm 0,0009, \\ \mathrm{BR}(K^+_{e3}) &= (5.078 \pm 0,031)\%. \end{aligned}$$

$G_l$ lässt sich somit nach Gleichung 1.4 durch die Messung von $\mathrm{BR}(K^+ \to \pi^+ \nu\bar{\nu})$ bestimmen. Dabei setzt sich $G_l$, nach der genannten Vernachlässigung der Beiträge mit up-Quarks in den Feynman-Diagrammen (siehe Abbildung 1.2), folgendermaßen zusammen:

$$G_l = \frac{\alpha G_F}{2\pi \cdot sin^2\Theta_W} \left[ V^*_{ts} V_{td} X(x_t) + V^*_{cs} V_{cd} X^l_{NL} \right], \tag{1.5}$$

$$x_t = \frac{m_t^2}{M_W^2}. \tag{1.6}$$

[3]Nach [7] existiert eine kleine Differenz zwischen den Kopplungskonstanten für $\nu_\tau$ und $\nu_{e,\mu}$. Definiert man $G_l$ als den Mittelwert dieser drei Werte, so erhält man einen vernachlässigbaren Fehler von unter 0,2%.

Dabei wurden die Koeffizienten $X_{NL}^{l}$ in führender Ordnung der QCD bestimmt und $X(x_t)$ ist eine bekannte kinematische Funktion [10] [11]. Zusammen mit den experimentell bestimmten Werten der verschiedenen Matrix-Elemente kann der Erwartungswert $G_l$ und somit das Verzweigungsverhältnis des Signalkanals, im Rahmen des Standardmodells bestimmt werden [12]:

$$\mathrm{BR}(K^+ \to \pi^+ \nu\bar{\nu})_{SM} = (7,81 \pm 0,75 \pm 0,29) \cdot 10^{-11}. \tag{1.7}$$

Die hohe Genauigkeit des theoretisch vorhergesagten Wertes in Kombination mit der extrem starken Unterdrückung des Prozesses nach dem SM motiviert die experimentelle Untersuchung dieses Verzweigungsverhältnisses. Schon kleine Beiträge durch neue Physik jenseits des SMs würden sich stark bemerkbar machen. Messungen könnten somit leicht Hinweise auf neue Physik liefern oder andererseits das SM weiter bekräftigen.

Bisher konnten erst sieben Kandidaten des Zerfalls $K^+ \to \pi^+ \nu\bar{\nu}$ am BNL E949-Experiment gemessen werden. Dabei wurde das Verzweigungsverhältnis nur mit einem relativen Fehler von über 60 % bestimmt [13]:

$$\mathrm{BR}(K^+ \to \pi^+ \nu\bar{\nu})_{Exp} = (17,3^{+11,5}_{-10,5}) \cdot 10^{-11}.$$

## 1.2. Das NA62-Experiment

Das NA62-Experiment ist ein Fixed-Target-Experiment am CERN bei Genf. Ziel des Experiments ist die Bestimmung des Verzweigungsverhältnisses des Kaon-Zerfalls $K^+ \to \pi^+ \nu\bar{\nu}$ mit einem relativen Fehler von höchstens 10%. Innerhalb von zwei Jahren sollen dazu ca. $10^{13}$ Kaon-Zerfälle untersucht werden. Bei der Signalakzeptanz von rund 10% und dem durch das SM prognostizierte Verzweigungsverhältnis von rund $8 \cdot 10^{-11}$ sollen somit rund 100 dieser extrem seltenen Zerfälle gemessen werden.

Zur Produktion der geladenen Kaonen werden Protonen mit einer kinetischen Energie von 400 GeV aus dem **S**uper **P**roton **S**ynchrotron (SPS) auf ein Beryllium-Target gelenkt. Ein aus zwei Dipolmagneten bestehender Achromat filtert anschließend alle positiv geladenen Teilchen mit einem Impuls von $(75 \pm 1)\mathrm{GeV}/c$ heraus. Dabei wirkt der Achromat für diesen Impuls nicht richtungsändernd, Teilchen mit anderen Impulsen werden dagegen auf Absorbermaterial gelenkt. Der verbleibende Strahl wird einen Teilchenfluss von ca. 750 MHz besitzen und nur zu ca. 6% aus Kaonen bestehen. Die restlichen Teilchen des Strahls sind vor allem Pionen und Protonen.

Um die hohe Genauigkeit in der Bestimmung des Verzweigungsverhältnisses zu gewährleisten, wird ein Signal-zu-Untergrund-Verhältnis von mindestens 10 zu 1 angestrebt. Das bedeutet, dass nach der Analyse von den $10^{13}$ zerfallenen Kaonen nur rund 110

Ereignisse (engl. Events) übrig bleiben dürfen. Dies stellt eine enorme Anforderung an das Experiment, wozu mehrere Methoden verwendet werden müssen.

Die Signatur des gesuchten Zerfalls ist durch die Spur des $\pi^+$-Meson gegeben, da Neutrinos nicht detektiert werden können. Es gilt also dieses Pion nachzuweisen und gleichzeitig auszuschließen, dass es ein Zerfallsprodukt eines Untergrundereignisses (z.B. $K^+ \to \pi^+\pi^0$) ist. Tabelle 1.1 zeigt die häufigsten Untergrundzerfälle des geladenen Kaons, die zur Untergrundunterdrückung identifiziert werden müssen.

| Zerfall | Verzweigungsverhältnis | Zur Unterdrückung benötigter Detektor |
|---|---|---|
| $K^+ \to \mu^+\nu$ | 63% | $\mu$ Veto |
| $K^+ \to \pi^+\pi^0$ | 21% | $\gamma$ Veto [4] |
| $K^+ \to \pi^+\pi^+\pi^-$ | 6% | Veto geladener Teilchen |
| $K^+ \to \pi^0 e^+\nu$ | 5% | $\gamma$ Veto |
| $K^+ \to \pi^0\mu^+\nu$ | 3% | $\gamma, \mu$ Veto |
| $K^+ \to \pi^+\pi^0\pi^0$ | 2% | $\gamma$ Veto |

**Tabelle 1.1.:** Die häufigsten Zerfälle des $K^+$-Mesons, deren Verzweigungsverhältnisse und die zur Unterdrückung benötigten Veto-Detektoren.

Als Erstes dienen kinematische Bedingungen zur Untergrundunterdrückung. Dazu werden Impuls und Richtung der Kaonen und der Zerfallsprodukte gemessen und das Quadrat der invarianten Masse $m^2_{miss}$ bestimmt:

$$m^2_{\mathrm{miss}} = (P_{\mathrm{K}} - P_\pi)^2 \tag{1.8}$$

$$= m^2_{\mathrm{K}}\left(1 - \frac{|\vec{p_\pi}|}{|\vec{p_{\mathrm{K}}}|}\right) + m^2_\pi\left(1 - \frac{|\vec{p_{\mathrm{K}}}|}{|\vec{p_\pi}|}\right) - |\vec{p_{\mathrm{K}}}||\vec{p_\pi}|\theta^2_{\pi\mathrm{K}}. \tag{1.9}$$

Dabei ist $\theta_{\pi\mathrm{K}}$ der Winkel zwischen dem zerfallenen Kaon und dem entstandenen Pion. $P_{\mathrm{K}}$ und $P_\pi$ sind die Vierer-Impulse im Laborsystem des Kaons und der sekundären Spur im Detektor. $\vec{p_{\mathrm{K}}}$ und $\vec{p_\pi}$ sind die dazugehörigen dreidimensionalen Impulse und $m_{\mathrm{K}}$ sowie $m_\pi$ die entsprechenden Massen.

Trägt man nun $m^2_{miss}$ für die drei häufigsten Zerfälle und für den Signalkanal auf, so ergibt sich Abbildung 1.3. Der Zerfall $K^+ \to \mu^+\nu$ fällt in den negativen Bereich, da hier das Myon fälschlicherweise als Pion identifiziert wird. Da sich der Bereich vom Zerfall $K^+ \to \pi^+\nu\bar{\nu}$ mit zwei anderen Zerfällen überschneidet, können nur die beiden markierten Bereiche zur Messung in Betracht gezogen werden.

Durch dieses Verfahren wird eine Untergrundunterdrückung von $\mathcal{O}(10^{-5})$ erreicht [13]. Dabei wird in Gleichung 1.9 deutlich, dass neben der Messung von Impuls und Richtung

[4] Das neutrale Pion zerfällt quasi instantan in zwei Photonen

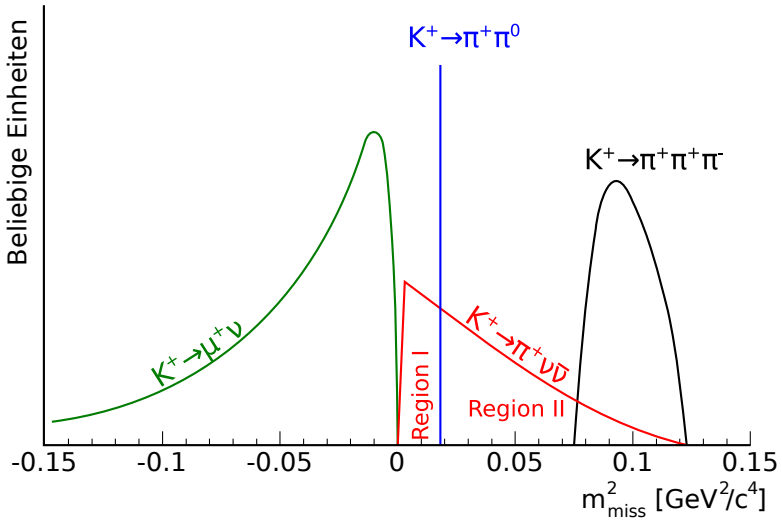


**Abbildung 1.3.:** Spektrum der fehlenden Masse der drei häufigsten Kaon-Zerfälle und des Signalkanals $K^+ \to \pi^+ \nu\bar{\nu}$. Basiert auf Figur 6 in [7].

der Teilchen, auch die Teilchenidentifikation eine wichtige Rolle spielt, um die richtigen Massen $m_K$ und $m_\pi$ anzunehmen. Für diesen Zweck und zur direkten Unterdrückung einiger Zerfälle wird eine Vielzahl von Detektoren zur Teilchenidentifikation eingesetzt. Auf der Seite der Zerfallsprodukte werden so Ereignisse direkte bei der Identifikation von Myonen oder Photonen verworfen. Gleichzeitig kann durch eine Kaon-Identifikation vor dem Zerfall sichergestellt werden, dass die nachgewiesenen Pionen tatsächlich von einem Kaon-Zerfall stammt. Für diesen Zweck wird eine sehr gute zeitliche Auflösung von $\mathcal{O}(100\ \text{ps})$ einzelner Detektoren benötigt.

### 1.2.1. Der Aufbau des NA62-Experiments

Abbildung 1.4 zeigt eine Übersicht des Experiments. Insgesamt kann man den Aufbau in drei Bereiche unterteilen. Der vordere Bereich ist ca. 100 m lang und dient, mit dem Target und zwei Achromaten, der Kaon-Produktion und Identifikation. Hier befinden sich die ersten drei Detektoren: der CEDAR zur Kaon-Identifikation, der GTK zur Impulsmessung und das CHANTI zur Unterdrückung von Ereignissen aus inelastischen Stößen im GTK.

Bei dem mittleren Bereich handelt es sich um die ca. 65 m lange Zerfallsregion innerhalb einer Vakuumröhre. Nur die Zerfallsprodukte jener Kaonen, die in diesem Bereich zerfallen, befinden sich im Akzeptanzbereich des Detektors. Innerhalb der Röhren befinden sich Szintillatoren des LAV-Detektors zur Photonen-Detektion (siehe Abschnitt

1.2.2.5), sowie die einzelnen Stationen des Magnet-Spektrometers, dem so genannten STRAW-Detektor (siehe Abschnitt 1.2.2.4).

Der hintere Bereich besteht aus verschiedenen Veto-Detektoren (siehe Abschnitt 1.2.2.6 bis 1.2.2.8). Der Strahl wird durch diesen ca. 50 m langen Abschnitt in einem Vakuumrohr geführt und schließlich durch einen aus Stahl und Beton bestehenden Beam-Dump absorbiert.

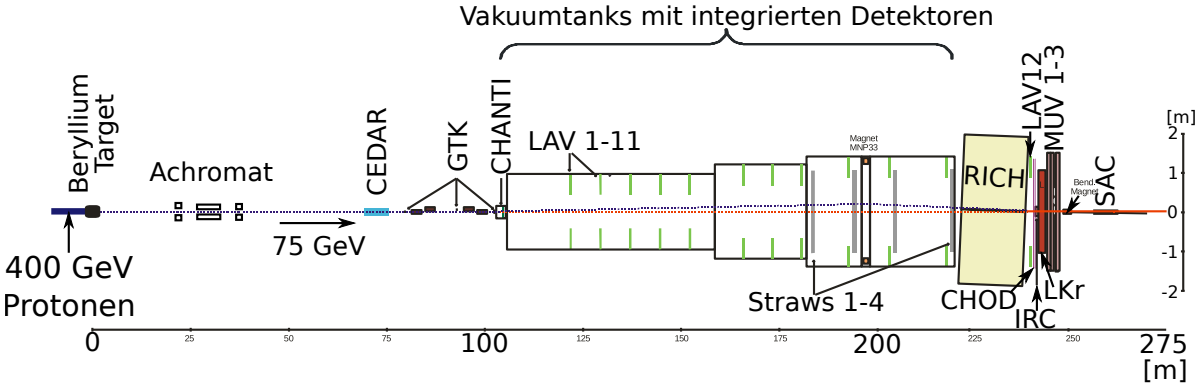


**Abbildung 1.4.:** Übersicht des NA62-Experiments [13]. Elemente zur Fokussierung des Strahls wurden nicht eingezeichnet.

## 1.2.2. Die Detektoren

Die Notwendigkeit verschiedener Veto-Detektoren und Kalorimeter wurde bereits motiviert. Die in Abbildung 1.4 gezeigten Detektoren werden in den folgenden Abschnitten näher beschrieben.

### 1.2.2.1. Der differentielle Tscherenkowzähler - CEDAR

Wie bereits erläutert, werden durch den Achromaten positiv geladene Teilchen mit einem Impuls von $(75 \pm 1)$ GeV/$c$ selektiert. In diesem Teilchenstrahl befinden sich jedoch nur rund 6% Kaonen. Um Untergrund durch die restlichen Teilchen des Strahls zu unterdrücken wird mit Hilfe des CEDARs eine explizite Kaon-Identifikation durchgeführt. Um Korrelationen mit anderen Detektoren mit einer hohen Genauigkeit bestimmen zu können, muss der CEDAR eine sehr hohe zeitliche Auflösung besitzen. Diese wird voraussichtlich unter 100 ps betragen.

In einem mit Wasserstoff gefüllten Gastank entsteht, abhängig von der Geschwindigkeit der Teilchen, ein Tscherenkow-Lichtkegel. Da der Impuls der Teilchen bereits durch den

Achromaten stark eingegrenzt und der Gasdruck auf $3,6$ bar festgesetzt ist, ist der Winkel des Kegels ausschließlich abhängig von der Masse des Teilchens. Spiegel am Ende des CEDAR reflektieren die Tscherenkowstrahlung so auf einen Sensor, dass nur im Falle von passierenden Kaonen ein Signal entsteht. Somit kann das Trigger-System des NA62-Experiments alle Ereignisse verwerfen, bei denen kein Signal im CEDAR gemessen wurde.

#### 1.2.2.2. Der Gigatracker - GTK

Sollte sich innerhalb eines kurzen Zeitfensters vor oder nach einem Kaon ein anderes Strahlteilchen befinden, so könnten dessen Zerfallsprodukte fälschlicher Weise dem Kaon zugeordnet werden. Um diese Fehlidentifikationen zu verhindern, erkennt der Gigatracker jedes Teilchen des Strahls, ohne dabei den Teilchentyp zu bestimmen (dies wird als Tagging bezeichnet). Sieht der GTK also bei einem positiven Signal des CEDARs zwei nahe beieinander liegende Teilchen (wobei eines davon, auf Grund das Signals im CEDAR, ein Kaon sein muss), so wird das Ereignis verworfen. Das Tagging muss entsprechend der Teilchenrate im Strahl mit einer Frequenz von 0,75 GHz stattfinden, wobei eine zeitliche Auflösung von weniger als 200 ps erreicht wird. Diese hohe Rate gab dem Detektor seinen Namen.

Gleichzeitig wird durch den GTK die Richtung und der Impuls der Strahlteilchen gemessen. Für diesen Zweck werden die Teilchen durch vier achromatische Magnete kurzzeitig abgelenkt und der Krümmungsradius bestimmt. Zwischen den Magneten befinden sich drei Stationen, welche aus Pixeldetektoren mit 18.000 Pixeln bei einer Fläche von jeweils 90 $\text{nm}^2$ pro Pixel bestehen. Diese werden sowohl zum Tagging, als auch zur Impuls- und Richtungsmessung verwendet.

#### 1.2.2.3. Der Charged-Anti-Detektor - CHANTI

Während die Strahl-Partikel den GTK passieren, kann es zu inelastischen Stößen mit dem Detektormaterial kommen. Dabei kann unter anderem ein geladenes Pion entstehen, welches dann im STRAW-Detektor die Signatur eines $K^+ \to \pi^+\nu\bar{\nu}$ Zerfalls hinterlässt. Der CHANTI-Detektor wird dazu eingesetzt Teilchen einer inelastischen Wechselwirkung zu detektieren und besteht aus sechs Stationen von Szintillatoren hinter dem GTK (siehe Abbildung 1.5). Teilchen, die als Produkt des inelastischen Stoßes entstehen, hinterlassen ein Signal in diesen Szintillatoren. Diese Signale können ausgelesen als Veto für ein Ereignis verwendet werden.

Simulationen zeigen, dass Fehlzuweisungen durch Teilchen aus den genannten inelastischen Stößen durch den CHANTI zu 99% unterdrückt werden.

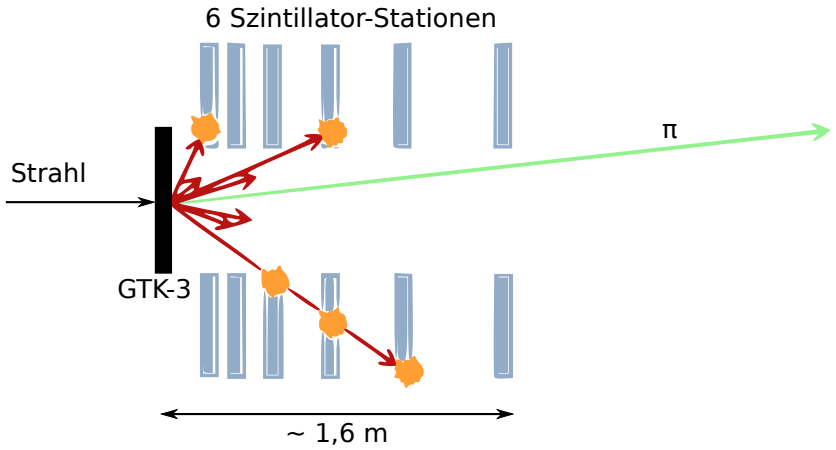


**Abbildung 1.5.:** Identifikation inelastischer Stöße an der letzten GTK-Station (GTK-3). Das entstandene Pion (grün) könnte ein Zerfall des Signalkanals imitieren. Die restlichen Partikel (rot) lösen jedoch ein Veto-Signal im CHANTI aus.

#### 1.2.2.4. Der STRAW-Tracker

Zur Messung von Impuls und Richtung geladener Zerfallsprodukte wird der STRAW-Tracker als Spektrometer eingesetzt. Dazu wird die Flugbahn der Teilchen vor und hinter einem Dipolmagneten bestimmt (Bahnverfolgung, engl. Tracking) und der Winkel zwischen beiden Bahnabschnitten gemessen. Bei gegebener Masse (Tagging) ergibt sich der Impuls aus diesem Winkel.

Das Funktionsprinzip des STRAW-Trackers entspricht dem einer Vieldraht-Proportionalkammer. Um die Wechselwirkung zwischen Teilchen und dem benötigten Gas möglichst gering zu halten, befinden sich die Drähte jeweils in einem dünnen Kupferrohr bei Normaldruck. Die Röhren selbst befinden sich im Vakuum mit einem Druck von $\mathcal{O}(10^{-10}$ bar). Diese insgesamt 1792 dünnen langen Rohre geben dem Detektor den Namen STRAW (dt.: Stroh) (siehe Abbildung 1.6).

Der Detektor unterteilt sich in vier Stationen mit je vier Lagen und 112 Drähten pro Lage. Die Lagen einer Station sind untereinander um einen Winkel von jeweils 45° versetzt, um eine zweidimensionale Auflösung zu erhalten. Zwischen der zweiten und der dritten Station befindet sich der Magnet mit einem Feld von 0,36 T.

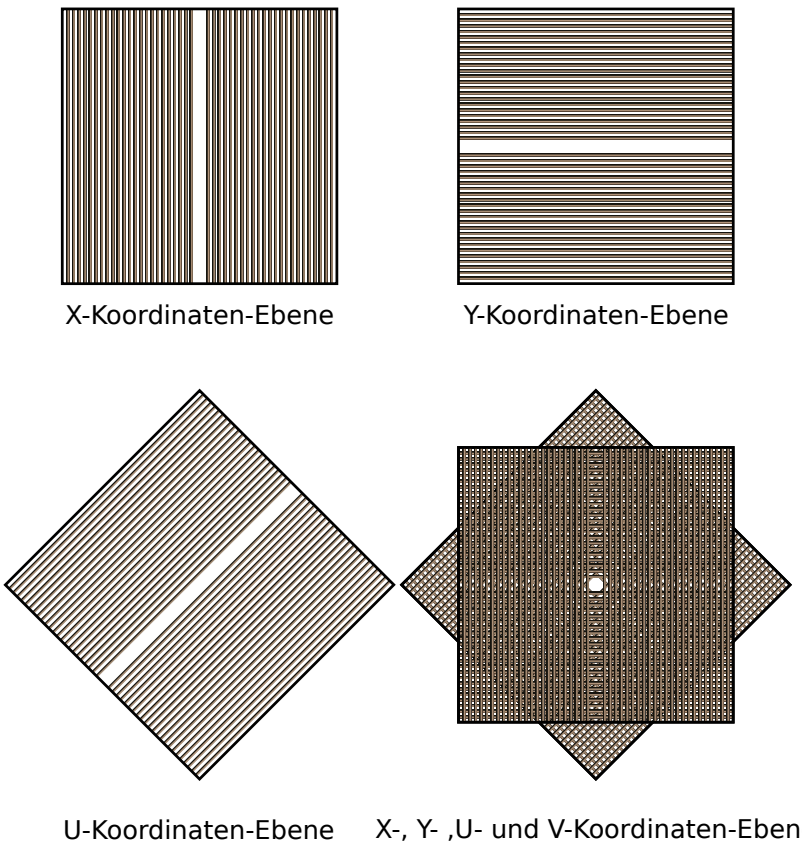


**Abbildung 1.6.:** Schematische Zeichnung der vier Lagen einer Station des STRAW-Trackers. Dabei wurden nur 46 der eigentlich 112 STRAWs einer Lage eingezeichnet.

#### 1.2.2.5. Das Large Angle Veto - LAV

In verschiedenen Zerfällen des Kaons entstehen neutrale Pionen, welche hauptsächlich in zwei Photonen zerfallen[5]. Diese Photonen müssen in einem großen Winkelbereich zuverlässig nachgewiesen werden um diese Zerfälle zu unterdrücken. Dazu kommen mehrere Detektoren für verschiedene Winkelbereiche zum Einsatz. Für den größten Bereich von ca. 8,5 bis 50 mrad zwischen Strahl und Photon wird das LAV mit 12 Stationen eingesetzt. Die ersten 11 Stationen befinden sich innerhalb der Vakuumröhren des Zerfallsbereiches. Die letzte Station ist hinter dem RICH-Detektor montiert. Jede Station besteht aus vier bis fünf Ringen aus Bleiglasblöcken. Das Bleiglas stammt vom OPAL-Experiment am LEP.

Die Ineffizienz des LAV darf nicht größer als $10^{-8}$ sein.

[5]Die mittlere Lebensdauer des $\pi^0$ beträgt $(8,4 \pm 0.5) \cdot 10^{-17}$s.

#### 1.2.2.6. Das LKr-Kalorimeter

Dieses elektromagnetische Kalorimeter hat sich bei dem Vorgänger-Experiment NA48 sehr gut bewährt und wird daher auch beim NA62-Experiment wieder eingesetzt. Es wird hauptsächlich zur Photon- und Myon-Unterdrückung eingesetzt, wird aber auch als Kalorimeter zur Untersuchung weiterer seltener Kaon-Zerfällen nützlich sein.

In dem zylindrischem Detektor befindet sich flüssiges Krypton als aktives Medium zusammen mit 13248 Cu-Be Elektroden (siehe Abbildung 1.7). Die Elektroden ermöglichen die transversale Messung von elektromagnetischen Schauern innerhalb des rund $6,7\ \mathrm{m}^3$ großen Kalorimeters. Zur Photonen-Unterdrückung deckt es den wichtigsten Bereich zwischen 1 und $8,5$ mrad ab. Die Ineffizienz für Photonenenergien oberhalb von 35 GeV darf einen Wert von $10^{-5}$ nicht überschreiten. Die Erfüllung dieser Anforderung wurde durch Messungen des NA48/2-Experiments bestätigt.

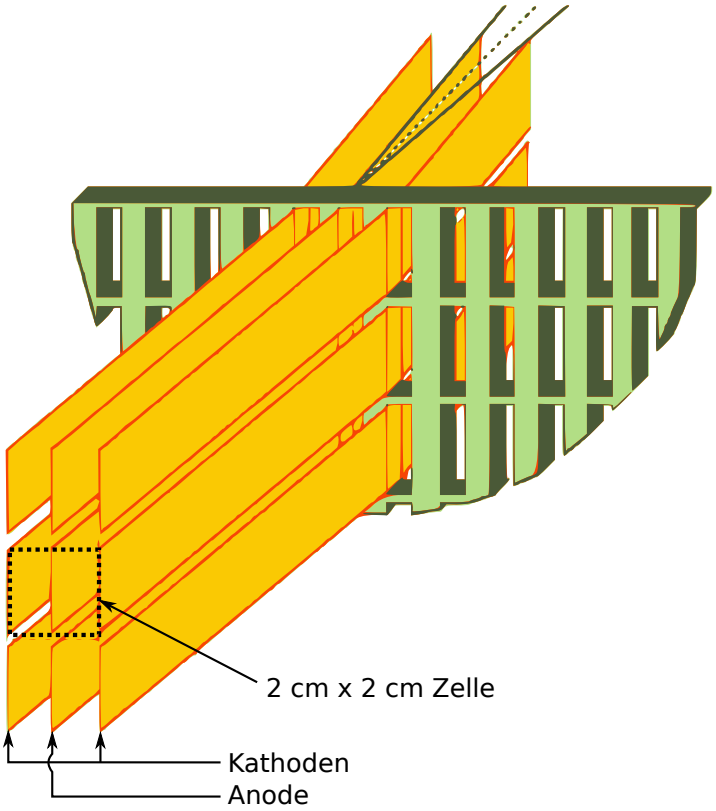


**Abbildung 1.7.:** Die Elektroden-Zellen des LKr-Kalorimeters [13].

#### 1.2.2.7. Das Intermediate Ring Calorimeter - IRC

Als Ergänzung zu dem LAV- und LKr-Detektor dient das IRC zur Detektion von Photonen mit kleineren Winkeln als 1 mrad zum Strahl. Zusätzlich werden so genannte Halo-Myonen durch das IRC unterdrückt. Dabei handelt es sich um hochenergetische Myonen, die durch den Achromaten kaum abgelenkt wurden und als minimal ionisierende Teilchen das Absorbermaterial durchdrangen. Diese Myonen befinden sich daher ebenfalls im Bereich kleiner Winkel um den Strahl herum.

Das IRC ist ringförmig um die Strahlröhre positioniert und besteht aus einem Blei- und Szintillator-Schaschlik-Kalorimeter mit insgesamt 70 Lagen. Es befindet sich direkt hinter dem LKr-Kalorimeter.

#### 1.2.2.8. Das Small-Angle-Calorimeter - SAC

Zur zusätzlichen Messung von Halo-Myonen und Photonen innerhalb der Strahlröhre wird dieses Kalorimeter eingesetzt. Es befindet sich in der Verlängerung des Strahlrohrs und besteht ebenfalls aus einem Blei- und Szintillator-Schaschlik. Um die Ereignisrate an diesem Detektor gering zu halten, wird der Hauptteil des Strahls, bestehend aus geladenen Teilchen mit einem Impuls von rund 75 GeV/c, aus dem Bereich des SAC gelenkt. Der für diesen Zweck verwendete Dipolmagnet beeinflusst die Flugbahn der hochenergetischen Halo Myonen kaum.

Die Ineffizienzen von IRC und SAC werden jeweils eine Ineffizienz von $10^{-5}$ nicht übersteigen.

#### 1.2.2.9. Der RICH-Detektor

Zum Nachweis von Pionen und Myonen im Bereich von 15 bis 35 GeV/$c$ wird ein Ring-Imaging-Cherenkov-Detektor (RICH) eingesetzt. Dieser muss eine hohe zeitliche Auflösung besitzen, um Korrelationen mit dem GTK erkennen zu können.

Der RICH-Detektor besteht aus einem 18 m langen, mit Neongas gefüllten Rohr. Darin entsteht Tscherenkowlicht, welches im hinteren Bereich auf piezoelektrisch verstellbare Spiegel trifft. Das Licht wird auf eine Matrix von 2000 Photomultiplier-Röhren am Anfang des Detektors reflektiert. Die Zeitauflösung ist besser als 100 ps.

Da die Rekonstruktion der Tscherenkow-Ringe innerhalb der 2000 Pixel sehr zeitaufwändig ist, werden hierfür **G**raphical **P**rocessing **U**nits (GPUs) zur Beschleunigung verwendet. Auf diese Weise können die Informationen des RICH bereits im L1-Trigger (siehe 1.4) verwendet werden.

#### 1.2.2.10. Das geladene Hodoskop - CHOD

Innerhalb des RICH-Detektors kann es, hauptsächlich durch Photonen vom Zerfall neutraler Pionen, zu photonuklearen Wechselwirkungen mit dem Neon-Gas kommen. Simulationen haben gezeigt, dass die dabei entstehenden geladenen Teilchen die Detektion der Photonen durch das LKr-Kalorimeter (siehe 1.2.2.6) verhindern können [13]. Daher verwendet man das „geladene“ Hodoskop (engl. **ch**arged **Hod**oscope, kurz **CHOD**), bestehend aus zwei Ebenen mit jeweils 64 Szintillatoren, zum Nachweis der geladenen Teilchen aus den Photonuklearen Wechselwirkungen. Dieser vom NA48-Experiment übernommene Detektor erreicht eine Zeitauflösung von etwa 200 ps und kann direkt im L0-Trigger (siehe Abschnitt 1.4.1) verwendet werden. Eine Verbesserung der zeitlichen und örtlichen Auflösung durch eine Neukonstruktion des Detektors ist geplant.

#### 1.2.2.11. Das Myon-Veto - MUV

Neben dem RICH, dem STRAW und dem LKr ist ein weiterer Detektor zur Myon-Unterdrückung notwendig. Dieser ist in drei getrennte Detektoren unterteilt, dem MUV1, MUV2 und dem MUV3, welche zusammen eine Ineffizienz von $10^{-5}$ nicht überschreiten dürfen.

Genauso wie die Halo-Myonen sind auch Myonen aus den Zerfällen des Kaons, etwa $K^+ \rightarrow \mu^+\nu$, minimal ionisierend. Das bedeutet, dass diese Teilchen sehr viel Masse durchdringen können, ohne dabei ihre komplette Energie zu verlieren. Diesen Effekt macht man sich beim MUV3 zu nutze. Dieser Detektor besteht aus 148 quadratischen Szintillatoren die sich direkt hinter einem 80 cm dicken Stahlquader befindet. Teilchen die in den Szintillatoren ein Signal hinterlassen, müssen also den Stahl durchdrungen haben und können somit keine Pionen sein. Signale im MUV3 werden also direkt als Veto verwendet und realisieren die größte Datenreduktion im L0-Trigger (siehe Abschnitt 1.4).

Das MUV3 erreicht eine Myonerkennungsineffizienz von $\mathcal{O}(10^{-4})$. Diese wird um einen weiteren Faktor zehn durch das MUV1 und MUV2 reduziert. Dazu werden Myonen detektiert, die durch katastrophale Bremsstrahlung ihre komplette Energie bereits vor dem MUV3 verlieren. Für diesen Zweck werden die Breiten der Schauer gemessen, die durch die einzelnen Teilchen in Materie entstehen. Myonen erzeugen einen schmalen elektromagnetischen Schauer. Die stark wechselwirkenden Pionen hinterlassen jedoch breitere hadronische Schauer.

Die Myon-Veto Detektoren MUV1 und MUV2 bestehen aus einem Eisen-Szintillator-Sandwich-Kalorimeter (siehe Abbildung 1.8) mit jeweils 24 Lagen Szintillatoren, welche untereinander um je 90° verdreht sind um eine zweidimensionale Auflösung zu erhalten.

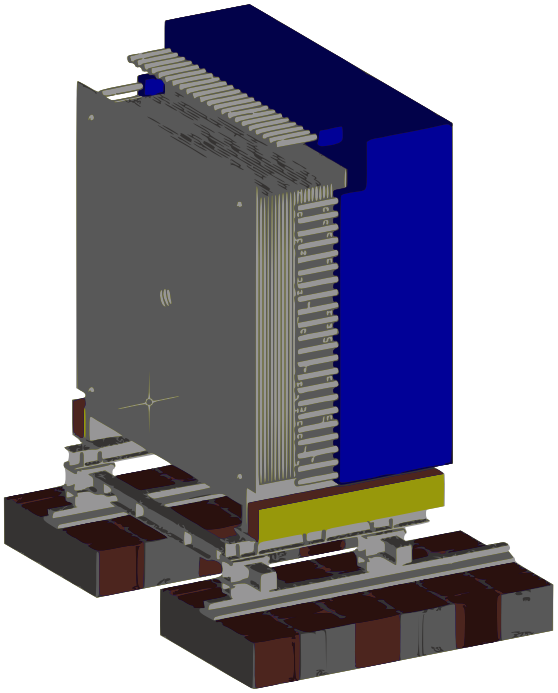

**Abbildung 1.8.:** 3D-Modell des MUV1 (grau) und MUV2 (blau) [13].

### 1.2.3. Der Duty-Cycle

Das SPS erzeugt keinen kontinuierlichen Strahl, sondern so genannte Bursts von einigen Sekunden[6]. Zwischen zwei Bursts muss das SPS erneut mit Protonen gefüllt werden. Da dieser Teilchenbeschleuniger gleichzeitig als Protonenquelle für weitere Experimente wie dem CNGS[7]-Experiment dient, wird nicht jeder Burst zum NA62-Experiment geleitet. Daher entsteht zwischen zwei Bursts eine Pause deren Länge, abhängig von den Aktivitäten anderer Experimente, variiert.

