Entwurf und Implementierung eines Monitors für Dienstqualitäten

Studienarbeit am Institut für Telematik
Fakultät für Informatik
Prof. Dr. Dr. h.c. Gerhard Krüger
Universität Karlsruhe (TH)

von

Roland Bless

Betreuer:

Dipl.-Inform. Claudia Schmidt
Dr. Jochen Seitz

23. August 1995

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Messen in Systemen 2

1.1.1 Meßwerkzeuge 2

1.2 Kommunikations-Modelle 5

1.3 Motivation 6

1.4 Ziele 9

1.5 Aufbau 9

2 Grundlegende Anforderungen **11**

2.1 Randbedingungen 11

2.2 Anforderungen an einen QoS-Monitor 14

2.2.1 QoS-Parameter 15

Durchsatz 15

Ende-zu-Ende Verzögerung 16

Verzögerungsschwankung (Delay-Jitter) 16

Fehler 19

2.2.2 QoS-Parameterklassen 20

2.2.3 Integration des QoS-Monitors 23

2.2.4 Sende- und Empfangsseite 25

2.2.5 Meßintervall und Auswertung 26

Meßintervall 26

Auswertung 30

2.2.6 Messung der QoS-Parameter 31
Durchsatz 31
Verzögerung 32
Verzögerungsschwankung 33
Fehler 34
2.2.7 Aspekte des Multicasting 34
2.2.8 Monitor-Schnittstellen 35
MP-Schnittstelle 35
MM-Schnittstelle 36
2.3 Zusammenfassung der Merkmale 37

3 QoS-Monitoring und Netzwerkmanagement 39
3.1 OSI Netzwerkmanagement 39
3.2 Integration einer QoS-Architektur 40
3.3 SNMP 42
3.4 QoS-Monitoring mit SNMP 42

4 Implementierung 47
4.1 Generische Implementierung 47
4.1.1 Protokollerweiterung 47
Zeitstempel 47
4.1.2 Sendeseite 50
4.1.3 Empfangsseite 51
4.2 Konkrete Implementierung 54
4.2.1 OSF/1 54
4.2.2 Verwendete Objekte 55
Kommunikationsobjekte 56
QoS-Parameterobjekte 56
Ablauf 59
4.2.3 PATROCLOS 59
Notwendige Änderungen 59
4.3 Mögliche Erweiterungen 61

5 Zusammenfassung und Ausblick **63**

Literaturverzeichnis **65**

Index **69**

Abbildungen

1.1 Aufbau eines Meßinstruments 3
1.2 Das 3-Komponentenmodell 6
1.3 Einsatz des Monitoring 7

2.1 Verbindungsaufbau - QoS-Verhandlung 12
2.2 Delay-Jitter und Raten-Jitter 18
2.3 Verzögerungsschwankung und absolute Verzögerung 19
2.4 Gleitendes Fenster vs. springendes Fenster 22
2.5 Ein QoS-Monitoring Modell 23
2.6 Beziehung zwischen System- zu Schichten-QoS-Instanzen 25
2.7 Ereigniszeitpunkte werden auf Uhrauflösung gerundet 27
2.8 Durchsatzmessung mittels statischem und dynamischem Intervall . . 28
2.9 Schwellwerte und Grenzen der QoS-Parameter 31

3.1 Netzwerkmanagement - Agenten und Manager 40
3.2 Eine QoS-Architektur basierend auf Netzwerkmanagement 41
3.3 konzeptionelle Tabelle für TCP-Verbindungen 44
3.4 QoS-Monitoring mit SNMP 45

4.1 Ablaufschema der Protokoll- und Monitor-Threads 52
4.2 Monitor als Fließband organisiert 53
4.3 PATROCLOS Architektur 62

1. Einleitung

Der Dienstqualität (auch Dienstgüte, engl. *Quality of Service*, kurz *QoS*, sprich: „Kwoss“) kommt vor dem Hintergrund der Hochgeschwindigkeitskommunikation und Diensteintegration eine wichtigere Bedeutung zu, als dies bisher in traditionellen Kommunikationssystemen[1] der Fall war. Diese Kommunikationssysteme bieten den Benutzern verschiedene *Dienste* an, z.B. zum Datenaustausch, zur Bewegtbildübertragung, Sprachübertragung usw. Die *Dienstqualität* bezeichnet dabei im Sinne des ISO/OSI Modells eine Menge von Eigenschaften, die dem (N)-Dienstnutzer aus seiner Sicht vom (N)-Dienst angeboten werden. Damit sind qualitative und quantitative Parameter gemeint, die den (N)-Dienst näher spezifizieren. Als Beispiele für Dienstgüteparameter seien hier kurz Datenverlust, Fehlerwahrscheinlichkeiten für Verbindungsauf- und -abbau, Durchsatz, Verzögerungszeit, Schutz, Kosten, etc. genannt. In Zukunft sollen verschiedene Dienste, wie etwa Video-, Audio-, Telefon-, Telefax- und Datendienste, mittels einer einzigen Netztechnologie (Breitband-ISDN) angeboten werden (Diensteintegration). Diese verschiedenen Dienste verlangen auch sehr unterschiedliche Werte und Verpflichtungen für die Dienstgüteparameter. Für viele Anwendungen, die durch die Hochgeschwindigkeitsnetze erst möglich werden, ist es nötig, daß die Einhaltung verschiedener Dienstgüteparameter-Vorgaben *garantiert* wird. So z.B. bei multimedialen Anwendungen: ein unkomprimierter Videostrom benötigt einen *garantierten* minimalen Durchsatz und garantierte maximale Verzögerungsschwankung (Delay-Jitter), die Anwendung kann jedoch einen gewissen Verlust an Daten tolerieren. Nun ist die vom Netzwerkbetreiber bzw. Dienstanbieter erbrachte Dienstqualität aufgrund der Eigenschaften der Hochgeschwindigkeitsnetze mit Schwankungen versehen (z.B. Paketverluste durch „Verkehrsstau“ in Switches[2], dadurch bedingte Nachrichtenverluste der „höheren“ Transportschicht). Die (sich verändernde) Dienstgüte muß also ständig beobachtet werden, damit die Protokollinstanzen rechtzeitig Korrekturen (an Ressourcen) vornehmen oder eine Herabstufung – eventuell sogar Neuverhandlung – der gewünschten Dienstqualität einleiten können. Die Aufgabe eines Dienstgüte-Monitors (*QoS-Monitor*) ist es nun, die entsprechenden QoS-Parameter zu beobachten und die Protokollinstanzen über die momentan vom System erbrachte Dienstqualität zu informieren. Damit über-

[1] nach [LoKK 93] ist ein Kommunikationssystem ein System, daß sich auf die ordnungsgemäße Abwicklung der Kommunikationsfunktionen innerhalb eines Informationssystems konzentriert.

[2] Netzwerkelement, das für das Weiterleiten der Pakete zuständig ist.

wacht der Dienstgüte-Monitor einmal die Einhaltung der Dienstgüte-Garantie und hilft gleichzeitig diese Garantie zu erbringen.

1.1 Messen in Systemen

Nach [Ferr 78] hat jedes System[3], das entworfen wird, eine Menge von Funktionen und einige vorherbestimmte Leistungsmerkmale zu erfüllen. Die wichtigste Bedingung ist, daß die Funktionen *korrekt* ausgeübt werden. Leistungsspezifikationen beziehen sich darauf, *wie gut* ein System seine Funktion erfüllt. Die Wahl der Leistungsanforderungen ist oft subjektiver, als die der Korrektheitsspezifikation, denn die verschiedenen Aspekte der Leistung werden je nach Betrachter unterschiedlich stark gewichtet. Oftmals läßt sich keine klare Grenze zwischen Korrektheit und Leistung ziehen, so ist z.B. für multimediale Dienste die tatsächlich erbrachte Leistung (Dienstgüte) wesentlich. Für den Gebrauch ist daher die Leistung genauso wichtig wie die Korrektheit, die allein wohl in den seltensten Fällen genügt, um ein System akzeptabel zu machen. Unter *Leistung* (engl. performance) wird in [Ferr 78] verstanden, *wie gut* ein System (unter Annahme der Korrektheit) arbeitet. Da es oft möglich ist, die subjektiven Definitionen der Leistung in technische Ausdrücke umzusetzen und dann zu quantifizieren, kann die Leistung objektiv ermittelt werden. Aktivitäten zur *Leistungsermittlung* haben das Ziel einer Beurteilung der Leistung. [Ferr 78] unterscheidet zwischen *Studien zur Leistungsbewertung* und *kontinuierlichen Monitoring-Aktivitäten*. Letztere dienen dazu, *die Leistung eines Systems zu beobachten*, um Leistungsprobleme sofort nach Auftreten zu entdecken und werden normalerweise für einen wesentlichen Abschnitt der Lebensdauer eines existierenden, laufenden Systems eingesetzt. [Ferr 78] geht dann hauptsächlich auf die Studien zur Leistungsbewertung (die kein existierendes System voraussetzen und meist für kürzere Zeit eingesetzt werden, nachdem die Existenz eines Leistungsproblems entdeckt wurde) ein, beschäftigt sich aber auch mit Meßtechniken, auf die hier weiter eingegangen wird.

1.1.1 Meßwerkzeuge

Aufgrund der diskreten Natur der Computersysteme messen *Meßwerkzeuge* Parameter durch Wahrnehmen von *Ereignissen*. Ein Ereignis ist durch einen *Systemzustandswechsel* charakterisiert. Ein Meßwerkzeug muß in der Lage sein, Ereignisse wahrzunehmen und aufzuzeichnen. In vielen Fällen werden die zu messenden Parameter bei Eintreten des Ereignisses oder später anhand der gesammelten Daten

[3] unter System ist hier hauptsächlich Computersystem gemeint, man kann aber auch Kommunikationssystem dafür einsetzen.

berechnet. Daher sind viele Messungen indirekt, weil die Daten durch einen Datenreduktionsprozeß erhalten werden. Eine solche Reduktion ist z.B. die Ermittlung des Durchsatzes durch Zählen bestimmter Ereignisse (Ankunft eines Paketes) innerhalb eines Intervalls. Bei einer Echtzeitselektion werden nur wichtige Ereignisse herausgefiltert und unnötige verworfen. Ein *Software-Ereignis* ist nach [FeSZ 83] mit einer Programmfunktion assoziiert und tritt auf, wenn das Programm eine bestimmte Stelle bei seiner Ausführung erreicht. Ein *Hardware-Ereignis* ist durch das Auftauchen ein oder mehrerer Signale in den Schaltkreisen einer Systemkomponente definiert und unabhängig vom logischen Gehalt des Programms das gerade ausgeführt wird (z.B. die Bewegung eines Plattenarms). Hardware-Ereignisse können oft durch Software erkannt werden, da sie meist mit einer Inhaltsänderung einer Speicherstelle einhergehen, was wiederum ein Software-Ereignis darstellt.