Bezeichnet man die Dauer eines Bursts durch $T_{\text{Burst}}$ und die Dauer der Pause zwischen zwei Bursts durch $T_{\text{Pause}}$, so wird der so genannte Duty-Cycle $\eta$ definiert durch $\eta = \frac{T_{\text{Burst}}}{T_{\text{Burst}}+T_{\text{Pause}}} = \frac{T_{\text{Burst}}}{T_{\text{Zyklus}}}$. Dabei ist $T_{\text{Zyklus}}$ die Zeit eines Strahlzyklus, also der Summe aus Burst-Zeit und Dauer der Pause zwischen zwei Bursts. Für den größten Teil der Datennahme ist ein Duty-Cycle von $\eta = 0,25$ mit einer Burst-Zeit von rund fünf Sekunden realistisch [13].

[6] Innerhalb eines Bursts ist der Strahl konstant. Es besteht also keine weitere Substruktur des Strahles in Form von Bunches, wie es im LHC der Fall ist.

[7] CNGS steht für CERN Neutrinos to Gran Sasso. Dieses Experiment produziert also den Neutrino-Strahl zum OPERA-Experiment in Italien.

## 1.3. Datenauslese

Mit Datenauslese (engl. Data Acquisition, kurz DAQ) ist die benötigte Elektronik zur Auslese des aktuellen Zustandes der Detektoren gemeint. Dabei werden in der Regel die analogen Signale (etwa das Ausgangssignal eines Photomultipliers an einem Szintillator) digitalisiert. Die digitalen Rohdaten enthalten alle Informationen über die Ereignisse und werden daher zur eigentlichen Physik-Analyse benötigt.

### 1.3.1. Die Ausleseelektronik

Aus praktischen Gründen wird bei den meisten Detektoren das so genannte TEL62-Board zur Auslese der Rohdaten und zur Prozessierung des L0-Triggers (siehe Abschnitt 1.4) verwendet. Dabei handelt es sich um eine angepasste Version des TELL1-Boards, wie es am LHCb-Experiment verwendet wird [13]. Zur Digitalisierung können **T**ime to **D**igital **C**onverter (TDC) Tochter-Karten verwendet werden. Die Datenverarbeitung wird mit vier vorhandenen **F**ield **P**rogrammable **G**ate **A**rrays (FPGAs) durchgeführt und zum Versenden der Daten stehen vier Gigabit-Ethernet-Netzwerkverbindungen zur Verfügung (mehr dazu in Abschnitt 2.2.5). Insgesamt sollen voraussichtlich bis zu 120 TEL62-Boards installiert werden.

Für das LKr wird eine spezielle Ausleseelektronik entwickelt, das so genannte CREAM-Board. Dieses Board basiert ebenfalls auf FPGAs mit Gigabit-Ethernet-Netzwerkanschlüssen zur Kommunikation. Zukünftig sollen bis zu 400 CREAM-Boards eingesetzt werden. Für die ersten Testläufe des Experiments im Sommer 2012 werden die CREAM-Boards durch PCs simuliert.

Die Datenverarbeitung innerhalb der Ausleseelektronik findet in den FPGAs statt und muss somit durch Firmware implementiert werden. Somit muss auch die Kommunikation zwischen FPGAs und Netzwerkkarte in der Firmware realisiert werden. Komplexe Netzwerkprotokolle wie etwa **T**ransmission **C**ontrol **P**rotocol (TCP) (siehe 2.3.6) würden dabei bereits einen erheblichen Platz innerhalb der FPGAs einnehmen, weshalb darauf verzichtet wird (siehe Abschnitt 4.2).

## 1.4. Trigger

Zerfälle von Kaonen und Pionen des Hadronenstrahls werden im Detektor eine Signalrate von ca. 10 MHz induzieren. Alleine das LKr-Kalorimeter mit seinen 13248 Kanälen produziert dabei etwa $2,2$ TB/s an Rohdaten. Die maximale Datenrate, mit der die

Rohdaten des Experiments gespeichert werden können, beträgt[8] 100 MB/s. Somit kann nur ein kleiner Teil der Ereignisse archiviert werden. Mit einem Trigger-System bestehend aus drei Ebenen (engl. Level; kurz L0, L1 und L2) werden interessante Ereignisse herausgefiltert, um so die Datenrate zu reduzieren. Der erste Trigger (L0) ist direkt in der Ausleseelektronik implementiert. Die höheren Stufen (L1 und L2) werden auf PCs ausgeführt. Diese PCs bilden die NA62-Online-PC-Farm.

Das Grundprinzip des Trigger-Systems ist wie folgt zu beschreiben:

1. Die TEL62-Boards der am L0-Trigger teilnehmenden Detektoren (siehe Tabelle 1.2) senden im Falle eines Signals ein Trigger-Paket[9] an ein Elektronik-Board, genannt **L**evel-**0**-**T**rigger-**P**rozessor (L0TP), mit einer maximalen Rate von 10 MHz.

2. Empfängt der L0TP genügend positive Trigger-Signale, jedoch keine negativen von den Veto-Detektoren, so sendet er 1 ms nach dem Ereignis ein Trigger-Signal[10] (positive L0-Trigger-Entscheidung) an alle Auslese-Boards. Der primäre Trigger-Modus wird dabei voraussichtlich eine Spur im CHOD, einen Cluster im LKr und kein Signal vom MUV3 fordern. Als maximale Rate positiver L0-Trigger-Entscheidungen ist 1 MHz vorgesehen.

3. Sollte die Auslese-Elektronik nach einer Millisekunde kein Trigger-Signal erhalten, so verwirft sie das Ereignis. Durch den L0-Trigger wird auf diese Weise die Datenrate um einen Faktor von 10 gesenkt.

4. Bei einem Trigger-Signal schickt die gesamte Auslese-Elektronik die Daten ihres Detektors an die L1-Trigger-Stufe (LKr ausgenommen). Hierbei schickt jeder Detektor ein Datenpaket, auch wenn etwa auf Grund von Nullenunterdrückung[11] keine Daten vorhanden sein sollten.

5. Die L1-Trigger-Stufe überprüft die Integrität der empfangenen Daten und führt eine erste Spurrekonstruktion, teilweise mit Hilfe von GPUs, durch. Dabei sollte ursprünglich jeder Detektor für sich betrachtet werden (siehe Abschnitt 3.2.2).

6. L1-Trigger-Entscheidung: Abhängig von den Ergebnissen der Datenanalyse aller Detektoren wird das Ereignis akzeptiert oder verworfen. Dazu sollte in einem ursprünglichen Konzept der so genannte **L**evel-**1**-**T**rigger-**P**rozessor (L1TP) als getrennter PC zur Verteilung des L1-Trigger-Ergebnisses über alle Teilnehmer der L1-Stufe verwendet werden (siehe Abschnitt 3.2.2).

[8] Das entspricht der maximalen Schreibgeschwindigkeit eines Tape-Schreibkopfes. Aus Kostengründen wird nur ein einziger Schreibkopf verwendet.

[9] Dabei handelt es sich um ein Ethernet-Paket (siehe Abschnitt 2.2.5).

[10] Dies ist durch das Aussetzen eines Taktes des globalen Zeitsignals kodiert.

[11] Damit ist gemeint, dass alle Kanäle mit einem Signal unterhalb einer festen Schwelle unterdrückt werden.

7. Im positiven Fall sendet die L1-Trigger-Stufe die Daten des akzeptierten Ereignisses an die L2-Stufe. Dabei werden zusätzlich die Daten des LKr zu dem Ereignis angefordert und an die L2-Trigger-Stufe gesendet. Zwischen einem positiven L0-Trigger-Signal und der L1-Trigger-Entscheidung zu einem Ereignis darf maximal eine Sekunde vergehen.

8. Die L2-Trigger-Stufe sammelt die Daten aller Detektoren (Eventbuilding). Nun stehen die Daten des gesamten Detektors für weitere Rekonstruktionen zur Verfügung. Die L2-Trigger-Stufe entscheidet endgültig, ob das Ereignis gelöscht oder gespeichert werden soll, und sendet im positiven Fall die Daten an einen persistenten Speicher. Dabei ist die Datenrate auf 100 MBps limitiert, da dem Experiment nur ein Kassettenschreibkopf im Datenzentrum des CERN zur Verfügung steht, welcher eine serielle Schreibrate von ca. 100 Megabyte pro Sekunde erreicht[12].

### 1.4.1. Beiträger der Detektoren zu den Trigger-Stufen

Wie oben angedeutet, nehmen nicht alle Detektoren in allen Trigger-Ebenen an der Entscheidung teil. Innerhalb unterschiedlicher Trigger-Modi können verschiedene Detektor-Daten für die einzelnen Trigger-Ebenen verwendet werden. Die Teilnahme an den Trigger-Entscheidungen der L0-Trigger-Stufe wird voraussichtlich, wie in Tabelle 1.2 beschrieben, stattfinden.

So werden z.B. die CEDAR-Daten bei einem positiven L0-Trigger ausgelesen und an die L1-Trigger-Stufe gesendet. Dort wird nur die fehlerfreie Datenübertragung (Datenintegrität) überprüft. Diese Daten nehmen also nicht direkt an der L1-Trigger-Entscheidung teil. Erst bei einem positiven L1-Trigger werden die CEDAR-Daten an L2 weitergegeben und dort analysiert.

Das LKr-Kalorimeter bildet Energiesummen, um an der L0-Trigger-Entscheidung teilzunehmen. Da dieser Detektor rund 98% der Daten des NA62-Experiments produziert, werden diese nicht an die L1-Trigger-Stufe gesendet. Erst bei einer positiven L1-Trigger-Entscheidung senden die CREAM-Boards ihre Daten an die L2-Stufe zur Verarbeitung.

[12] Es ist geplant, zusätzlich Daten auf Festplatten direkt am Experiment zu speichern. Dieser zusätzliche Speicher soll jedoch hauptsächlich als Puffer dienen und wird daher vernachlässigt.

| Name | L0 | Ereignis-Größe [B] |
|---|---|---|
| CEDAR | | 216 |
| GTK | | 2250 |
| CHANTI | | 192 |
| LAV | ✓ | 160 |
| STRAW | | 768 |
| RICH | | 160 |
| CHOD | ✓ | $\ll 1\ k$ |
| MUV | ✓ | 768 |
| IRC & SAC | | 576 |
| LKr | ✓ | 222 k |
| **Summe** | | **≈227 kB** |

**Tabelle 1.2.:** Die Detektoren des NA62-Experiments, deren vorgesehene Teilnahme an den L0-Trigger-Entscheidungen und die geschätzten Ereignis-Größen [13]. Neben den 222 kB des LKr-Kalorimeters ergeben die Restlichen Detektoren eine Datenmenge von $S_{\mathrm{L0}} \approx 5$ kB.

In Tabelle 1.2 ist eine Schätzung der erwarteten Ereignis-Größen für die verschiedenen Detektoren zu finden. Diese Daten teilen sich über mehrere Ausleseelektronik-Boards auf. Die Summe dieser Größen muss bei jedem L0-Trigger ausgelesen werden und entspricht etwa der Größe eines gesamten Ereignisses, wie es bei positiven Trigger-Entscheidungen schlussendlich für spätere Offline-Analysen abgespeichert wird.

# Kapitel 2.

# Grundlagen

Bevor auf die Entwicklung der PC-Farm eingegangen wird, sollen zunächst einige häufig verwendete Begriffe der **I**nformations-**T**echnik (IT) erläutert werden, welche zum Verständnis der Diplomarbeit wesentlich sind.

## 2.1. Dezimalpräfixe und Datenraten

Da am NA62-Experiment nur sogenannte IBM-PCs verwendet werden, entsprechen einem Byte genau acht Bit (1 B = 8 b), wobei ein Bit als Binärziffer (i.d.R. kann diese die Werte 0 oder 1 annehmen) die kleinstmögliche Informationseinheit darstellt.

Als Präfixe werden in der Informatik sowohl die auf Zehnerpotenzen basierenden Dezimalpräfixe[1], als auch die auf Zweierpotenzen basierenden Binärpräfixe[2] verwendet. Innerhalb dieser Arbeit werden fast ausschließlich Dezimalpräfixe verwendet:

$$\begin{aligned} 1\ \text{kB} &= 1000\ \text{B} = 8000\ \text{b}, \\ 1\ \text{MB} &= 1000\ \text{kB} = 1 \cdot 10^6\ \text{B}, \\ 1\ \text{GB} &= 1000\ \text{MB} = 1 \cdot 10^9\ \text{B}. \end{aligned}$$

Zum Vergleich die Definition der Binärpräfixe nach [14]:

$$\begin{aligned} 1\ \text{kiB} &= 1024\ \text{B} = 8192\ \text{b}, \\ 1\ \text{MiB} &= 1024\ \text{kiB} = 2^{20}\ \text{B}, \\ 1\ \text{GiB} &= 1024\ \text{MiB} = 2^{30}\ \text{B}. \end{aligned}$$

Die Verwendung der Dezimalpräfixe ist nützlich, da bei Kommunikationsnetzen, im Vergleich zu anderen Bereichen der IT, nur selten Binärpräfixe verwendet werden. So ist

[1] Im *Internationalen Einheitensystem* definiert. Daher auch als SI-Präfix bezeichnet.
[2] Definiert durch die *International Electrotechnical Commission* (IEC) im Jahre 1998.

mit dem Gigabit-Ethernet nach dem Standard IEEE 802 eine Datenrate von genau 1 Gb (also $10^9$ Bit) pro Sekunde erreichbar.

Bei der Angabe von Datenraten werden sowohl Bits pro Sekunde, als auch Bytes pro Sekunde angegeben. Eine „Zehn-Gigabit-Netzwerkkarte“ erlaubt es 10 Gb/s zu übertragen. Diese Übertragungsrate wird häufig mit 10 Gbps abgekürzt:

$$\begin{aligned} 10\ \text{Gbps} &:= 10\ \text{Gb/s}, \\ 10\ \text{GBps} &:= 10\ \text{GB/s}, \\ 10\ \text{Gbps} &= 1,25\ \text{GBps}. \end{aligned}$$

## 2.2. Kommunikationsnetze

Ein Kommunikationsnetz besteht aus einer Menge räumlich getrennter Stationen (z.B. PCs oder TEL62-Boards), die mittels eines Übertragungsmediums (z.B. Kupferkabel oder Glasfaser) miteinander kommunizieren. Man klassifiziert Kommunikationsnetze abhängig von deren Ausdehnung. Bei einigen zehn bis hundert Metern spricht man von dem häufig genannten **L**ocal **A**rea **N**etwork (LAN).

Wichtige Terminologien im Bereich der Kommunikationsnetze werden in diesem Kapitel eingeführt und erläutert. Eine vertiefende Auseinandersetzung mit dem Thema findet sich in [15].

### 2.2.1. Standards für Netzwerktechnologien

Das **I**nstitute of **E**lectrical and **E**lectronics **E**ngineers (IEEE) bildet ein wichtiges Gremium zur Standardisierung von Hard- und Software im Bereich der Kommunikationsnetze. Von dem IEEE definierte Standards werden in der Form „IEEE X.Y“ gekennzeichnet, wobei Unterlagen zu diesen in [16] zur Verfügung stehen.

### 2.2.2. Netzwerk-Topologien

Eine Netzwerk-Topologie beschreibt die physikalische Anordnung der Stationen innerhalb eines Kommunikationsnetzes.

Wegen der Analogie zu Graphen werden im folgenden Text Teilnehmer bzw. Stationen eines Netzwerkes auch als Knoten bezeichnet und die Netzwerkverbindungen (das Medium zwischen zwei Knoten) auch als Kanten.

### 2.2.3. LAN-Topologien

In einem LAN muss prinzipiell jeder Rechner mit jedem anderen kommunizieren können[3]. Um dies zu realisieren, sind eine Vielzahl an Topologien denkbar (siehe Abbildung 2.1):

- Punkt-zu-Punkt: Direkte Verbindung zweier Knoten (Basis für weitere Topologien).
- Bus: Alle Teilnehmer teilen sich ein Übertragungsmedium.
- Stern: Eine Zentrale Einheit regelt die Kommunikation.
- Baum: Hierarchischer Zusammenschluss mehrerer Busse oder Sterne.
- Ring: Nachrichten werden von Teilnehmern im Kreis weitergereicht.
- Vermaschtes Netz: Hierarchieloser Baum.

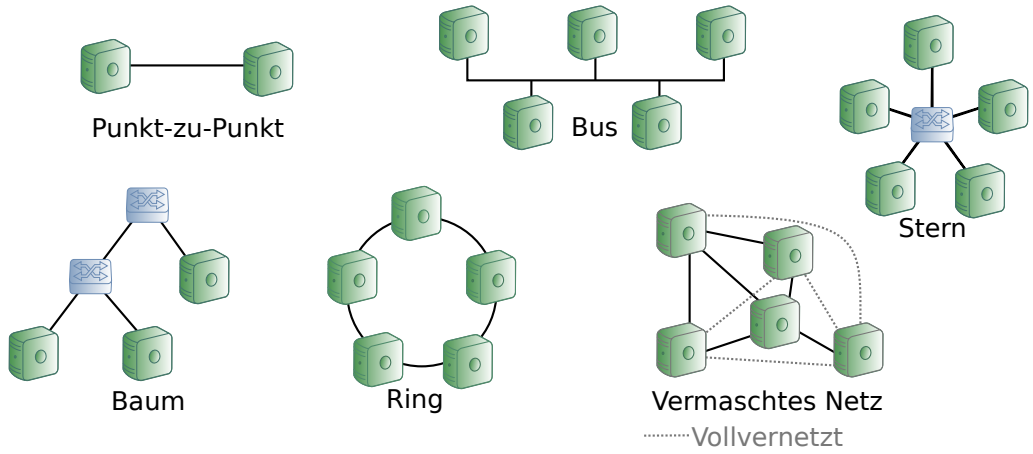


**Abbildung 2.1.:** Verschiedene LAN-Topologien.

Prinzipiell haben alle diese Netzwerkformen ihre Existenzberechtigung. Man muss daher abhängig von der Problemstellung entscheiden, welche Topologie die passendste ist. Zu beachtende Kriterien sind dabei unter anderem:

- Sicherheit:
  - Angriffssicherheit (engl. security).
  - Betriebssicherheit (engl. safety).
- Effizienz (beeinträchtigen sich einzelne Kommunikationspfade gegenseitig?).

[3]Dabei kann, die Kommunikation zwischen zwei Teilnehmern auch indirekt über einen dritten Knoten stattfinden.

- Komplexität (Fehlersuche soll einfach bleiben).
- Preis.

Als Beispiel ist die Angriffssicherheit ein häufiges Argument gegen eine Bus-Topologie, da jeder PC die Kommunikationen der anderen Teilnehmer mithören kann. Im Falle der NA62-PC-Farm ist dies jedoch unkritisch, da die Farm-Rechner nur über eine abgesicherte Leitung (Firewall) mit der Außenwelt (Internet) kommunizieren können. Dafür sind Bus-Topologien nicht für die gleichzeitige Kommunikation verschiedener Kanten mit hoher Datenrate geeignet, da sich einzelne Datenpakete überlagern können und in dem Übertragungsmedium nicht mehr voneinander getrennt werden können. Daher setzt man meistens auf Punkt-zu-Punkt-basierte Netze.

Das am stärksten verbreitete Kommunikationsnetz is das so genannten Ethernet (siehe Abschnitt 2.2.5) mit einer Baum-Topologie. Aus Kostengründen wurde entschieden, dass für die Kommunikation innerhalb der PC-Farm des NA62 Experiments das weit verbreitete Gigabit (1GbE)- und Zehn-Gigabit (10GbE)-Ethernet als Netzwerk verwendet werden soll. Dabei können Datenraten von 1 Gbps und 10 Gbps in beide Richtungen einer Kante erreicht werden. Die genaue Form des Netzwerkes bleibt dabei jedoch noch offen.

### 2.2.4. Das OSI-Referenzmodell

Das OSI-Referenzmodell (engl. Open Systems Interconnection Reference Model) wird zur Bewältigung der Komplexität einer Kommunikation innerhalb eines Netzwerkes verwendet. Dabei werden folgende sieben Schichten definiert [15]:

7. Anwendung (Benutzerdaten)
6. Darstellung
5. Sitzung
4. Transport
3. Vermittlung
2. Sicherung
1. Bit

Jede Schicht definiert ihr eigenes Kommunikationsprotokoll und bietet der darüber liegenden Schicht einen Dienst an. Die Protokolle sehen so genannte **P**rotocol **D**ata **U**nits (PDUs) vor, welche aus einem Kopfdatenbereich und einem Nutzdatenteil bestehen. Dabei beinhaltet der Nutzdatenteil der PDU der n-ten Schicht die PDU der (n+1)-ten Schicht, wie in Abb. 2.2 zu sehen ist.

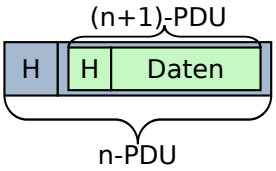


**Abbildung 2.2.:** Rekursive Verschachtelung von PDUs der einzelnen Schichten des OSI-Referenzmodells. Hierbei steht H für Header, dem englischen Wort für Kopfzeile, also dem Kopfdatenteil.

Ein Datenpaket der n-ten Schicht durchläuft also alle Schichten von oben (Schicht n) nach unten (Schicht 1) und wird dabei immer weiter verschachtelt. Wird es schließlich durch die unterste Schicht über das Übertragungsmedium verschickt, so besteht es aus n ineinander verschachtelten PDUs. Auf der Empfängerseite durchläuft das Paket in entgegengesetzter Richtung die Schichten (also von der ersten bis zur n-ten Schicht) und wird wie eine russische Matrjoschka stückweise ausgepackt.

Diese Verschachtelung dient zur Kommunikation der einzelnen Schichten auf beiden Seiten einer Kante untereinander. Dabei können die jeweiligen Kopfdaten beliebige Informationen enthalten und somit auch leer sein.

Tatsächlich implementieren heutige, weit verbreitete Protokolle wie das **I**nternet-**P**rotokoll (IP) einige Schichten im OSI-Referenzmodell nicht. Daher wurden alternative Modelle entwickelt, wie das TCP/IP-Modell oder das häufig von A. Tannenbaum verwendeten Fünf-Schichten-Modell. Trotzdem ist es üblich, sich an den Schichten des OSI-Modells zu orientieren. Daher werden diese nun einzeln eingeführt.

#### 2.2.4.1. Schicht 1 – Bitübertragungsschicht

Die unterste Schicht definiert das übertragungstechnische Verfahren zur bitweisen Kommunikation zwischen zwei Knoten. Somit wird zunächst das Übertragungsmedium (z.B. Kupferkabel oder Lichtwellenleiter) inklusive der dazugehörigen Steckverbindungen definiert. Darüber hinaus wird klargestellt, wie einzelne Bits in dem Medium kodiert übertragen werden. Bei dieser so genannten Leitungskodierung muss häufig die Gleichstromfreiheit[4] und eine möglichst effiziente Taktrückgewinnung[5] gewährleistet werden.

[4]Zur galvanischen Trennung werden die Signale in den Netzwerkkarten meist über Kapazitäten angekoppelt. Diese dürfen sich jedoch nicht aufladen, damit das zeitliche Mittel der Spannung des Übertragungssignals verschwindet.

[5]Damit der Empfänger die einzelnen Bits voneinander trennen kann, darf die Abtastfrequenz nicht zu sehr von der Sendefrequenz abweichen. Die Synchronisation findet über den Bitstrom selbst statt.

Häufig werden in der Bitübertragungsschicht bereits Prüfsummen eingesetzt um die korrekte Übertragung der Daten zu überprüfen.

Zur Realisierung der verschiedenen Netzwerk-Topologien können in dieser Ebene Hubs, Repeater und Bridges als so genannte Kopplungselemente eingesetzt werden. Diese bereiten ein an einer Kante empfangenes Signal auf und leiten es an alle übrigen Kanten weiter. Bei großen Netzwerken kommt es hier jedoch häufig zu Kollisionen[6] der einzelnen Pakete, wodurch das Netzwerk nur sehr ineffizient genutzt werden kann (mehr dazu in Abschnitt 2.2.5.1).

#### 2.2.4.2. Schicht 2 – Sicherungsschicht

Diese Schicht wird nach IEEE 802.2 in folgende zwei Teilschichten unterteilt:

**Medium Access Control (MAC):** Dieses Protokoll regelt den konkurrierenden Zugriff mehrerer Nutzer auf ein gemeinsames Medium (siehe z.B. Abschnitt 2.2.5.1).

**Logical Link Control (LLC):** Dieses Protokoll definiert das Format zur Adressierung entfernter Teilnehmer. Gleichzeitig kann eine Fehlerkontrolle durch Prüfsummen realisiert werden. Zudem wird nach IEEE 802.3x eine Flusskontrolle für dieses Protokoll vorgesehen, worauf in Abschnitt 2.2.5.4 näher eingegangen wird.

Mit dieser Schicht können so genannte Switches als Kopplungselement verschiedener Netzwerksegmente eingesetzt werden. Diese lesen die Hardware-Adresse (oft auch MAC-Adresse genannt) innerhalb der Kopfdaten der Sicherungsschicht-PDU aus. In einer Tabelle innerhalb des Switches wird gespeichert, über welche Kante der zur Adresse gehörige Knoten erreicht werden kann. Somit wird das Paket nur über diese Kante weitergeleitet und die Anzahl an Kollisionen verringert. Im Vollduplex-Modus können Kollisionen sogar komplett verhindert werden (siehe Abschnitt 2.2.5.1).

#### 2.2.4.3. Schicht 3 – Vermittlungsschicht

Betrachtet man die heutige Größe des Internets, so ist klar, dass die ersten beiden Schichten nicht ausreichen um die Kommunikation zwischen einer so großen Zahl an Knoten zu realisieren. So müsste etwa jeder Switch alle MAC-Adressen des kompletten Internets in einer Tabelle gespeichert haben. Typischerweise können heutige Switches jedoch nur ca. zehntausend Adressen speichern.

[6] Damit sind Überlagerungen mehrerer Pakete innerhalb des Übertragungsmediums gemeint, welche zur fehlerhaften Datenübertragung führten und somit durch die Sicherungsschicht detektiert werden müssen.

Um den Datenverkehr in großen Netzen effizient zu nutzen, ist das Wegesuchen (Routing) die Hauptaufgabe der dritten Schicht. Dies wird anhand des verbreitetsten Protokolls IP in Abschnitt 2.3.1 vertieft.

Gleichzeitig ermöglichen einige Protokolle dieser Schicht, dem Empfänger die Korrektheit des Paketes durch eine Prüfsumme zu bestätigen. Diese Prüfsumme bildet sich meist aus den Kopfdaten der PDU, und wird über diese übertragen. Sollte die Prüfsumme nicht rekonstruierbar sein (d.h. ein Teil der Kopfdaten inklusive Prüfsumme wurde falsch übertragen), so wird das Paket gelöscht[7]. Der Sender wird darüber nicht informiert.

Das Routing wird häufig innerhalb von Level-3-Switches oder Router als Kopplungselemente realisiert. Darauf wird in Abschnitt 2.3.1 näher eingegangen.

#### 2.2.4.4. Schicht 4 – Transportschicht

Die Hauptaufgaben dieser Schicht sind:

- Realisierung von verbindungsorientiertem oder verbindungslosem Datentransport[8].
- Segmentierung des Datenstroms.
- Flusssteuerung und Stauvermeidung (siehe Abschnitt 2.3.6.1 und 2.3.6.2).
- Qualitätsgarantie (Fehlerbehandlung).

Häufig verwendete Protokolle hierbei sind TCP/IP (siehe Abschnitt 2.3.6) und UDP/IP (siehe Abschnitt 2.3.5). Dazu werden unter anderem Prüfsummen über das komplette Paket, also alle Kopfdaten und Nutzdaten von Schicht vier bis Schicht sieben, verwendet.

Für diese und folgende Schichten existieren keine speziellen Kopplungselemente. Firewalls, welche häufig in Form von Software innerhalb eines Routers realisiert sind, lesen teilweise die Kopfdaten der Schicht Vier und Fünf aus, um unbefugte Datenströme zu unterbinden.

[7] Manche Protokolle versuchen aus der Prüfsumme die fehlerhaften Bits zu korrigieren. Innerhalb des Internet-Standards wird dies jedoch nicht getan.

[8] Eine Verbindung kann hier wie ein virtuelles Telefonkabel betrachtet werden: Daten, die über die Verbindung geschickt werden, kommen garantiert und in geordneter Reihenfolge an der Empfängerseite an.

#### 2.2.4.5. Schicht 5 – Sitzungsschicht

Diese Schicht ist für die Realisierung der Interprozesskommunikation (engl. **I**nter-**P**rocess **C**ommunication) (IPC) zuständig. Damit ist gemeint, dass z.B. einem Prozess auf einem PC ermöglicht wird, nicht nur Daten an einen zweiten PC zu senden, sondern zusätzlich einen konkreten Prozess auf dem zweiten PC zu adressieren. Zudem ermöglicht diese Schicht eine verbindungsorientierte Kommunikation zwischen den beiden Prozessen. Dabei ist zu betonen, dass die darunter liegenden Schichten eins bis vier nicht verbindungsorientiert realisiert sein müssen.

Bei der Internetprotokollfamilie (siehe Abschnitt 2.3) wird diese Schicht zusammen mit der vierten Schicht unter anderem durch TCP und UDP implementiert. Dabei ist nur TCP verbindungsorientiert.

#### 2.2.4.6. Schicht 6 – Darstellungsschicht

Analog zur Leitungskodierung aus Schicht 1 definiert die Darstellungsschicht eine Syntax mit der abstraktere Daten korrekt übertragen werden können. So unterscheidet man z.B. zwischen Big-Endian-Systemen, bei denen das Byte mit den höchstwertigen Bits eines Datenwortes an erster Stelle gespeichert wird, und Little-Endian-Systemen[9], bei denen das Byte mit den höchstwertigen Bits eines Datenwortes an letzter Stelle gespeichert wird. Würde man nun zwischen zwei verschiedenen solcher Systeme über die Sitzungsschicht etwa die vier-Byte Zahl „4C 1E FC 31“ senden, so würde sie auf der anderen Seite fälschlicherweise als „31 FC 1E 4C“ interpretiert werden. Die Darstellungsschicht soll diese Fehlinterpretationen verhindern und gleichzeitig Datenkompression und Verschlüsselung ermöglichen.

#### 2.2.4.7. Schicht 7 – Anwendungsschicht

Innerhalb der obersten Schicht des OSI-Referenzmodells sind verschiede ne Dienste implementiert, welche dem Endnutzer als Schnittstelle zu dem Netzwerk dienen. Bekannte Protokolle hierbei sind SSH, Telnet, FTP, NFS, SMTP und viele weitere.

Das innerhalb dieser Arbeit entwickelte Framework der NA62-Online-PC-Farm, wie es in Kapitel 4 vorgestellt wird, ist den OSI-Schichten 5 bis 7 zuzuordnen.

[9] Der Ausdruck „Endian“ stammt aus Jonathan Swifts satirischem Roman „Gulliver's Travels“ in dem ein Krieg über die Frage ausbricht, auf welcher Seite ein weichgekochtes Ei geöffnet werden sollte. Dieser Sachverhalt betont, dass verschiedenen Reihenfolgen, mit der die Bytes gelesen werden, gleichwertig und reine Definitionssache sind.

### 2.2.5. Ethernet

Das Ethernet entspricht weitestgehend der IEEE-Norm 802.3. Dabei handelt es sich um eine Kombination aus Software- (Protokoll) und Hardware-Spezifikationen, welche seit 1980 entwickelt werden. Obwohl zu Beginn der Entwicklung schon einige alternative Standards auf dem Markt waren[10], ist das Ethernet heute der mit Abstand verbreitetste Standard unter den LAN-Technologien.

Das Ethernet definiert aus Sicht des OSI-Modells die Schichten Eins (physikalische Schicht) und Zwei (Sicherungsschicht).

#### 2.2.5.1. CSMA/CD

CSMA/CD steht für Carrier Sense Multiple Access/Collision Detection und stellt die Basis für die frühen Standards der IEEE 802.3 Familie dar. Es ist der Sicherungsschicht des OSI-Referenzmodells zuzuordnen und ermöglicht die gleichzeitige Nutzung eines Mediums durch mehrere Knoten. Dabei werden so genannte Kollisionen (Überlagerung von mehreren Datenpaketen) minimiert und detektiert. Im Falle einer Kollision wird ein Störsignal (jam) an das Netzwerk gesendet, damit die jeweiligen Pakete, diesmal zeitversetzt, erneut versendet werden.

Bei schnellen Ethernet-Standards wie dem Gigabit-Ethernet verliert dieser Mechanismus jedoch an Gewicht: Heutzutage ist es üblich, Netzwerkkarten im so genannten Vollduplex-Modus zu betreiben. Das bedeutet, dass in einer Punkt-zu-Punkt-Verbindung gleichzeitig empfangen und gesendet werden kann. Dies wird beispielsweise durch Verwendung getrennter Drähte oder verschiedener Wellenlängen (Moden) bei optischer Übertragung realisiert. Somit kann es nicht mehr zu Kollisionen kommen, weshalb CSMA/CD in diesen Schnittstellen deaktiviert wird.

#### 2.2.5.2. Format

Das fast ausschließlich verwendete Format von Ethernet-Paketen ist in Abbildung 2.3 gezeigt. Dabei werden innerhalb des Ethernet-Standards alle Worte im Big-Endian-Format übertragen.

**Präambel:** Diese sieben Byte dienen der Synchronisation zwischen Sender und Empfänger. In modernen Netzwerken werden fast ausschließlich dauerhaft eingeschwungene Punkt-zu-Punkt Verbindungen genutzt, bei denen diese Präambel überflüssig ist. Aus Kompatibilitätsgründen werden diese Bytes weiterhin mit übertragen.

[10] Token ring, FDDI und ATM, um einige zu nennen [15].

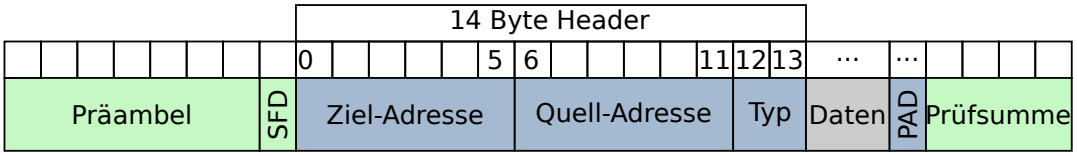


**Abbildung 2.3.:** Das Ethernet-Format nach IEEE 802.3 ohne VLAN-Tag. Dabei ist in blau der i.d.R. von der Software beeinflussbare Kopfdatenteil der Ethernet-PDU (LLC) markiert. In grün wird der von der Elektronik realisierte MAC-Teil dargestellt.

**SFD:** SFD steht für Start Frame Delimiter und markiert den Beginn der eigentlichen Ethernet-Kopfdaten. Analog zur Präambel ist dieser nur bei paketweiser Synchronisation notwendig und wird nur noch aus Kompatibilitätsgründen implementiert.

**Ziel- und Quell-Adressen:** Die in Abschnitt 2.2.5.3 beschriebenen MAC-Adressen der Sender und Empfänger des Paketes.

**Typ:** Beschreibt den Datentyp der darauf folgenden (im Datenfeld enthaltenen) PDU. Hier steht im Falle eines IP-Paketes das Wort 0x0800.

**Daten:** Das Datenfeld mit der PDU der Schicht 3 im OSI-Referenzmodell.

**PAD:** Um Kollisionen in alten Systemen sicher erkennen zu können, mussten Ethernet-Pakete, ohne SFD und Präambel, mindestens 64 Byte lang sein. Um dies zu gewährleisten, wird das PAD-Feld (kurz für Padding) mit entsprechend vielen Bytes aufgefüllt, welche auf Empfängerseite ignoriert werden. Dies wird auch bei modernen System mit deaktiviertem CSMA/CD aus Kompatibilitätsgründen durchgeführt.

**Prüfsumme:** Diese vier Byte lange **C**yclic **R**edundancy **C**heck (CRC)-Prüfsumme wird über das eigentliche Ethernet-Paket, also vom ersten Byte der Ziel-Adresse bis zum letzten Byte des PAD-Feldes, berechnet. Dadurch können alle Eins-, Zwei- und Drei-Bit-Fehler, sowie alle Fehlermuster mit ungerader Bitfehleranzahl identifiziert werden. Andere Fehlerarten bleiben mit einer Wahrscheinlichkeit von rund $10^{-5}$ unidentifiziert [17].

#### 2.2.5.3. MAC-Adressen

In Abschnitt 2.2.4.2 wurde bereits die MAC-Adresse eingeführt. Sie dient der Adressierung der Teilnehmer innerhalb eines Kommunikationsnetzes und besteht beim Ethernet

aus einem Sechs-Byte-Wort. Die drei höchstwertigen Bytes sind dabei herstellerspezifisch (Herstellerkennung). Die letzten drei Bytes dienen der eindeutigen Identifikation der Hardware innerhalb von Bauteilen der gleichen Herstellerkennung[11].

In einer Stern- oder Baum-Topologie werden häufig Switches als Kopplungselement verwendet. Diese lesen die MAC-Adresse der einzelnen Datenpakete innerhalb des Netzwerkes aus und erstellen so eine **S**ource-**A**ddress-**T**able (SAT). Diese Tabelle bildet alle im Netzwerk befindlichen MAC-Adressen auf einen Port (also eine Kante) des Switches ab. Pakete mit bekannten MAC-Adressen werden somit nur an die so definierte Kante weitergeleitet (Unicast). Unbekannte Pakete werden an alle Kanten, außer der Empfänger-Kante, weitergeleitet (Broadcast).

Möchte man bewusst Pakete an alle Teilnehmer des Kommunikationsnetzes senden (dies entspricht einem Broadcast), so kann die reservierte Broadcast-Adresse FF:FF:FF:FF:FF:FF verwendet werden. Häufig möchte man jedoch nur eine spezielle Gruppe von Knoten erreichen. Diese als Multicast bezeichnete Kommunikationsmethode kann durch spezielle MAC-Adressen in Kombination mit dem IP realisiert werden. Darauf wird in Abschnitt 2.3.3 genauer eingegangen.

#### 2.2.5.4. Flusskontrolle nach IEEE 802.3x

Dieser oft vergessene Standard könnte für die NA62-Farm eine wichtige Rolle spielen. Er ermöglicht den Teilnehmern eines Ethernet-Netzwerkes eine Überlastung, und somit den Verlust von Paketen zu verhindern. Dazu werden so genannte Pause-Pakete eingesetzt: Sobald ein Knoten überlastet wird, sendet dieser ein spezielles Paket an eine feste Hardware-Adresse[12] mit der Anfrage, eine bestimmte Zeit zu warten[13].

Der Nachteil dieser Methode kann durch folgendes Beispiel leicht gezeigt werden: Es seien in einem Netzwerk zwei PCs (A und B) über 10GbE mit einem Switch verbunden. Weiter sei ein dritter PC (C) über eine 1GbE-Kante mit dem gleichen Switch vernetzt. Sendet nun PC A mit voller Datenrate an die beiden anderen PCs, so kann er die Netzwerkkarte des PCs C schnell überlasten. Somit würde C ein Pause-Paket an den Switch senden, welcher dann die Übertragung zu C pausiert. Da sich nun die Daten von A zu C im Switch stauen, sendet kurze Zeit darauf der Switch ein Pause-Paket an PC A um den eigenen Puffer zu entlasten. Jetzt stoppt PC A den kompletten Datentransfer in Richtung Switch. Somit würde auch die Kommunikation mit PC B zum Erliegen kommen, obwohl diese zu keinem Stau geführt hatte und weiterhin vom Switch bearbeitet werden könnte.

[11] Z.B. haben die Hardware-Adressen der innerhalb dieser Arbeit verwendeten Intel® Netzwerkkarten die Form 00:07:E9:XX:XX:XX. Einer Firma können mehrere Herstellerkennungen zugeordnet sein.

[12] Diese lautet 01-80-C2-00-00-01 im IEEE 802.3x Standard.

[13] Über ein Zwei-Byte-Wort wird die Anzahl an 512-Bitzeiten definiert. Dabei ist eine N-Bitzeit jene Zeit, die der Empfänger des Pause-Pakets benötigt, um N Bits zu senden.

## 2.3. Die Internetprotokollfamilie

Die Internetprotokollfamilie ist im OSI-Modell in den Schichten Drei und Vier angesiedelt. Sie besteht aus rund 500 Protokollen, wobei wir hier auf fünf der wichtigsten eingehen möchten: **I**nternet-**P**rotokoll (IP), **T**ransmission **C**ontrol **P**rotocol TCP, **U**ser **D**atagram **P**rotocol (UDP), **A**ddress **R**esolution **P**rotocol (ARP) und **I**nternet **G**roup **M**anagement **P**rotocol (IGMP). Dabei setzen, bis auf ARP, alle Protokolle auf IP auf.

### 2.3.1. Das Internet-Protokoll (IP)

Wie bereits in Abschnitt 2.2.4.3 erläutert, dient die Vermittlungsschicht und somit das IP hauptsächlich der Wegesuche. Neben der MAC-Adresse erhält nun jeder Teilnehmer eine, innerhalb des Netzwerkes eindeutige, IP-Adresse. Diese Adresse wird über die sogenannte Subnetzmaske einem Bereich (Subnetz) des Netzwerkes zugewiesen. Diese Unterteilung findet hierarchisch statt, so dass das Wegesuchen iterativ durchgeführt werden kann. Einzelne Pakete werden somit von Subnetz zu Subnetz durch die dazwischen befindlichen Router geleitet. Dies ermöglicht es, extrem große Netzwerke, wie etwa das Internet, zu realisieren.

Obwohl diese Fähigkeit des IP innerhalb der NA62-Farm nicht benötigt wird, da die Wegewahl auf Basis von Hardware-Adressen in dem relativ kleinen Netzwerk ausreicht, wird dieses Protokoll trotzdem eingesetzt. Der Grund ist, dass IP durch die Größe des Internets sehr verbreitet ist und somit eine große Auswahl an Standard-Software existiert.

Innerhalb dieser Arbeit wird ausschließlich IPv4 eingesetzt, welches z.Z. das am meisten verbreitete Protokoll darstellt[14] (im folgenden Text als IP bezeichnet). Das Format der PDU-Kopfdaten ist in Abbildung 2.4 zu sehen.

Die einzelnen Felder der Kopfdaten sind wie folgt zu beschreiben (siehe Abbildung 2.4):

**IP Header Länge (IHL):** Anzahl der 32-Bit-Worte innerhalb der IP-Kopfdaten. Dieser Wert beträgt 5 für den Fall, dass kein Options-Feld gesendet wird.

**Type Of Service:** Kann zur Priorisierung einzelner Pakete verwendet werden.

**Länge:** Die Anzahl an Bytes innerhalb des IP-Paketes inklusive Kopfdaten.

[14] Wie in Abbildung 2.4 dargestellt, wird innerhalb von IPv4 ein 4-Byte-Wort zur Adressierung verwendet. Dies beschränkt die Größe des Internets auf maximal $4,3 \cdot 10^9$ Knoten, wobei durch reservierte Adressen und durch die Unterteilung in Subnetze tatsächlich deutlich weniger Adressen genutzt werden können. Laut [18] werden innerhalb der nächsten fünf Jahre die letzten freien IPv4-Adressen vergeben. Daher wurde bereits 1998 der IPv6-Standard mit einem 16-Byte-Wort zur Adressierung eingeführt.

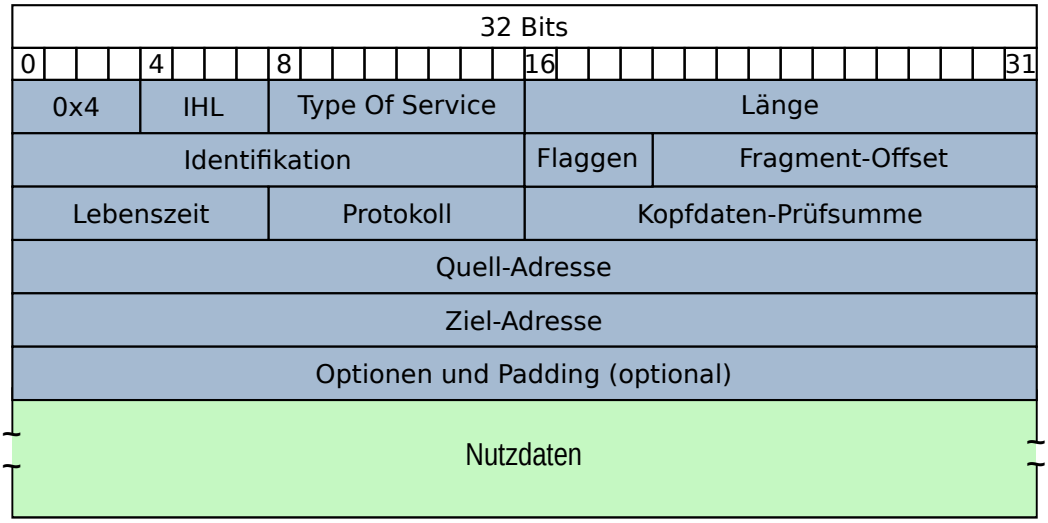


**Abbildung 2.4.:** Das Format der IPv4-PDU nach RFC791 (1981).

**Identifikation:** Dient der Identifikation dieses Paketes innerhalb der fragmentierten Daten zur Reassemblierung der ursprünglichen Nutzdaten (siehe Abschnitt 2.3.1.1.

**Flaggen:** Hiermit kann eine bereits stattgefundene Fragmentierung signalisiert und eine (weitere) Fragmentierung unterbunden werden.

**Fragment-Offset:** Im Falle einer Fragmentierung definiert dieses Feld die Position der Nutzdaten innerhalb des ursprünglichen Paketes.

**Lebenszeit:** Verhindert ein endloses Weiterleiten von Paketen. Jeder Knoten verringert diesen Wert um Eins, bis er Null ergibt und das Paket verworfen wird.

**Protokoll:** Beschreibt den Typ des im Nutzdatenfeld enthaltenen Protokolls. Hier stehen z.B. die Werte 0x01 (ICMP), 0x02 (IGMP), 0x06 (TCP) oder 0x11 (UDP).

**Kopfdaten-Prüfsumme:** Prüfsumme mit der die korrekte Übertragung des IP-Kopfdatenteils überprüft werden kann.

**Quell- und Ziel-Adresse:** Die eigentliche IP-Adresse des Senders und Empfängers.

**Optionen und Padding:** Hier können zusätzliche Informationen übertragen werden. Bei Echtzeitanwendungen wie dem L0TP kann hier der Zeitpunkt des Eintreffens des Paketes gespeichert und von der Anwendung wieder ausgelesen werden.

#### 2.3.1.1. Fragmentierung

Da, wie in Abbildung 2.4 gezeigt, innerhalb der IP-Kopfdaten zwei Byte für die Längenangabe zur Verfügung stehen, können Pakete mit einer Länge von bis zu 65535 Byte erstellt werden. Die **M**aximum **T**ransmission **U**nit (MTU) beschreibt die maximale Anzahl an Bytes, die von der Vermittlungsschicht (hier also IP) an die darunter liegende Sicherungsschicht (hier also Ethernet) übertragen werden kann. Das Ethernet sieht dabei nur eine minimale MTU von 1500 Bytes vor. Viele moderne Kopplungselemente und Netzwerkkarten erlauben eine maximale MTU von 9000 Bytes. Dabei werden Pakete oberhalb der Standard MTU (1500 B) als Jumbo-Frames bezeichnet.

Um nun tatsächlich 65 kB große Daten-Pakete (Datagramme) versenden zu können, erlaubt das IP eine Fragmentierung eines Datagramms in mehrere Pakete, welche jeweils kleiner als die MTU sein müssen. Dazu werden die in Abbildung 2.4 dargestellten Datenfelder „Identifikation“, „Flaggen“ und „Fragment Offset“ auf die im Folgenden beschriebene Art und Weise gesetzt.

Soll ein IP-Datagramm der Länge $N$ mit $N > MTU$ gesendet werden, so wird dieses in $M$ Pakete unterteilt. In den Nutzdaten dieser $M$ IP-Pakete steht jeweils ein Teil der Nutzdaten des eigentlichen Datagramms inklusive des Offsets. Das erste Paket hat somit Daten mit einem Offset von Null. Überträgt dieses nun 1480 Bytes (Byte 0 bis Byte 1479) des Datagramms, so erhält das zweite Paket einen Offset von 1480 und überträgt Byte 1480 und weitere Bytes des Datagramms.

Da das Datenfeld „Fragment Offset“ nur in 8-Byte-Blöcken angegeben ist, müssen die ersten $M - 1$ Pakete ein Vielfaches von 8 Byte im Nutzdatenteil beinhalten. Diese Pakete haben gleichzeitig im Datenfeld „Flagge“ das niedrigste Bit auf eins gesetzt. Dies kennzeichnet, dass weitere Pakete folgen. Das letzte Paket hat dieses Bit nicht gesetzt und kann beliebig viele Datenfeld-Bytes enthalten, wobei auch hier die MTU nicht überschritten werden darf.

All diese $M$ Pakete tragen das gleiche Wort im Identifikations-Wort der IP-Kopfdaten. Mit diesem Feld, in Verbindung mit den Datenfeldern „Protokoll“, „Quell-Adresse“ und „Ziel-Adresse“ können Fragmente vom Empfänger zu einem Datagramm zusammengesetzt werden. Dabei ist darauf zu achten, dass die einzelnen „Fragment-Offset“-Werte lückenlos von 0 bis zum Wert des Paketes ohne gesetzte Flagge anschließen müssen. Sollte dies durch verloren gegangene oder fehlerhafte Pakete nicht der Fall sein, so werden alle Pakete, also das komplette Datagramm, verworfen.

### 2.3.2. Address Resolution Protocol (ARP)

Möchte man einen speziellen Knoten innerhalb eines IP-Netzwerkes adressieren, so muss sowohl die IP-Adresse als auch die MAC-Adresse bekannt sein. Das ARP dient dazu, die

einer IP-Adresse zugehörige MAC-Adresse bestimmen zu können.

Bei einer normalen ARP-Anfrage wird ein spezielles Broadcast-Ethernet-Paket, genannt ARP-Anfrage-Paket, versendet. Darin befinden sich die gesuchte IP-Adresse und die IP- und MAC-Adressen des Senders. Empfängt der Rechner mit der genannten IP-Adresse dieses Paket, so antwortet dieser mit einem Unicast-ARP-Paket. Dabei ist in diesem Paket die IP- und MAC-Adresse des gesuchten PCs eingetragen.

Zusätzlich ermöglichen so genannte Gratuitous-ARP-Anfragen das aktive Verbreiten der MAC-IP-Adressen-Zuordnung. Dabei sendet jeder Rechner in regelmäßigen Abständen ein Broadcast-ARP-Paket mit seiner eigenen MAC- und IP-Adresse im Sender- und Empfänger-Feld. Dieses Paket beinhaltet somit gleichzeitig Anfrage und Antwort. Alle Empfänger dieses Paketes können damit ihre ARP-Tabellen aktualisieren.

### 2.3.3. Multicast

Bisher wurde beschrieben, wie mit Hilfe der MAC-Adressen Unicast- und Broadcast-Pakete erzeugt werden können. Für die Kommunikation zwischen L1-Farm und CREAMs ist jedoch eine Multicast-Kommunikation nötig. Das bedeutet, dass ein PC ein Paket versendet, welches durch die Switches innerhalb des Netzwerkes an eine spezielle Gruppe von Knoten (z.B. alle CREAMs) weitergeleitet wird. Dies ist in Verbindung mit dem Internet-Protokoll möglich.

Hierzu werden Multicast-Gruppen definiert, bei der sich die Teilnehmer des Netzwerkes über das IGMP anmelden können (siehe Abschnitt 2.3.4). Jede dieser Gruppen wird durch eine IP-Adresse im Bereich 224.0.0.0 bis 239.255.255.255 identifiziert. IP-Pakete an eine dieser Adressen werden dann von allen Teilnehmern empfangen, die sich in der entsprechenden Gruppe angemeldet haben. Bei diesen Paketen muss dabei eine spezielle Empfänger-MAC-Adresse gesetzt sein, damit Switches die Pakete korrekt weiterleiten können, ohne Informationen der Vermittlungsschicht auslesen zu müssen. Hierzu werden die untersten 23 Bits der Multicast-IP-Adresse mit der MAC-Adresse 01:00:5E:00:00:00 addiert. Die Gruppe 224.0.1.2 würde somit die MAC-Adresse 01:00:5E:00:01:02 erhalten. Auf diese Weise werden mehrere Multicast-IP-Adressen auf die gleiche MAC-Adresse abgebildet. Man muss also darauf achten, dass innerhalb eines Netzwerkes nur Multicast-Gruppen verwendet werden, deren Adressen sich innerhalb der untersten 23 Bits unterscheiden.

### 2.3.4. Internet Group Management Protocol (IGMP)

Dieses Protokoll ermöglicht einem Knoten, sich bei Switches und Routern für eine Multicast-Gruppe zu registrieren. Dazu sendet ein Teilnehmer einer Multicast-Gruppe

ein IGMP-Paket mit der Multicast-Adresse und einer Flagge „Anmeldung“ wie in Abschnitt 2.3.3 beschrieben an die Multicast-Gruppen-IP. Analog kann eine Abmeldung mit entsprechender Flagge durchgeführt werden. Zudem gibt es die Möglichkeit für Switches und Router Mitgliedschaften von anderen Knoten anzufragen. Dazu wird erneut ein Multicast-IGMP-Paket mit entsprechender Flagge an die gesuchte Gruppe gesendet. Als Antwort müssen alle Teilnehmer der angefragten Gruppe mit einem Anmelde-Paket reagieren.

### 2.3.5. User Datagram Protocol (UDP)

UDP dient dem verbindungslosen Transport von Daten an ein spezifisches Programm innerhalb des adressierten Rechners und setzt zu diesen Zwecken auf dem IP auf. Verbindungslos bedeutet hierbei, dass jedes Paket einzeln betrachtet werden (Zustandslosigkeit) und die Reihenfolge der versendeten Pakete nicht erhalten bleiben muss.

Durch den in Abschnitt 2.3.1.1 beschriebenen Mechanismus der Fragmentierung kann ein UDP-Datagramm inklusive Kopfdaten eine Größe von bis zu 65535 Bytes erreichen. Diese Limitierung ist durch die Größe des Längen-Wortes in den UDP-Kopfdaten (siehe Abbildung 2.5) gegeben.

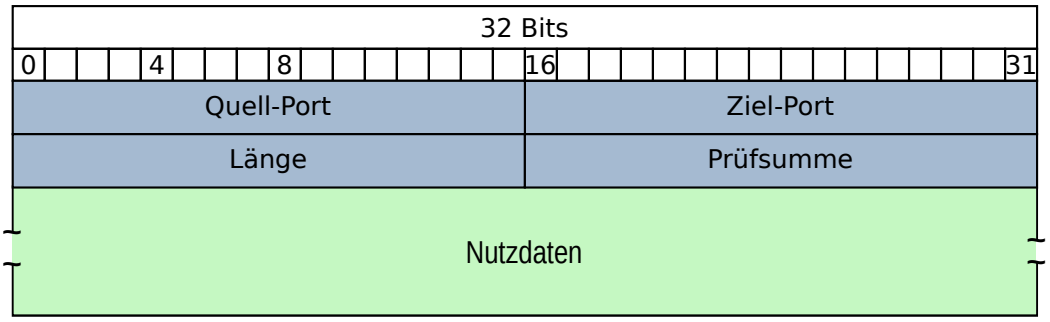


**Abbildung 2.5.:** Format der UDP-PDU nach RFC768 (1980). Diese Daten werden selbst innerhalb der Nutzdaten eines IP-Paketes übertragen (siehe Abbildung 2.4).

Die einzelnen in Abbildung 2.5 vorkommenden Felder sind wie folgt zu beschreiben:

**Quell- und Ziel-Port:** Zur Adressierung der Programme auf Quell- und Ziel-Seite der Kommunikation.

**Länge:** Die Anzahl an Bytes innerhalb des Datagramms inklusive Kopfdaten.

**Prüfsumme:** Die nach RFC768 definierte Prüfsumme über den Kopf[15]- und Nutzdatenteil.

Die verwendete Prüfsumme dient, zusätzlich zur Ethernet-Prüfsumme, der Erkennung fehlerhafter Übertragungen. Da fehlerhafte Pakete auf Empfängerseite verworfen werden und der Sender darüber nicht informiert wird, bezeichnet man dieses Protokoll als nicht-zuverlässig. Zuverlässigkeit muss durch alternative Protokolle innerhalb der Transportschicht, oder durch auf UDP aufsetzende Protokolle in höheren Ebenen realisiert werden.

### 2.3.6. Transmission Control Protocol (TCP)

TCP stellt eine zuverlässige, verbindungsorientierte Alternative zu UDP dar. Hier werden Daten nicht mehr in einzelnen Datagrammen, sondern durch einen kontinuierlichen Datenstrom übertragen. Dabei ist zu beachten, dass TCP weiterhin auf das verbindungslose IP aufsetzt und die Verbindung innerhalb von TCP nur virtuell ist.

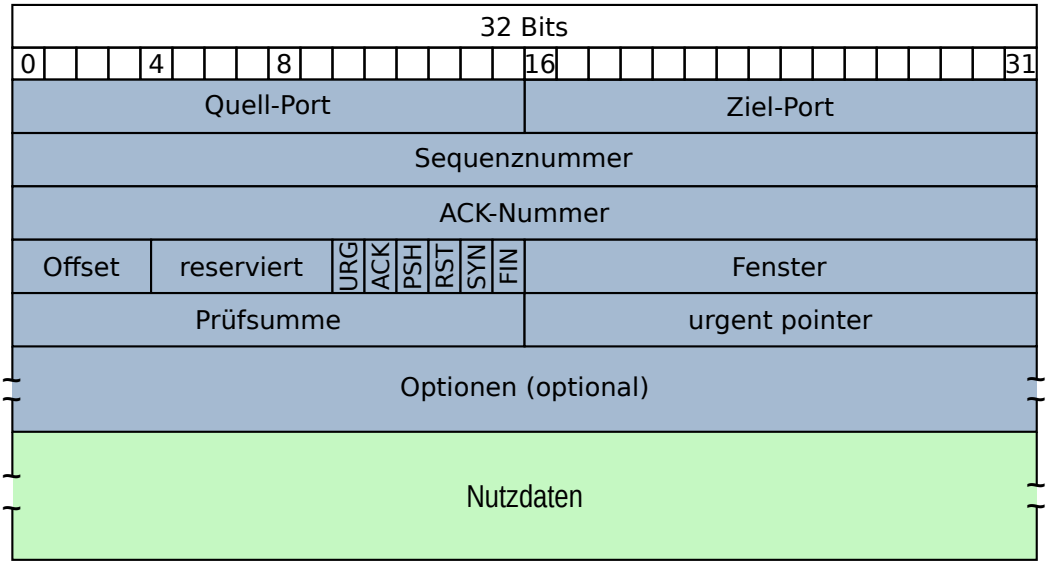


**Abbildung 2.6.:** Format der TCP-PDU nach RFC793 (1981) und RFC1323 (1992). Diese Daten werden selbst innerhalb der Nutzdaten eines IP-Paketes übertragen (siehe Abbildung 2.4).

[15]Zum Erstellen der Prüfsumme wird das Prüfsummen-Wort zunächst auf Null gesetzt und 12 Byte mit Informationen aus dem IP-Kopfdatenteil vor das UDP-Datagramm gesetzt. Es entsteht der so genannte Pseudo-Header, welcher nur für die Erstellung der Prüfsumme dient.

Da, wie in Abbildung 2.6 ersichtlich, dieses Protokoll deutlich komplexer als die bisher beschriebenen ist, soll hier nur auf die wichtigsten Funktionen eingegangen werden:

**Quell- und Ziel-Port:** Adressen der Programme auf Quell- und Ziel-Seite der Kommunikation. Diese definieren die Verbindung.

**Sequenznummer:** Dient der Ordnung einzelner Pakete (Verbindungsorientiertheit).

**ACK-Nummer:** Dient der Bestätigung (engl. **ack**nowledgement) eines empfangenen Paketes (Zuverlässigkeit).

**Offset:** Länge des Kopfdatenteils.

**ACK:** Zeigt an, dass ACK-Nummer gesetzt ist (Flagge).

**SYN:** Dient dem Aufbau einer Verbindung (Flagge).

**FIN:** Dient der Beendigung einer Verbindung (Flagge).

**Fenster:** Dient der Flusssteuerung (siehe Abschnitt 2.3.6.1).

**Prüfsumme:** Die Prüfsumme wird wie bei UDP-Kopfdaten erzeugt (siehe Abschnitt 2.3.5).

Die Sequenznummer wird beim Verbindungsaufbau auf einen zufälligen[16] Wert gesetzt. Nach dem Senden eines Paketes mit $N$ Bytes im Nutzdaten-Feld, wird die Sequenznummer um $N$ erhöht[17]. Der Empfänger bestätigt ein Paket mit der Sequenznummer $S$ und $N$ Bytes im Nutzdaten-Feld durch die Sendung eines Paketes mit der ACK-Nummer $S + N$. Die ACK-Nummer gibt also die nächste erwartete Sequenznummer an. Auf diese Weise wird sowohl die richtige Reihenfolge der Pakete als auch die zuverlässige Übertragung gewährleistet: Sollte ein Paket nach einer gewissen Zeit nicht bestätigt worden sein, so wird dieses vom Sender erneut übertragen.

Abbildung 2.7 zeigt den benötigten Datenfluss zur Übertragung von $Z$ Bytes von Rechner A zu Rechner B. Dabei wird deutlich, dass alleine sechs Pakete zum Verbindungsauf- und Abbau benötigt werden. TCP kann also nur für effiziente Datenübertragungen verwendet werden, wenn die Verbindung über lange Zeit aufrechterhalten wird.

[16] Aus Sicherheitsgründen beginnt man nicht mit der Sequenznummer Null.

[17] Eine Ausnahme stellt der Verbindungsauf- und Abbau dar. Bei den hierfür benötigten Paketen wird die Sequenznummer jeweils um eins erhöht.

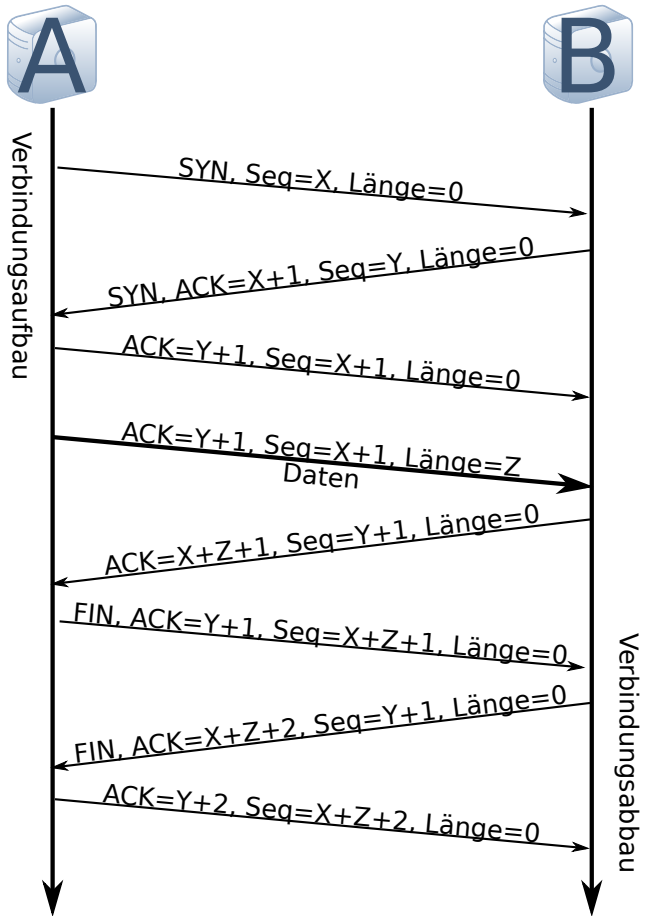


**Abbildung 2.7.:** Datenfluss einer TCP-Verbindung vom Aufbau bis zur Schließung der Verbindung. Hierbei zeigen SYN und FIN an, dass die entsprechenden-Flaggen in den Kopfdaten auf 1 stehen. ACK steht dafür, dass die entsprechende Flagge auf 1 steht und der angegebene Wert im ACK-Nummer-Feld steht. Seq bezeichnet das Datenfeld „Sequenznummer“ im Datenkopf und Länge beschreibt die Anzahl an Bytes im Nutzdaten-Feld. Die Länge muss über die IP-Kopfdaten ermittelt werden.

#### 2.3.6.1. Flusssteuerung

In kleineren Netzwerken mit sehr großen Datenraten ist die Flusssteuerung die wahrscheinlich wichtigste Funktionalität des TCP-Protokolls. Sie stellt sicher, dass nur dann Pakete versendet werden, wenn der adressierte Empfänger genügend Speicherplatz in seinem Puffer besitzt, um das Paket auch empfangen zu können.

Für diesen Zweck wird die Fenster-Größe eingeführt, welche die Größe freien Speichers für die jeweilige Verbindung bezeichnet. Sie wird über das „Fenster“-Datenfeld im TCP-Kopfdatenteil übertragen und kann jederzeit durch Senden eines evtl. leeren TCP-Pakets aktualisiert werden. Auf diese Weise wird der Sender informiert, wie viele Daten er maximal verschicken darf, bis er das nächste Bestätigungs-Paket empfängt.

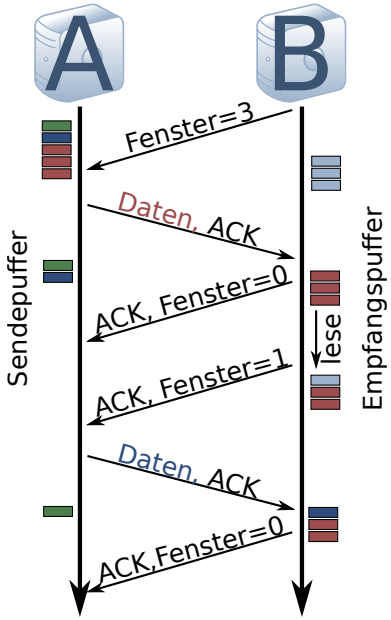


**Abbildung 2.8.:** Prinzip der Flusssteuerung im TCP. Die Fenster-Größe gibt an wie viele Daten der Sender unbestätigt versenden darf. Somit kann der Empfangspuffer nicht überlaufen.

In Abbildung 2.8 wird der Mechanismus der Flusssteuerung erläutert. Dabei hat der Empfänger (B) eine Fenster-Größe von 3 Byte[18]. Der Sender (A) erfährt von dieser Größe durch ein TCP-Paket mit entsprechendem Fenster-Wort. Da A 5 Byte senden möchte und damit den Empfängerpuffer überlasten würde, sendet er nur 3 Byte. Der Empfänger bestätigt den Empfang und gibt an, dass sein Speicher nun voll ist (Fenster-Größe von Null). Nach einer gewissen Zeit hat ein Empfänger-Programm $N$ Bytes aus dem Puffer

[18]Dies ist ein vereinfachendes Beispiel. Realistische Fenster-Größen liegen im kB bis MB Bereich.

gelesen. Dann sendet der Empfänger ein neues TCP-Paket mit der Information, dass die Fenster-Größe nun auf $N$ gestiegen ist. Der Sender kann also die nächsten $N$ Bytes senden. Für kleine $N$ kann dies zu dem so genannten „Silly Window Syndrome“ führen. Dabei werden ständig sehr kleine Datenmengen ($N$ Bytes) in einem Ethernet-Paket mit Ethernet-, IP- und TCP-Kopfdaten übertragen, die in der Summe deutlich größer als $N$ Bytes sind. Um diese ineffiziente Übertragung zu verhindern, wird der so genannte Nagle-Algorithmus verwendet (Definition im RFC896). Dabei werden kleine Pakete nur dann gesendet, wenn keine unbestätigten Pakete im Umlauf sind.