Grundsätzlich besteht jedes Meßinstrument aus drei Teilen mit unterschiedlicher Funktionalität (siehe Abbildung 1.1): Der *Sensor* hat die Aufgabe den Meßwert zu

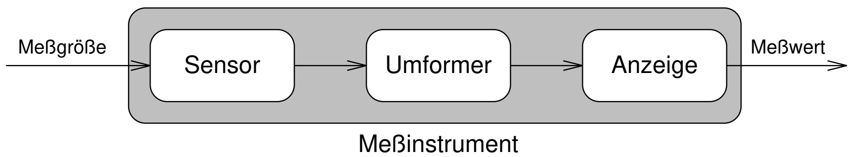


Abbildung 1.1: Aufbau eines Meßinstruments

erfassen, der *Umformer* führt eine Transformation auf den Meßdaten aus (kann auch entfallen) und die *Anzeige* ermöglicht das Ablesen des Ereignisses. Bei einem Thermometer beispielsweise wird die Temperatur durch die Kapillarflüssigkeit (Quecksilber, Alkohol) gemessen (Sensor), in Ausdehnung umgesetzt (Transformation) und die Höhe der Flüssigkeitssäule mittels einer Skala ablesbar gemacht (Anzeige). Bei einem Meßinstrument im Computer wird die Anzeige oft durch eine Speicherung der Daten ersetzt, da die Schnelligkeit mit der die Daten anfallen, nicht mehr in Echtzeit von einem Mensch abgelesen werden könnten. Der Umformer kann ein Ereignis vorverarbeiten oder reduzieren, indem nur ein Zähler bei Eintreten des Ereignisses hochgezählt wird. Auch das Auslesen einer Systemuhr oder Registern kann vom Umformer zusätzlich durchgeführt werden. Die Aufgabe des Sensors ist, das Ereignis zu erkennen.

Wenn ein Meßwerkzeug selbst Systemressourcen[4] benötigt, können die zu messenden Größen dadurch beeinflußt werden. Die Beeinflussung kann sowohl eine Leistungseinbuße des Systems als auch Ungenauigkeiten in der Messung bedeuten. Die Leistungseinbuße bezeichnet dabei jede Verschlechterung der Leistung bezogen auf die

[4] Unter (System-)Ressourcen versteht [Ferr 78] die Summe der Hard- und Softwarekomponenten, die ein funktionierendes Computersystem ausmachen. In deutscher Literatur wird für *technische Komponenten* auch der Begriff „Betriebsmittel“ verwendet, vgl. [Wett 93]. Dazu zählen also insbesondere Zentralprozessor, Arbeitsspeicher und Peripherie-Geräte.

Leistung *ohne Messung*. Es lassen sich folgende Hauptcharakteristiken eines Meßwerkzeugs ausmachen:

- Beeinflussung, Störung
  Jede Messung beeinflußt das Meßergebnis (hier durch gemeinsame Nutzung von Systemressourcen).
- Genauigkeit
  Fehler, Abweichung vom gemessenen Wert zur tatsächlichen Größe. Interne Genauigkeit der Darstellung von Meßergebnissen, etc.
- Auflösung
  Maximale Meßfrequenz mit der Ereignisse erkannt und aufgezeichnet werden können.
- Bereich
  Bezeichnet mögliche Ereignisklassen, die erkannt werden können.
- Vorverarbeitungsfähigkeiten
  Ableiten anderer Größen aus Ereignissen, wie z.B. Ereigniszähler. Andernfalls wird die genaue zeitliche Abfolge der Ereignisse – samt den Ereignissen selbst – festgehalten.
- Kosten
- Einfachheit der Installation
- Einfachheit im Gebrauch

Ein *Monitor* ist nach [FeSZ 83] ein Mechanismus, um Informationen über die Systemaktivität zu sammeln und er besteht aus einer Menge von Anweisungen und Daten, die *Software-Ereignisse* registriert. Ein Software-Ereignis kann durch hinzufügen von Anweisungen (trap code, hook) an die entsprechenden Stellen eines Programms erkannt und durch einen damit verbundenen Aufruf des Monitors registriert werden. Ein *ereignisgesteuerter* Monitor wird durch das Eintreten eines Ereignisses aktiviert. [Ferr 78] teilt Meßwerkzeuge in drei Klassen ein: *Hardware-Werkzeuge* bestehen aus zusätzlicher Hardware zum Messen der Größen, *Software-Werkzeuge* stellen zusätzliche Befehle dar, die vom System zur Messung auszuführen sind und Firmware-Werkzeuge sind Mikro-Befehle, die ebenfalls zusätzlich vom System auszuführen sind. Oft werden *hybride* Werkzeuge (also eine Kombination obiger Werkzeuge) eingesetzt, um die Leistung von Systemen zu messen. Für die Aufgabe des Beobachtens der Dienstqualität bei der Kommunikation sind hier eigentlich nur Software-Monitore von Interesse, da die zu messenden Ereignisse eng an das Protokoll gekoppelt und somit per Software zugänglich sind.

Da Software-Monitore die Messung der Daten durch zusätzliche Befehle realisieren, beeinflussen sie in der Regel das Meßergebnis. Die Beeinflussung hängt von der Häufigkeit ab, mit der die Ereignisse auftreten und der Anzahl der darauf vom Monitor auszuführenden Operationen für jedes Ereignis. Es lassen sich Raum- und Zeitbeeinflussung unterscheiden: die Raumbeeinflussung entsteht durch die anfallenden Meßdaten und kann durch oben genannte Datenreduktionsmechanismen verringert werden. Die Zeitbeeinflussung entsteht durch Vorverarbeitung und Datenreduktion. Wie so oft stehen beide Arten (Raum, Zeit) im Widerstreit. Die Zeitungenauigkeiten können jedoch teilweise oder sogar ganz beseitigt werden, wenn die Zeit für die Verarbeitung gemessen und abgezogen wird. Die Auflösung der Software-Monitore ist geringer als bei Hardware-Monitoren und sie beobachten daher weniger häufige makroskopische Ereignisse. Sie haben allerdings den Vorteil, daß sie flexibler sind. In [Ferr 78] wird noch die Unterscheidung zwischen *internen* und *externen* Ereignissen getroffen: Interne Ereignisse sind z.B. Zugriff auf Datenstrukturen oder Variablen, sowie das Ausführen bestimmter Befehle bzw. Befehlstypen. So kann beispielsweise bei Passieren eines bestimmten Punktes (Checkpoint) die Systemuhr ausgelesen werden. Externe Ereignisse sind bestimmte Zeitpunkte zu denen dann eine Stichprobe (engl. sample) entnommen wird. Man nimmt Stichproben (das reduziert auch die anfallende Datenmenge) um einige wichtige Größen mit ihrer Hilfe zu *schätzen*. Das Auftreten der bestimmten Zeitpunkte (z.B. jedesmal wenn ein Zähler einen Wert überschreitet, in diesem Fall ist also ein internes Ereignis Auslöser; oder bei verstreichen eines bestimmten Zeitintervalls) veranlaßt den Monitor Daten zu sammeln, zu transformieren und zu speichern. Die anfallenden Daten werden *Beobachtungen* genannt. Die Zeitintervalle können von konstanter Länge sein oder zufällig gewählt werden. Auf die gewonnenen Stichproben können statistische Verfahren angewendet werden (z.B. Ermittlung eines Vertrauensintervalls).

1.2 Kommunikations-Modelle

Obwohl im Rahmen dieser Arbeit die Begriffe des ISO/OSI-Basis Referenzmodells verwendet werden, wird auf keine strenge Einhaltung dieser Schichtenarchitektur Wert gelegt (obwohl sich letztere aufgrund von Normungen noch in vielen Arbeiten wiederfindet). Man sollte heute eher von einer Schichtung nach dem 3-Komponentenmodell (vgl. [Zitt 91]) ausgehen (siehe Abbildung 1.2). Es besteht aus drei Komponenten: eine *anwendungsbezogene Komponente* (in etwa OSI Schichten 5-7), eine *transportorientierte Komponente* (OSI Schichten 2b-4) und eine *netzwerkbezogene Komponente* (OSI Schichten 1-2a). Das Hauptaugenmerk für den Einsatz des QoS-Monitors wird in dieser Arbeit auf der transportorientierten Komponente liegen.

Das ISO/OSI Basis-Referenzmodell (7-Schichtenmodell) hat u.a. folgende Nachteile:

- Redundante Funktionalität in mehreren Schichten

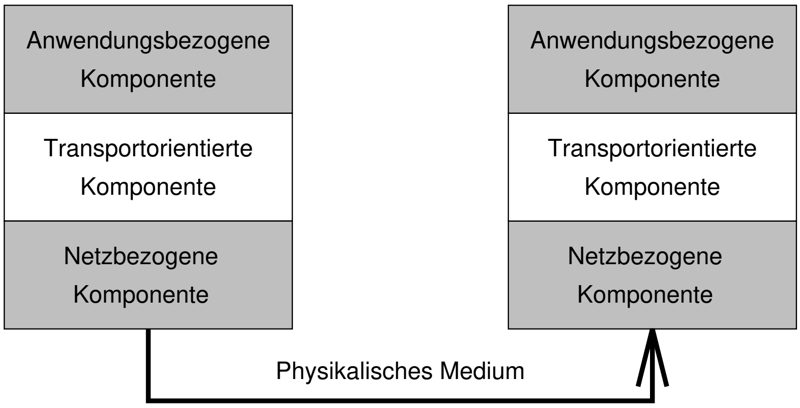


Abbildung 1.2: Das 3-Komponentenmodell

(z.B. Segmentieren u. Reassemblieren, Fehlererkennung, Multiplexing u. Demultiplexing).

- Hoher Implementierungsaufwand, da viele Schnittstellen zwischen den Schichten zu implementieren sind, sowie Leistungsverlust durch Schnittstellen (z.B. Funktionsaufrufe, Kopieren der Daten oder Maßnahmen zur Flußkontrolle an Schichtengrenzen).
- Schichtenlokale statt systemglobale Optimierung wird angestrebt.
- Schichteninterne Kommunikation
  Großer Protokolloverhead durch Einpacken der Dienstdateneinheiten und hinzufügen von protokollinterner Information in jeder Schicht.
- Nur eingeschränkte Auswahl der Dienstqualität möglich.

Die Einrichtung von Unterschichten (sublayers, z.B. LLC/MAC[5] der Schicht 2), um Industriestandards wie Ethernet oder FDDI[6] zu integrieren, deutet ebenfalls auf Inflexibilität des starren Schichtenmodells hin.