#### 2.3.6.2. Überlaststeuerung

Die Flusssteuerung aus Abschnitt 2.3.6.1 berücksichtigt nur die Puffergrößen der Sende- und Empfangsknoten. Dazwischen können jedoch beliebig viele weitere Kopplungselemente, etwa Switches und Router, liegen. Um diese nicht zu überlasten, verfügt das TCP über zwei Algorithmen zur Steuerung der Datenrate, genannt „Slow Start“ und „Congestion Avoidance“ (siehe **R**equest **F**or **C**omments (RFC)2581[19]) [17].

Dabei wird, analog zur Fenster-Größe in der Flusskontrolle, das Überlastfenster eingeführt. Das kleinere dieser beiden Fenster beschreibt nun die maximale Datenmenge, die zwischen zwei Bestätigungen versendet werden darf. Nun wird das Überlastfenster, von einer **M**aximum **S**egment **S**ize (MSS)[20] aus, zunächst exponentiell (Slow Start) und später ab einem Schwellwert nur noch linear (Congestion Avoidance) erhöht (siehe Abbildung 2.9). Sobald ein Paket verloren geht (was ein Hinweis für eine Überlastung ist) wird der Schwellenwert halbiert und das Überlastfenster wieder auf eine MSS gesetzt.

## 2.4. Parallele Programmierung

Heutzutage beinhalten fast alle CPUs auf dem Markt mindestens zwei Hauptprozessoren (Prozessorkerne). Vorteil solcher Mehrkernprozessoren ist einerseits, dass mehrere Prozesse gleichzeitig ausgeführt werden können. Andererseits ist man bei modernen Prozessoren, welche mit bis zu 4 GHz getaktet werden, an technische Grenzen gestoßen. Um also kostengünstig mehr Rechenkapazität zu erlangen, werden mehrere Prozessoren zu einem Mehrkernprozessor vereinigt. Um diese Rechenkapazität effizient ausnutzen zu

[19] In RFC2581 werden noch die Algorithmen „fast retransmit“ und „fast recovery“ definiert, welche jedoch für die Zwecke innerhalb der Online-PC-Farm nicht geeignet sind und hier nicht weiter erläutert werden.

[20] Das ist nach RFC879 die maximale Anzahl an Bytes die, innerhalb der Nutzdaten eines TCP-Pakets versendet werden darf. Dieser Wert entspricht in der Regel der Differenz zwischen MTU und der Summe aus IP- und TCP-Kopfdatenlänge um IP-Segmentierungen zu vermeiden.

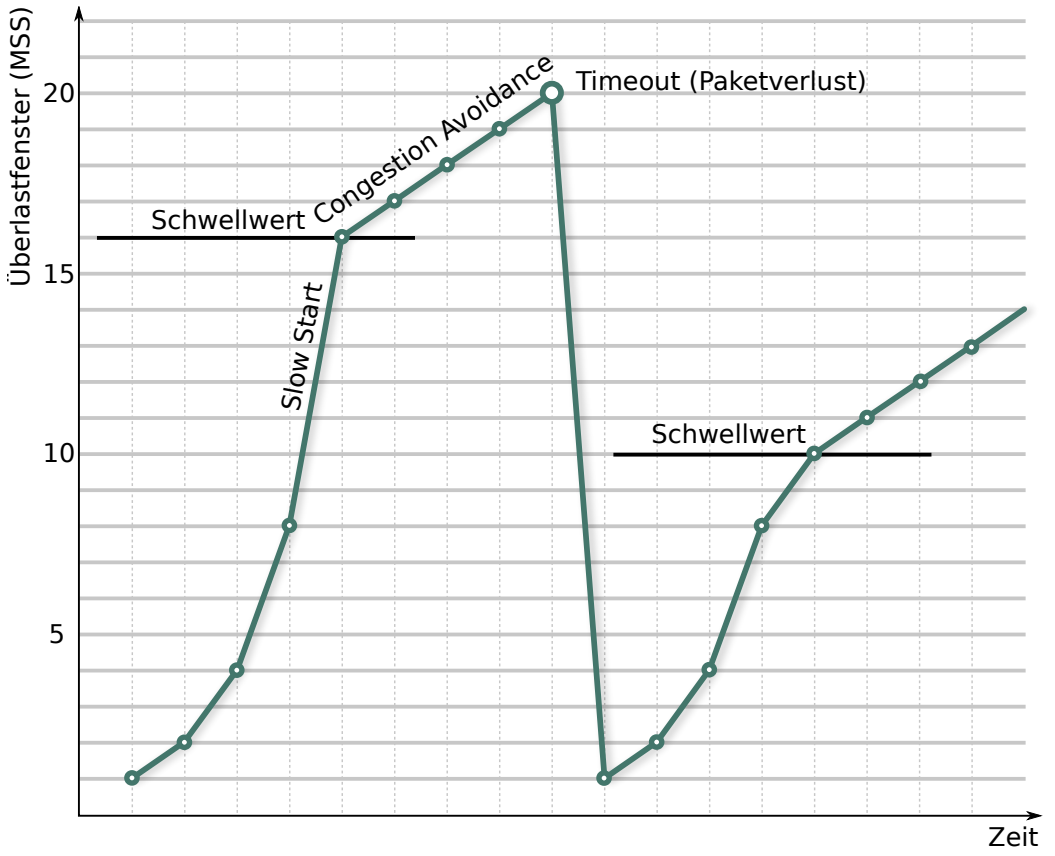


**Abbildung 2.9.:** Überlaststeuerung durch Slow Start und Congestion Avoidance. Von einer MSS aus beginnend wird das Überlastfenster bei jeder erfolgreichen Übertragung vergrößert: unterhalb des Schwellwertes verdoppelt, oberhalb um eine MSS vergrößert. Geht ein Paket verloren, wird der Schwellwert halbiert und das Überlastfenster erneut auf eine MSS gesetzt.

können, bedarf es eines speziellen Programmierparadigmas: die parallele Programmierung.

Ziel der parallelen Programmierung ist es, Software so zu unterteilen, dass die einzelnen Teilstücke parallel prozessiert werden können. Diese Nebenläufigkeit kann auf verschiedene Arten realisiert werden. Man unterscheidet hierbei zwischen Multitasking und Multithreading:

**Multithreading:** Dies bedeutet, dass innerhalb eines Prozesses die Verarbeitung in mehrere nebenläufige „Fäden“ (Threads) aufgeteilt wird. Diese können, als Teil des übergeordneten Prozesses, auf die gleichen Ressourcen zugreifen. Nebenläufige Zugriffe mehrerer Threads auf eine Ressource (Speicheradresse) können zu so genannten kritischen Wettläufen (engl. Race Conditions) führen und müssen daher synchronisiert werden. Darauf wird in Abschnitt 2.4.1 näher eingegangen.

**Multitasking:** Dies bezeichnet die Nebenläufigkeit von verschiedenen Prozessen mit unterschiedlichen Speicherbereichen. Ein Beispiel wäre ein HTTP-Server und eine Datenbank. Beides sind unabhängige Programme, die nur hin und wieder miteinander kommunizieren müssen (etwa eine Datenbankabfrage von einer Webanwendung). Die Kommunikation findet dabei typischerweise über Shared Memory oder Netzwerkprotokolle statt. Auch hier muss i.d.R. eine Synchronisierung stattfinden.

### 2.4.1. Kritischer Wettlauf

Für den Fall, dass das Ergebnis einer Operation von der Reihenfolge und dem zeitlichen Verhalten verschiedener Teiloperationen abhängt, kann die nebenläufige Ausführung verschiedener Teiloperationen zu unerwarteten und nicht reproduzierbaren Ergebnissen führen. Dieses Verhalten wird als Kritischer Wettlauf (engl. Race Condition) bezeichnet.

Kritische Wettläufe sollen an Hand zweier Threads, die die gleiche Speicheradresse um den Wert Eins inkrementieren, erläutert werden: Zur Inkrementierung muss der aktuelle Wert, sagen wir 5, zunächst aus dem Systemspeicher ausgelesen werden. Dann wird der neue Wert innerhalb des Prozessors bestimmt ($5+1 = 6$) und anschließend der alte Wert (5) im Systemspeicher durch den neu bestimmten (6) überschrieben.

Während der Bestimmung des neuen Wertes im Prozessor liegt der ausgelesene Wert im Prozessor-Cache. Sollte sich der Originalwert im Systemspeicher währenddessen ändern, so bekommt der inkrementierende Thread davon nichts mit. Möchten also zwei Threads auf unterschiedlichen Prozessorkernen den gleichen Wert um eins inkrementieren, so kann es dazu kommen, dass beide quasi gleichzeitig den Wert 5 auslesen und mit dem Wert 6 überschreiben. Nach den beiden Inkrementierungen wäre jedoch eigentlich der Wert

7 zu erwarten. Um diese Problematik zu umgehen, müssen Threads, die sich Speicherbereiche teilen, synchronisiert werden. Ein mögliches Verfahren ist die Verwendung von wechselseitigen Ausschlüssen, wie sie in Abschnitt 2.4.2 eingeführt werden.

Programmteile, welche nebenläufig von mehreren Threads ausgeführt werden können, ohne dabei Kritische Wettläufe hervorzurufen, werden als threadsicher bezeichnet.

### 2.4.2. Wechselseitiger Ausschluss

Ein wechselseitiger Ausschluss (engl. **mut**ual **ex**clusion, kurz Mutex) bezeichnet das häufigste Verfahren zur Synchronisierung von nebenläufigen Threads oder Prozessen. Dazu wird ein Objekt implementiert, welches mindestens die Methoden „sperren“ (engl. lock) und „entsperren“ (engl. unlock) beinhaltet. Sobald dieses oft selbst als Mutex bezeichnete Objekte durch einen Thread gesperrt wird, verweigert es anderen Threads den Zugriff auf einen kritischen Abschnitt des Programms. Wenn der sperrende Thread den kritischen Programmteil überwunden hat, entsperrt es den Mutex, so dass der nächste Thread den kritischen Abschnitt bearbeiten kann[21].

Dies wird in der Regel so realisiert, dass die Methode `lock()` im gesperrten Zustand blockiert. Ein Thread, der also einen gesperrten Mutex sperren möchte, wird durch die Methode `lock()` so lange aufgehalten, bis der ursprünglich sperrende Thread die Methode `unlock()` aufruft und den Mutex entsperrt. Kritische Bereiche im Programmcode können somit durch das Sperren und Entsperren eines Mutex threadsicher gemacht werden.

Diese Methode erlaubt eine Synchronisierung mit wenig Aufwand. Bei objektorientierten Sprachen kann so z.B. jede Klasse mit kritischen Instanzvariablen mit einem Mutex als Instanzvariable bestückt werden. Jeder Zugriff auf die kritischen Variablen innerhalb der Klasse kann dann durch den Mutex eingekapselt werden. Bei Erweiterungen der Klasse muss nur innerhalb der neuen Programmteile auf den vorhandenen Mutex zugegriffen werden.

Ein großer Nachteil dieser Methode ist jedoch offensichtlich: sollte auf eine nebenläufig verwendete Speicheradresse mit hoher Frequenz zugegriffen werden, so ist die Wahrscheinlichkeit groß, dass der dazu verwendete Mutex bereits gesperrt ist und ein Thread blockiert wird. Somit sinkt die Effizienz, da im Extremfall keine Nebenläufigkeit mehr stattfindet und immer nur ein Thread aktiv ist. Um dieses Problem zu umgehen, wurde

[21] Mutexe werden häufig mit binären Semaphoren gleichgesetzt, wobei eine $N$-Semaphore maximal $N$ Prozessen oder Threads den Zugriff auf den kritischen Abschnitt ermöglicht. Tatsächlich gehören Mutexe zur Gruppe der Sperrvariablen, welche im Gegensatz zu Semaphoren nur von dem Thread entsperrt werden können, welcher das Objekt ursprünglich selbst gesperrt hat.

innerhalb dieser Arbeit eine Methode entwickelt, mit der keine blockierenden Objekten benötigt werden und gleichzeitig eine hohe Parallelität und Effizienz der PC-Farm-Software realisiert wird. Dieses Verfahren wird in Abschnitt 4.5.1.1 eingeführt.

### 2.4.3. Rundlauf-Verfahren

Das Rundlauf-Verfahren (englisch Round-Robin) ist ein häufig verwendetes Verfahren zur Lastverteilung. Es dient der gleichmäßigen Zuordnung einer Ressource an mehrere konkurrierende Prozesse oder Threads. Dazu wird die Ressource, jeweils für einen festen Zeitschlitz, den verschiedenen Prozessen oder Threads nacheinander zugewiesen. Die Reihenfolge wird i.d.R durch eine feste Tabelle (Round-Robin-Tabelle), oder durch die Reihenfolge in der die Prozesse oder Threads einen Zugriff auf die Ressource reserviert haben, festgelegt.

# Kapitel 3.

# Konzept der Online-PC-Farm

In Abschnitt 2.2.3 wurde der Begriff der Topologie bereits erörtert. In diesem Kapitel wird nun die Entwicklung einer optimalen Topologie für die Online-PC-Farm in Verbindung mit einem Konzept für den Datentransfer beschrieben, und die Vor- und Nachteile der jeweiligen Alternativen besprochen.

## 3.1. Ursprüngliches Konzept

Vor Beginn dieser Arbeit wurde bereits ein Konzept für die PC-Farm durch die Kollaboration entwickelt. Dabei hat man sich an der Infrastruktur anderer Hochenergie-Experimente und der Realisierung am Vorgänger-Experiment NA48 orientiert. Das entstandene Konzept eines dreistufigen Online-Triggersystems kann wie folgt zusammengefasst werden:

- Die L1- und L2-Trigger-Stufen bestehen aus physisch wie logisch getrennten PC-Farmen mit ca. 10 PCs (L1) und ca. 30 PCs (L2).
- Es werden detektorspezifische L1-PCs eingesetzt. Das bedeutet, dass jeder L1-PC nur für einen oder eine kleine Zahl von Detektoren zuständig ist. Die L1-PCs für das RICH und evtl. noch weiteren Detektoren werden mit Grafikkarten ausgestattet.
- Die L0-Elektronik wird über 1GbE-Switches mit zusätzlichen 10GbE-Ports gebündelt und mit der L1-Farm inkl. L1TP über ein 10GbE-Netzwerk verbunden. Dabei kann prinzipiell für jeden Detektor ein getrenntes Netzwerk verwendet werden.
- Für die Kommunikation zwischen den Farmen wird ein weiteres 10GbE-Netzwerk verwendet.
- Die LKr-Elektronik wird auf die gleiche Weise gebündelt und mit dem 10GbE-Netzwerk zwischen den Farmen verbunden.

- Die L2-Farm wird über ein 1GbE-Netzwerk mit einem Speicher-Cluster und über eine gemeinsame 10GbE-Glasfaser mit dem CERN-Rechenzentrum verbunden.

Diese Topologie wird in Abbildung 3.1 dargestellt.

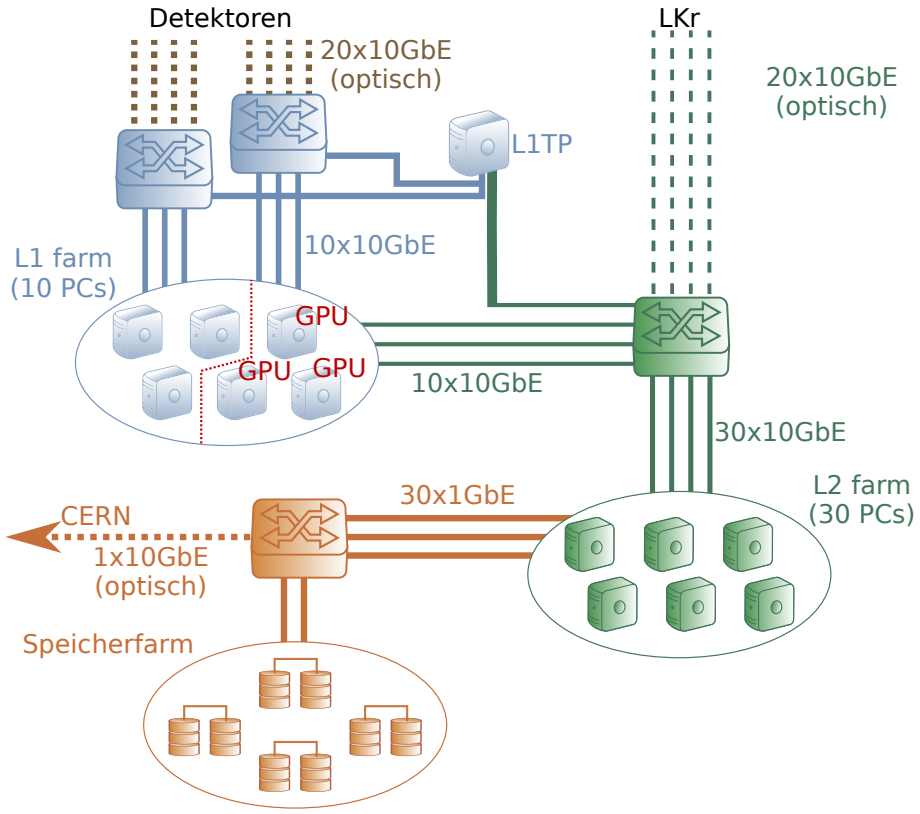


**Abbildung 3.1.:** Das ursprüngliche Farm-Konzept.

Dabei ist zu betonen, dass auf Grund der enormen Datenmengen des LKr-Kalorimeters dessen Daten erst nach einer positiven L1-Trigger-Entscheidung an die L2-Farm gesendet werden. Um die Speichermenge zu reduzieren, erhält die Ausleseelektronik des LKr-Kalorimeters (CREAM-Boards) ebenfalls das Trigger-Signal des L0TP und verwirft Ereignisse im Falle einer negativen L0-Trigger-Entscheidung. Eine weitere Reduzierung der benötigten Speichergröße innerhalb der CREAM-Boards wird durch eine zeitliche Restriktion für die Prozessierung des L1-Triggers erreicht. Dabei soll vom positiven L0-Trigger bis zur Verteilung des L1-Triggers nicht mehr als eine Sekunde vergehen.

Bei diesem dreistufigen System sinkt von Stufe zu Stufe die Datenrate und gleichzeitig auch die zeitlichen Restriktionen für die Trigger-Prozesse. Letzteres wurde bereits für L0 und L1 mit 1 ms bzw. 1 s angegeben (siehe Abschnitt 1.4). Für die L2-Prozesse steht

die Zeit eines ganzen Strahlzyklus (siehe Abschnitt 1.2.3), also rund 15 bis 20 Sekunden, zur Verfügung.

Die Prozessierungen in den Trigger-Stufen L0 und L1 fangen in guter Näherung[1] gleichzeitig mit dem Burst an, und L0 endet unmittelbar nach Ende des Bursts. Da die Dauer für die Generierung einer L1-Trigger-Entscheidung auf eine Sekunde limitiert ist, endet die L1-Prozessierung spätestens eine Sekunde nach Ende des Bursts. Außerdem beginnt im schlechtesten Fall die Prozessierung der L2-Trigger-Entscheidung erst eine Sekunde nach Beginn des Bursts. Die L2-Farm muss dabei immer vor Beginn des darauf folgenden Bursts mit der Verarbeitung abschließen. Diese Zeitverteilung wird in Abbildung 3.2 dargestellt.

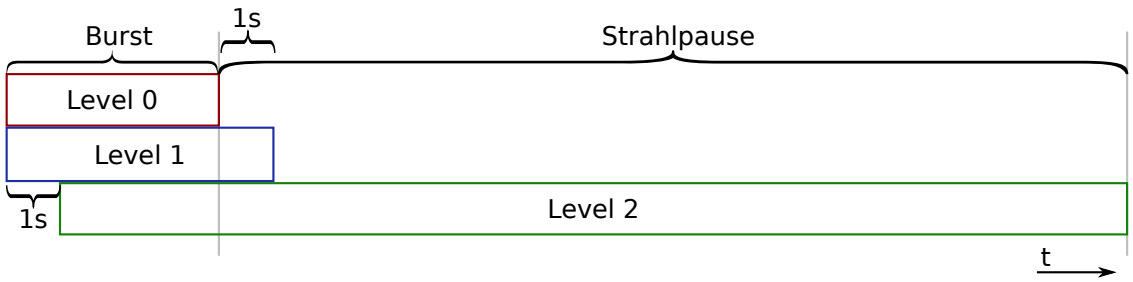


**Abbildung 3.2.:** Aufteilung der Prozessierzeiten der verschiedenen Trigger-Stufen bei getrennten L1 und L2 Farmen.

In Tabelle 1.2 wurde bereits die Größe eines gesamten Ereignisses mit $S_{\text{Ereignis}} = 227$ kB angegeben. Bei der mittleren Ereignisrate von 10 MHz muss die Ausleseelektronik aller Detektoren insgesamt eine Datenrate von $S_{\text{Ereignis}} \cdot 10$ MHz $\approx 2,3$ TBps auslesen. Die Ereignisrate wird durch die L0-Trigger auf 1 MHz reduziert. Während des Bursts werden mit dieser Frequenz die Daten aller Detektoren, ausschließlich des LKr-Kalorimeters, an die L1-Farm gesendet. Die Datenmenge dieser Detektoren beträgt nur $S_{\text{L0}} \approx 5$ kB pro Ereignis (siehe Tabelle 1.2) und somit ergibt sich eine Datenrate $D_{\text{L0}\to\text{L1}}$ von L0 zu L1:

$$D_{\text{L0}\to\text{L1}} \;=\; S_{\text{L0}} \cdot 1 \text{ MHz} \;\approx\; 5,0 \text{ GBps}.$$

Für jedes Ereignis muss innerhalb einer Sekunde die L1-Entscheidung verteilt werden. Die CREAM-Boards können mit dieser Information die Daten zu einem Ereignis mit einer negativen L1-Trigger-Entscheidung verwerfen. Umgekehrt werden die Daten des LKr, bei positiven L1-Trigger-Entscheidungen (10%) an die L2-Farm gesendet. Da die L2-Prozessierung während eines ganzen Burst-Zyklus stattfinden kann (siehe Abbildung 3.2), müssen die LKr-Daten nicht unmittelbar nach einem positiven L1-Trigger gesendet

[1] Wie bereits erläutert, werden die Daten 1 ms nach einem L0-Trigger-Signal an die L1-Farm geschickt. Im Verhältnis zur Dauer eines Bursts ($\mathcal{O}(5$ s)) ist diese Zeit vernachlässigbar klein.

werden. Stattdessen kann die Übertragung über die Zeit eines Burst-Zyklus verteilt stattfinden[2], jedoch erst dann beginnen, nachdem das erste Ereignis von L1 prozessiert wurde, eine Sekunde nach Beginn des Bursts. Somit ergeben sich die Ereignisrate $F_{\mathrm{LKr}\to\mathrm{L2}}$ zwischen den CREAM-Boards und der L2-Farm sowie die resultierende Datenrate $D_{\mathrm{LKr}\to\mathrm{L2}}$ zu:

$$F_{\mathrm{LKr}\to\mathrm{L2}} = 1\ \mathrm{MHz} \cdot 10\% \cdot \frac{T_{\mathrm{Burst}}}{T_{\mathrm{Zyklus}} - 1} \approx 32\ \mathrm{kHz}, \tag{3.1}$$

$$\Rightarrow D_{\mathrm{LKr}\to\mathrm{L2}} = S_{\mathrm{LKr}} \cdot F_{\mathrm{LKr}\to\mathrm{L2}} \approx 7,1\ \mathrm{GBps}. \tag{3.2}$$

Dabei beträgt die Datenmenge $S_{\mathrm{LKr}} \approx 222$ kB pro Ereignis (siehe Tabelle 1.2), und der Duty-Cycle $\eta$ wurde auf 0,3 bei einer Burst-Dauer von 5 Sekunden geschätzt (siehe Abschnitt 1.2.3)[3]. Mit der gleichen Ereignisrate werden die Daten von L1 zu L2 übertragen, woraus sich die Datenrate $D_{L1\to L2}$ zwischen L1 und L2 ergibt:

$$\Rightarrow D_{\mathrm{L0}\to\mathrm{L1}} = S_{L0} \cdot F_{\mathrm{LKr}\to\mathrm{L2}} \approx 160\ \mathrm{MBps}. \tag{3.3}$$

Somit erhält man den in Abbildung 3.3 dargestellten Datenfluss.

## 3.2. Verbesserungsvorschläge

Im Laufe dieser Arbeit hat sich herausgestellt, dass vor allem den folgenden Punkten in Hinsicht auf die Entwicklung der PC-Farm besondere Beachtung geschenkt werden muss:

- Skalierbarkeit des Netzwerkes.
- Skalierbarkeit der Rechenleistung.
- Latenz zwischen L0 und L1.
- Latenz zwischen L1 und L2.
- Handhabbarkeit des Netzwerkes.

[2] Abhängig von der Dauer zur Prozessierung eines Ereignisses in L2 muss das letzte Ereignis eines Bursts kurz vor dem darauf folgenden Burst versendet werden. Diese Zeit soll jedoch zunächst vernachlässigt werden.

[3] In Abschnitt 1.2.3 wurde angegeben, dass ein Duty-Cycle von 0,25 realistisch ist, dieser Wert jedoch variieren wird. Größere Werte führen zu einer höheren Last innerhalb der PC-Farm. Daher wurde in dieser Rechnung der Duty-Cycle als Puffer um 20% höher angenommen.

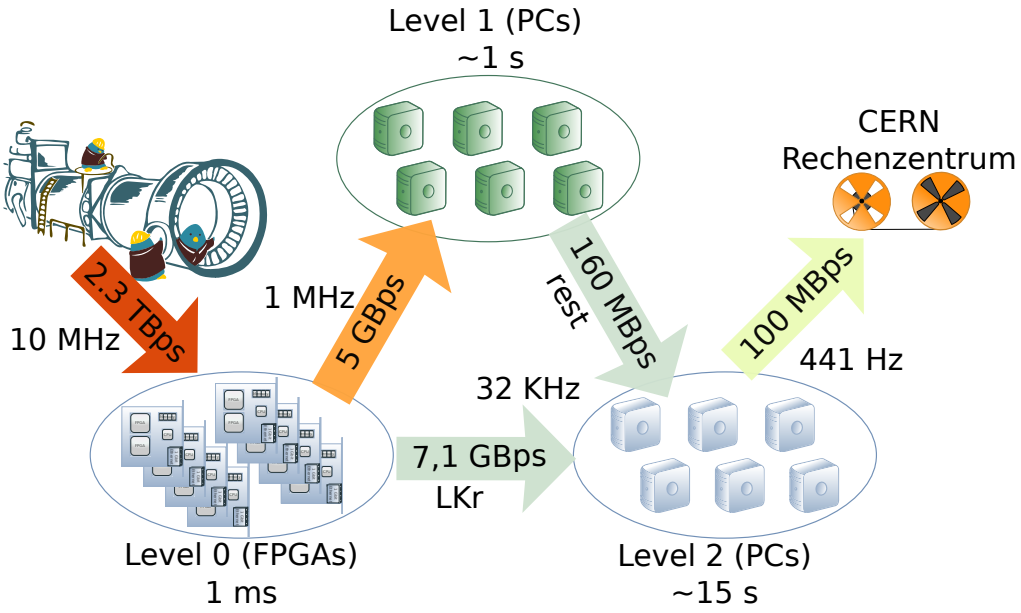


**Abbildung 3.3.:** Logischer Datenfluss vom Detektor bis zum persistenten Speicher. Dabei wurde der Duty-Cycle $\eta = 0,3$ und $T_{Burst} = 5$ s angenommen.

- Komplexität der benötigten Software.
- Preis.

In dieser Hinsicht konnte im Rahmen dieser Arbeit eine deutliche Optimierung der PC-Farm realisiert werden. Darauf wird in den folgenden Abschnitten eingegangen, wobei als Ansatz folgende Punkte dienen:

**Die L1-Farm erhält eine höhere Skalierbarkeit:** Mit dem Einsatz von detektorspezifischen PCs muss die Rechenkapazität der L1-Farm für jeden Detektor getrennt angepasst werden. Daher soll nur noch eine einzige, homogene Farm für den L1-Trigger verantwortlich sein, so dass jeder Detektor seine Daten an einen beliebigen PC dieser Farm schicken kann.

**Die L1- und L2-Farm werden nicht mehr physisch voneinander getrennt:** Im Hinblick auf den Duty-Cycle aus Abschnitt 1.2.3 soll durch eine Vereinigung beider Farmen eine Kostenersparnis bei gleichzeitiger Effizienzsteigerung erreicht werden.

### 3.2.1. Skalierbarkeit der L1-Farm

Um die Skalierbarkeit der L1-Farm zu erhöhen, ist es sinnvoll, sich vom Konzept mit detektorspezifischen PCs zu lösen. Stattdessen sollte man eine homogene Farm einsetzen.

Das heißt, dass jeder L1-Rechner in der Lage sein muss, die Daten von allen Detektoren zu empfangen und zu verarbeiten. Zur Lastverteilung würde dann jeder Detektor seine Daten eines Bursts homogen an alle PCs der L1-Farm verteilen (z.B. ereignisweise nach dem Round-Robin-Schema).

Auf diese Weise kann man mit dem Kauf eines weiteren PCs nicht mehr nur die Rechenkapazität für einzelne Detektoren erhöhen; stattdessen verteilt sich die zusätzliche Kapazität über alle Detektoren. Zudem kann mit diesem Ansatz eine zentrale Softwarearchitektur verwendet werden. Das entstehende System wird dadurch deutlich übersichtlicher und strukturierter, da einzelne Programmteile nicht mehr für jeden Detektor getrennt entwickelt, sondern von den einzelnen Arbeitsgruppen gemeinsam genutzt werden können.

Bei einer homogenen L1-Farm muss jedoch jeder Rechner mit einer Grafikkarte bestückt werden, und nicht nur die zum RICH-Detektor gehörigen Rechner. Da sich die Last durch die RICH-Daten jedoch homogen über die Farm verteilt, benötigt jeder L1-Rechner nur eine kleine GPU-Leistung, also günstigere Grafikkarten. Mit Detektor-spezifischen PCs würden nur wenige PCs mit sehr teuren Grafikkarten eingesetzt werden, so dass sich die Kosten für die Grafikkarten bei beiden Varianten nicht stark unterscheiden. Da zudem die Grafikkarten, verglichen zu den Kosten für die PCs (ca. 3.000-4.000 EUR pro PC), relativ günstig sind (weniger als 1.000 EUR pro Grafikkarte), können eventuelle zusätzliche Kosten durch eine homogene Farm vernachlässigt werden.

### 3.2.2. Fusion der L1- und L2-Farmen

Bei der in Abbildung 3.2 gezeigten Zeitverteilung der einzelnen Trigger-Stufen ist folgendes auffällig: Die L0-Boards und L1-Rechner werden nur während des Bursts benötigt, bleiben aber innerhalb der weit längeren Pause unbenutzt. Daher wurde ein Konzept entwickelt, mit dem die Ressourcen effizienter genutzt werden. Dabei trennt man die L1- und L2-Rechner nicht mehr physisch, sondern nur noch logisch voneinander. Es entsteht also eine vereinigte L1-L2-Farm auf der sowohl die L1-Software, als auch die Programme zur L2-Prozessierung inklusive Eventbuilding ausgeführt werden[4]. Dadurch ergibt sich die in Abbildung 3.4 dargestellte Zeitverteilung.

Der Datenfluss der einzelnen Ereignisse findet bei einer vereinigten L1-L2-Farm weiterhin sequentiell von L0 über L1 zu L2 statt. Zudem könnten theoretisch weiterhin detektorspezifische L1-Rechner eingesetzt werden.

[4]Prinzipiell könnte man auch die TEL62 Boards des L0-Triggers in diese Farm integrieren. Da für L1 und L2 jedoch Software verwendet wird, die nicht problemlos auf FPGAs übertragen werden kann, müsste man CPUs auf den FPGAs simulieren und ein minimales Betriebssystem auf den TEL62-Board-FPGAs laufen lassen. In den FPGAs ist jedoch nur Platz für eine Anwendung, also entweder L0-Firmware oder CPU-Simulation. Das Wechseln zwischen diesen beiden Firmware-Versionen würde jedoch zu lange dauern, um die TEL62-Boards tatsächlich für die L2-Prozessierung zu verwenden.

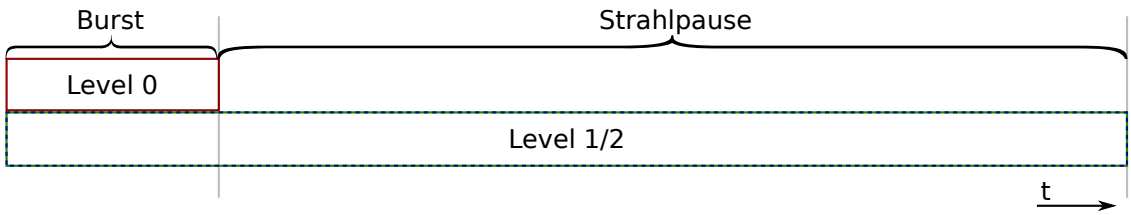


**Abbildung 3.4.:** Aufteilung der Prozessierzeiten der verschiedenen Trigger-Stufen mit einer vereinigten L1-L2-Farm.

Zum Eventbuilding müssen die auf den L1-L2-Rechnern verteilten Daten weiterhin nach der L1-Prozessierung in einem Rechner gesammelt werden. Hierbei fungiert jeder einzelne Rechner also erst als L1-Prozessor. Im Falle einer positiven L1-Entscheidung würden die Rechner anschließend entweder ihre Daten an den Eventbuilder-PC (was selbst ein L1-L2-Farm-Rechner ist) senden, oder selbst als Eventbuilder-PCs fungieren. Die Entscheidung, welche Ereignisse auf welchen Rechnern zusammengefügt werden sollen, wird durch den L1TP koordiniert. Dabei entsteht der in Figur 3.5 gezeigte Datenfluss.

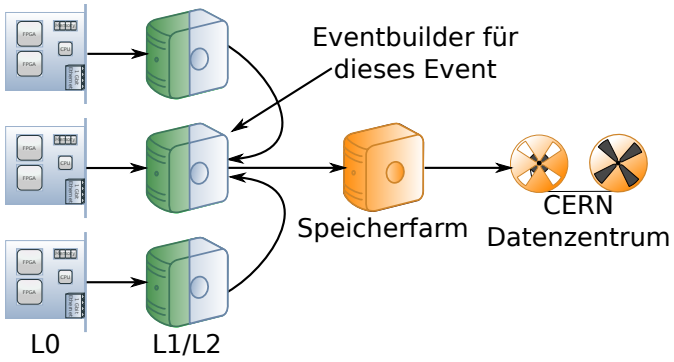


**Abbildung 3.5.:** Physischer Datenfluss bei einer kombinierten L1-L2-Farm. Nach der L1-Prozessierung müssen die Daten zu einem Event an den dazugehörigen Eventbuilder-PC übertragen werden.

Die ersten Ereignisse eines Bursts werden die Kette durch L0 und L1 schon durchlaufen haben, bevor der Burst endet. In dieser Situation wird mit Abbildung 3.5 eine Problematik deutlich: Noch während Daten von der Ausleseelektronik an einen Rechner zur L1-Prozessierung gesendet werden, müssen gleichzeitig Daten von verschiedenen PCs zu einem L2-Prozess auf einem der Rechner zum Eventbuilding gesendet werden. Es überlagern sich somit zwei Datenströme zeitlich, mit Datenraten von ca. 5 GBps und 7 GBps (siehe Abbildung 3.3). Innerhalb der Topologie aus Abbildung 3.1 werden diese Daten über getrennte Netzwerke transportiert. Auch bei einer fusionierten Farm sind diese un-

abhängigen Netzwerke denkbar. Die Problematik durch die simultane Übertragung der beiden Datenströme entsteht also nicht im Netzwerk, sondern auf Seiten der Rechner. Man muss also sicherstellen, dass die einzelnen PCs genügend Kapazitäten frei haben, um weiterhin Daten von den TEL62-Boards zu empfangen, also nicht durch Daten von L1-Prozessen zum Eventbuilding überlastet werden. Dazu sind prinzipiell zwei Konzepte denkbar:

**Dynamische Flusssteuerung:** Sobald ein PC zu sehr mit der L1-Prozessierung und Datenaufnahme weiterer Daten von der L0-Elektronik ausgelastet ist, wird dieser Rechner für eine gewisse Zeit nicht mehr für L2-Rechnungen und somit als Eventbuilder verwendet. Sollte kein Rechner genügend Kapazitäten für das Eventbuilding frei haben, so müssen die Daten nach der L1-Prozessierung gehalten werden bis wieder ein Computer frei wird. Im Zweifel kann dieser Zustand bis zum Ende der L1-Prozessierung, also etwa eine Sekunde nach Ende des Bursts, anhalten.

Zur Implementierung einer dynamischen Flusssteuerung muss ein Protokoll entwickelt werden, um die Last-Informationen der einzelnen Rechner zu verteilen. Der Algorithmus zur Bestimmung, wann ein Rechner entlastet werden muss, ist dabei ein kritisches Element der Farm, da Fehler hierbei zum Verlust von Ereignissen führen können.

**Statische Flusssteuerung:** Geht man davon aus, dass die benötigte Rechenkapazität für die L1-Prozessierung in etwa jener für die L2-Prozessierung entspricht, so würde sich der Aufwand für eine dynamische Flusssteuerung nicht lohnen. Solange die Farm nicht deutlich mehr Kapazitäten zur Verfügung stellt, als benötigt, können in diesem Fall L1 und L2 nicht gleichzeitig berechnet werden. Die dynamische Flusssteuerung führt in diesem Fall dazu, dass die L1-L2-Kommunikation zum Erliegen kommt, bis alle L1-Daten prozessiert sind. Somit würde es sich hier empfehlen, grundsätzlich mit dem Eventbuilding erst dann zu beginnen, wenn keine Daten mehr von der Ausleseelektronik gesendet werden, also unmittelbar nach dem Ende des Bursts.

### 3.2.3. Stern-Topologie

Durch die statische Flusssteuerung erhöht sich die Datenrate für die Kommunikation zwischen LKr und L2, da die gleiche Datenmenge so in kürzerer Zeit übertragen werden muss. Analog zu Gleichungen 3.1 und 3.2 ergeben sich nun, bei gleichen Annahmen, die Ereignisrate $F^{\text{stat}}_{\text{LKr}\to\text{L2}}$ und die Datenrate $D^{\text{stat}}_{\text{LKr}\to\text{L2}}$ für diese Kommunikation, sowie die Datenrate $D^{\text{stat}}_{\text{L1}\to\text{L2}}$ für die Kommunikation zwischen L1- und L2 zu:

$$F^{\text{stat}}_{\text{LKr}\to\text{L2}} = 1\ \text{MHz} \cdot 10\% \cdot \frac{T_{\text{Burst}}}{T_{\text{Zyklus}} - T_{\text{Burst}}} \approx 43\ \text{kHz}, \tag{3.4}$$

$$\Rightarrow D^{\text{stat}}_{\text{LKr}\to\text{L2}} = S_{\text{LKr}} \cdot F^{\text{stat}}_{\text{LKr}\to\text{L2}} \approx 9,5\ \text{GBps}, \tag{3.5}$$

$$\Rightarrow D^{\text{stat}}_{\text{L1}\to\text{L2}} = S_{\text{L0}} \cdot F^{\text{stat}}_{\text{LKr}\to\text{L2}} \approx 0.2\ \text{GBps}. \tag{3.6}$$

Somit werden zunächst während des Bursts ca. 5,0 GBps Daten für die L1-Prozessierung übertragen. Anschließend, mit Ende des Bursts, beginnt die Übertragung der Daten von L1 und LKr an die L2-Prozesse (L1/LKr-L2-Kommunikation) mit einer Datenrate von insgesamt rund 9,7 GBps bis zum darauf folgenden Burst.

Für den Fall, dass genügend PCs vorhanden sind, würden sich bei einer dynamischen Flusssteuerung die Datenraten wie im ursprünglichen Fall verhalten (siehe Abbildung 3.3). Es würden sich also ca. eine Sekunde nach Beginn, bis Ende des Bursts, eine Datenrate von ca. 5,0 GBps für die L0-L1-Kommunikation und eine Rate von rund 7,1 GBps für die L1/LKr-L2-Kommunikation zeitlich überlagern. In der Summe entspricht dies einer maximalen Rate von 12,1 GBps während eines Strahlzyklus.

Da die kombinierte PC-Farm aus voraussichtlich mindestens 32 PCs bestehen wird (siehe Abschnitt 3.2.4) und sich die oben berechneten Datenraten auf alle PCs gleichmäßig verteilen, können für beide Flusssteuerungen die maximalen Datenraten problemlos mit einer 10GbE-Verbindung pro PC realisiert werden. Es ist also nicht nötig, wie im ursprünglichen Konzept vorgesehen, zwei getrennte Netzwerke für die L0-L1 und L1/LKr-L2-Kommunikation zu verwenden. Es bietet sich somit an, nur ein einziges 10GbE-Netzwerk zu verwenden. Dieses Netzwerk würde sowohl für den Datentransfer von L0 nach L1, als auch für die Übertragung der Daten von L1 und den CREAMs zu den jeweiligen Eventbuilder-Rechnern dienen, und dabei mit weniger Kanten im Netzwerk auskommen. Die so entstandene Sterntopologie ist in Abbildung 3.6 gezeigt.

Als zentrales Kopplungselement kann hier ein Switch oder ein Router eingesetzt werden. Da die TEL62- und CREAM-Boards mit voraussichtlich 40 optischen 10GbE-Kanten verbunden werden, muss dieses Kopplungselement mindestens 72 Ethernet-Ports besitzen. In dieser Größenordnung stehen nur wenige Switche und Router auf dem Markt zur Verfügung. Daher kann es sich eventuell rentieren, mehrere kleine Kopplungselemente zu einer Baum-Topologie zu verbinden und die PCs und Elektronik-Boards an die „Blätter“ dieses Baumes zu koppeln (siehe Abbildung 3.7). In dieser Topologie entstehen Flaschenhälse[5] zwischen den einzelnen Kopplungselementen. Für diesen Zweck kann die durch IEEE 802.1AX definierte Link Aggregation verwendet werden. Dabei verbindet

[5] Damit werden die langsamsten Ressourcen innerhalb einer Netzwerkkommunikation bezeichnet. Diese definieren die maximal erreichbare Datenrate.

man zwei Kopplungselemente mit mehreren 10GbE-Kanten. Durch eine spezielle Konfiguration der Switche oder Router werden diese Verbindungen als eine einzige (virtuelle) Kante mit entsprechend vielfacher Datenrate verwendet.

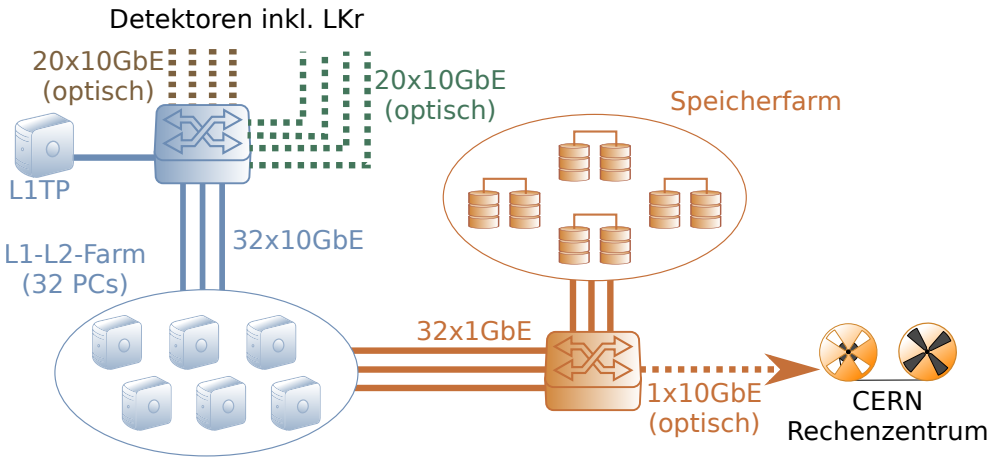


**Abbildung 3.6.:** Übersicht des Netzwerkes im Falle der gemeinsamen L1-L2-Farm und einer einzigen 10GbE-Stern-Topologie.

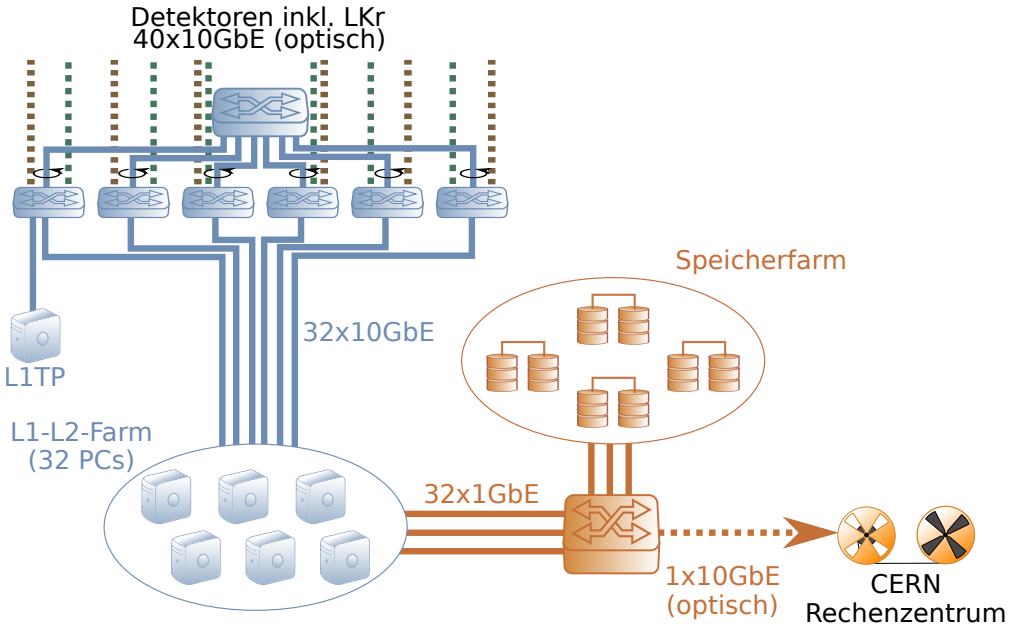


**Abbildung 3.7.:** Übersicht des Netzwerkes im Falle der kombinierten L1-L2-Farm und einer Baum-Topologie. Die schwarzen Pfeile an den Kanten zwischen den Kopplungselementen markieren durch Link Aggregation verbundene 10GbE-Verbindungen.

### 3.2.4. Einsparungen

Seien $N_{L1}$ und $N_{L2}$ die minimalen Anzahlen benötigter PCs für die Farmen L1 und L2 im ursprünglichen Ansatz. Dann kann für die statische Flusssteuerung die Anzahl benötigter PCs $N^{stat}_{L1L2}$ im Falle der kombinierten Farm folgendermaßen abgeschätzt werden[6]:

$$N_{\mathrm{L2}} \cdot (T_{\mathrm{Zyklus}} - 1s) \approx N^{\mathrm{stat}}_{\mathrm{L1L2}} \cdot (T_{\mathrm{Zyklus}} - T_{\mathrm{Burst}})\,, \tag{3.7}$$

$$\Rightarrow N^{\mathrm{stat}}_{\mathrm{L1L2}} \approx N_{\mathrm{L2}} \cdot \left( \frac{T_{\mathrm{Zyklus}} - 1\ \mathrm{s}}{T_{\mathrm{Zyklus}} \cdot (1-\eta)} \right), \tag{3.8}$$

$$\Rightarrow N^{\mathrm{stat}}_{\mathrm{L1L2}} \approx N_{\mathrm{L2}} \cdot \frac{1}{1-\eta} \left( 1 - \frac{\eta}{T_{\mathrm{Burst}}[\mathrm{s}]} \right). \tag{3.9}$$

Geht man von $N_{\mathrm{L2}} = 30$ aus, so müsste die kombinierte Farm bei $\eta = 0,3$ und $T_{\mathrm{Burst}} = 5$ s bereits aus mindestens 41 Rechnern bestehen. Da bisher ca. zehn Rechner für die

[6]Der Ansatz hierbei ist, dass die Rechenleistung der Farmen proportional zu der Anzahl der Rechner ist. Gleichung 3.7 ist dabei äquivalent zum Energieerhaltungssatz.

eigenständige L1-Farm vorgesehen waren, also insgesamt $N_{\mathrm{L1}} + N_{\mathrm{L2}} = 40$ Rechner für beide Farmen zusammen, würde man somit für die vereinigte Farm mehr PCs benötigen als im separierten Fall. Grund dafür ist, dass während des Bursts zwar alle Rechner für die L1-Prozessierung verwendet werden, diese dann jedoch nicht voll ausgelastet sind. Erst während der Strahlpause zwischen zwei Bursts wird die Farm effizient durch die L2-Prozesse genutzt. Statische Flusssteuerung lohnt sich also nur, wenn $N_{\mathrm{L1}} \approx N_{\mathrm{L2}}$. Dies kann z.B. dadurch erreicht werden, dass Teile der L2-Algorithmen auf die L1-Prozesse ausgelagert werden. Dies ist jedoch mit einem hohen Aufwand verbunden.

Geht man andererseits von einer dynamischen Flusssteuerung aus, so kann die dabei benötigte Anzahl $N_{\mathrm{L1L2}}^{\mathrm{dyn}}$ an PCs für die kombinierte Farm folgendermaßen abgeschätzt werden:

$$N_{\mathrm{L2}} \cdot (T_{\mathrm{Zyklus}} - 1\mathrm{s}) \approx \left(N_{\mathrm{L1L2}}^{\mathrm{dyn}} - N_{\mathrm{L1}}\right) \cdot T_{\mathrm{Burst}} + N_{\mathrm{L1L2}}^{\mathrm{dyn}} \cdot (T_{\mathrm{Zyklus}} - T_{\mathrm{Burst}})\,, \tag{3.10}$$

$$= N_{\mathrm{L1L2}}^{\mathrm{dyn}} T_{\mathrm{Zyklus}} - N_{\mathrm{L1}} T_{\mathrm{Burst}}, \tag{3.11}$$

$$\Rightarrow N_{\mathrm{L1L2}}^{\mathrm{dyn}} \approx N_{\mathrm{L2}} \cdot \left(1 - \frac{1\ \mathrm{s}}{T_{\mathrm{Zyklus}}}\right) + N_{\mathrm{L1}} \frac{T_{\mathrm{Burst}}}{T_{\mathrm{Zyklus}}}, \tag{3.12}$$

$$= N_{\mathrm{L2}} \cdot \left(1 - \frac{\eta}{T_{\mathrm{Burst}}[\mathrm{s}]}\right) + N_{\mathrm{L1}} \cdot \eta. \tag{3.13}$$

Nimmt man wieder die Werte $N_{\mathrm{L1}} = 10$, $N_{\mathrm{L2}} = 30$, $\eta = 0,3$ und $T_{\mathrm{Burst}} = 5$ s an, so ergibt sich $N_{\mathrm{L1L2}}^{\mathrm{dyn}} = 31,2$. Mit einer dynamischen Flusssteuerung kann man somit im Vergleich zum ursprünglichen Konzept ca. 8 PCs einsparen, da in diesem Fall alle Rechner während des kompletten Strahlzyklus gleichmäßig ausgelastet werden.

Für die ersten Testläufe des Experiments im Sommer 2012 sollen PCs für je rund 3.700 EUR eingekauft werden[7].

Somit werden durch die Aggregation der beiden Farmen, in Verbindung mit einer dynamischen Flusssteuerung, PCs im Wert von fast 30.000 EUR eingespart. Dies entspricht ca. 20% der urprünglichen Kosten für die Farm-PCs.

Beim Konzept mit separierten L1- und L2-Farmen hätte man insgesamt 50 Ports (also 50 Kanten zwischen PC und einem der 10GbE-Switches) für die PCs der beiden Farmen

[7] Hierbei handelt es sich um vier Dell PowerEdge R610 mit zwei X5675 Prozessoren und 24 GB RAM (1333 MHz) im Wert von je 3.741,36 EUR inkl. MwSt. Dieses Angebot wurde über spezielle CERN-Konditionen direkt beim Hersteller eingeholt.

benötigt (siehe Abbildung 3.1). Durch die Verwendung einer einfachen Stern-Topologie wird jeder Farm-Rechner nur noch mit einem 10GbE-Switch verbunden. Bei einer dynamischen Flusssteuerung innerhalb einer kombinierten L1-L2-Farm sind nur noch 32 PCs notwendig und entsprechend auch nur noch 32 der 10GbE-Ports des zentralen Switches im Stern (siehe Abbildung 3.6). Man spart somit durch die neue Topologie 18 Ports, und es muss nur noch ein Netzwerk konfiguriert und gepflegt werden. Für die Testläufe im Sommer 2012 wurde ein Switch mit 24 Ports im Wert von über 27.000 EUR gekauft[8] (1145 EUR pro Port).

Somit wird durch die Aggregation der beiden Farmen, in Verbindung mit einer dynamischen Flusssteuerung und einer Stern-Topologie, Netzwerk-Komponenten im Wert von etwa 20.000 EUR eingespart. Dies entspricht ca. 36% der Kosten für das Netzwerk-Zubehör bei separierten L1- und L2-Farmen.

### 3.2.5. L1-Eventbuilding

Auf Basis der in Abschnitt 3.2.1 und 3.2.2 vorgestellten Konzepte, also einer homogenen, kombinierten L1-L2-PC-Farm, konnte eine weitere Vereinfachung realisiert werden.

Der Ansatz für dieses weiterentwickelte Konzept besteht darin, von L1 an alle Daten eines Ereignisses auf einem einzigen Rechner zu sammeln.

Das bedeutet, dass nach einem positiven L0-Trigger alle Detektoren ihre Daten zu einem Ereignis an den gleichen PC schicken. Somit sind L1-Prozesse, Eventbuilder und L2-Prozesse für ein Ereignis nicht mehr nur in der gleichen Farm, sondern auf dem gleichen Rechner lokalisiert. Auf diese Weise wird die Anzahl an Kommunikationen über das Netzwerk verringert. Ein L1TP wird nicht mehr benötigt, da die L1-Entscheidung nur noch innerhalb eines PCs verteilt werden muss. Außerdem wird die Implementierung der effizienteren dynamischen Flusssteuerung deutlich einfacher (siehe Abschnitt 4.7). Ein weiterer Zugewinn ist, dass die Daten nach einer positiven L1-Trigger-Entscheidung nicht mehr durch das Netzwerk an den Eventbuilder transportiert werden müssen, sondern im Speicher des L1-Rechners verweilen können. Durch das Wegfallen dieser Kommunikation wird das Netzwerk weiter entlastet[9] und es sinkt die Latenz zwischen L1 und L2. Vor

[8] Dabei handelt es sich um einen modularen Switch von HP (J9641A) mit drei achtfachen SFP+-Modulen (J9538A) im Wert von insgesamt 27.489 EUR inkl. MwSt. Dieser wurde über das CERN direkt erworben.

[9] Da nach einem positiven L1-Trigger weiterhin die Daten des LKr angefordert werden müssen, spart man so allerdings nur $\eta \cdot 0,5$ GBps.

allem aber fällt die aufwändigere Implementierung der Netzwerkkommunikationen weg, da der Datenaustausch innerhalb eines Rechners deutlich einfacher programmiert werden kann und gleichzeitig weniger fehleranfällig ist.

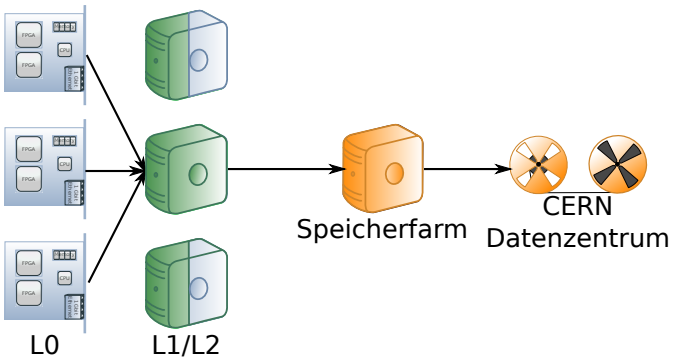


**Abbildung 3.8.:** Physischer Datenfluss bei kombinierter L1-L2-Farm mit L1-Eventbuilding.

Der Datenfluss für dieses neue Konzept wird in Abbildung 3.8, analog zu Abbildung 3.5, dargestellt. Dabei entsteht eine absolut homogene PC-Farm, da nun jeder PC die Daten von allen Detektoren verarbeiten können muss. Es ist also nicht mehr möglich, die Farm in PCs mit und ohne Grafikkarten aufzuteilen. Diese Problematik wurde bereits in Abschnitt 3.2.1 angesprochen. Jetzt müssen sich die GPU-Ressourcen auf noch mehr PCs verteilen, da die ursprüngliche L2-Farm auf mindestens 30 PCs geschätzt wurde, die reine L1-Farm aber nur auf rund 10 PCs. Da die Preise der benötigten Grafikkarten, verglichen zu den Kosten für die einzelnen PCs, sehr gering ausfallen, ist man sich innerhalb der Kollaboration einig, dass die Vorteile der homogenen Farm deutlich überwiegen. Daher wurde diese Variante für das endgültige Design der Farm gewählt.

Bei der Implementierung der Farm-Software wurde zusätzlich klar, dass durch die hier genannte Herangehensweise, fast ohne Performanceverlust (siehe Abschnitt 4.6), bereits vor der L1-Prozessierung ein Eventbuilding stattfinden kann (L1-Eventbuilding). Dabei würde man die Daten aller Detektoren, abgesehen von den LKr-Daten, sammeln und gebündelt verarbeiten. Das bedeutet, dass man bei den L1-Algorithmen nicht mehr nur jeden Detektor für sich separat analysieren muss. Stattdessen können schon auf dieser Stufe Zusammenhänge zwischen verschiedenen Detektoren untersucht werden. Das in Kapitel 4 dargestellte Framework, wie es innerhalb dieser Arbeit entwickelt wurde, ermöglicht dieses L1-Eventbuilding. Ob davon später tatsächlich Gebrauch gemacht wird, ist abhängig von der Geschwindigkeit der endgültigen L1-Algorithmen und ob diese die zeitlichen Restriktionen einhalten können.

# Kapitel 4.

# Implementierung des Frameworks

Neben der Planung einer geeigneten Topologie wurde das für die Realisierung der L1- und L2-Trigger-Algorithmen notwendige Framework entwickelt und in der Programmiersprache C++ implementiert. Dabei handelt es sich um ein Programm, dass den Datenstrom für den Benutzer unsichtbar verwaltet. Als Benutzer gelten dabei diejenigen Personen, die die endgültigen Trigger-Algorithmen implementieren. Für diesen Zweck stellt das Framework eine Schnittstelle dar, die diese Implementierung ermöglicht, ohne dabei Kenntnisse über den Datenfluss, die verwendeten Protokolle oder die Topologie haben zu müssen (Transparenz[1]). Das Framework in Kombination mit den Trigger-Algorithmen wird im folgenden Text als Farm-Software bezeichnet.

## 4.1. Das Konzept

Als Grundlage für die Konzipierung gilt das in Abschnitt 3.2.5 beschriebene L1-Eventbuilding, das ohne L1TP und ohne Kommunikation zwischen den Farm-PCs auskommt. Jeder PC stellt somit ein eigenständiges System dar, welches nur mit den verschiedenen Elektronik-Boards (inklusive L0TP) und dem persistenten Speicher kommuniziert. Das Framework besteht daher nur aus einem einzigen Programm, welches auf allen Farm-PCs ausgeführt wird. Für ein einzelnes Ereignis läuft die Verarbeitung durch das Framework innerhalb eines PCs folgendermaßen ab:

1. Empfang und Integritätsprüfung der L0-Daten (Ausleseelektronik der Detektoren exklusive LKr, sowie L0TP).
2. Durchführung des L1-Eventbuilding.
3. Ausführung der L1-Trigger-Algorithmen.

[1] Transparente Programmteile beschreiben in der Informatik jene Programmabschnitte, deren Existenz für den Benutzer nicht direkt erkennbar sind. Das Wort transparent ist hier somit im Sinne von (beinahe) unsichtbar zu verstehen.

4. Verteilung der L1-Entscheidung an die CREAM-Boards (auch im negativen Fall, damit die Daten gelöscht werden können).
5. Empfang und Integritätsprüfung der LKr-Daten.
6. Durchführung des Eventbuilding.
7. Ausführung der L2-Trigger-Algorithmen.
8. Senden der fertig prozessierten Daten an den persistenten Speicher.

Sollte dabei eine negative L1- oder L2-Entscheidung erhalten werden, so bricht der Datenfluss nach Punkt 4 bzw. nach Punkt 7 ab.

## 4.2. Die Protokolle

Zur Wiederverwendung der TELL1-Firmware für die TEL62-Boards wurden auf Basis der innerhalb des LHCb-Experiments verwendeten Protokolle [19], die Protokolle für die Kommunikation zwischen den Elektronik-Boards und der PC-Farm entwickelt. Diese sind im OSI-Referenzmodell der Darstellungs- und Anwendungsschicht zuzuordnen und werden im folgenden Text als Farm-Protokolle bezeichnet. Innerhalb der Darstellungsschicht ist kein spezielles Protokoll definiert. Stattdessen wurde die Konvention eingeführt, dass alle Datenworte innerhalb der Protokolle im Little-Endian-Format übertragen werden[2].

Die Farm-Protokolle setzen in den untersten drei OSI-Schichten auf Ethernet und IP auf, um weit verbreitete Ethernet-Switche und -Router verwenden zu können (siehe Abschnitt 2.3). Für die Transport- und Sitzungsschicht wird UDP verwendet. Da TCP ein sehr komplexes Protokoll ist, kann dies nicht innerhalb der FPGAs der TEL62-Boards implementiert werden. Innerhalb der CREAM-Boards wäre dies zwar möglich, man entschied sich jedoch für eine einheitliche Methode, so dass auch für die Kommunikation zwischen den CREAM-Boards und der PC-Farm UDP eingesetzt wird.

### 4.2.1. Die Eventnummer

Der L0TP definiert für jedes Ereignis mit einer positiven L0-Trigger-Entscheidung eine fortlaufende Eventnummer. Über diese Nummer können die Daten der verschiedenen Detektoren zu einem Ereignis im Experiment zusammengefasst werden, also das (L1-)Eventbuilding durchgeführt werden. Die Eventnummer wird nach jedem Burst zurückgesetzt. Somit erhält das erste vom L0-Trigger akzeptierte Ereignis eines Bursts die

[2]Da innerhalb der Farm ausschließlich Intel-Prozessoren (Little-Endian-Systeme) verwendet werden sollen, müssen somit die Datenworte nicht konvertiert werden.

Eventnummer 0, das zweite akzeptierte Ereignis die Eventnummer 1 und so fort. In den Farm-Protokollen wird diese Nummer durch ein 24-Bit-Datenwort übertragen. Somit können innerhalb eines Bursts bis zu $2^{24} \lesssim 1,7 \cdot 10^7$ Ereignisse von dem L0-Trigger akzeptiert werden.

### 4.2.2. Multi-Event-Packet (MEP)

Für die Übertragung der L0-Daten von den TEL62-Boards zur Farm wird das so genannte **M**ulti-**E**vent-**P**acket (MEP) eingeführt. Ein MEP beinhaltet die Daten eines TEL62-Boards zu mehreren Ereignissen, wobei die Daten zu einem Ereignis innerhalb eines MEPs als MEP-Event (kurz MEPE) bezeichnet wird. Der Vorteil von dieser Methode ist, dass viele kleine MEP-Events zu einem großen Paket gebündelt werden können und somit weniger Kopfdaten (Ethernet, IP, UDP) pro Ereignis übertragen werden müssen. Innerhalb dieses Protokolls werden die Kopfdaten des MEP und der MEP-Events definiert (siehe Abbildung 4.1).

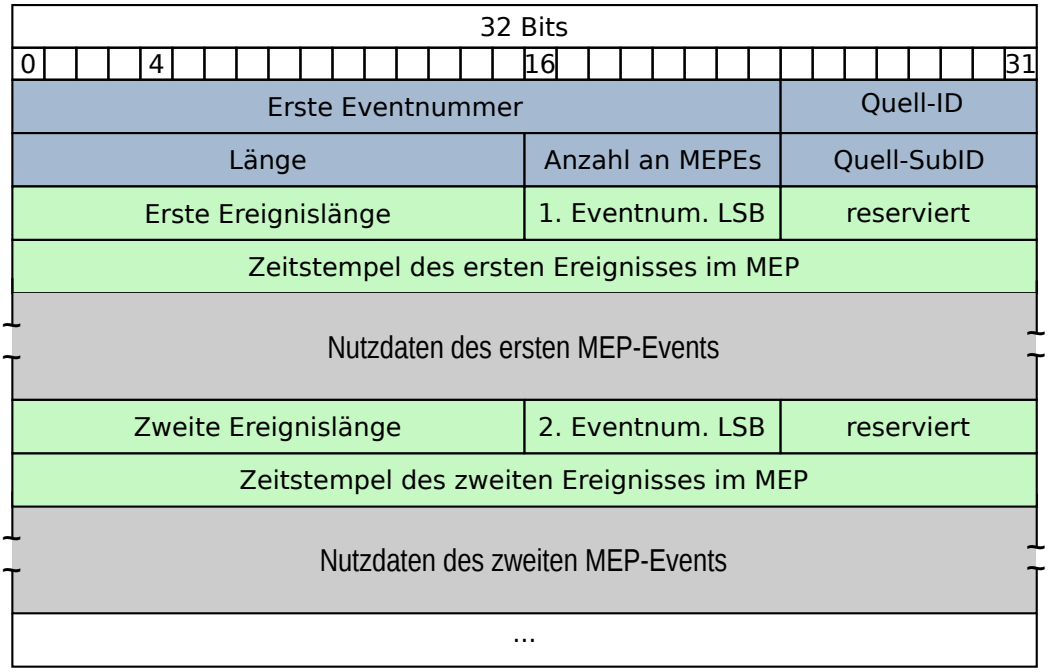


**Abbildung 4.1.:** Das Datenformat eines Multi-Event-Packets (Kopfdaten in blau) und der darin befindlichen MEP-Events (Kopfdaten in grün). Das N-te LSB ergibt zusammen mit dem Datenwort „Erste Eventnummer“ die Eventnummer des N-ten MEP-Events. Die Datenfelder „Länge“ und „N-te Ereignislänge“ beschreiben die Summe der Bytes innerhalb des Kopf- und Nutzdatenteils des MEPs bzw. MEP-Events.

Jedes MEP wird über ein UDP-Datagramm übertragen, wobei eine IP-Fragmentierung bisher nicht erlaubt wird. Wie in Abschnitt 4.10 diskutiert, wird z.Z. über eine mögliche Notwendigkeit von Fragmentierungen diskutiert.

Jedem Detektor wird eine einmalige Quell-ID zugewiesen. Jedes TEL62-Board eines Detektors erhält eine, innerhalb des Detektors eindeutige, Quell-SubID. Da zu jedem Ereignis im Detektor von jedem Elektronik-Board exakt ein MEP-Event übertragen wird, kann innerhalb der PC-Farm über diese Nummern bestimmt werden, ob alle Daten zu einem Ereignis empfangen wurden. Dies wird für das L1-Eventbuilding benötigt[3].

Die MEP-Events sind innerhalb eines MEPs chronologisch sortiert. Somit genügt es innerhalb der MEP-Event-Kopfdaten nur das LSB der Eventnummer, also das Byte an der kleinsten Speicheradresse im Little-Endian-Format, zu übertragen. Da im MEP-Kopfdatenteil die komplette Eventnummer des ersten Ereignisses gespeichert ist und die Eventnummern der MEP-Events lückenlos fortlaufend sind, kann die Eventnummer jedes MEP-Events bestimmt werden. Auf diese Weise werden zwei Byte pro MEP-Event (durchschnittlich ca. 40-50 Byte) bei der Übertragung eingespart.