1.3 Motivation

Monitoring bezeichnet nach Abschnitt 1.1.1 die *Beobachtung* bzw. Messung eines Systemzustandes. Dazu werden bestimmte Werte einiger Größen aus der Menge der für den Systemzustand relevanten Zustandsvariablen gemessen. Die Größen werden je nach Aspekt, an dem der Beobachter interessiert ist, ausgewählt. Die Beobachtung

[5]LLC= **L**ogical **L**ink **C**ontrol= Sicherungsmechanismen, MAC= **M**edium **A**ccess **C**ontrol= Zugriffskontrolle auf geteiltem Medium.

[6]**F**ibre **D**istributed **D**ata **I**nterface, meist optischer Doppel-Tokenring mit 100MBit/s.

der mit einem (Kommunikations-)Dienst verbundenen Dienstqualität ist aufgrund verschiedener Aspekte wichtig. Um:

- Reaktive Maßnahmen ergreifen zu können, falls die erbrachte Dienstqualität von der vereinbarten abweicht.
- Den Dienstanbieter überwachen zu können (schließlich wird der Benutzer für die genutzten Dienste in Abhängigkeit von der geforderten Dienstgüte zahlen müssen).

Die Aufgabe an sich ähnelt einem Regelkreis:

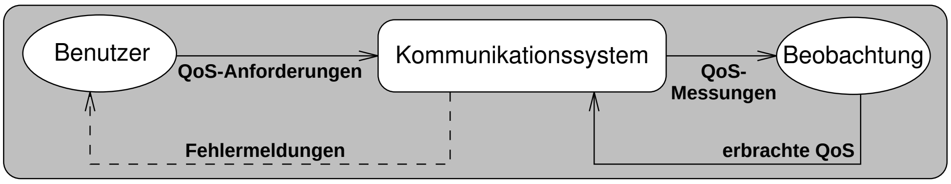


Abbildung 1.3: Einsatz des Monitoring

Ein Anwender wendet sich mit einer bestimmten Dienstqualitätsanforderung (QoS-Vorgabe, QoS-Anforderung) an das Kommunikationssystem. Dieses Kommunikationssystem (entspricht im 3-Komponentenmodell den beiden unteren Komponenten, d.h. den OSI-Schichten 1-4) bildet diese Anforderung auf entsprechende Kommunikationsparameter ab und versucht die verlangte Dienstgüte zu erbringen. Dazu muß die tatsächlich vom System erbrachte Dienstgüte überwacht werden. Dies geschieht mittels einer Messung, deren Beurteilung dem System mitgeteilt wird. Das System versucht im Falle eines Verfehlens der erwünschten Dienstgüte, Korrekturen vorzunehmen. Führen diese Korrekturmaßnahmen zu keinem Erfolg, informiert das System den Benutzer mit einer Fehlermeldung, der daraufhin möglicherweise eine neue Anforderung mit „geringerer“ Dienstqualität an das System stellt.

Besonders im Bereich der Hochgeschwindigkeitskommunikation und den damit verbundenen Anwendungen (wie z.B. verteilte Multimedia-Anwendungen) wird die erbrachte Dienstgüte und somit die „Leistung des Systems“ zu einem Aspekt der Funktionalität, bzw. Korrektheit. Denn ein System, welches zwar prinzipiell in der Lage ist, die geforderten Leistungen zu erbringen, sie praktisch aber nicht erreicht, wird für den Benutzer nur von geringem Nutzen sein. Daher müssen *Garantien* für die Dienstqualität gegeben werden können. Weil in einem Netz die Ressourcen jedoch meist geteilt werden (besonders bei ATM[7], kann z.B. durch Überlast in einigen Netzwerkkomponenten eine zeitweilige Verschlechterung der Dienstqualität entstehen. Diese Verschlechterung (engl. QoS-Degradation) muß zuerst *erkannt* werden

[7] **A**synchronous **T**ransfer **M**ode: Zellenbasiertes (Zellen:=Blöcke fester Länge mit Nutzdaten á 48 Bytes + 5 Bytes Header) Verfahren für Hochgeschwindigkeits-Netze [Zitt 95].

und dann sollte darauf angemessen *reagiert* werden. Die Erkennung erfolgt durch eine permanente Beobachtung der tatsächlich erbrachten Dienstgüte (QoS-Monitoring). Die momentan erbrachte Dienstqualität kann streng genommen nur *geschätzt* werden, da sich der Systemzustand in einigen Teilen des Netzes aufgrund von physikalischen Laufzeiten bei Vorliegen des Meßergebnisses schon wieder geändert haben kann (Statusproblem in verteilten Systemen).

In [CCGH 93] wird zwischen *Policing* und *Monitoring* unterschieden: *Policing* bedeutet dabei eine *Überwachung des Benutzers*, um sicherzustellen, daß die vereinbarten QoS-Parameter auch nicht verletzt werden (z.B. Daten werden mit höherer Rate an das Kommunikationssystem geschickt, als vereinbart wurde) und *Monitoring* stellt durch *Beobachtung* sicher, daß die vereinbarten QoS-Stufen vom *Dienstanbieter* eingehalten werden. Policing wird in [CCGH 93] als Aufgabe der Netzwerkschicht betrachtet, während Monitoring auf der Transportschicht durchgeführt wird. Es stellt sich die Frage, ob auf das QoS-Monitoring an der Netzschnittstelle verzichtet werden kann. QoS-Monitoring an der Transportschnittstelle zwischen je zwei TSAPs[8] (Sender u. Empfänger) zu betreiben ist notwendig, weil trotz QoS-Monitoring in der Netzwerkschicht die erbrachte Dienstgüte auf Transportebene nicht ausreichen kann, da einmal die Last in den Endsystemen Einfluß auf die Verbindung haben kann und zum zweiten zwar möglicherweise auf der Netzwerkebene in Teilnetzen die Dienstgüte ausreicht, aber nicht mehr insgesamt von Ende zu Ende. Ist aber der Dienstgüte-Monitor in der Transportebene alleine auch hinreichend, um die Ursache für eine Dienstgüte-Verschlechterung festzustellen? Wäre das der Fall, so könnte man auf einen weiteren QoS-Monitor auf Netzwerkebene verzichten. Das Protokoll in der Transportschicht hat die Aufgabe, die Differenz in der Funktionalität zwischen dem erwünschten Dienst, den der Transportdienstnutzer (z.B. eine Anwendung) fordert und dem Dienst, den die Netzwerkschicht anbietet, auszugleichen. Dies geschieht mittels bestimmter Mechanismen und Ressourcen, die aufgrund der geforderten Dienstqualität ausgewählt und reserviert werden. Sind die Mechanismen in der Lage ein Fehlverhalten (aufgrund einer Ausnahme) oder eine Überlast der Ressourcen zu erkennen, könnte das Protokoll bei einer durch den QoS-Monitor an der Transportdienstschnittstelle festgestellten Dienstgüte-Verletzung und einem *garantierten Netzwerkdienst* die Ursache selbst herausfinden. Bietet jedoch der Netzwerkdienst nur einem unzuverlässigen oder statistischen Dienst an, so ist die Ursache nicht mehr so leicht ausfindig zu machen. Diese Thematik kann Gegenstand weiterer Untersuchungen sein, soll für diese Arbeit jedoch nicht weiter interessieren, da das Erarbeitete in weiten Teilen ebenfalls für einen Dienstgüte-Monitor an der Netzwerkschnittstelle gültig ist.

[8] **T**ransport **S**ervice **A**ccess **P**oint, Transportdienstzugangspunkt.

1.4 Ziele

Ziel dieser Arbeit ist es, die Komponenten eines *QoS-Monitors* zu *definieren*, einen solchen zu *entwerfen* und dann in ein bestehendes Transportprotokoll zu integrieren und den Monitor zu *implementieren*. Das Hauptaugenmerk soll hierbei auf die Transportkomponente (im Sinne des 3-Komponentenmodells nach Abschnitt 1.2) gelegt werden, obwohl viele Ausführungen auch für die Netzwerkschicht passen und daher manchmal allgemein (N)-Schicht benutzt wird.

Der Schwerpunkt dieser Arbeit liegt auf einer Untersuchung der Grundlagen des QoS-Monitoring, da in der Literatur nur sehr selten in der notwendigen Tiefe auf dieses eingegangen wird.

1.5 Aufbau

Es werden zuerst die Anforderungen an einen QoS-Monitor allgemein untersucht. Dabei werden grundlegende Prinzipien und Mechanismen für das QoS-Monitoring gegenübergestellt, bewertet und ausgewählt. Die Nähe zu den Aufgaben des Netzwerkmanagements und mögliche Realisierung mit Hilfe des Netzwerkmanagements werden in Kapitel 3 untersucht. Schließlich wird in Kapitel 4 zuerst eine plattformunabhängige Implementierung vorgestellt, die dann auf ein konkretes Transport- und Betriebssystem abgebildet wird.

2. Grundlegende Anforderungen

2.1 Randbedingungen

Es wird von folgenden Randbedingungen ausgegangen:

1. Die Kommunikation ist *verbindungsorientiert*.
2. Es werden *unidirektionale* Medienströme betrachtet.
3. Steuerungs- und Kontrollnachrichten werden über separate (duplex) „out-of-band“[1] Signalisierungskanäle verschickt.

Um Echtzeit-Anwendungen zu ermöglichen, muß das Kommunikationssystem die Leistungsanforderungen und Verkehrscharakteristik (allgemeiner: die Dienstgüte) der Anwendung kennen, um Ressourcen entsprechend reservieren und dadurch Garantien bieten zu können [Ferr 90]. Daher ist Punkt 1 Voraussetzung für eine sinnvolle Anwendung der Dienstgüte: Es muß ein *Kontext*[2] über die gewünschte Dienstqualität hergestellt werden. Es macht wenig Sinn, die Dienstqualität separat für jede zu sendende Dateneinheit festzulegen, da das darunterliegende System meist gar nicht so schnell auf die Anforderung reagieren kann, denn letztendlich wird die geforderte Dienstqualität auf eine Reservierung von Ressourcen (etwa Anzahl und Größe der Puffer, CPU-Zeit, „Bandbreite“, etc.) abgebildet. Da die Ressourcen im Netzwerk möglichst effizient genutzt werden sollen, könnten keine so kurzfristigen Zusagen für Datagramme[3] gemacht werden. Verbindungslose Dienste können über entsprechende Mechanismen wie „Connectionless Server“ (siehe [Zitt 95]) auf verbindungsorientierte Netze abgebildet werden. Darauf soll hier jedoch nicht näher eingegangen werden. Da sich der hier geforderte Kontext nur auf die Vereinbarung der QoS bezieht, ist es also auch denkbar und möglich, für Teilstrecken diese QoS-Verbindung einzurichten und auf ihnen dann Datagramme, für die die vereinbarte QoS gilt, zu übertragen.

[1] vom normalen Datenverkehr getrennte Leitungen.

[2] vgl. hierzu auch [LoKK 93].

[3] ein Datagramm ist eine verbindungslose Dateneinheit, die alle notwendigen Informationen enthält, um von der Quelle zum Ziel zu gelangen. Es besteht also diesbezüglich kein Kontext zwischen Sender und Empfänger.

Jede Verbindung besteht aus drei Phasen:

- Verbindungsaufbau
- Datenaustauschphase
- Verbindungsabbau

Durch den Verbindungsaufbau wird ein gemeinsamer *Kontext* zwischen Sender und Empfänger(n) vereinbart, der dem Datenaustausch zugrundeliegt (z.B. Sender- und Empfängeradresse, Dienstqualitäten), so daß diese Information nicht für jede Dateneinheit mit übertragen werden muß.

Unter dem Aspekt der Dienstqualität stellt sich das Szenario folgendermaßen dar (siehe Abbildung 2.1):

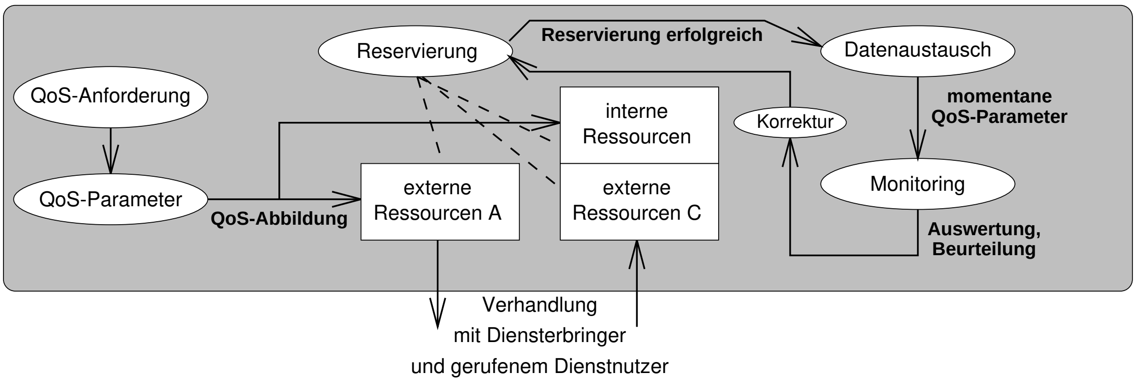


Abbildung 2.1: Verbindungsaufbau - QoS-Verhandlung

Der (sendeseitige) Dienstnutzer spezifiziert die Dienstgüte, die er für seine Verbindung benötigt, an der Dienstschnittstelle des in Anspruch genommenen Dienstes, mittels einer „QoS-Anforderung“ (QoS-Vorgabe). Die einzelnen QoS-Parameter können dabei entweder als strikt einzuhaltende Werte oder als Richtwerte, die innerhalb eines Intervalls schwanken dürfen, vorgegeben werden. Diese Vorgaben der QoS-Parameter werden möglicherweise auf andere QoS-Parameter der darunterliegenden Schichten abgebildet (QoS-Abbildung) und die entsprechenden Ressourcen in Netzwerkelementen (Switches, Router) und Endsystemen reserviert. Die direkte Reservierung der Ressourcen ist nötig, weil ohne Reservierung nach Feststellen der Verfügbarkeit die Ressourcen durch andere (neue) Verbindungen zwischenzeitlich bereits belegt worden sein könnten. Der gerufene Dienstnutzer kann die Dienstqualität noch abschwächen und so können dann zum Teil vorreservierte Ressourcen auf dem Weg der Rückantwort zum Verbindungsaufbau wieder freigegeben werden[4].

[4] Bei einem impliziten Verbindungsaufbau ist diese ausführliche Aushandlung nicht möglich!

Es können daher andere QoS-Parameterwerte ausgehandelt werden, als diejenigen, die vom Initiator angegeben wurden, weil die entsprechenden Ressourcen nicht zur Verfügung standen (in Abb. 2.1: externe Ressourcen A und externe Ressourcen C). Drei Parteien verhandeln über die Dienstgüte: rufender Dienstnutzer, Diensterbringer, gerufener Dienstnutzer. Der Diensterbringer verhandelt also i.a. mit über die Dienstgüte (s. auch [ISO 93]). Wurde sich auf bestimmte Werte geeinigt (es kommt also ein *Vertrag* zwischen den Kommunikationspartnern zustande [Ferr 90], *QoS-Vertrag* oder *Dienstvertrag* nach [CaCH 94]), sollten diese dann allerdings vom Diensterbringer eingehalten werden.

Dies ist der *statische* Aspekt der Dienstqualität. Jetzt kann sich aber die Dienstgüte aus drei Gründen während der Verbindung ändern (*dynamischer* Aspekt):

- die geforderte Dienstqualität kann vom Netzwerk nicht mehr erbracht werden (Dienstgüte durch Netzwerk verschlechtert).
- QoS-Vorgaben werden durch den Dienstnutzer geändert.
- Der Diensterbringer kann durch Ausnahmebehandlungen oder Überlast im lokalen System am Einhalten der Vorgaben gehindert werden.

Ersteres festzustellen ist die Aufgabe des *QoS-Monitors*. Mögliche Reaktionen sind eine *transparente*[5] Reaktion des Kommunikationssystems (Feinjustierung der Protokollressourcen und -mechanismen, Neuverhandlung der Dienstgüte anhand eines vorgegebenen Schemas ohne Verbindungsabbau) oder eine Anzeige der QoS-Verschlechterung an den Dienstnutzer. Der Benutzer kann auch von sich aus eine Änderung der Dienstgüte wünschen (z.B. Herabsetzen einer Video-Verbindung von Farbe auf Monochrom, um zusätzlich einen Audio-Kanal zu öffnen), oder verschiedene Medienströme mit daraus resultierenden unterschiedlichen QoS-Anforderungen sollen nacheinander über dieselbe Verbindung übertragen werden. Bei garantierten Diensten können nur Ausnahmesituationen die Dienstqualität mindern, bei statistischen Diensten[6] kann durch gleichzeitige Inanspruchnahme der Spitzenlast mehrerer Benutzer dieser Dienste eine Überlast entsprechender Ressourcen entstehen.

Eine *Simplex-Verbindung* anzunehmen (Punkt 2 von S. 11) ist im Hinblick auf Flexibilität, Ressourcenschonung und multimediale Anwendungen sinnvoll. Duplex-Verbindungen lassen sich durch Aufbau zweier Simplex-Verbindungen in verschiedene Richtungen realisieren. Simplex-Kommunikation liegt in der Natur von Verteildiensten, so z.B. kontinuierlichen Medienströmen wie Video und Audio. Da sich die Dienstgüte möglicherweise für Hin- und Rückrichtung unterscheidet (Farbvideo Hinrichtung, Monochromvideo Rückrichtung), können so Ressourcen gespart werden.

[5] transparent in dem Sinne, daß es dem Dienstnutzer verborgen bleibt.

[6] mehrere Benutzer belegen teilweise Ressourcen gemeinsam, s.S. 20, Abschnitt 2.2.2

Ebenso wäre es eine Verschwendung von Ressourcen immer eine Duplex-Verbindung aufbauen zu müssen, wenn lediglich eine Simplex-Verbindung nötig ist. Diese Einschränkung erfordert einige allgemeinere Lösungen bestimmter Probleme. So ist z.B. die Verzögerung nicht einfach über eine „Roundtrip-Time“ zu messen.

Ein separater Signalisierungskanal (Punkt 3) ist in den meisten Standards üblich (z.B. bei B-ISDN) und wegen Punkt 2 (Simplex-Verbindung) nötig. Bei Verbindungsaufbau muß der Empfänger ja auch die Möglichkeit haben, QoS-Parameter auszuhandeln, bzw. der Sender während des Datentransfers Benachrichtigungen über die momentane Dienstgüte entgegenzunehmen.

Hier sei auch darauf hingewiesen, daß (ähnlich wie bei den Flußkontroll–Mechanismen) in Hochgeschwindigkeits-Weitverkehrsnetzen mit hohen Bandbreiten-Verzögerungs–Produkten (Pfadkapazität) QoS-Monitoring möglicherweise wirkungslos ist, da sich unter Umständen bereits die gesamten zu sendenden Daten schon im Netzwerk befinden. Typische Medienanwendungen sind jedoch Langzeitanwendungen, so daß auf diese Problematik nicht weiter eingegangen wird.

2.2 Anforderungen an einen QoS-Monitor

Der QoS-Monitor soll möglichst unabhängig vom zugehörigen Protokoll und den darunterliegenden Schichten entworfen werden.

Es stellen sich folgende grundlegende Fragen:

1. Welche QoS-Parameter sind zu überwachen?
2. Wie sieht ein allgemeines Monitoring-Konzept aus?
3. Wie sieht sende- und empfangsseitiges Monitoring aus?
4. Wie werden die QoS-Parameter gemessen?
5. Wie oft sollen QoS-Parameterwerte gemessen und die Meßergebnisse gemeldet werden?
6. Ändern sich die Anforderungen unter dem Aspekt des Multicasting[7]?
7. Welche Funktionalität sollten die Monitor-Schnittstellen haben?

[7] Gruppenkommunikation: 1 Sender, n Empfänger.

2.2.1 QoS-Parameter

In der Menge von spezifizierbaren Dienstgüteparametern sind für das Monitoring nur diejenigen wichtig, die während einer aktiven Verbindung durch den Diensterbringer und die unterliegenden Schichten (d.h. die in Anspruch genommenen Dienste) verändert werden können. Das sind also mindestens:

Durchsatz

Der Durchsatz gibt die Anzahl der zwischen Dienstnutzern übertragenen Nutzdaten (d.h. Dienstdateneinheiten, SDUs[8]) pro Zeiteinheit an. Im folgenden werden die Begriffe *Paket* und *Rahmen*[9] gleichermaßen für eine SDU verwendet; ist dagegen eine PDU gemeint, so wird dies explizit betont. Der Durchsatz wird genauer durch folgende Größen charakterisiert [CaCH 94]:

- Größe der Dateneinheit (Rahmengröße)
  normalerweise feste Größe, ansonsten kann auch hier ein Maximalwert und ein Durchschnittswert angegeben werden.
- Mittlere Rate
  Zu erwartende Durchschnittsrate über die Dauer der Verbindung hinweg. Exakter wäre die Angabe der Größe I des Mittelungsintervalls und die Durchschnittsrate über diesem Intervall [ZhFe 93].
- Spitzenrate (Burst)
  Maximale Rate, mit der aufeinanderfolgende Dateneinheiten gesendet oder empfangen werden. Diese Rate erhält man, indem man den Kehrwert der minimalen Zeitdistanz zweier aufeinanderfolgenden Dateneinheiten unter allen Dateneinheiten ermittelt.
- Burstgröße
  Maximale Anzahl aufeinanderfolgender Nutzdaten, die mit der Spitzenrate gesendet oder empfangen werden.