### 4.2.3. LKr-MEPs

Für die Übertragung der Daten von den CREAMs zur PC-Farm werden so genannte LKR-MEPs verwendet (siehe Abbildung 4.2). Diese können, analog zu den MEPs aus Abschnitt 4.2.2, die Daten zu mehreren Ereignissen beinhalten (LKr-MEP-Events). Allerdings sind die LKr-MEP-Events nicht sortiert, und es wird kein gemeinsamer Kopfdatenteil verwendet[4].

Ähnlich wie beim MEP-Protokoll wird für das Eventbuilding eine Quell-SubID definiert. Da die CREAM-Boards in einzelnen Crates gruppiert sind, unterteilt sich die Quell-SubID der LKR-Events in eine Crate-ID und eine CREAM-ID. Beides sind fortlaufende 8-Bit-Worte[5].

[3] Prinzipiell hätte man hierfür auch die MAC- oder IP-Adressen verwenden können. Da man die Länge der Protokoll-Kopfdaten jedoch zur besseren Übersicht auf ein Vielfaches von 4 Byte setzt, ist für die Quell-ID und Quell-SubID genügend Speicher übrig. Somit können diese Informationen ohne zusätzlichen Aufwand übertragen werden, und das L1-Eventbuilding bleibt unabhängig von MAC- oder IP-Adressen.

[4] Es war ursprünglich nicht geplant, mehrere LKr-MEPs in einem Paket zu übertragen. Um dies zu ermöglichen, ohne dabei das Format neu zu definieren, werden die einzelnen Event-Daten in einem Paket hintereinander gereiht. Der Empfänger kann die einzelnen LKr-MEP-Events durch die Längenangabe voneinander trennen und dabei durch das feste Datenwort mit dem Wert 0x24 die Integrität überprüfen.

[5] Das zweite CREAM-Board im fünften Crate erhält somit die Quell-ID „01 04“.

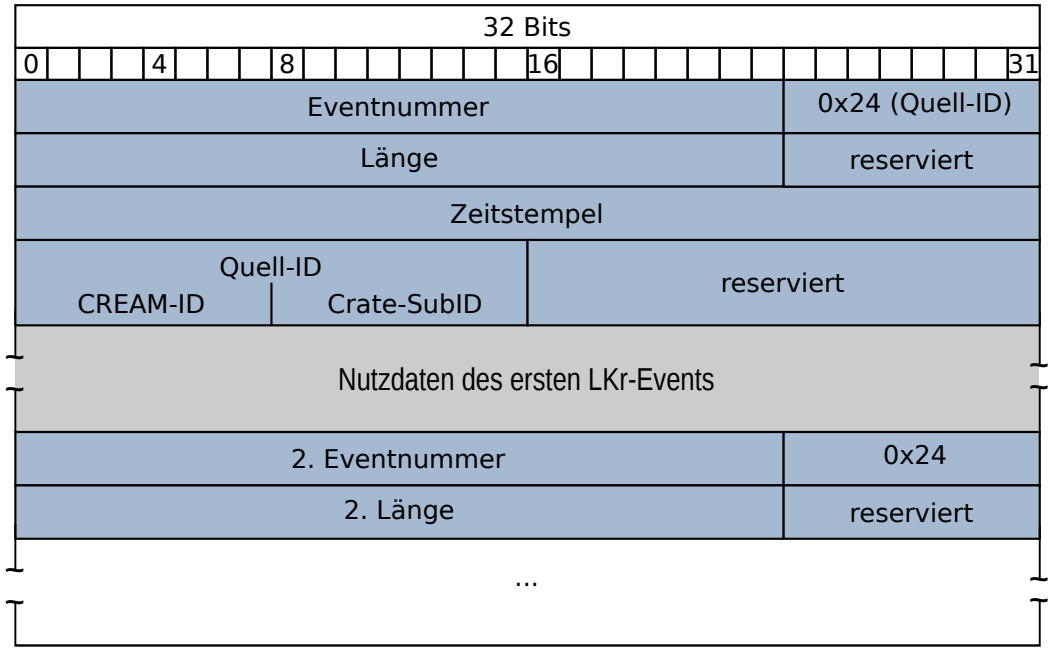


**Abbildung 4.2.:** Das Datenformat der LKr-MEPs. Hierbei werden, im Gegensatz zu den MEPs aus Abschnitt 4.2.2, keine gemeinsame Kopfdaten verwendet. Das Datenfeld „Länge“ beschreibt die Anzahl an Bytes zu einem Ereignis inklusive Kopfdaten. Dabei kann ein LKr-MEP aus mehreren solcher LKr-MEP-Events bestehen, welche durch die Längenangabe separierbar sind.

### 4.2.4. Multi-Trigger-Packet (MTP)

In jeder der drei Trigger-Ebenen wird pro Ereignis ein so genanntes Trigger-Typ-Wort definiert. In diesen 8-Bit-Daten ist kodiert, ob das Ereignis akzeptiert wurde, und falls dem so ist, zu welcher Kategorie akzeptierter Ereignisse es gehört. Der Wert Null beschreibt dabei die Ablehnung des Ereignisses, und so können bis zu $2^8 - 1 = 255$ Kategorien pro Trigger-Stufe definiert werden.

Die PC-Farm erhält das L0-Trigger-Typ-Wort über ein MEP vom L0TP. Nachdem die L1-Trigger-Entscheidung innerhalb der PC-Farm generiert wurde, werden das L0- und L1-Trigger-Typ-Wort an die CREAMs übertragen. Dabei können mehrere Trigger-Entscheidungen in Form von einem **M**ulti-**T**rigger-**P**acket (MTP) gesendet werden (siehe Abbildung 4.3). Ein MTP besteht aus einem Kopfdatenteil und einem Nutzdatenteil, welcher die einzelnen Trigger-Entscheidungen enthält. In dem Kopfdatenteil wird die Anzahl der Trigger-Entscheidungen und der Bytes innerhalb eines MTPs, sowie eine fortlaufende MTP-Nummer, übertragen. Die fortlaufende Nummer wird bei jedem gesendeten MTP um die Anzahl der Trigger in dem Paket erhöht. Analog zur Sequenznummer im TCP

(siehe Abschnitt 2.3.6) dient dies der Überprüfung, ob alle Pakete empfangen wurden.

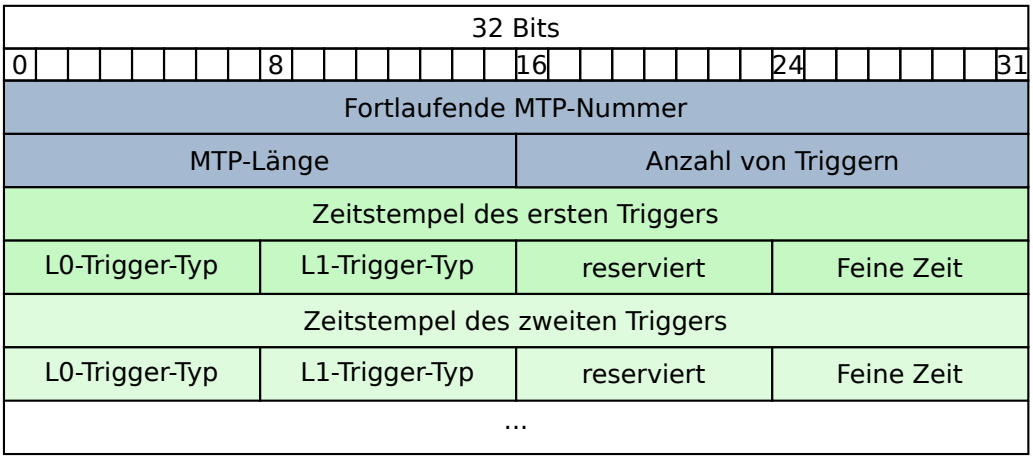


**Abbildung 4.3.:** Das Format des Multi-Trigger-Packets (MTP). Der Zeitstempel zusammen mit der „Feinen Zeit“ ergibt die Information, die zum Auslesen der LKr-Daten verwendet wird, falls das L1-Trigger-Typ-Wort größer Null ist.

Im Nutzdatenteil des MTPs werden die Trigger-Entscheidungen mit dem Zeitstempel und den Trigger-Typ-Worten von L0 und L1 in beliebiger Reihenfolge übertragen. Der Zeitstempel hat dabei eine Genauigkeit von 25 ns (LHC-Clock). Zusätzlich wird beim L1-Eventbuilding eine genauere Zeit ermittelt, welche durch das Datenwort „Feine Zeit“ übertragen wird (ca. 100 ps Genauigkeit). Sobald das L1-Trigger-Typ-Wort größer Null ist, werden diese Zeitinformationen von den CREAM-Boards ausgelesen und die Daten des LKr-Kalorimeters zu dem jeweiligen Zeitpunkt an die PC-Farm verschickt.

## 4.3. Das Testsystem

Für die Evaluation des Farm-Frameworks wurde ein eigenes Rechencluster aufgebaut. Dazu wurde ein Switch[6] mit 48 1GbE und 4 10GbE Ports verwendet. Dieser diente als Kopplungselement in einer Stern-Topologie mit insgesamt 15 PCs. Neben 13 älteren PCs[7] (im folgenden als L0-PCs bezeichnet), wurden zwei neue PCs mit jeweils 12 Prozessorkernen und zwei 10GbE-Konnektoren (Intel® X520 DA) erworben. Diese Rechner der Marke Dell vom Typ R510 werden im Folgenden mit Dell1 und Dell2 bezeichnet:

[6] Hierbei handelte es sich um einen Dell PowerConnect 6248 mit zwei Dualport SFP+ Modulen. Für die 10GbE Verbindungen über SFP+ wurden Direct-Attach-Kabel (Kupferkabel) verwendet.

[7] Hierbei handelte es sich um drei verschiedene Typen von PCs. Diese beinhalteten einen Ein-, Zwei- oder Vier-Kern-Prozessor und je ein 1GbE-Port pro Kern. Insgesamt standen so 25 Prozessorkerne zur Verfügung.

**Dell1:** Dieser PC beinhaltete zwei Intel X5670 Prozessoren mit jeweils sechs Prozessorkernen. Diese Prozessoren waren mit 2,93 GHz[8] getaktet.

**Dell2:** Dieser PC war mit zwei energieeffizienten Intel L5640 Prozessoren mit jeweils sechs Prozessorkernen ausgestattet. Diese waren mit 2,26 GHz[9] getaktet.

Beide Rechner waren mit 24 GB Arbeitsspeicher bei einer Taktrate von 1,333 GHz ausgerüstet.

## 4.4. Netzwerkprogrammierung

Zu Beginn der Arbeit wurden erste Prototypen des Frameworks implementiert. Dabei wurden Standard-Kernel-Sockets verwendet. Sockets stellen die Schnittstellen für die Interprozesskommunikation innerhalb der Sitzungsschicht dar (siehe Abschnitt 2.2.4.5). Standard-Kernel-Sockets sind somit innerhalb des Betriebssystemkerns (Kernel) implementiert und ermöglichen z.B. die Kommunikation über UDP und TCP. Für diesen Zweck werden über so genannte Systemaufrufe die benötigten Informationen wie Ziel-Adresse und Nutzdaten an den Kernel übertragen. Der Vorteil von dieser Methode ist, dass die jeweiligen Kommunikations-Protokolle für den Benutzer transparent implementiert sind und durch das Betriebssystem eine hohe Vereinheitlichung realisiert wird. So unterscheiden sich z.B. Zugriffe auf eine Datei oder auf eine TCP-Verbindung nur bei der Initialisierung.

Möchte man über Standard-Kernel-Sockets UDP-Pakete empfangen, so wird zunächst ein Socket-Objekt initialisiert. Dabei wird unter anderem die gewünschte Port-Nummer angegeben (beim Empfangen eines Pakets entspricht dies dem Ziel-Port aus Abbildung 2.5). Der Ablauf sieht dann schematisch folgendermaßen aus[10]:

1. Das Benutzerprogramm fragt über einen Systemaufruf (read) ein Paket an. Die Kontrolle wird somit an den Kernel übergeben.

2. Falls ein Paket im Speicher der Netzwerkkarte mit entsprechendem Ziel-Port vorhanden ist, werden diese Daten in den Speicherbereich des Benutzerprogramms

[8] Abhängig von der Hitzeentwicklung werden diese Frequenzen von einzelnen Kernen temporär in Schritten von 133 MHz überschritten (Turbo-Modi). Dabei können maximal zwei Kerne im dritten (3,33 GHz) Turbo-Modus laufen.

[9] Analog zum X5670 können hier bis zu vier Turbo-Modus-Stufen erreicht werden. Dabei können maximal zwei der Prozessorkerne mit 2,8 GHz getaktet werden.

[10] In diesem Beispiel wird die blockierende Variante dargestellt. Bei einem nichtblockierenden Socket gibt die Anfrage (read) sofort einen Wert zurück, auch wenn kein Paket im Puffer der Netzwerkkarte gefunden wurde. Das Benutzerprogramm müsste also überprüfen, ob read tatsächlich Nutzerdaten zurückgegeben hat.

kopiert. Anschließend erhält das Benutzerprogramm die Kontrolle zurück, wobei ein Zeiger auf die Daten des empfangenen Pakets übergeben wird.

3. Ist kein Paket vorhanden, so werden die CPU-Ressourcen für weitere Threads freigegeben. Das Benutzerprogramm schläft somit, da der Kernel die Kontrolle weiterhin behält.

4. Sobald die Netzwerkkarte ein Paket empfängt, kopiert sie dieses in einen neu allozierten Speicherbereich (Eine Allokation und Deallokation pro Paket). Anschließend wird ein Interrupt-Signal (engl. to interrupt, unterbrechen) an den Kernel[11] gesendet. Sollten keine Ressourcen für den Kernel zur Verfügung stehen, entzieht dieser einem beliebigen Thread die CPU-Ressourcen (Kontextwechsel).

5. Der Kernel kopiert das empfangene Paket in den Speicherbereich des Benutzerprogramms und gibt die Kontrolle an das Benutzerprogramm zurück.

Sobald das Benutzerprogramm „schläft“ und ein anderer Prozess Ressourcen benötigt, speichert der Kernel den aktuellen Zustand des Benutzerprogramms (Kontext) und lädt das neue Programm (Kontextwechsel). Dann muss beim „Aufwachen“ des ursprünglichen Benutzerprogramms dessen Kontext wieder in das Register des Prozessors geladen werden (erneuter Kontextwechsel). Somit finden für jedes Paket bis zu drei Kontextwechsel statt. Dieser Prozess kann bei einem 2.0 GHz Intel Pentium Xeon Prozessor zwischen einigen zehn Mikrosekunden bis hin zu einigen Millisekunden, abhängig von der Last des Systems, dauern [20]. Wie in Abschnitt 4.4.1 gezeigt wird, führt dies selbst bei deutlich schnelleren Prozessoren zu erheblichen Problemen.

### 4.4.1. Performance-Tests

Um die Auswirkung der Kontextwechsel und Last durch die Verarbeitung der Interrupts auf die effektive Performance einer 10GbE-Kommunikation zu untersuchen, wurden verschiedene Tests durchgeführt. Zunächst wurde die Auswirkung der Paketgrößen analysiert, indem eine Sekunde lang Pakete einer festen Größe von drei Threads auf Dell2 zu einem Thread auf Dell1 gesendet wurden. Gleichzeitig wurden die Datenraten, die CPU-Last, der relative Paketverlust, die Interruptrate und die Rate der Kontextwechsel bestimmt. Für jede Paketgröße wurde dies 15 mal wiederholt und daraus der Mittelwert mit Standardabweichung bestimmt. Dieser Vorgang wurde mit Hilfe des UDP- und des TCP-Protokolls durchgeführt. Das Ergebnis ist in Abbildung 4.4 zu sehen.

Hierbei wird deutlich, dass die volle Datenrate von 10 Gbps bei beiden Protokollen erst ab einer gewissen Paketgröße oberhalb der Standard-MTU von 1.500 B, also so genannten Jumbo-Frames (siehe Abschnitt 2.3.1.1), erreicht wird.

[11] Wie in Abschnitt 4.4.1 beschrieben, existieren verschiedene Methoden zur Reduzierung der Interruptrate. Dabei werden mehrere Pakete durch ein Interrupt „übertragen“.

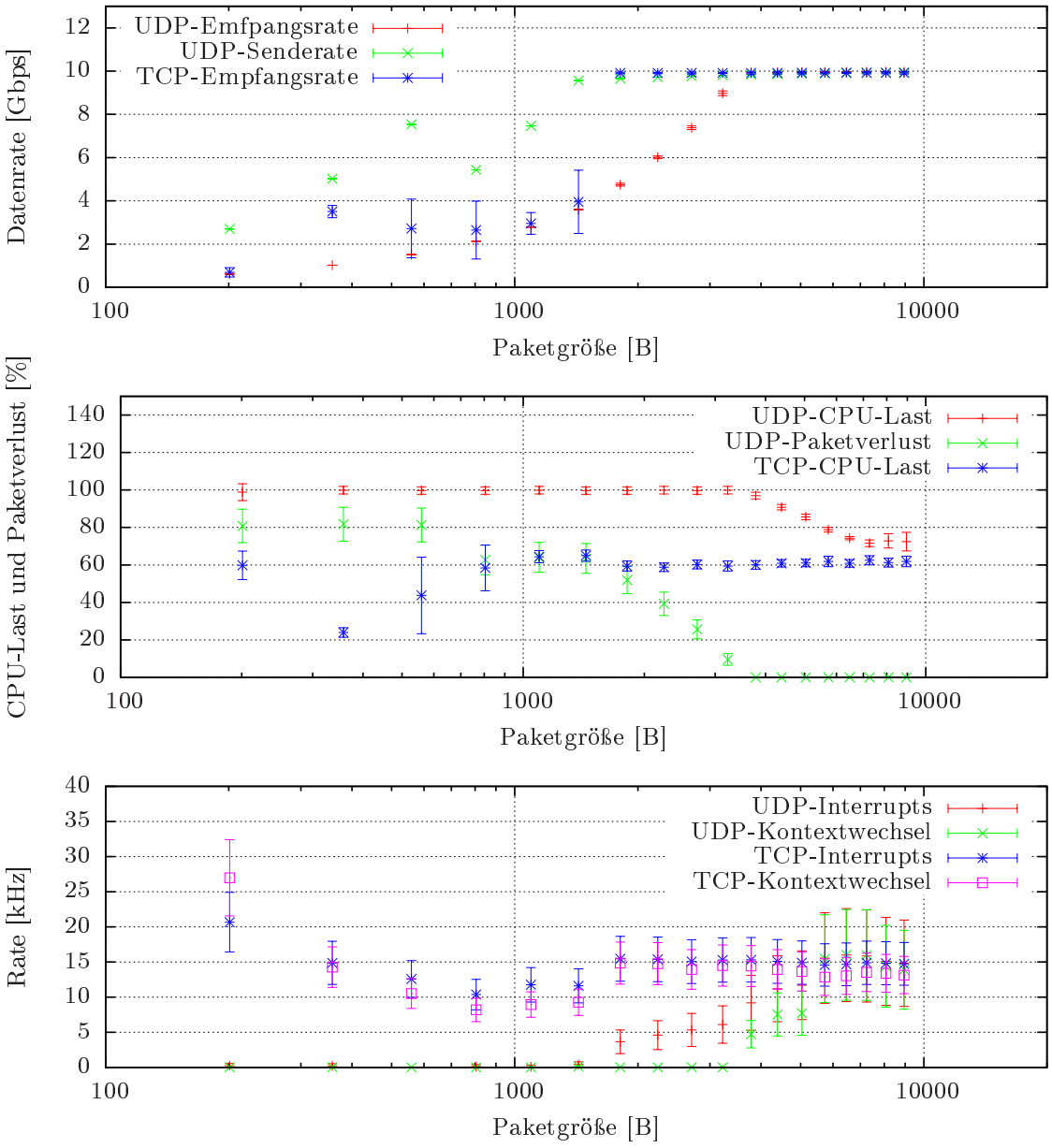


**Abbildung 4.4.:** Performance von UDP und TCP unter Verwendung von Standard-Kernel-Sockets.

Für UDP wird bei kleineren Paketgrößen eine höhere Senderate erreicht, als der Empfänger verarbeiten kann. In den Kernel-Statistiken konnte eingesehen werden, dass alle Pakete tatsächlich von diesem erkannt wurden und nicht schon im Switch oder im Puffer der Netzwerkkarte verloren gingen. Da die Rate jedoch zu hoch war, musste der Kernel einige Pakete verwerfen. Die relative Anzahl von verworfenen Paketen wird in Abbildung 4.4 durch „UDP Paketverlust“ dargestellt.

Bei der TCP-Verbindung wird durch die Flusssteuerung die Senderate angepasst. Somit wird der Empfänger nicht überlastet, und es müssen deutlich seltener Pakete vom Kernel verworfen werden (die genaue Anzahl verworfener Pakete konnte nicht bestimmt werden, da diese vom Sender erneut gesendet werden und so das Empfängerprogramm immer alle Daten erhält). Auf diese Weise erreicht TCP insgesamt eine höhere Datenrate bei geringerer CPU-Last, obwohl im Gegensatz zu UDP hierbei auch vom Empfänger Pakete zur Bestätigung des Empfangs der Daten gesendet werden müssen.

Eine weitere Auffälligkeit sind die geringen Raten an Interrupts für beide Protokolle. Bei einer Paketgröße von 1096 B wird bei beiden Protokollen eine Datenrate von ca. 3,5 Gbps erreicht. Dies entspricht einer Paketrate von rund 400 kHz. Würde bei jedem Paket ein Interrupt gesendet werden, so wäre eine Interruptrate von ebenfalls 400 kHz zu erwarten. Tatsächlich wurden bei UDP mit der genannten Paketgröße fast keine Interrupts registriert. Auch bei TCP liegt die Interruptrate mit rund 12 kHz deutlich unter den erwarteten 400 kHz. Dieser Effekt wird durch die so genannte Interrupt-Koaleszenz und durch NAPI erreicht: Bei der Interrupt-Koaleszenz wird von der Netzwerkkarte zunächst eine gewisse Anzahl an Paketen „gesammelt“, bevor ein Interrupt gesendet wird [21]. Der Kernel kann dann alle gesammelten Pakete auf einmal verarbeiten. Das Verhältnis „Pakete pro Interrupt“ kann dabei direkt, also über die Anzahl der Pakete definiert werden, oder durch eine Zeit, die mindestens zwischen zwei Interrupts vergehen soll. Bei den Testläufen zu diesem Kapitel wurde die Standardkonfiguration verwendet. Dabei wird der Parameter abhängig von der Last automatisch angepasst. Wie in Abschnitt A.1 gezeigt, entspricht dies der optimalen Einstellung. Zu große Werte führen zu einem Überlauf des Netzwerkkarten-Puffers.

NAPI steht für New API und ermöglicht die temporäre Deaktivierung von Interrupts [22]. Der Kernel deaktiviert somit Interrupts solange noch Pakete im Empfangspuffer der Netzwerkkarte vorhanden sind.

Die Interruptrate ist bei TCP deutlich höher als bei UDP, da durch die Flusssteuerung der Empfänger nicht überlastet wird. Somit kann der Kernel regelmäßig den Empfangspuffer leeren und innerhalb von NAPI das Senden von Interrupts wieder aktivieren.

Der starke Einbruch der Senderate bei Paketgrößen von 559 B zu größeren Paketen hin ist, wie die vernachlässigbaren Fehler andeuten, reproduzierbar. Grund hierfür könnten datengrößenabhängige Effizienzen innerhalb der Hardware sein.

Es hat sich somit gezeigt, dass bei Verwendung von Standard-Kernel-Socket Jumbo-Frames verwendet werden müssen, um die volle Datenrate der 10GbE-Netzwerkkarten ausnutzen zu können. Gleichzeitig würde die Wahl auf TCP fallen, da einerseits schon bei kleineren Paketgrößen die volle Datenrate erreicht wird, und andererseits dieses Protokoll eine zuverlässige Übertragung realisiert bei gleichzeitig geringerer CPU-Last. Die Problematik ist jedoch, dass TCP das deutlich komplexere Protokoll ist und daher nicht in den TEL62-Boards implementiert werden kann. Für die Kommunikation zwischen L0 und der PC-Farm muss daher UDP verwendet werden. Die CREAM-Boards könnten über TCP ausgelesen werden, da diesen höhere Rechenkapazitäten zur Verfügung stehen. Wie oben erwähnt, wird dies aus Gründen der Einheitlichkeit des Systems jedoch nicht gemacht.

### 4.4.2. Paketverluste

Bei der Realisierung des L1-Eventbuilding (siehe Abschnitt 3.2.5) sieht der Datenfluss so aus, dass alle TEL62-Boards ein Bündel von $N$ Ereignissen an den gleichen Farm-PC senden, bevor die gleiche Datenmenge an einen anderen PC gesendet wird (siehe Abbildung 4.5). Ein Bündel kann dabei aus mehreren MEPs bestehen, und die Summe der Datenmenge der Bündel aller Detektoren Beträgt $S_{Bündel} = S_{L0} \cdot N$ (siehe Tabelle 1.2).

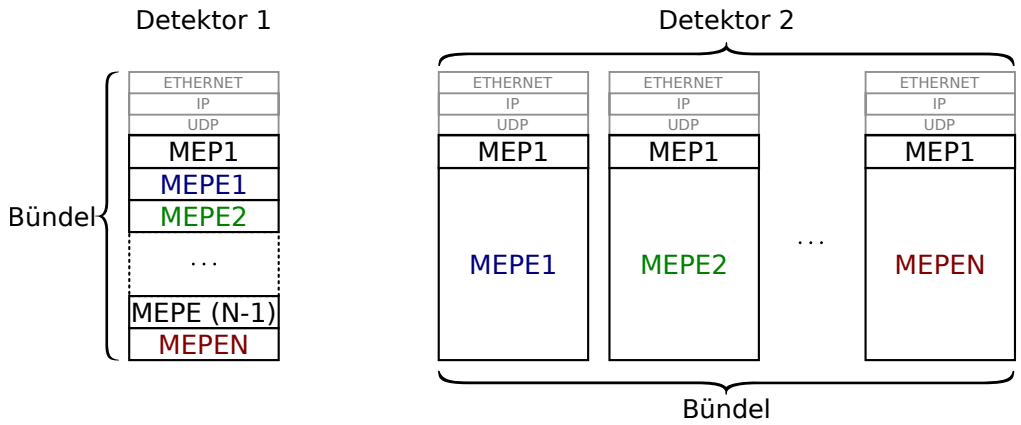


**Abbildung 4.5.:** Bündelung mehrerer Ereignisse zum L1-Eventbuilding: Jeder Detektor sendet Bündelweise $N$ Ereignisse zu einem Farm-PC. Abhängig von der Datenmenge kann ein Bündel aus mehreren MEPs bestehen.

Um diesen Datenstrom zu simulieren und zu evaluieren, ob diese gepulste Übertragung zu Problemen führt, wurden auf den L0-PCs Programme ausgeführt, die beim Empfang eines bestimmten Broadcast-Paketes ein solches Bündel simulieren und an Dell1 senden.

Dabei sind die Bündelgröße und die Rate der Broadcast-Pakete (Bündelfrequenz) variiert worden. So konnten 21 TEL62-Boards simuliert werden, welche Bündel in Form von ein oder zwei UDP-Paketen der Größe zwischen 1 kB und 9 kB, abhängig von der Bündelgröße, versendet haben[12]. Dabei wurde der Quotient aus der Anzahl an erwarteten und tatsächlich empfangenen Paketen bestimmt (relativer Paketverlust). Das Ergebnis ist in Abbildung 4.6 dargestellt.

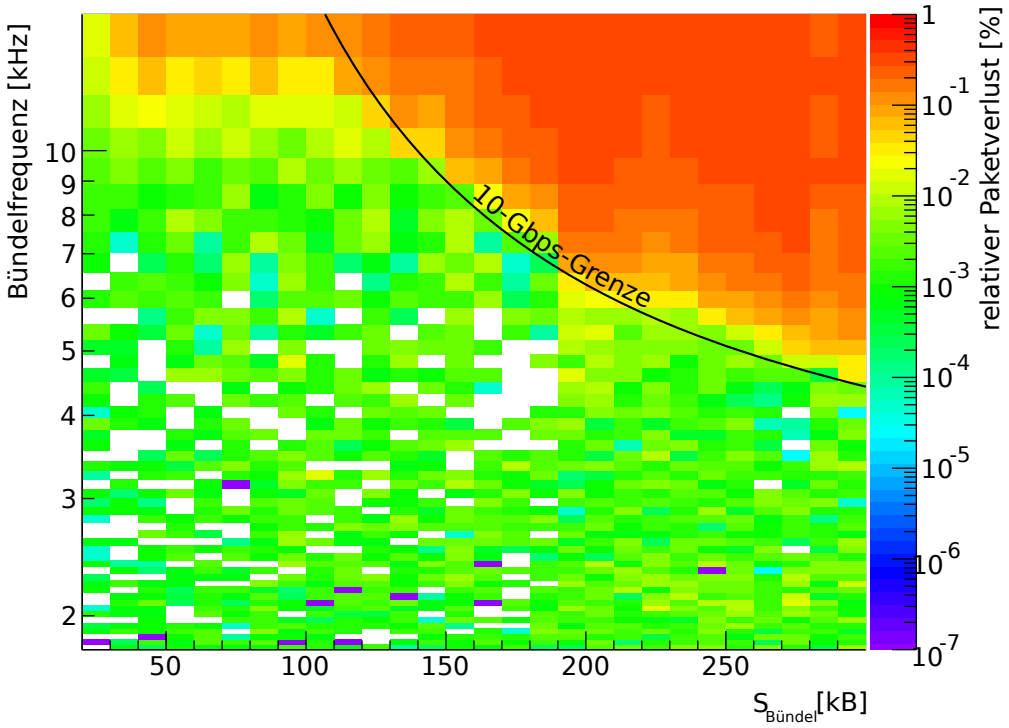


**Abbildung 4.6.:** Paketverluste bei simuliertem L1-Eventbuilding. Jeder Datenpunkt entspricht einer Messung bei der 21 PCs jeweils $10^7$ Bündel sendeten, wobei ein Bündel für $S_{\text{Bündel}} < 190$ kB aus einem UDP-Paket, darüber aus zwei Paketen bestand (jeweils zwischen 1 kB und 9 kB pro Paket). Bei weißen Bereichen gingen dabei keine Pakete verloren.

Bei Messwerten oberhalb der mit „10-Gbps-Grenze“ markierten Linie senden die L0-PCs insgesamt mit einer höheren Datenrate als der Switch über die 10GbE-Verbindung an den Empfänger-PC senden kann. In diesem Bereich gehen daher nicht nur Pakete innerhalb des PCs verloren, sondern auch innerhalb des Switches[13]. Unterhalb dieser Grenze konnten keine Paketverluste innerhalb des Switches beobachtet werden.

[12] Als Vorgabe galt hierbei die gesamte Datenmenge $S_{\text{Bündel}}$. Jeder PC hat somit ein Bündel der Größe $S_{\text{Bündel}}/21$ versendet. Ein Bündel wurde wiederum in gleich große UDP-Pakete mit einer Mindestgröße von 4 kB unterteilt. Somit variiert die Anzahl gesendeter Pakete mit der Bündelgröße.

[13] Für diese Messung wurde die Ethernet-Flusskontrolle (siehe Abschnitt 2.2.5.4) deaktiviert.

Interessant ist, dass unterhalb der 10-Gbps-Grenze, größtenteils unabhängig von der resultierenden Datenrate, ein relativer Paketverlust von rund $10^{-3}$ erreicht wird. Weitere Tests konnten zeigen, dass dieser Effekt auch bei einer einzelnen Punkt-zu-Punkt-Kommunikation zu beobachten ist[14]. Zudem ist auffällig, dass nur selten Paketverluste im Bereich zwischen $10^{-3}$ und $10^{-7}$ gemessen wurden, häufig jedoch kein einziges Paket verloren geht. Dieser Effekt könnte durch Fluktuationen der Last durch Hintergrundprozesse auf dem Empfänger-PC entstehen, welche zu Kontextwechseln führen. Da nach diesem Test entschieden wurde, keine Standard-Kernel-Sockets für die NA62-Online-PC-Farm zu verwenden um Datenverlust zu vermeiden, wurden die genannten Effekte nicht genauer untersucht.

Nach Aussagen von Luca Deri[15] sind diese Paketverluste bei Verwendung von Standard-Kernel-Socket unumgänglich. Aus diesem Grund wird für die endgültige Implementierung des Farm-Frameworks ein spezieller Netzwerktreiber, genannt pf_ring DNA, der Firma ntop, verwendet (siehe Abschnitt 4.4.3) [23]. Unabhängig davon konnten diese Tests zeigen, dass die gepulste Übertragung durch das L1-Eventbuilding keine Probleme innerhalb des Switches und der Netzwerkkarten verursachten.

### 4.4.3. pf_ring DNA

Der Netzwerktreiber pf_ring DNA ermöglicht das Empfangen von Ethernet-Paketen ohne dabei Systemaufrufe oder Interrupts verwenden zu müssen. Zu diesem Zweck wird der Netzwerkkarte ein fester Speicherbereich (Ringpuffer) zugewiesen, auf welchen direkt vom Benutzerprogramm zugegriffen werden kann. Es findet also nur eine einmalige Speicher-Allokation statt und der Kernel wird komplett umgangen (siehe Abbildung 4.7).

Das Benutzerprogramm muss somit empfangene Pakete aktiv im Ringpuffer abrufen. Dieser Vorgang wird als Polling bezeichnet (engl. polling, Sendeabruf).

Mit dieser Methode muss das Benutzerprogramm den Ringpuffer kontinuierlich nach neu empfangenen Paketen untersuchen (Polling). Sobald ein Paket empfangen wurde, kann dieses direkt innerhalb des Ringpuffers verarbeitet werden (Zero-Copy). Sollte die Verarbeitung jedoch eine längere Zeit in Anspruch nehmen, so muss das Paket kopiert und der Speicher wieder freigegeben werden, um ein Überlaufen des Ringpuffers zu verhindern (siehe Punkt 3 in Abbildung 4.7).

Tests haben gezeigt, dass sowohl auf Dell1 als auch auf Dell2 die volle Datenrate von 10 Gbps mit jeder beliebigen Paketgröße über Zero-Copy erreicht wird. Dabei wurde nur

[14] Hierbei wurde nur ein TEL62-Board durch Dell2 simuliert, wobei Dell1 und Dell2 durch eine 10GbE Verbindung direkt miteinander kommunizierten.

[15] Luca Deri ist Dozent an der Universität Pisa und Gründer der Firma ntop. Er hat unter anderem den speziellen Netzwerktreiber pf_ring DNA entwickelt (siehe Abschnitt 4.4.3).

ein einziger Thread zum Empfangen der Daten eingesetzt und kein einziger Paketverlust festgestellt. Die obere Schranke für den relativen Paketverlust konnte innerhalb des Test-Clusters mit 100-Byte-Paketen auf $4 \cdot 10^{-11}$ festgesetzt werden.

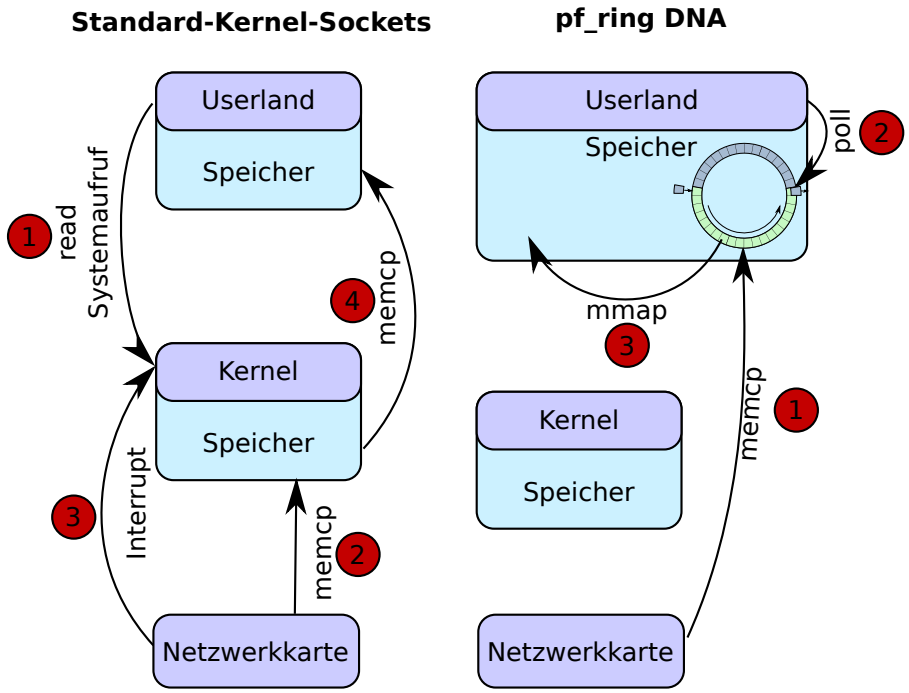


**Abbildung 4.7.:** Vergleich des Datenflusses bei Standard-Kernel-Socket und bei pf_ring DNA. Bei Standard-Kernel-Sockets erfährt der Kernel vom Empfang eines Paketes durch ein Interrupt von der Netzwerkkarte (3), und das Benutzerprogramm (Userland) muss Pakete über Systemaufrufe abfragen (1). Dabei findet für jedes empfangene Paket eine Speicher-Allokation statt (2). Bei pf_ring DNA werden empfangene Pakete von der Netzwerkkarte direkt in den Ringpuffer innerhalb des Userlands geschrieben (1), wobei keine Speicher-Allokation stattfindet. Das Benutzerprogramm muss durch Polling die Daten abrufen (2) und evtl. kopieren (3), um den Ringpuffer für neue Pakete freizuhalten. (3) kann dabei nebenläufig durchgeführt werden. pf_ring DNA verwendet bei aktivem Polling keine Systemaufrufe, der Kernel wird also umgangen (siehe Abschnitt 4.4.4).

Durch die Verwendung von pf_ring DNA kann das Benutzerprogramm nicht mehr von den Implementierungen der Netzwerkprotokolle innerhalb des Kernels profitieren. Stattdessen erhält das Benutzerprogramm beim Empfang die Daten von der ersten Schicht des OSI-Referenzmodells (Ethernet) aufwärts. Aus diesem Grund musste innerhalb die-

ser Arbeit Ethernet (siehe Abschnitt 2.2.5), IP (siehe Abschnitt 2.3.1), UDP (siehe Abschnitt 2.3.5) und ARP (siehe Abschnitt 2.3.2) für die Verwendung innerhalb des Frameworks implementiert werden.

### 4.4.4. Polling-Methoden

Beim permanenten (aktiven) Polling wird, unabhängig davon, ob Pakete empfangen werden oder nicht, ein Prozessorkern zu 100% beansprucht, da der Polling-Thread ständig auf den Speicher zugreift. Diese Zugriffe belasten den Datenbus zwischen Prozessor und Systemspeicher und können somit das System verlangsamen. Aus diesem Grund erlaubt pf_ring DNA beim Polling eine blockierende Methode (passives Polling). Falls kein Paket im Ringpuffer vorhanden ist, wird dabei der Thread „schlafen gelegt“ und in der Netzwerkkarte das Senden von Interrupts aktiviert. Sobald ein Paket empfangen wird, sendet die Netzwerkkarte ein Interrupt an den pf_ring-Treiber (Teil des Kernels). Dieser deaktiviert Interrupts innerhalb der Netzwerkkarte und weckt den Polling-Thread wieder auf. Auf diese Weise entsteht nur eine sehr geringe zusätzliche Last durch Interrupts, da diese nur auftreten, wenn der Ringpuffer keine Daten mehr enthält. In der Zeit des „Aufweckens“ des Polling-Threads müssen die Daten von diesem Puffer aufgefangen werden, so dass keine Paketverluste entstehen. Diese Zeit kann jedoch stark variieren: Vor allem wenn ein Kontextwechsel stattgefunden hat kann dieser Prozess besonders viel Zeit beanspruchen. Wie in Abschnitt 4.4 erläutert, können Kontextwechsel bei Systemen mit hoher Last (etwa durch die L1- oder L2-Prozessierung) bis hin zu einigen Millisekunden dauern.

Die innerhalb dieser Arbeit verwendeten Netzwerkkarten erlauben eine maximale Größe des Ringpuffers von 32768 (32 ki) Slots. Das bedeutet, dass zwischen zwei „Polls“ maximal 32 ki Pakete empfangen werden können. Bei einer durchschnittlichen Paketgröße von $S_{\text{Paket}}$ Byte darf ein Kontextwechsel daher nicht länger als $32\ \text{ki} \cdot S_{\text{Paket}}\ \text{B}/1,25\ \text{GBps} \approx S_{\text{Paket}} \cdot 26\ \mu\text{s}$ andauern, um während des Aufweckens des Polling-Threads keine Daten zu verlieren. Vor allem bei der Übertragung von kleinen Paketen müssen daher Kontextwechsel vermieden werden

Um die Häufigkeit von Kontextwechseln zu reduzieren, können die einzelnen Threads des Frameworks an bestimmte Prozessorkerne gebunden werden. Gleichzeitig ermöglicht der Kernel eine Konfiguration, so dass diese Prozessorkerne nicht von anderen Prozessen verwendet werden dürfen und keine Interrupts an diese Kerne gesendet werden[16]. Durch diese Methode konnte die Kontextwechselrate deutlich gesenkt werden; allerdings erlaubt der aktuelle Linux-Kernel[17] noch nicht Interrupts der Systemuhr an bestimmte

[16] Dies wird durch den Kernelparameter „isolcpus“ erreicht.
[17] Innerhalb dieser Arbeit wurde die Kernelversion 3.2.11 verwendet.

Prozessorkerne zu verbieten (diese werden mit einer Rate von 100 Hz durchgeführt). Somit können Kontextwechsel nicht komplett ausgeschlossen werden.

Um Kontextwechsel zu vermeiden, muss daher der Polling-Thread aktiv bleiben. Um jedoch den Datenbus zwischen Prozessor und Systemspeicher zu entlasten und gleichzeitig eine höhere Energie-Effizienz zu erhalten, erlaubt das Farm-Framework einen besonderen aktiven Polling-Modus: Falls der Ringpuffer bei einem Polling leer ist, wird der Polling-Thread eine gewisse Zeit „schlafen gelegt“. Wenn also keine Pakete empfangen werden, wacht der Polling-Thread regelmäßig auf um den Ringpuffer nach neuen Paketen zu überprüfen. Dadurch entsteht nur eine geringe CPU-Last. Bei einer Datenübertragung schläft der Polling-Thread erst wieder ein, wenn alle Pakete im Ringpuffer gelesen wurden. Jetzt ist im Gegensatz zum passiven Polling die Zeit, die der Thread schläft, fest definiert. Wenn diese „Wartezeit“ klein genug gesetzt ist und der Polling-Thread mit höchster Priorität ausgeführt wird, wird dessen Prozessorkern nicht für Interrupts oder andere Prozesse vom Kernel verwendet. Bei Testläufen mit der endgültigen Farm-Software konnte so eine verlustfreie Übertragung realisiert werden, wobei im Ruhezustand für den Polling-Thread eine CPU-Last von unter 1% erreicht wurde.

Die Wartezeit kann innerhalb der Farm-Software durch den Parameter „pollingSleep“ konfiguriert werden. Bei dem Wert 0 wird ein permanentes Polling eingesetzt, bei dem Wert -1 wird passives Polling aktiviert. Werte $N > 0$ konfigurieren ein aktives Polling mit einer Wartezeit von $N$ Mikrosekunden.

## 4.5. Datenfluss

Um die Mehrkernprozessoren der Farm-PCs effizient zu nutzen, werden mehrere Ereignisse parallel verarbeitet. Innerhalb eines Ereignisses muss daher keine Nebenläufigkeit stattfinden, was die Implementierung deutlich erleichtert.

Um eine effiziente Implementierung zu realisieren, sollte die Anzahl an aktiven Threads innerhalb der Farm-Software nicht größer als die Anzahl der Prozessorkerne $\Sigma$ sein, da so häufige Kontextwechsel vermieden werden. Eine weitere Effizienzsteigerung kann dadurch erzielt werden, dass nur ein Thread für den Empfang, also für das Polling, verwendet wird. pf_ring DNA ermöglicht zwar das nebenläufige Empfangen von Daten, verliert dabei jedoch deutlich an Effizienz, da die Zugriffe auf den Ringpuffer in diesem Modus durch Mutexe (siehe Abschnitt 2.4.2) synchronisiert werden.

Der Ansatz innerhalb des Farm-Frameworks besteht also darin, einen Thread für das Polling und weitere $\Sigma - 1$ Threads für die Verarbeitung der empfangenen Daten zu verwenden. Zum (L1-)Eventbuilding müssen alle Daten zu einem Ereignis von dem gleichen Thread verarbeitet werden. Da die Daten jedoch paketweise empfangen werden und MEPs sowie LKr-MEPs (siehe Abschnitt 4.2.2 und 4.2.3) Daten von verschiedenen

Ereignisse beinhalten, müssen die Daten innerhalb eines Pakets über mehrere Threads verteilt werden. Daher wurde das Framework in fünf Ebenen aufgeteilt:

**1: Polling (ein Thread)**
Dieser Thread dient dem reinen Empfangen der UDP-Pakete.

**2: Packethandler (mindestens ein Thread)**
Hier werden Datenstrukturen zu den Paketen angelegt und deren Konsistenz überprüft. Dazu gehören die Berechnung der UDP-Checksumme und Überprüfung der Validität der Längenangaben in den MEP- und LKr-MEP-Kopfdaten.

**3: Eventbuilding und Trigger (mindestens ein Thread)**
Diese Threads führen das (L1-)Eventbuilding und die L1- und L2-Prozessierung durch.

**4: L1-Verteilung (ein Thread)**
Dieser Thread sammelt L1-Trigger-Entscheidungen und versendet MTPs an die CREAM-Boards (siehe Abschnitt 4.2.4).

**5: Speicherung (ein Thread)**
An diesen Thread werden Ereignisse übertragen, die einen positiven L2-Trigger erhalten haben. Diese werden dann an den persistenten Speicher gesendet.

Nach dem Empfang eines MEPs wird dieses vom Polling-Thread an einen Packethandler-Thread übertragen (Round-Robin). Dort wird dieses Paket in die einzelnen MEP-Events zerlegt und jedes dieser Ereignisse, abhängig von deren Eventnummer, an einen speziellen Eventbuilder-Thread übertragen[18]. Dort findet das L1-Eventbuilding statt: Sobald das letzte notwendige MEP-Event übertragen wurde, führt dieser Thread die L1-Algorithmen aus und überträgt die L1-Trigger-Entscheidung an den L1-Verteiler-Thread. Sobald er genügend Trigger-Entscheidungen erhalten hat, sendet dieser Thread ein MTP an die CREAM-Boards.

Wird ein LKr-MEP empfangen, so durchläuft dieses ebenfalls die ersten drei Ebenen. Der Eventbuilder-Thread führt demnach das Eventbuilding durch. Anschließend werden die L2-Algorithmen ausgeführt und, im Falle einer positiven Entscheidung, das Ereignis an den Speicherungs-Thread übertragen. Dieser sammelt und sortiert die Ereignisse, um sie anschließend burstweise in Dateien auf Datenkassetten im Rechenzentrum des CERN zu speichern.

[18] In der finalen Software werden Ereignisse mit der Eventnummer E an den Eventbuilder-Thread E modulo N übertragen, wobei N die Anzahl an Eventbuilder-Threads ist.

### 4.5.1. Datenverteilung

Bei den im letzten Kapitel eingeführten Ebenen des Frameworks dienen die Ebenen Eins bis Drei als Daten-„Produzent“ für eine der höheren Ebenen. So sind z.B. die Packethandler „Konsumenten“ der Daten des Polling-Threads und gleichzeitig Produzent von Daten, welche von den Eventbuilding-Threads konsumiert werden.

Wie in Abschnitt 4.4.2 erläutert, müssen die UDP-Pakete, nach dem Empfangen durch den Polling-Thread kopiert werden, um den Ringpuffer zu entlasten. Tests haben gezeigt, dass dieser Prozess bei Paketen unterhalb von ca. 500 B nicht mehr von nur einem Thread alleine bewältigt werden kann. Zu kleine Pakete würden daher zu Paketverlusten führen. Deshalb wird der Kopiervorgang durch die Packethandler durchgeführt. Diese Threads erhalten vom Polling-Thread einen Zeiger auf die Pakete im Ringpuffer. Nach dem Kopieren der Pakete werden diese ausgelesen und auf eine korrekte Übertragung überprüft. Anschließend werden Zeiger auf die einzelnen Ereignisse innerhalb der Pakete erzeugt und an die jeweiligen Eventbuilder-Threads übertragen. Somit bleiben die UDP-Pakete so lange im Speicher, bis alle Ereignisse verarbeitet wurden. Bis dahin findet keine weitere Kopie mehr statt. Auf diese Weise wird die Effizienz des Frameworks deutlich erhöht. In einer anfänglichen Version des Frameworks wurden die Ereignisse innerhalb eines (LKr-)MEPs kopiert, bevor sie an den Eventbuilder weitergegeben wurden. Dies vereinfachte zwar die Implementierung, da so jedes Ereignis für sich gelöscht werden kann und keine Synchronisierung innerhalb eines UDP-Pakets stattfinden musste. Allerdings hat dieser Ansatz zu einem Überlaufen des Ringpuffers geführt, da der Polling-Thread die Pakete nicht schnell genug an die Packethandler weitergeben konnte.

Es empfiehlt sich, für die Kommunikation zwischen den einzelnen Ebenen so genannte Warteschlangen zu verwenden. Dabei handelt es sich um Datenstrukturen, die nach dem FIFO[19]-Prinzip funktionieren. Daten werden der Warteschlange also in der Reihenfolge entnommen, wie sie ihr hinzugefügt wurden. Die Warteschlangen können zwischen den Farm-Threads als Puffer dienen, falls ein Produzent zeitweise deutlich schneller Daten produziert, als die Konsumenten sie abarbeiten können. Da auf die Warteschlangen von verschiedenen Threads zugegriffen wird, müssen diese threadsicher implementiert sein (siehe Abschnitt 2.4.1).

Die Standard C++-Bibliothek beinhaltet eine Warteschlange namens `std::queue`. Da diese selbst nicht threadsicher implementiert ist, wurden die Lese- und Schreibzugriffe dieser Warteschlange durch einen Mutex gesichert (erste Methode). Betrachten wir nun die Übertragung der Zeiger auf die UDP-Pakete vom Polling-Thread zu den Packethandler-Threads: Verwendet man hierbei nur eine Warteschlange, und somit nur einen Mutex, so blockiert ein lesender Thread alle anderen lesenden und den schreibenden Polling-Thread. Daher wurde zu Testzwecken für jede Produzenten-Konsumenten-

[19] FIFO steht für First In, First Out (zu deutsch zuerst hinein, zuerst hinaus).

Kommunikation eine eigene Warteschlange mit eigenem Mutex eingesetzt (zweite Methode). Somit kann immer nur ein Konsument den Produzenten (Polling-Thread) blockieren, und umgekehrt kann der Produzent nur einen der Konsumenten blockieren.

Um dieses Verfahren zu evaluieren, wurde ein Produzenten-Thread implementiert, welcher permanent Zeiger alloziert und an eine feste Anzahl Konsumenten-Threads überträgt. Dabei wurde einerseits nur eine Warteschlange für alle Konsumenten und andererseits eine Warteschlange pro Konsument verwendet, um beide Methoden vergleichen zu können. Die Konsumenten führen dabei ebenfalls ein Polling durch, überprüfen also die Warteschlange regelmäßig nach neuen Daten. Genauso wie in Abschnitt 4.4.4 wurde dies mit einem aktiven (mit variabler Wartezeit) und einem passiven Polling implementiert[20]. Bei einem passiven Polling konnte bei 11 Konsumenten-Threads eine Übertragungsrate von $(220 \pm 15)$ kHz mit einer gemeinsamen Warteschlange und $(805 \pm 11)$ kHz bei der zweiten Methode erreicht werden. Die Verwendung mehrerer Warteschlangen ist also, wie erwartet, deutlich effizienter. In Abbildung 4.8 sind die Messwerte mit aktivem Polling bei einem und bei 11 Konsumenten-Threads aufgetragen. Hierbei ist ebenfalls die zweite Methode signifikant schneller, wobei maximale Übertragungsraten von $(7.00 \pm 0.03)$ MHz bei einem, und $(10.8 \pm 0.2)$ MHz bei 11 Konsumenten erreicht wurden.

Vergleicht man aktives und passives Polling, so zeigt sich erneut, dass die hohe Rate an Kontextwechseln durch das passive Polling diese Methode deutlich ineffizienter macht.

Obwohl durch die zweite Methode die Häufigkeit, dass ein Thread blockiert wird, deutlich sinkt, wurde für die endgültige Farm-Software eine dritte, deutlich effizientere Methode verwendet. Bei diesen speziellen Warteschlangen werden keine Mutexe verwendet, weshalb sie im folgenden Text als lockfreie Warteschlangen bezeichnet werden (siehe Abschnitt 4.5.1.1).

#### 4.5.1.1. Lockfreie Warteschlangen

Die Implementierung dieser speziellen Warteschlangen basiert auf einem Vorschlag von [24]: Bei Verwendung einer eigenen Warteschlange für jede Produzenten-Konsumenten-Kommunikation (zweite Methode in Abschnitt 4.5.1) greift immer nur ein Thread lesend (pop) und ein Thread schreibend (push) auf die einzelnen Warteschlangen zu. Wenn sichergestellt ist, dass auf alle Speicherbereiche innerhalb der Warteschlangen immer nur durch einen Thread lesend und durch den anderen Thread schreibend zugegriffen wird, dann sind diese Zugriffe auf eine gewisse Art threadsicher: Der schreibende Zugriff wird nicht vom lesenden gestört, allerdings kann es dazu kommen, dass beim lesenden Zugriff ein veralteter Wert erhalten wird.

[20] Für diesen Zweck wurden Konditionsvariablen aus der Boost-Bibliothek verwendet (boost::condition_variable).

Verwendet man nun einen Ringpuffer als Warteschlange, so muss dazu die Position des ersten freien (In-Pointer) und des letzten belegten (Out-Pointer) Speicherbereichs (Slot) gespeichert werden (siehe Abbildung 4.9). Der Produzent schreibt also Daten auf die Position des In-Pointers und erhöht diesen anschließend. Der Konsument liest Daten aus dem Slot an der Position des Out-Pointers und inkrementiert diesen Zeiger. Bei einem Lesezugriff muss sichergestellt werden, dass die beiden Pointer nicht auf die gleiche Position zeigen, da in diesem Fall der Ringpuffer leer ist. Beim Schreiben muss zwischen den beiden Zeigern mindestens noch zwei Positionen Abstand sein, da die Warteschlange sonst voll gelaufen ist und Daten überschreiben würden[21]. Auf diese Weise greift der Produzent lesend auf den Out-Pointer und schreibend auf den In-Pointer zu. Umgekehrt greift der Konsument nur lesend auf den In-Pointer, jedoch schreibend auf den Out-Pointer zu. Sollte der Konsument durch die Nebenläufigkeit einen veralteten In-Pointer auslesen, so könnte dieser Thread fälschlicherweise die Warteschlange als leer interpretieren und somit keine Daten auslesen, obwohl der Produzent gerade neue Daten geschrieben hat. Umgekehrt kann der Produzent fälschlicherweise die Warteschlange als voll interpretieren, obwohl der Konsument gerade einen Slot freigegeben hat. In beiden Fällen führt dies höchstens zu Ineffizienzen, jedoch zu keinem Datenverlust. Eine lockfreie Warteschlange ist somit threadsicher, solange nur ein Thread lesend und nur ein Thread schreibend darauf zugreift.

Analog zu der Evaluation in Abschnitt 4.5.1 wurde der gleiche Test mit lockfreien Warteschlangen, bei aktivem Polling mit variabler Wartezeit, durchgeführt. Dabei wird die Wartezeit beim ersten erfolglosen Polling (leere Warteschlange) auf den aktuellen Wert gesetzt. Anschließend wird die Wartezeit bei jedem erfolglosen Polling verdoppelt, bis sie einen Wert von 128 ms erreicht. Somit kann im inaktiven Zustand die CPU-Last deutlich gesenkt werden. Bei dieser Messung wurde zusätzlich die Anzahl an Slots innerhalb des Ringpuffers variiert. Das Ergebnis ist in Abbildung 4.10 aufgetragen.

Mit lockfreien Warteschlangen können somit die Zeiger zu empfangenen UDP-Paketen vom Polling-Thread an 11 Packethandler mit einer Rate von bis zu $(38 \pm 1)$ MHz übertragen werden. Bei gleichen Tests wurde, unter Verwendung von nur vier Konsumenten-Threads, eine Rate von bis zu $(25 \pm 3)$ MHz erreicht, und bei nur zwei Konsumenten-Threads ist eine Übertragungsrate von rund 16 MHz erreichbar (siehe Abschnitt A.2). Selbst bei der Verwendung eines einzigen Konsumenten-Threads, und somit nur einer einzigen lockfreien Warteschlange, wird mit etwa 11 MHz eine ähnliche Übertragungsrate erreicht, wie sie bei Verwendung von `std::queue` und Mutexen erst bei 11 Konsumenten erreicht wurde.

Innerhalb der endgültigen Farm-Software wird für jede Produzenten-Konsumenten-Kommunikation eine lockfreie Warteschlange mit aktivem Polling eingesetzt. Die Wartezeit

[21] Da nach dem Schreiben der In-Pointer inkrementiert wird, muss immer ein Speicherbereich zwischen In- und Out-Pointer freigehalten werden, da sonst nach einem Schreibzugriff nicht unterschieden werden kann, ob der Ringpuffer vollgelaufen oder leer ist.

wird dabei dynamisch auf 1 ms bis 128 ms gesetzt. Bei hoher Last wird der Konsument nur selten schlafen gelegt und somit bleibt die Wartezeit konstant bei 1 ms. Die Anzahl an Slots kann für die Warteschlangen zwischen den einzelnen Ebenen (siehe Abschnitt 4.5) getrennt konfiguriert werden.

## 4.6. Das Eventbuilding

Durch das L1-Eventbuilding-Konzept, wie es in Abschnitt 3.2.5 eingeführt wurde, erreichen alle L0-Daten zu einem Ereignis nach einem positiven L0-Trigger den gleichen Farm-PC. Durch die Bündelung mehrerer Event-Daten zu einem MEP können die einzelnen Teildaten zu einem Ereignis den jeweiligen Farm-PC zeitlich versetzt erreichen. Durch die Ausleseelektronik wird jedoch sichergestellt, dass jedes MEP-Event spätestens rund 10 $\mu$s nach Empfang eines L0-Trigger-Signals an die PC-Farm geschickt wird[22]. Im Vergleich zur Dauer eines Strahlzyklus kann diese Zeit vernachlässigt werden. Daher können die L0-Daten bereits vor den L1-Trigger-Algorithmen gesammelt werden (L1-Eventbuilding).

Dieser Ansatz erweitert die Möglichkeiten innerhalb der L1-Trigger-Algorithmen. So können Informationen zwischen verschiedenen Detektoren schon vor L2 verknüpft werden. Zudem steigt die Effizienz der L1-Trigger-Stufe, da die Algorithmen sequentiell innerhalb eines Threads abgearbeitet werden, und somit schon früh (etwa durch ein Veto) abgebrochen werden können. Somit werden Daten eines Ereignisses nur so lange ausgelesen und verarbeitet, bis eine negative L1-Trigger-Entscheidung gefällt wird, oder der ganze L1-Algorithmus abgearbeitet wurde (positive Entscheidung). Bei einer verteilten L1-Trigger-Stufe im ursprünglichen Konzept hätten immer alle Bestandteile der L1-Algorithmen prozessiert werden müssen, bevor es zur Trigger-Entscheidung kommt.

Der Datenfluss zum L1-Eventbuilding wird in Abbildung 4.12 dargestellt. Dabei wird das in Abbildung 4.11 dargestellte Datenmodell verwendet: Beim Programmstart wird eine Event-Liste von konfigurierbar vielen Event-Objekten initialisiert. Diese Objekte beinhaltet für jeden Detektor ein leeres Subevent-Objekt. Beim Empfang eines MEPs kopiert ein Packethandler-Thread dieses Paket und erstellt MEP-Event-Objekte. Von nun an werden nur Zeiger auf die MEP-Event-Objekte inklusive Daten transportiert, so dass keine weiteren Kopien stattfinden. Sobald einer dieser Zeiger an einen Eventbuilder-Thread

[22] Diese Zeit ist abhängig von der Anzahl an MEP-Events in einem MEP: Passen alle MEP-Events eines Bündels in nur ein MEP, so ist die Verzögerung maximal, da das erste Ereignis erst verschickt wird, wenn $N$ Ereignisse in der Ausleseelektronik gesammelt wurden. Da eine L0-Trigger-Rate von 1 MHz erwartet wird, entspricht die maximale Verzögerung also $N \cdot 1$ $\mu$s. $N$ wird dabei voraussichtlich 9 betragen.

übertragen wird, entnimmt dieser das Event-Objekt mit der entsprechenden Eventnummer aus der Event-Liste und fügt den MEP-Event-Zeiger dem jeweiligen, durch die Quell-ID definierten, Subevent-Objekt zu. Da die Anzahl an MEP-Events für jeden Detektor definiert ist, kann der Eventbuilder-Thread bestimmen, ob alle Subevent-Objekte „vollständig“ sind und das L1-Eventbuilding somit abgeschlossen ist. Ist dies der Fall, so wird das Event-Objekt direkt an die L1-Trigger-Algorithmen übertragen, und das L1-Trigger-Typ-Wort bestimmt. Dieses wird anschließend an den L1-Verteiler-Thread übertragen und im negativen Fall wird das Event-Objekt geleert. Beim Leeren bleibt die Datenstruktur zur Verwendung im folgenden Burst erhalten. Es werden lediglich die MEP-Events gelöscht und einige Klassenvariablen zurückgesetzt.

Der L1-Verteiler sendet die L1-Trigger in Form von Multicast-MTPs (siehe Abschnitt 2.3.3 und 4.2.4) an alle CREAM-Boards. Im Falle einer positiven L1-Trigger-Entscheidung werden anschließend die LKr-MEPs zu dem Event empfangen. Diese werden auf die gleiche Weise wie MEPs verarbeitet und ebenfalls in ein spezielles Subevent-Objekt innerhalb des gleichen Event-Objekts abgespeichert. Ist dieses Subevent-Objekt vollständig, so wird der L2-Trigger-Algorithmus ausgeführt und das L2-Trigger-Typ-Wort bestimmt. Im positiven Fall wird das Event-Objekt an den Speicherungs-Thread übertragen und anschließend, auch bei negativer L2-Trigger-Entscheidung, geleert.

Da die Kommunikation mit dem Rechenzentrum am CERN noch nicht spezifiziert ist, werden akzeptierte Ereignisse vom Speicherungs-Thread zu Testzwecken auf einem konfigurierbaren lokalen Datenträger abgelegt.

## 4.7. Dynamische Flusssteuerung

Für jede der Kommunikationen zwischen den Packethandler- und Eventbuilder-Threads werden zwei verschiedene lockfreie Warteschlangen eingesetzt: Eine für die Übertragung von MEP-Event- und eine für die Übertragung von LKr-MEP-Event-Objekten. Die L1-Prozessierungen haben aufgrund der zeitlichen Restriktionen (siehe Abschnitt 3.1) eine höhere Priorität. Daher führt jeder Eventbuilder-Thread so lange ein Polling der MEP-Event-Warteschlangen mit anschließender L1-Prozessierung durch, bis all diese Warteschlangen leer sind. Erst dann werden die LKr-MEP-Event-Warteschlangen ausgelesen, wobei nur ein einziges LKr-MEP-Event-Objekt verarbeitet wird (L2-Trigger-Algorithmus), so dass anschließend wieder die MEP-Event-Warteschlangen geleert werden. Auf diese Weise wird verhindert, dass L2-Trigger-Algorithmen durchgeführt werden, obwohl noch Daten für die L1-Prozessierung vorhanden sind.

Wenn jetzt jedoch bei jeder positiven L1-Trigger-Entscheidung die LKr-Daten angefordert würden, könnte dies zu einem schnellen Überlaufen der LKr-MEP-Event-Warteschlange führen. Aus diesem Grund wurde ein Algorithmus zur dynamischen Fluss-

steuerung entwickelt. Dabei überträgt jeder Eventbuilder-Thread die L1-Trigger-Entscheidungen, unabhängig von der Last, an den L1-Verteiler-Thread. Sobald eine der LKr-MEP-Event-Warteschlangen zwischen Packethandlern und Eventbuildern eine konfigurierbare Größe (erster Parameter) überschritten hat, pausiert der L1-Verteiler-Thread das Senden von MTPs. Zusätzlich erlaubt ein zweiter Parameter die Pausierung der MTP-Verteilung für den Fall, dass die Empfangsrate durch die Netzwerkkarte eine gewisse Datenrate überschreitet.

Solange diese beiden Schwellwerte unterschritten sind, sendet der L1-Verteiler-Thread MTPs mit einer konfigurierbaren maximalen Anzahl an L1-Trigger-Entscheidungen pro MTP (dritter Parameter). Prinzipiell möchte man dabei möglichst viele Trigger-Entscheidungen gleichzeitig senden um Ineffizienzen durch die Übertragung der Kopfdaten zu verhindern. Allerdings ist die Datenmenge, welche nach einem MTP von den CREAM-Boards zu dem Farm-PC versendet wird, proportional zur Anzahl positiver Trigger-Entscheidungen innerhalb des MTPs. Mann muss also sicherstellen, dass die CREAM-Boards das Netzwerk nicht temporär überlasten. Für diesen Zweck ist die Implementierung der in Abschnitt 2.2.5.4 eingeführten Flusskontrolle innerhalb der Firmware der CREAM-Boards vorgesehen. Daher ist aus Sicht des Frameworks der dritte Parameter nur durch die MTU begrenzt.

## 4.8. Das Überwachungssystem

Zusätzlich zu den in Abschnitt 4.5 eingeführten Threads wurde ein, als Überwachungssystem bezeichneter, Dienst implementiert. Dieser Thread sammelt regelmäßig Statistiken zum aktuellen Zustand der Farm-Software und sendet diese an einen zentralen Datenserver, welcher von allen Farm-PCs genutzt wird. Bisher werden dabei folgende Informationen gespeichert:

- Empfangsrate (Datenmenge, Anzahl an Paketen und Ereignissen) in Abhängigkeit von der Quell-ID.
- Statistische Verteilung der Trigger-Typ-Worte.
- Senderate der MTPs.
- Ereignis- und Datenrate der Speicherung akzeptierter Ereignisse.
- CPU-Last.
- Relativer Paketverlust.
- Warnungen und Fehlermeldungen.

Wie in Abschnitt 1.2.3 erläutert, wird ein Burst und somit der Empfang von MEPs nur einige Sekunden andauern. Daher sollte die zeitliche Auflösung des Überwachungssystems höchstens eine Sekunde betragen. Dem Autor waren ausschließlich Systeme zur Speicherung und Visualisierung bekannt, welche eine maximale zeitliche Auflösung von 10 Sekunden ermöglichen[23]. Aus diesem Grund wurde ein eigenes Verfahren konzipiert. Dabei stellt das Überwachungssystem eine Schnittstelle zu einer MySQL-Datenbank dar. Innerhalb der Farm-Software kann dabei mit nur einer Zeile Quelltext und ohne SQL-Kenntnisse eine neue überwachte Variable hinzugefügt werden.

Zur Visualisierung der Daten wurde eine Webanwendung unter Verwendung von **G**oogle **W**eb **T**oolkit (GWT) implementiert [25]. Der Vorteil von GWT ist, dass dabei nur die weit verbreitete Programmiersprache Java verwendet werden muss und keine Kenntnisse über HTML oder CSS benötigt werden. Eine Weiterentwicklung dieser Software ist somit leicht durchführbar.

## 4.9. Evaluation

Da während dieser Arbeit noch keine L1- und L2-Algorithmen zur Verfügung standen, wurden diese durch eine Summenbildung aller empfangenen Bytes simuliert. Bei diesem Prozess werden sowohl Zugriffe auf den Systemspeicher, als auch auf den lokalen Prozessor-Cache, ausgeführt. Dabei wurde für Testzwecke die Summenbildung $N$-fach wiederholt. Um eine realistische Datenrate zu erhalten, wurde jedes Ereignis mit einer Wahrscheinlichkeit von 10% durch den simulierten L1-Trigger-Algorithmus akzeptiert und mit der gleichen Wahrscheinlichkeit wurden die übrigen Ereignisse auf die lokale Festplatte geschrieben (positive L2-Trigger-Entscheidungen).