Man beachte, daß die *Wahl der Zeiteinheit* einen großen Freiheitsgrad darstellt. So stellen zwar die Angaben 10Mbit/s und 100Mbit/10s den gleichen Durchsatz dar, jedoch ist bei letzterem die Schwankungsbreite größer (bezogen auf ein längeres Intervall, wie z.B. 60s).

[8] SDU=**S**ervice **D**ata **U**nit, Nutzdateneinheit des Dienstnutzers, die vom Diensterbringer eventuell in mehrere Protokolldateneinheiten (PDU) aufgeteilt wird (Segmentierung).

[9] der Begriff Rahmen (engl. „frame“) wird häufig im Zusammenhang mit kontinuierlichen Medienströmen verwendet.

Ende-zu-Ende Verzögerung

Die Ende-zu-Ende Verzögerung (end-to-end delay) ist die Zeit, die zwischen Empfang der (N)-SDU von der (N)-Schnittstelle beim Sender und dem Ausliefern der (N)-SDU an den (N)-Dienstnehmer beim Empfänger vergeht. In Abbildung 2.2 läßt sich leicht die Definition der Verzögerung für ein Paket P_i ablesen: $d_i := T_i^E - T_i^S$. Interaktive Anwendungen benötigen eine garantierte obere Schranke D_{max} für die Ende-zu-Ende Verzögerung. Anwendungen mit kontinuierlichen Medienströmen (z.B. ein Audio- oder Videostrom) benötigen zusätzlich die Definition einer unteren Schranke D_{min}, damit die Abfolge der Stromelemente (Samples oder Bilder) nicht plötzlich zu schnell wird. Für solche Anwendungen sind meistens Pakete, die „zu spät" kommen, unbrauchbar und können verworfen werden, womit diese den Datenverlusten hinzuzurechnen wären [Ferr 90].

Verzögerungsschwankung (Delay-Jitter)

Die Verzögerungsschwankung einer Dateneinheit P_i ist die Differenz $J_i := d_i - D$ der Ende-zu-Ende Verzögerung d_i zu einer *gewünschten* Ende-zu-Ende Verzögerung D (Definition aus [Ferr 90]). In einem *idealen Übertragungskanal* erfährt jede Dateneinheit die gleiche Verzögerung d_{min}, so daß die Verzögerungsschwankung 0 beträgt (für $D = d_{min}$) und somit das Ankunftsmuster der Daten exakt dem Sendemuster entspricht. Dieses ideale Verhalten ist bei paket- oder zellvermittelnden Netzen allerdings kaum zu erwarten.

Für eine Anwendung, die einen *gleichmäßigen Datenfluß* wünscht, ist eine garantierte Beschränkung dieser Verzögerungschwankung und der absoluten Verzögerungswerte wichtig, d.h. die Vorgabe einer *maximalen Intervallbreite* J_{max}, die als maximal zulässige Differenz zwischen *gemessener* maximaler und minimaler Ende-zu-Ende Verzögerung definiert ist ($d_{max} - d_{min} \leq J_{max}$). Innerhalb dieses Intervalls dürfen die gemessenen Verzögerungswerte liegen, deshalb gilt: $\forall i : |d_i - D| \leq J_{max}$ für $D \in [d_{min}, d_{max}]$. Die Beschränkung der *Intervallbreite der Verzögerungswerte* erlaubt eine Vorhersage, zwischen welchen Werten die Verzögerung für die nächste Dateneinheit zu erwarten ist, und somit – bei bekanntem Sendemuster – die Ankunftszeit. Es folgt also $T_i^E \in [T_i^S + d_{min}, T_i^S + d_{max}]$.

Es gibt in der Literatur allerdings unterschiedliche Definitionen des „Jitters". Ein *Raten-Jitter* (s. [ZhFe 93]) definiert die maximal tolerierbare Schwankung der Zwischenankunftszeit zweier aufeinanderfolgender (Dienst-)Dateneinheiten und ist daher gleichbedeutend mit einer maximalen Schwankung der effektiven Empfangsrate. Ein *Delay-Jitter* hingegen ist alleine – wie oben – über die Ende-zu-Ende Verzögerung definiert. Die Motivation für beide Definitionen sind hauptsächlich *kontinuierliche* Medienströme: Man möchte bei einem Video keine zu große (oder zu kleine)

Verzögerung *zwischen aufeinanderfolgenden* Einzelbildern haben, damit der Bewegungsablauf flüssig bleibt (Audio entsprechend). Ein vorgegebener maximaler Jitter sorgt also für einen *gleichmäßigen Datenstrom*. Die Kontrolle des Jitters erleichtert außerdem die (intermediale) Synchronisation verschiedener Medienströme auf Anwendungsebene. Der oben erwähnte Zusammenhang zwischen Ende-zu-Ende Verzögerung und Jitter wird deutlich, wenn man bedenkt, daß sich ein Rahmen *relativ* zu einem anderen Rahmen verzögert, wenn sich dessen Ende-zu-Ende Verzögerung *verändert* (wodurch sich auch die Zwischenankunftszeit ändert). Diese *unterschiedlichen Verzögerungen* entstehen durch unterschiedliche Bedienzeiten in den Zwischen- und Endsystemen. Einem Anwachsen des Jitters kann man durch Zwischenpuffern und Weiterleiten nach dem gewünschten Ankunftsmuster entgegenwirken, damit allerdings zu Lasten einer wachsenden Ende-zu-Ende Verzögerung, da lediglich die Auslieferung „zu früher“ Pakete verzögert werden kann und somit die größtmögliche Ende-zu-Ende Verzögerung für alle Pakete als Zielverzögerung angenommen werden muß.

Abbildung 2.2 veranschaulicht den Zusammenhang zwischen Raten- und Delay-Jitter. Der Raten-Jitter wird aber durch den Delay-Jitter beschränkt (s. folgenden Abschnitt), während ein konstanter Raten-Jitter eine stetig wachsende, also unbeschränkte Verzögerung nicht ausschließt. Dies ist beispielsweise der Fall, wenn die Sendeseite mit konstanter Rate r_1 Nachrichten verschickt und die Empfangsseite mit konstanter Rate $r_2 < r_1$ empfängt. Dadurch nimmt die Ende-zu-Ende Verzögerung ständig zu, der Raten-Jitter ist aber 0. Daher ist bei einer Kontrolle des Raten-Jitters immer die Einhaltung der absoluten Grenzen (D_{min} und D_{max}) der Verzögerung mit zu überwachen. Zum Zeitpunkt T_i^S sei ein Paket P_i (entsprechend einer SDU) an der Schnittstelle des Senders vollständig übergeben worden. Nach einer möglichen Pause von $\epsilon_{i,i+1} := T_{i+1}^S - T_i^S$ folgt die nächste SDU und so weiter. Der Zeitpunkt der vollständigen Auslieferung der SDU beim Empfänger sei T_i^E. Somit ergibt sich eine Ende-zu-Ende Verzögerung von $d_i = T_i^E - T_i^S$. Zur Zeit T_{i+1}^E sei nun das nächste Paket P_{i+1} an den Empfänger ausgeliefert worden, dessen Verzögerung $d_{i+1} = T_{i+1}^E - T_{i+1}^S$ beträgt und nach einer Zeit von $\delta_{i,i+1} := T_{i+1}^E - T_i^E$ auf Paket P_i folgend beim Empfänger eintrifft. Die Schwankung in der Zwischenankunftszeit $\Delta\delta$ (Raten-Jitter) ist durch den Delay-Jitter beschränkt:

$$\begin{aligned}
\Delta\delta &= |\delta_{j,j+1} - \delta_{j-1,j}| \\
&= |(T_{j+1}^E - T_j^E) - (T_j^E - T_{j-1}^E)| \\
&= |T_{j+1}^S + d_{j+1} + T_{j-1}^S + d_{j-1} - 2(T_j^S + d_j)| \\
&= |\epsilon_{j,j+1} - \epsilon_{j-1,j} + (d_{j+1} - d_j) - (d_j - d_{j-1})| \\
&\leq |\epsilon_{j,j+1} - \epsilon_{j-1,j} + J_{max} - 0| \\
&\leq |\underbrace{\epsilon_{j,j+1} - \epsilon_{j-1,j}}_{\substack{\text{Schwankung in} \\ \text{der Senderate}}} + J_{max}|
\end{aligned}$$

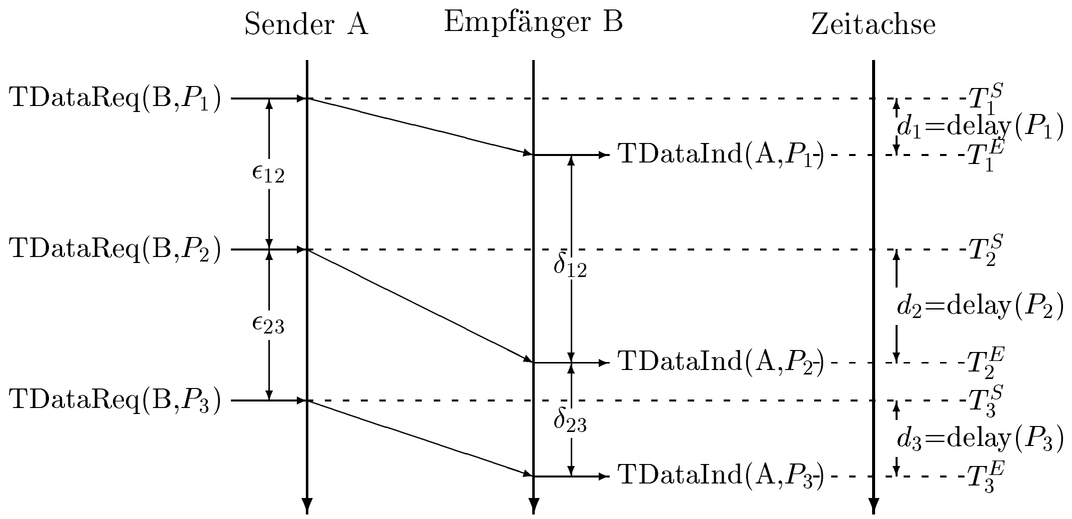


Abbildung 2.2: Delay-Jitter und Raten-Jitter

Die Abschätzung entstand dadurch, daß $d_{j+1} - d_j$ durch $d_{max} - d_{min} = J_{max}$ und $d_j - d_{j-1}$ durch 0 abgeschätzt wurde. Werden die Nachrichten mit konstanter Rate ($\epsilon_{j,j+1} = \epsilon_{j-1,j}$) gesendet, so wird der Raten-Jitter allein durch den Delay-Jitter J_{max} begrenzt.

Die Vorgabe eines Delay-Jitters J_{max} und einer mittleren Verzögerung ist gleichbedeutend mit der Vorgabe einer minimalen und maximalen Ende-zu-Ende Verzögerung (d_{min} und d_{max}). Es sollte also für den Delay-Jitter J_i eines jeden Pakets P_i mit $D = \overline{d} := \frac{1}{2}(d_{max} + d_{min})$ gelten: $|d_i - \overline{d}| \leq \frac{1}{2} J_{max}$. Nach [ZhFe 93] hat die Kontrolle des Delay-Jitters auch den Vorteil (gegenüber der alleinigen Kontrolle des Raten-Jitters), daß die Anwendungen keinen großen Puffer mehr benötigen. Schließlich sei noch bemerkt, daß die Angabe einer maximalen Ende-zu-Ende Verzögerung zwar auch den Delay-Jitter nach oben abschätzt, allerdings ist diese Angabe ohne zusätzliche Spezifikation von d_{min} noch zu ungenau. d_{min} hat mindestens den Wert der physikalischen Signallaufzeit zwischen Sender und Empfänger.

Eine praktische Variante entsteht, wenn man $J_{max} < D_{max} - D_{min}$ definiert und die aktuellen Werte d_{min}, d_{max} nur über einem Intervall I aufeinanderfolgender Dateneinheiten mitführt (s. Abb. 2.3). Dies wäre für einen nicht interaktiven Videostrom nützlich, da sich die absolute Ende-zu-Ende Verzögerung *langsam* verändern darf. Es könnte zum Beispiel länger bis zum Empfang des allerersten Bildes dauern, darauffolgende Bilder müssen aber die Vorgabe des maximalen Delay-Jitters erfüllen. Mit der Zeit kann sich dann aber das Intervall $[d_{min}, d_{max}]$ innerhalb von D_{min} und D_{max} verschieben. Eine verwandte Idee findet man in [WaCr 93] als *adaptive Synchronisation.*

Abbildung 2.3: Verzögerungsschwankung und absolute Verzögerung

Fehler

Es gibt auf den unterschiedlichsten Ebenen (Bits, Zellen oder Rahmen) Fehler:

- Datenverfälschung, Datenfehler
durch physikalische Störeinflüsse verursacht. Es gibt Einzelbitfehler, die ein einzelnes Bit invertieren, oder Bündelfehler, die eine Sequenz von Bits beeinflussen. Bei Lichtwellenleitern ist die Wahrscheinlichkeit für Einzelbitfehler jedoch sehr gering (etwa im Bereich von 10^{-12} bis 10^{-15} und kleiner).

- Datenverlust
kann durch Pufferüberlauf oder absichtliches Verwerfen als verfälscht erkannter Dateneinheiten entstehen. Je nach Anwendung sind auch Nachrichten, die „zu spät“ eintreffen oder nicht in geeigneter Reihenfolge ausgeliefert werden, für die Anwendung unbrauchbar und müssen verworfen werden. Diese „Verluste“ können in die Angabe des allgemeinen Verlustes mit einbezogen werden oder explizit aufgeführt werden (s. [Ferr 90]).

- Datenduplizierung
Duplizierte Daten können z.B. durch Übertragungswiederholung aufgrund Verlust einer Empfangsquittung bei einer gesicherten Übertragung oder auch durch kurzzeitigen Ausfall eines Netzwerkelements und dynamisches Rerouting in Kombination mit einer Übertragungswiederholung entstehen. Das gestörte Netzwerkelement könnte nämlich bei seiner Reaktivierung versuchen, die noch im Puffer befindlichen Nachrichten zuzustellen, welche u.U. schon durch Wiederholung auf der Alternativ-Route den Empfänger erreicht haben (siehe auch [Ferr 90]).

- Angezeigte Fehler
Oft besitzt die Schicht, deren Dienste in Anspruch genommen werden, ein Dienstprimitiv, um Fehler anzuzeigen. D.h. die (Transport-)Schicht ist in der Lage einige Fehler zu *erkennen*, kann sie aber *nicht beheben*. Auch der darunterliegende Diensterbringer, z.B. die Adaptionsschicht AAL[10] bei B-ISDN,

[10] **A**TM-**A**daption-**L**ayer, s. [Zitt 95]

könnte Fehler anzeigen, jedoch sollte das Transportprotokoll diese geeignet beheben können, so daß den Transportdienstnutzer nur die Fehler interessieren, die erkannt, aber nicht vom Transportprotokoll behoben werden können.

Anders als bei den vorhergenannten Parametern ist hier nur eine Angabe der maximal tolerierten Fehler innerhalb eines bestimmten Intervalls sinnvoll, da die Anzahl der Fehler mit der Zeit nur zunehmen kann. So könnte eine Anwendung beispielsweise fordern, daß höchstens 5 von 100 SDUs fehlerhaft sein dürfen. Ist das Intervall unendlich, so gilt die Einschränkung für die gesamte Verbindungsdauer (maximal 5 fehlerhafte SDUs absolut). Bei ATM-Technologien dürfte der Datenverlust ganz klar den dominierenden Faktor darstellen (z.B. Zellenverlust durch Verkehrsstau im ATM-Switch).

2.2.2 QoS-Parameterklassen

Um *flexibel* auf die unterschiedlichen Anforderungen der Medienströme reagieren zu können, wird *jeder* Parameter in eine von drei verschiedenen Klassen eingeteilt und erlaubt so eine qualitative Verfeinerung der Anforderung (s. [CaCH 94] und [Ferr 90]):

- garantiert (deterministisch)
  der Wert des Parameters muß in einem angegebenen Intervall liegen, Werte außerhalb dieses Intervalls sind nicht erlaubt, d.h. es wird vom System *garantiert*, daß der Parameterwert innerhalb dieses Intervalls liegt. Um eine Garantie zu erreichen, müssen die mit dem garantierten Parameter korrespondierenden Ressourcen in ausreichenden Maße (für Spitzenwerte) *exklusiv reserviert* werden. Theoretisch wäre also kein Monitoring nötig, da allein die dedizierten Ressourcen für die Garantie sorgen. In der Realität können trotzdem Einbrüche erfolgen (etwa durch Ausnahmebehandlungen, Überlastsituationen, etc.), so daß eine 100%ige Garantie nicht gegeben werden kann. Im Falle einer Verletzung sollte dann wenigstens geeignet reagiert werden. Daher ist also auch hier Monitoring notwendig.

  Bei [CaCH 94] wird diese Klasse allerdings mit Intervallbreite 0 definiert. In der Praxis wird der Parameter immer ein wenig um den einzuhaltenden Wert schwanken, daher ist es sinnvoller ein *Intervall* zuzulassen, innerhalb dessen der Parameterwert schwanken darf.

- statistisch
  der vorgegebene Wert entspricht einer Zielvorgabe und eine Abweichung innerhalb eines gewissen Intervalls ist mit einer angegebenen Wahrscheinlichkeit erlaubt. Die *Zielvorgabe* wird in der Regel ein gewünschtes arithmetisches Mittel

sein. Die einzuhaltende Bedingung lautet $P([b_{low} \leq t \leq b_{hi}]) \geq p$, wobei t der aktuelle QoS-Parameterwert ist und b_{low} die untere, bzw. b_{hi} die obere Grenze; p ist die Wahrscheinlichkeit. Wenn $p = 1$, so ist eine Abweichung von der Zielvorgabe nur innerhalb des Intervalls erlaubt und stellt den deterministischen Fall dar (auch in [Ferr 90] wird dieser Fall als deterministisch bezeichnet). b_{low}, b_{hi} und p sind also für die statistische Klasse zu spezifizieren. So kann man beispielsweise durch $P([30\text{kBit/s} \leq \text{Durchsatz} \leq 64\text{kBit/s}]) \geq 0.95$ spezifiziert werden, daß der erbrachte Durchsatz zwar die angegebenen Grenzen auch überschreiten darf, das jedoch nur in 5% aller Fälle.

Bei [CaCH 94] ist immer $p = 1$ und es wird ein Prozentsatz angegeben, um den der vorgegebene Mittelwert schwanken darf. Das Intervall ist dort also immer symmetrisch. Die hier verwendete Erweiterung stammt aus [ISO-QoS-WD 94] bzw. [Ferr 90] und begründet die Bezeichnung „statistisch" eigentlich erst, obwohl man auch den Standpunkt einnehmen kann, daß die Wertverteilung des QoS-Parameters innerhalb der Grenzen einer Wahrscheinlichkeitsfunktion unterliegt. Im Gegensatz zu der garantierten Klasse werden hier Ressourcen unter mehreren Benutzern geteilt, bzw. Ressourcen überbucht. Diese Vorgehensweise ist auch unter dem Begriff *statistisches Multiplexen* bekannt [Zitt 95] und geht von der Annahme aus, daß eben nicht alle Dienstnutzer *gleichzeitig* ihre maximale Dienstgüte benötigen, so daß Ressourcenreservierungen anhand einer „mittleren" Dienstgüte erfolgen. Unter wirtschaftlichen Aspekten ist diese Klasse durch eine geeignete Kostenpolitik zu forcieren (statistisch deutlich billiger als garantiert anbieten). Ein Monitoring ist also unbedingt notwendig, um geeignet reagieren zu können, falls Ressourcen knapp werden und dadurch eine Mindest-Dienstgüte unterschritten wird.