Für die Evaluation des Frameworks wurden 11 Detektoren mit jeweils 10 TEL62-Boards sowie 400 CREAMs simuliert und die verwendeten Warteschlangen sehr klein gewählt (1000 Slots). Sobald die simulierten Trigger-Algorithmen durch ein zu hohes $N$ für die einzelnen Ereignisse zu lange brauchen, stauen sich neue Ereignisse, bis alle Warteschlangen bis zum Polling-Thread gefüllt sind und somit dieser empfangene Pakete nicht mehr weiterleiten kann. Dies macht sich also durch Paketverluste bemerkbar.

Zur Optimierung der insgesamt 32 Parameter des Frameworks wurde jeweils ein Burst mit $2 \cdot 10^5$ Ereignissen simuliert und $N$ solange erhöht, bis Paketverluste auftraten. Nach einigen Iterationen konnte so für ein maximales $N^{\max} \approx 240 \pm 10$ eine Verlustfreie Prozessierung der Daten realisiert werden (unter Verwendung von Dell1) . Bei der

[23] Diese Programme lauten Cacti, Opsview, Munin, Ganglia und Cricket. Alle basieren jedoch auf dem Datenbanksystem RRDtool, welches bisher nur eine maximale zeitliche Auflösung von 10 s ermöglicht.

so erhaltenen Konfiguration und deaktivierten Trigger-Algorithmen wurde eine CPU-Last von $(25 \pm 2)\%$ der Gesamtkapazität erreicht. Dies stellt allerdings nur eine obere Grenze der verwendeten Kapazität dar, da bei deaktivierten Trigger-Algorithmen die Eventbuilder-Threads häufig schlafen gelegt werden und durch die dazu nötigen Systemaufrufe (usleep) eine zusätzliche Last entsteht. Zudem werden z.Z. rund 4% der Gesamtkapazität für die Kontrolle der Prüfsummen benötigt. Laut Luca Deri kann diese Berechnung zukünftig durch eine Erweiterung von pf_ring DNA von der Netzwerkkarte übernommen werden, so dass zukünftig voraussichtlich rund 79% der Gesamtkapazität des Dell1 für die Prozessierung der Trigger-Algorithmen verwendet werden kann.

Bei dem gleichen Test mit Dell2 als Farm-PC kam es bereits bei $N_2 \approx 220 \pm 10$ zu Datenverlusten. Bei deaktivierter Trigger-Simulation ($N_2 = 0$) wurde jedoch innerhalb der Fehler die gleiche CPU-Last (ca. 26%) gemessen. Dies war zu erwarten, da die Prozesse innerhalb des Frameworks hauptsächlich Zugriffe auf den Systemspeicher durchführen. Wie in Abschnitt 4.3 beschrieben, verwenden Dell1 und Dell2 jedoch den gleichen Speicher. Ob sich daher der Kauf von High-End-Prozessoren für die endgültige PC-Farm rentiert, müssen zukünftig Tests mit den Prototypen der ersten Trigger-Algorithmen zeigen[24].

Wie hoch die prozessierbare Ereignisrate eines Farm-PCs sein wird, ist vor allem von den endgültig verwendeten Trigger-Algorithmen abhängig. Durch die Homogenität der PC-Farm ist diese jedoch vollkommen skalierbar. Durch die in Abbildung 3.7 dargestellte Baum-Topologie kann zudem eine hohe Skalierbarkeit des Netzwerkes realisiert werden, so dass die Rechenkapazität der PC-Farm leicht durch den Kauf weiterer PCs zukünftig leicht angepasst werden kann.

## 4.10. Zukünftige Ergänzungen

Da erst Ende März 2012 einige Änderungen der verwendeten Farm-Protokolle eingeführt wurden, muss die aktuelle Version des Frameworks angepasst werden. Dabei handelt es sich hauptsächlich um kleine Änderungen im Format der Kopfdaten, welche sich nicht auf die Performance der Software auswirken. Allerdings wird z.Z. noch darüber diskutiert, ob die Übertragung fragmentierter IP-Datagramme benötigt wird (siehe Abschnitt 2.3.1.1). Das Framework realisiert bisher noch keine Reassemblierung, jedoch arbeitet Luca Deri an einer proprietären Implementierung des IP-Protokolls innerhalb des pf_ring DNA Treibers. Sollte eine Fragmentierung zukünftig benötigt werden, so wird dies ebenfalls mit nur wenig Aufwand realisierbar sein.

[24] Tests mit dem häufig verwendeten Benchmark-Programm LINPACK haben gezeigt, dass Dell1 mit ca 102 GFLOPs ca. 24% mehr Gleitkommaoperationen pro Sekunde (FLOPs) bewältigt als Dell2 (82 GFLOPs). Rechenintensivere Trigger-Algorithmen könnten daher vom schnelleren Prozessor des Dell1 profitieren.

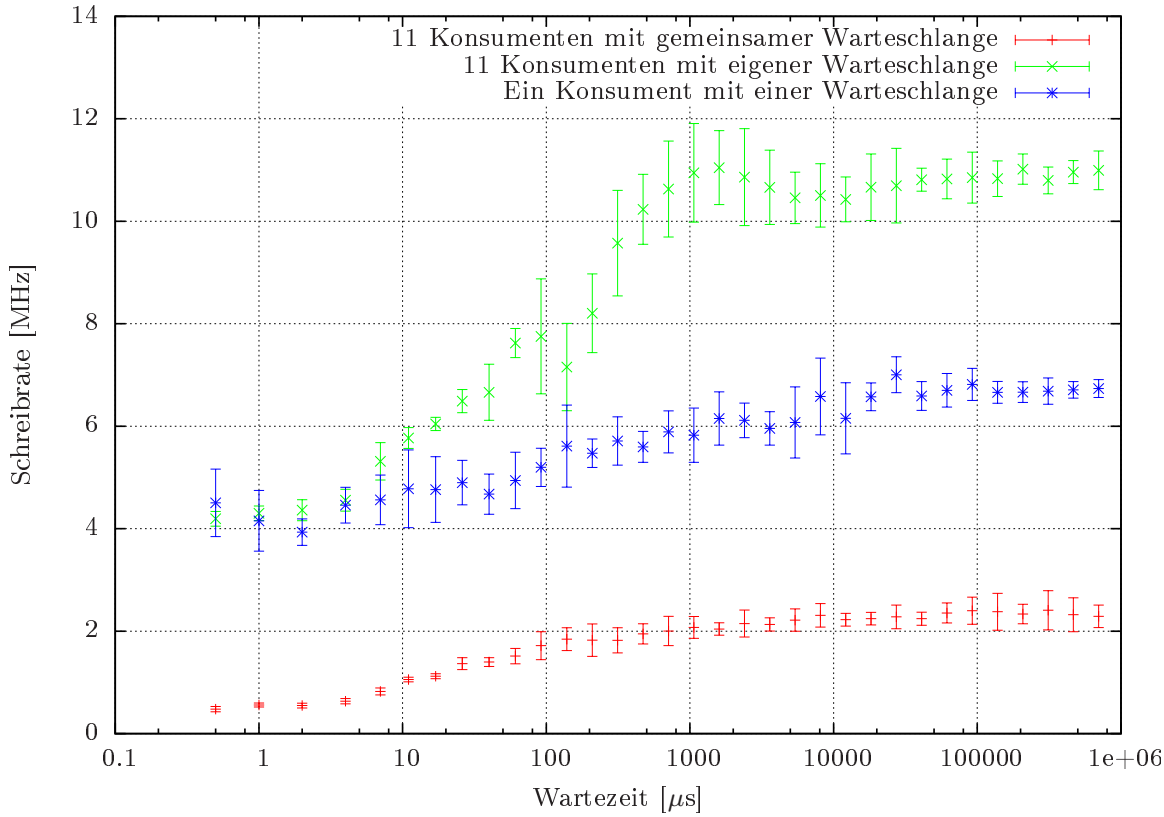


**Abbildung 4.8.:** Übertragungsraten von Zeigern von einem Thread zu 11 Konsumenten-Threads unter Verwendung von `std::queue` und Mutexen. Dabei wurde die Warteschlange aktiv „gepollt“ und im Falle einer leeren Warteschlange der Polling-Thread eine gewisse Zeit schlafen gelegt (siehe Abszisse). Die beiden Werte unterhalb von 1 $\mu s$ entsprechen einem aktiven Polling ohne Wartezeit zwischen zwei Polls.

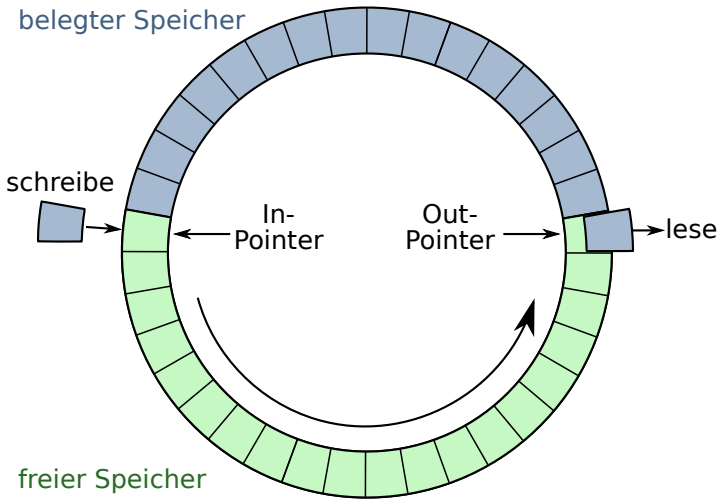


**Abbildung 4.9.:** Das Prinzip eines Ringpuffers: Bei jedem Lese- oder Schreibzugriff werden der Out- oder In-Pointer um eins erhöht. Diese Zeiger laufen hier im entgegengesetzten Uhrzeigersinn.

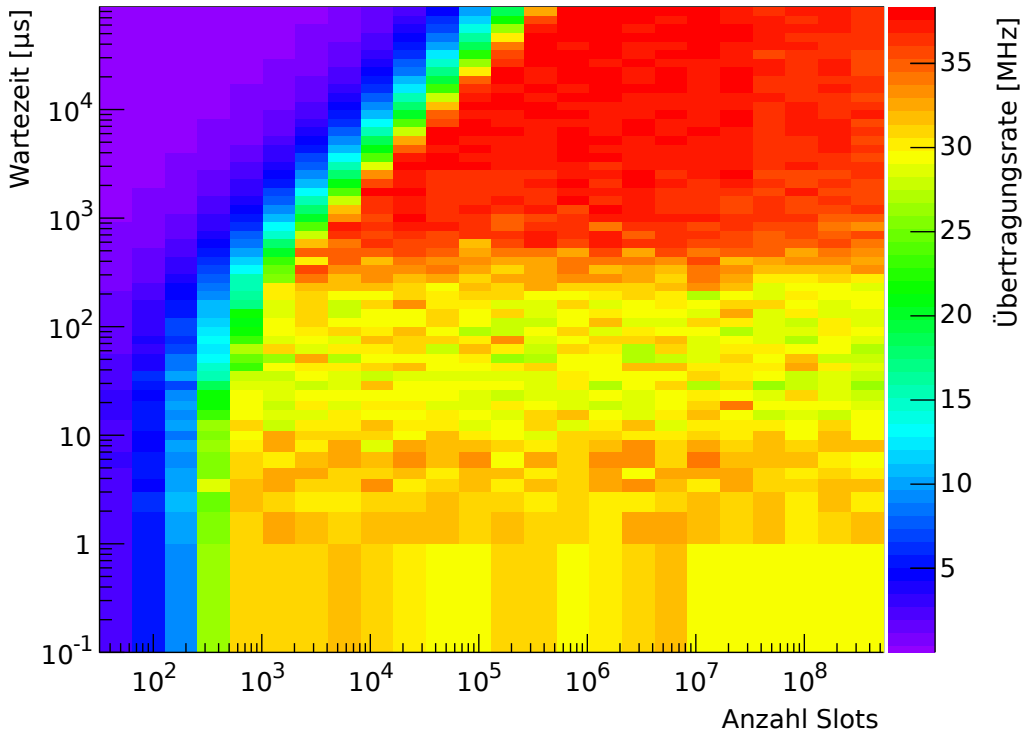


**Abbildung 4.10.:** Übertragungsraten von Zeigern von einem Produzenten- zu 11 Konsumenten-Threads unter Verwendung von lockfreien Warteschlangen bei aktivem Polling. Dabei entspricht jeder Datenpunkt der mittleren Rate innerhalb einer fünfsekündigen Übertragung.

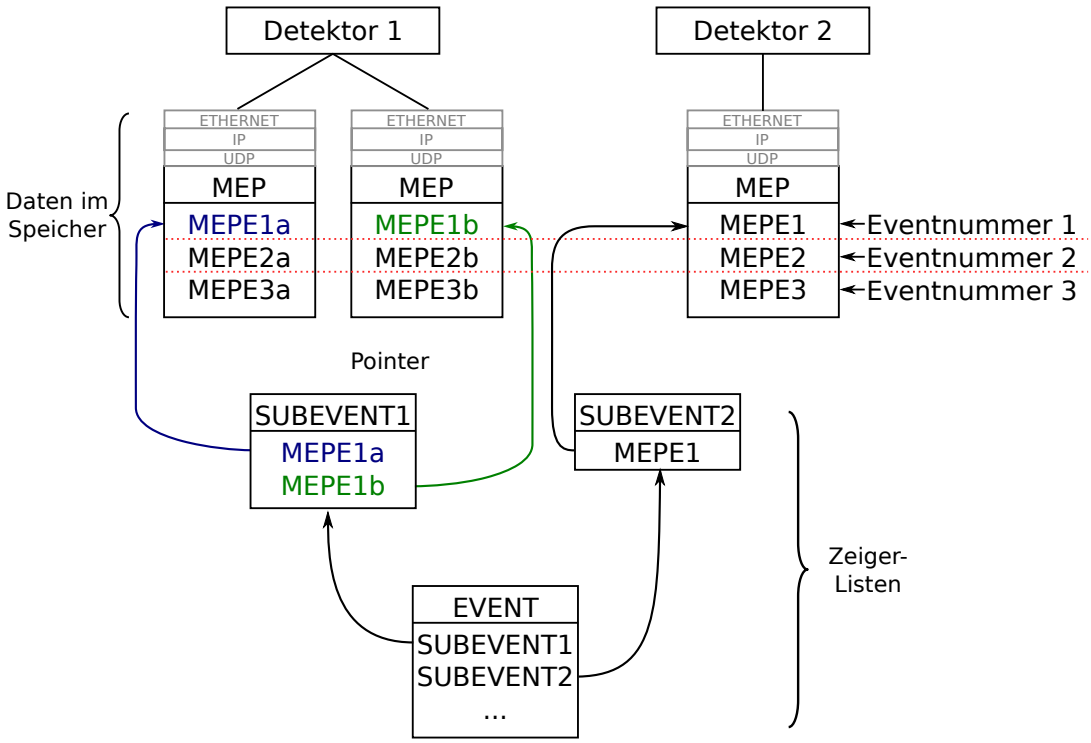


**Abbildung 4.11.:** Das Datenmodell zum (L1-)Eventbuilding.

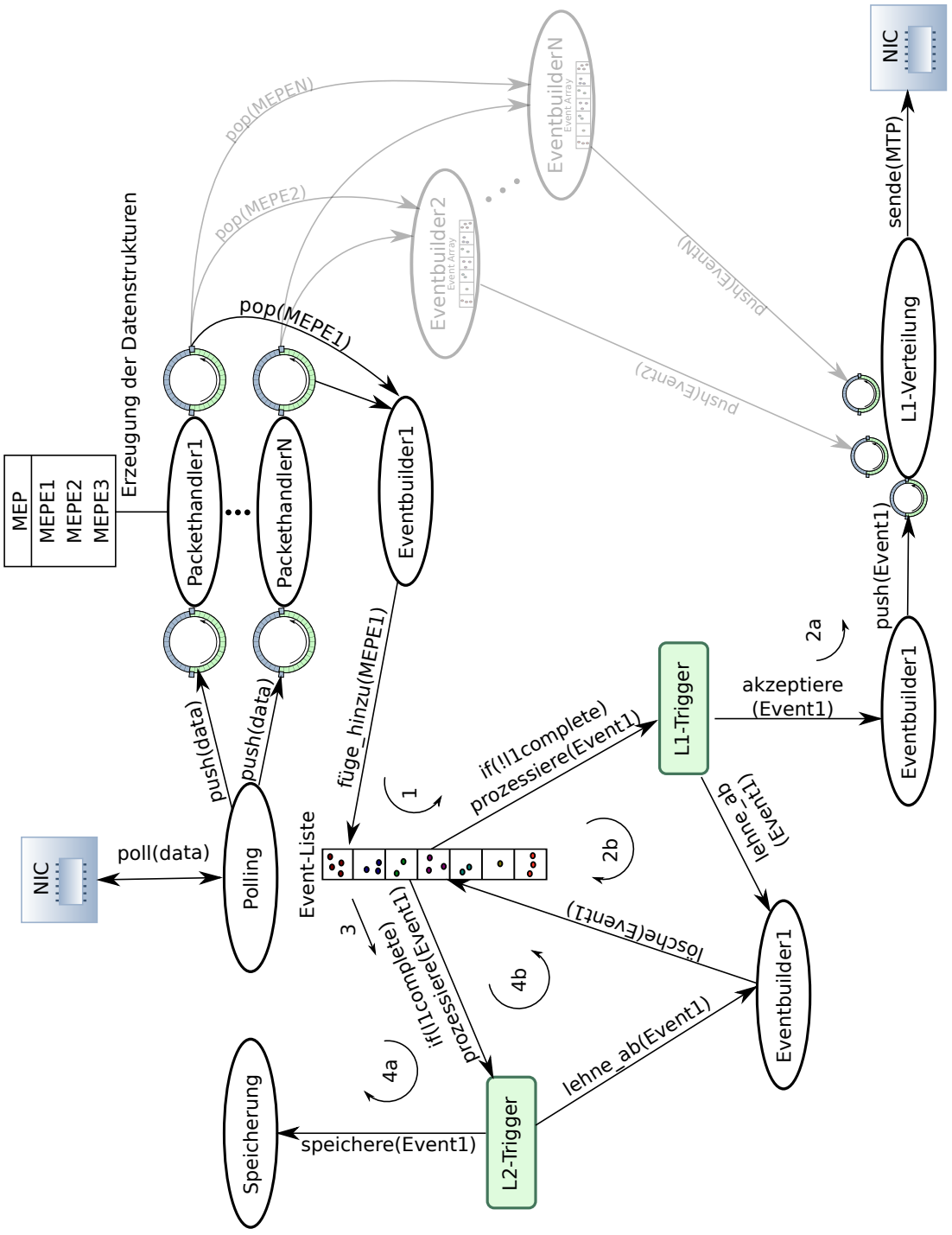


**Abbildung 4.12.:** Schematische Darstellung des Datenstroms innerhalb des Frameworks. Ellipsen beschreiben jeweils einen Thread. Die grünen Rechtecke stellen die Schnittstelle zu den Trigger-Algorithmen dar. Diese werden dabei innerhalb des Eventbuilder-Threads ausgeführt. Zur besseren Übersichtlichkeit wurden die Netzwerkkarte (NIC) und ein Eventbuilder-Thread mehrfach eingezeichnet. Dabei handelt es sich jedoch um das jeweils selbe Objekt.

# Kapitel 5.

# Zusammenfassung und Ausblick

Durch eine ausführliche Planung des Konzepts für die Online-PC-Farm des NA62-Experiments konnte eine deutliche Effizienzsteigerung erreicht werden. Dabei wurden zunächst die beiden Software-Trigger-Stufen nur noch als logisch getrennte Systeme betrachtet, wodurch sich direkt die Vorteile einer homogenen Farm zeigten: Das L1-Eventbuilding wird ermöglicht und die Implementierung und Verwaltung der Farm-Software wird deutlich erleichtert, da weniger verteilte Kommunikationen stattfinden müssen.

Das resultierende Konzept ermöglicht eine maximale Skalierbarkeit der PC-Farm und erlaubt schon innerhalb der L1-Trigger-Algorithmen Informationen zwischen verschiedenen Detektoren (LKr ausgenommen) zu verknüpfen.

Die bei der Implementierung des Frameworks aufgetretenen Schwierigkeiten werden beherrscht. Dabei konnten erste Evaluationen zeigen, dass die hohen Datenraten durch die Software bewältigt werden, und dabei noch ein großer Teil der CPU-Ressourcen zur Prozessierung der eigentlichen Trigger-Algorithmen verwendet werden können.

Im laufenden Jahr ist der Aufbau eines Prototypen der PC-Farm geplant. Dabei soll vor allem die Realisierbarkeit der Baum-Topologie mit mehreren kleineren Switchen getestet werden. Sobald einige der benötigten Trigger-Algorithmen vorhanden sind, kann die Anzahl benötigter PCs, welche bisher auf rund 30 geschätzt wurde, genauer bestimmt werden.

# Anhang A.

# Performance-Tests

## A.1. Interrupt-Koaleszenz

Der Begriff Interrupt-Koaleszenz wurde in Abschnitt 4.4.1 eingeführt. Um die Auswirkung dieses Parameters zu analysieren, wurden die gleichen Messungen wie in Abschnitt 4.4.1, bei einer Variation der Interrupt-Koaleszenz, durchgeführt. Eine Interrupt-Koaleszenz von 1 $\mu$s entspricht hierbei einem reservierten Wert. Dabei wird der Parameter abhängig von der aktuellen Last angepasst.

In den Abbildungen A.1 und A.2 ist zu sehen, dass bei beiden Protokollen eine maximale Datenrate im Bereich von $\mathcal{O}(100\ \mu\text{s})$ erreicht wird. Für kleinere Werte sinkt die Datenrate, da hier die Interruptrate steigt und somit mehr CPU-Last entsteht. Für höhere Werte sinkt die Datenrate, da nun zu viel Zeit zwischen zwei Interrupts vergeht und sich somit zu viele Daten im Netzwerkkarten-Puffer ansammeln, bis dieser überläuft und Pakete verloren gehen.

Bei beiden Protokollen entspricht der reservierte Wert 1 $\mu$s dem besten Wert oberhalb von 1 $\mu$s. Dies ist bei dem verwendeten Treiber[1] der Standard-Wert.

## A.2. Lockfreie Warteschlangen

Neben den in Abschnitt 4.5.1.1 dargestellten Evaluationen wurden die gleichen Testläufe nur zwei und vier Konsumenten-Threads durchgeführt. Die Ergebnisse sind in den Abbildungen A.3-A.5 zu finden.

[1] Die verwendeten Netzwerkkarten wurden mit dem ixgbe-Treiber betrieben.

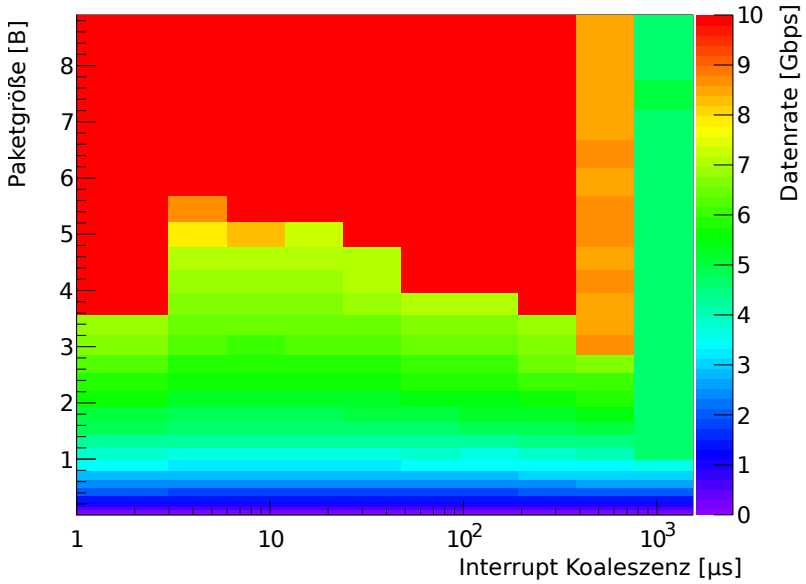


**Abbildung A.1.:** 10GbE-TCP-Performance abhängig von der Paketgröße und der Interrupt-Koaleszenz.

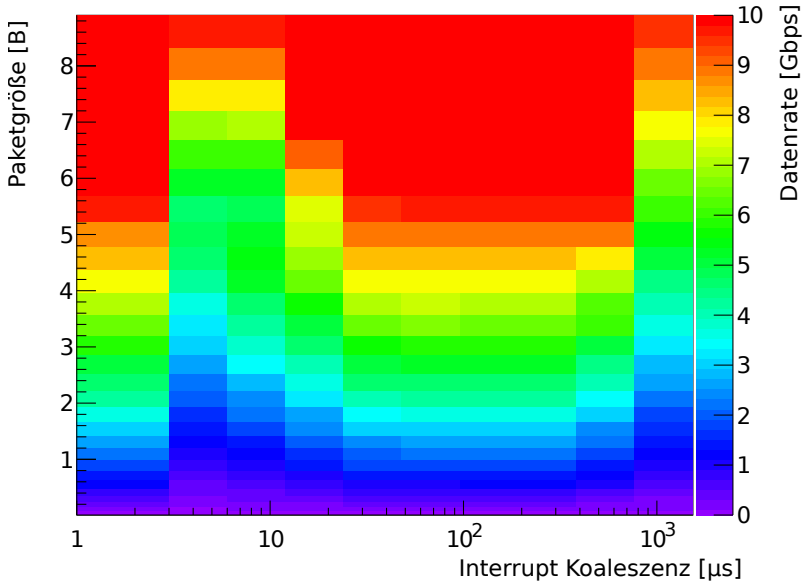


**Abbildung A.2.:** 10GbE-UDP-Performance abhängig von der Paketgröße und der Interrupt-Koaleszenz.

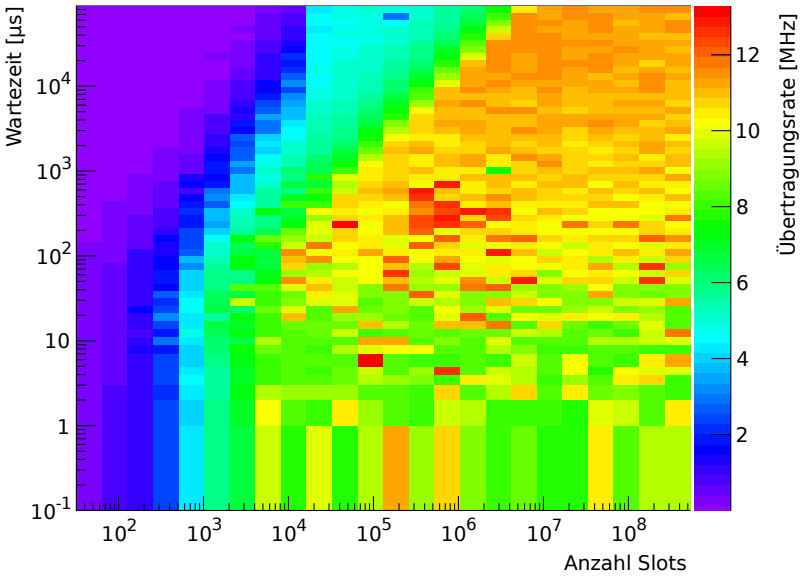


**Abbildung A.3.:** Übertragungsraten von Zeigern von einem Thread zu einem Konsumenten-Thread unter Verwendung von einer lockfreien Warteschlange bei aktivem Polling.

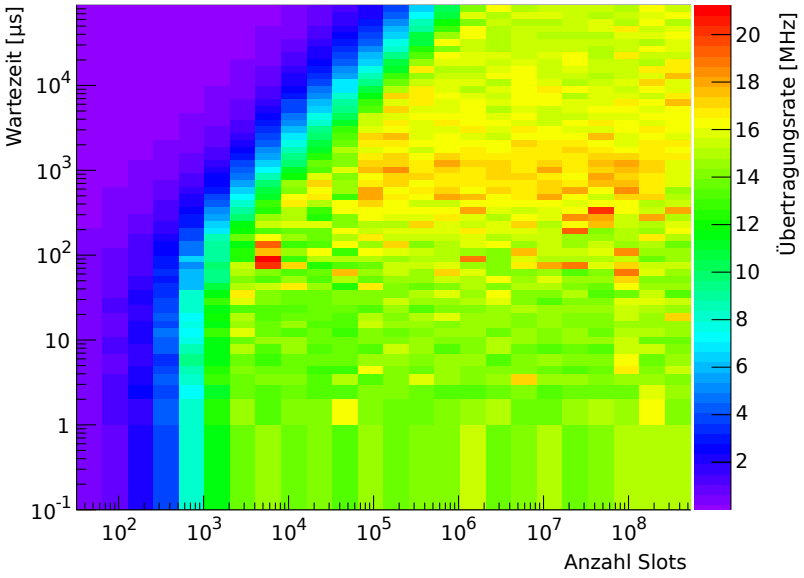


**Abbildung A.4.:** Übertragungsraten von Zeigern von einem Thread zu 2 Konsumenten-Threads unter Verwendung von lockfreien Warteschlangen bei aktivem Polling.

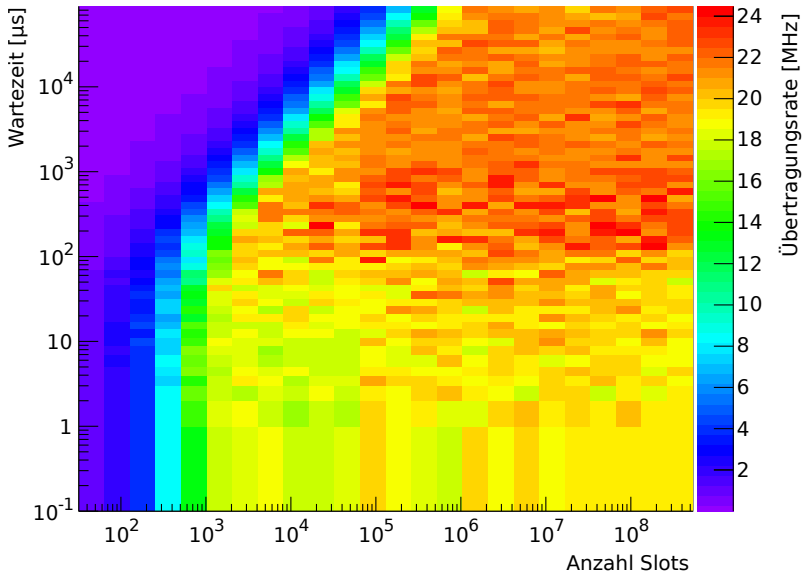


**Abbildung A.5.:** Übertragungsraten von Zeigern von einem Thread zu 4 Konsumenten-Threads unter Verwendung von lockfreien Warteschlangen bei aktivem Polling.

# Literaturverzeichnis


[1] B. W. Kernighan, P. J. Plauger. *The Elements of Programming Style 2nd Edition.* McGraw Hill, 1978.

[2] E. Rutherford. *The Scattering of $\alpha$ and $\beta$ Particles by Matter and the Structure of the Atom.* Philosophical Magazine, 1911.

[3] B. Povh et al. *Teilchen und Kerne.* Springer-Verlag, 2006.

[4] K. Nakamura et al (Particle Data Group). *Review of Particle Physics.* J. Phys. G: Nucl. Part. Phys. 37 075021, 2010.

[5] K. Freeman, G. McNamara. *In search of dark matter.* Springer-Verlag, 2006.

[6] A. J. Buras, F. Schwab, S. Uhlig. *Rev. Mod. Phys. 80 (2008) [hep-ph/0405132].*

[7] *Proposal to Measure the Rare Decay $K^+ \to \pi^+ \nu\bar{\nu}$ at the CERN SPS.* CERN-SPSC-2005-013, 2005.

[8] W.J. Marciano, Z. Parsa. *Phys. Rev. D 53 (1996) R1.*

[9] M. Antonelli et al. *An evaluation of $|V_{us}|$ and precise tests of the Standard Model from world data on leptonic and semileptonic kaon decays.* The European Physical Journal C - Particles and Fields, 2010.

[10] G. Buchalla, A. J. Buras. *Nucl. Phys. B 398 (1993) 285; 400 (1993) 225; 412 (1994) 106 [hep-ph/9308272].*

[11] M. Misiak, J. Urban. *Phys. Lett. B. 451 (1999) 161 [hep-ph/9901278].*

[12] J. Brod, M. Gorbahn, E. Stamou. *Phys. Rev. D 83.* 2011.

[13] F. Hahn et al. *Technical Design Document.* NA62 Kollaboration, Dezember 2010, `http://na62.web.cern.ch/na62/Documents/TD_Full_doc_v10.pdf`.

[14] International Electrotechnical Commission. *ISO/IEC 80000-13:2008.*

[15] Markus Kammermann. *CompTIA Network+.* Mitp-Verlag, 2010.

[16] IEEE.org. *Standards der IEEE.* `http://standards.ieee.org`.

[17] Dr. Gregor Schiele. *Vorlesungsskript Kommunikationsnetze, SS 2010.*

[18] www.golem.de. *Die letzten IPv4-Adressen werden verteilt.* `http://www.golem.de/1102/81104.html`, 2011.

[19] B. Jost and N. Neufeld. *LHCb Technical Note - EDMS 499933.* 2003.

[20] C. Li, C. Ding, K. Shen. *Quantifying The Cost of Context Switch.* `http://www.cs.rochester.edu/u/cli/research/switch.pdf`.

[21] Intel Corporation. *Interrupt Moderation Using Intel GbE Controllers.* `http://www.intel.com/content/dam/doc/application-note/gbe-controllers-interrupt-moderation-appl-note.pdf`, 2007.

[22] J. H. Salim, R. Olsson, A. Kuznetsov. *Proceedings of the 5th Annual Linux Showcase & Conference.* `http://static.usenix.org/publications/library/proceedings/als01/full_papers/jamal/jamal.pdf`, 2001.

[23] Webseite der Firma ntop. `http://www.ntop.org`.

[24] Autor unbekannt. *Creating a thread safe producer consumer queue in C++ without using locks.* msmvps.com, `https://msmvps.com/blogs/vandooren/archive/2007/01/05/creating-a-thread-safe-producer-consumer-queue-in-c-without-using-locks.aspx`, 2007.

[25] Google Web Toolkit (GWT). `https://developers.google.com/web-toolkit`.