In [Ferr 90] wird noch eine *„gebrochene Grenze"* vorgeschlagen, in der p genauer durch zwei Zahlen A und B spezifiziert wird, wobei A eine Anzahl unmittelbar aufeinanderfolgender Dateneinheiten und B die Anzahl der erlaubten „Grenzverletzungen" unter diesen A Dateneinheiten festlegt ($C_A(b_{low} \leq t \leq b_{hi}) \geq B$ und $C_A(\text{Pred})$ bezeichnet eine Zählfunktion, die die Erfüllung des Prädikats Pred für A aufeinanderfolgende Dateneinheiten mißt). Das ist wesentlich genauer, einfacher zu messen und zu verifizieren als die Angabe einer Wahrscheinlichkeit. Dabei stellt sich noch die Frage, ob in einem *springendem Fenster* oder in einem *gleitenden Fenster* gemessen wird. Beim *springenden Fenster* wird nach Erreichen von A Dateneinheiten überprüft, ob $C_A < B$ und der Zähler für die Verletzungen sowie der Zähler für die betrachteten Dateneinheiten wieder auf Null gesetzt. Beim *gleitenden Fenster* hingegen wird für jede neue Dateneinheit der Zustand der bis dorthin ältesten Dateneinheit vergessen und der Zähler für die Verletzungen aktualisiert. Es ist anhand von Abbildung 2.4 leicht einzusehen, daß das gleitende Fenster wesentlich restriktiver ist, da die Bedingung „weniger als B Verletzungen unter A aufeinanderfolgenden Dateneinheiten" zu jedem Zeitpunkt erfüllt ist. Jeder Strich auf der

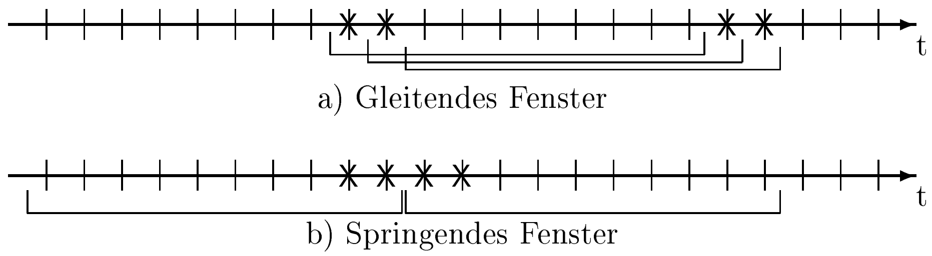


Abbildung 2.4: Gleitendes Fenster vs. springendes Fenster

Achse stellt eine Dateneinheit dar (zur Vereinfachung der Darstellung wird hier ein konstantes Ankunftsmuster angenommen), jedes X bezeichnet eine Dateneinheit, deren Parameterwert außerhalb des erlaubten Bereichs liegt. Im Beispiel seien 2 abweichende unter 10 aufeinanderfolgenden Dateneinheiten erlaubt. Die Situation in Abbildung 2.4 b) stellt beim gleitenden Fenster eine Verletzung dar, während sie beim springenden Fenster toleriert wird. Somit kann beim springenden Fenster maximal die doppelte Anzahl an Verletzungen *hintereinander* an den Fenstergrenzen auftreten! Dieser Einbruch in der Dienstqualität ist eigentlich unerwünscht, denn sonst hätte der Dienstnutzer ein größeres Intervall gewählt (z.B. 4 von 20). Damit ist das *gleitende Fenster* die zu bevorzugende Zähl-Methode, die zu dem den Vorteil besitzt, permanent Ergebnisse zu liefern, während beim springenden Fenster nur alle A Dateneinheiten ein Ergebnis vorliegt.

- bestmöglich
  dem System ist die Einhaltung einer Parametervorgabe freigestellt (keinerlei Garantie). Für Parameter aus dieser Klasse ist im Prinzip kein Monitoring nötig, es sei denn um Statistiken zu führen.

Beispiel für herkömmlichen Dateitransfer: Verzögerung, Durchsatz und Jitter werden bestmöglich unterstützt, aber Datenfehler deterministisch garantiert. In [Ferr 90] findet sich eine vergleichbare Klassifizierung (deterministisch, statistisch) wieder. Entscheidend für obige Klassifizierung war der Aspekt, daß man für die *garantierte* Klasse Ressourcen dediziert reservieren muß, bei der *statistischen* Klasse weniger Ressourcen reserviert und verbleibende Kapazität mit anderen Benutzern zu einem bestimmten Grad teilt, sowie bei der *bestmöglichen* Klasse gar keine Ressourcen reserviert.

Die *Aufgabe des QoS-Monitors* besteht darin, die entsprechenden Parameter innerhalb eines vorgegebenen Zeitintervalls zu messen und mit den geforderten QoS-Parameterwerten zu vergleichen. Wird die Dienstgüte vom System nicht wie vorgegeben erreicht, so sollte der Monitor das der zugehörigen Protokollinstanz oder einem

QoS-Manager (siehe Abbildung 2.5) anzeigen, damit weitere Maßnahmen (z.B. Anzeige an Dienstnutzer, QoS-Feinjustierung, QoS-Neuverhandlung) eingeleitet werden können. Der QoS-Monitor sollte mehrere Verbindungen gleichzeitig überwachen und verwalten können, damit nicht für jede Verbindung eine eigene Monitorinstanz vorhanden sein muß. Intern kann die Monitorinstanz durch ein Prozeß-Team (Threads[11]) realisiert werden. Es wird davon ausgegangen, daß die Protokollinstanz ebenso mehrere Verbindungen verwaltet. Generell sollten die zu messenden Parameter wählbar sein. Zu beachten ist noch, daß möglichst keine Verletzungen der Dienstgüte aufgrund von Meßungenauigkeiten oder -fehlern angezeigt werden.

2.2.3 Integration des QoS-Monitors

Wie integriert man den Monitor in eine sinnvolle Modellstruktur und wie sieht eine solche aus? Wo sollte der QoS-Monitor dort plaziert werden? Um möglichst allgemein zu bleiben, ist folgende Systemaufteilung nützlich (siehe Abbildung 2.5):

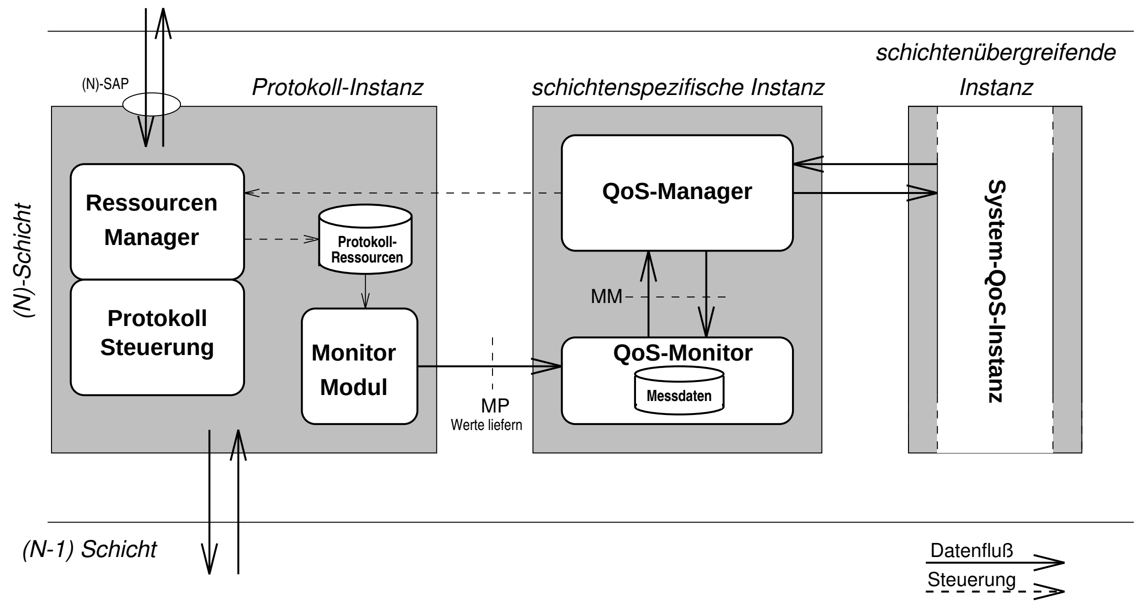


Abbildung 2.5: Ein QoS-Monitoring Modell

Grundsätzlich muß der QoS-Monitor mit der zugehörigen Protokoll-Instanz gekoppelt sein, damit er die zu messenden QoS-Daten ermitteln kann. Die Protokollinstanz besitzt ein *Monitor-Modul*, welches die Aufgabe hat, dem Monitor (lesenden) Zugriff

[11] Ein „Thread“ ist ein sogenannter „Leichtgewichtsprozeß“, der nebenläufig zu anderen Threads *innerhalb des gleichen Adreßraums* (Prozeßkontext) eines Prozesses läuft. Innerhalb dieses Adreßraums werden die Threads nach einer auszuwählenden Umschaltstrategie (scheduling policy) umgeschaltet. Der Wechsel zwischen Threads geht schnell vonstatten, da kein Adreßraumwechsel dazu nötig ist.

auf die protokollinternen Zustandsvariablen bzw. QoS-Parameter zu ermöglichen und ihn über Ereignisse zu informieren. Die *QoS-Monitorinstanz* greift über diese spezielle Monitorschnittstelle des Protokolls auf die zu beobachtenden Parameter zu. Diese Monitorinstanz errechnet möglicherweise erst aus den protokollinternen Variablen die zu überwachenden QoS-Parameterwerte und vergleicht diese mit den Vorgaben (Transformation nach Abschnitt 1.1.1). Über eine Monitor-Manager Schnittstelle wird die Monitorinstanz vom *QoS-Manager* gesteuert. Der *QoS-Manager* ist eine schichtenspezifische Instanz (Schichten-QoS-Instanz), die mit der *Erhaltung der Dienstgüte* beauftragt ist [CaCH 94]. Dazu kann sie über eine entsprechende Schnittstelle steuernd auf die Protokollressourcen zugreifen. Der QoS-Manager erhält auf Wunsch Meßergebnisse des QoS-Monitors oder eine QoS-Verschlechterungsnachricht, auf die er mit einer Feinjustierung der Ressourcen reagieren kann oder indem er eine System-QoS-Instanz (siehe Abbildung 2.6) über die QoS-Verletzung informiert, die wiederrum den (N)-Dienstnutzer informiert. Der QoS-Manager sollte auch dafür verantwortlich sein, die Partnerinstanzen zu informieren (z.B. QoS-Manager beim Empfänger benachrichtigt QoS-Manager beim Sender), denn möglicherweise liegt die Ursache der QoS-Verschlechterung beim Empfänger selbst und das sollte der QoS-Manager lokal zu beheben versuchen. Für einen solchen Fall muß eine *Monitorinstanz für Betriebssystemressourcen* im Endsystem existieren, damit festzustellen ist, ob die Ursache lokal ist (siehe auch [OiTo 93] und [ToKi 93][12]). Die lokale Verschlechterung der Dienstqualität kann nämlich (für garantierte Parameter) nur durch eine Überlast im Betriebssystem erzeugt werden, da das Kommunikationssystem normalerweise vor Annahme einer Verbindung prüft, ob genügend Ressourcen (sowohl lokale System- als auch Netzwerkressourcen) vorhanden sind, um die QoS-Anforderungen zu erfüllen und diese dann reserviert. Im Fall der Parameter aus der statistischen Klasse kann durch Überlastsituationen (alle Benutzer fordern Spitzenwerte) ebenfalls eine lokale Verschlechterung erfolgen, da die Ressourcen von mehreren Nutzern geteilt werden (und i.a. nur für Durchschnittswerte reserviert wird). Wird die Partnerinstanz benachrichtigt (Empfänger informiert Sender), so muß ihr eine Reaktionszeit zugestanden werden (z.B. für die Feststellung der Ursache und die Anpassung der Ressourcen) und abhängig von der Umgebung (z.B. Signalisierungskanal ohne Auslieferungsgarantie) kann die Nachricht nach einer gewissen Zeit wiederholt werden. Der QoS-Manager sollte auch das Monitoring ein- bzw. ausschalten können. Je nach Architektur könnten ihm auch weitergehende Aufgaben zukommen, wie z.B. Auswahl der Protokoll-Mechanismen, QoS-Parameter-Abbildung etc., was dann nicht mehr einer strengen Trennung von schichtenspezifischen und systemspezifischen (schichtenübergreifenden) Aufgaben entspricht.

Man kann den Monitor auch in die zugehörige Protokoll-Instanz integrieren, wodurch die Schnittstelle zwischen Monitor und Protokoll wegfällt, dafür aber auch die Selbständigkeit der Monitorinstanz aufgegeben wird, denn diese kann im an-

[12] Im Falle einer Nichteinhaltung der „Deadline“ eines Realtime-Threads durch den Scheduler wird ein spezieller Deadline-Handler gestartet, ein expliziter Monitor kann also auch entfallen.

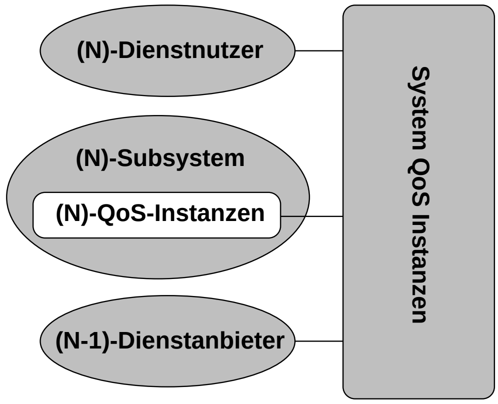


Abbildung 2.6: Beziehung zwischen System- zu Schichten-QoS-Instanzen

deren Fall *asynchron* zum Protokoll ablaufen und die Auswertung der erbrachten Dienstgüte nebenläufig vornehmen. Im Idealfall meldet sich der Monitor nur (beim QoS-Manager), falls die von ihm beobachtete Dienstqualität nicht mehr den Anforderungen entspricht. Im Hinblick auf die größere Flexibilität und Parallelität sollte also die Einrichtung einer eigenen QoS-Monitorinstanz erfolgen.

Damit ist eine *Schnittstelle* zwischen Protokoll-Instanz und QoS-Monitor, sowie eine zwischen QoS-Manager und letzterem zu definieren, worauf in Abschnitt 2.2.8 eingegangen wird.

2.2.4 Sende- und Empfangsseite

Es wird eine QoS-Monitorinstanz beim Sender und je eine bei den Empfängern vorhanden sein. Die Empfängerseite mißt alle unter Abschnitt 2.2.1 angegebenen Parameter. Was sollte aber die Sendeseite „monitoren“? Es kann lediglich der Zeitpunkt der Entgegennahme der (N)-SDU vom (N)-Dienstnehmer und die Größe der (N)-SDU ermittelt werden, sowie eventuelle Anforderungen einer Übertragungswiederholung oder Fehleranzeigen derselben (N)-SDU. Damit läßt sich nur die *Rate* messen, mit der der (N)-Dienstnehmer die Daten an den Diensterbringer übergibt, aber keiner der Parameter Ende-zu-Ende-Verzögerung oder Verzögerungsschwankung. Eine Messung der Parameter *Durchsatz* und *Datenfehler* beim Sender ist sinnvoll, damit (vom QoS-Manager) festgestellt werden kann, ob beispielsweise ein Einbruch im Durchsatz durch Protokollmechanismen wie etwa Übertragungswiederholungen verursacht wurde. Insbesondere läßt sich dadurch auch der (N)-Dienstnehmer überwachen, ob er sich z.B. an seine eigene Angabe der Spitzenrate hält (Policing, s.S. 8).

2.2.5 Meßintervall und Auswertung

Meßintervall

Grundsätzlich sind zwei verschiedene Ansätze möglich:

- Messung über einem statischem Zeitintervall
- Messung über einem dynamischem Zeitintervall

Die Messung über einem *statischen Zeitintervall* bedeutet, daß einmal die Länge des Zeitintervalls (Meßperiode) festgelegt wird und dann periodisch zu festen Zeitpunkten die Messung erfolgt. Dies hat den Nachteil, daß Meßzeitpunkte mitten in Rahmen hineinfallen können, sich also das System zum Meßzeitpunkt in einem für die Messung ungünstigen Zustand befindet (z.B. wurde u.U. bis zu diesem Zeitpunkt nur ein Teil eines Paketes empfangen, siehe Abbildung 2.8 zum Meßzeitpunkt t_{m4}). In diesem Fall muß man daher die Anzahl der in diesem Intervall empfangenen Bytes zählen (Stichprobe). Die Messung der Verzögerung hat jedoch auch die Bezugsgröße Rahmen (SDUs) statt Bytes und Schwankungen innerhalb eines Rahmens sind für den Anwender relativ uninteressant, da er die *Rahmenrate* vorgibt (man denke etwa an einen Video-Strom mit: 1 Rahmen=1 Bild und Vorgabe von 30 Bildern/s). Die Auflösung des internen Zeitgebers (Timer) begrenzt jedoch die Intervallänge, und die Fähigkeit eine periodische Unterbrechung (timer interrupt) zu generieren, ist bei dieser Meßmethode Voraussetzung (was keine Schwierigkeit darstellt). Ein schwerwiegender (implementierungsbedingter) Nachteil dieser Methode ist jedoch, daß die periodischen Unterbrechungen entsprechende Unterbrechungsbehandlungen auslösen, die erstens zu einer starken Systembelastung führen, da meist Kontextwechsel (Adreßraumwechsel oder Kerneintritte) mit ihnen verbunden sind und damit zweitens zu einer nicht zu vernachlässigenden Verzögerung in der Meßwertermittlung führen. Der Ablauf geschieht etwa wie folgt: zuerst wird der Timer beauftragt, periodisch zu unterbrechen. Läuft das entsprechende Intervall ab, so muß erst die Unterbrechungsbehandlung durchlaufen werden, die dann direkt die zu messenden Daten auslesen und speichern kann (d.h. den Monitor aufruft), danach wird die Unterbrechungsbehandlung verlassen und das Programm fortgesetzt. Bis zum Auslesen der zu messenden Größen vergeht also je nach Intervallänge eine beträchtliche Verzögerung, die das zu messende System beeinflußt.

Die Messung über einem *dynamischem Zeitintervall* kann dadurch geschehen, daß man die Länge des Zeitintervalls von einem Ereignis abhängig macht und so z.B. die Messung nach n empfangenen SDUs (Rahmen, Paketen) durchführt[13]. Das hat einmal den Vorteil, daß die Messung nun *rahmenorientiert* erfolgt und somit der Durchsatz für Rahmen ermittelt wird und nicht noch Schwankungen innerhalb von

[13] möglich ist auch die zufällige Wahl der Intervallänge, s. [Ferr 78].

Rahmen berücksichtigt werden. Dies entspricht auch eher der ISO-Definition des Durchsatzes. Zum zweiten braucht nicht jedesmal eine Unterbrechungsbehandlung stattzufinden, sondern es wird direkt nach Auftreten des Ereignisses, also an der entsprechenden Stelle im Programm die Systemuhr ausgelesen und danach ein Unterprogrammaufruf der Monitor-Routine[14] oder eine Prozeßsignalisierung an den Monitor-Prozeß stattfinden. Damit tritt lediglich eine Verzögerung durch das Auslesen der Uhr auf, wenn sich die Monitor-Instanz z.B. im gleichen Adreßraum wie die zu messende Instanz befindet. Ein Nachteil ist, daß die Intervallänge (zu messende Zeit) sehr kurz sein kann (beispielsweise bei der Durchsatzermittlung einer Spitzenrate) und dafür die *Auflösung der Zeitnehmer* möglicherweise nicht mehr ausreicht. Dem könnte man durch (eventuell sogar dynamische) Wahl anderer, zeitlich weiter voneinander entfernter Ereignisse (im obigen Beispiel Vergrößerung von n anhand der vorgegebenen Spitzenrate und Uhrauflösung) entgegenwirken. n_{max} bezeichnet im folgenden die Anzahl der SDUs, die zur Spitzenrate hintereinander empfangen werden und für die Durchsatzmessung zusammengefaßt werden müssen, weil die Uhrauflösung für einen kürzeren Zeitraum – also ein kleineres n_{max} – nicht mehr ausreicht.

Ist die Rate zeitweilig geringer, so kann z.B. auch $n = 1$ ausreichen, man hat allerdings jederzeit mit einem Burst zu rechnen, weshalb bei einer statischen Wahl von n durch n_{max} der „Worst-Case" berücksichtigt wird. Eine dynamische Wahl von n wird unten näher erläutert. Wie legt man nun den Parameter n_{max} fest? Die Auflösung der Uhr ist ein kritischer Punkt, da der tatsächliche Zeitpunkt des Ereignisses auf das nächstkleinere ganze Vielfache der Uhrauflösung Δt_{min} „gerundet" wird und damit um die kleinstmögliche Auflösung der Uhr schwanken kann (siehe Abbildung 2.7). Dies ist bei der Wahl des Parameters n_{max} zu berücksichtigen. Gibt man einen maxi-

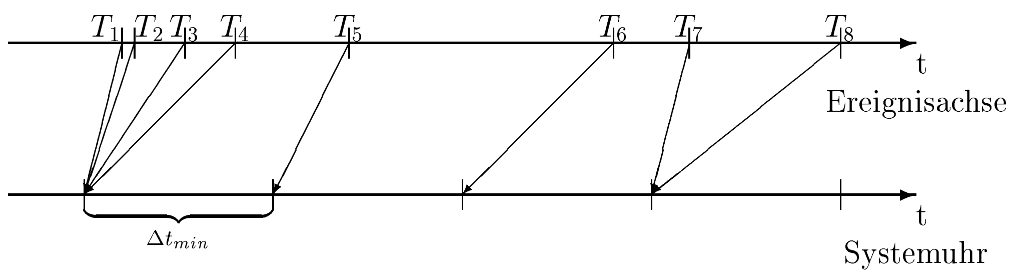


Abbildung 2.7: Ereigniszeitpunkte werden auf Uhrauflösung gerundet

malen relativen Meßfehler ϵ_{max} vor, so berechnet sich n_{max} mit der maximalen SDU-Rate R_{max} und der minimalen Uhrauflösung Δt_{min} zu $n_{max} := \lceil \frac{1}{\epsilon_{max}} \cdot R_{max} \cdot \Delta t_{min} \rceil$.

[14]sinnvoll nur bei sehr kurzer Monitor-Routine, denn das Protokoll wird blockiert. Ein Monitor-Unterprogramm ist also zu vermeiden!

Die Gleichung erhält man aus der Vorgabe $\frac{\Delta t_{min}}{\Delta t} \leq \epsilon_{max}$, wobei Δt der Meßzeitraum ist. In diesem Zeitraum können maximal $n = \lceil \Delta t \cdot R_{max} \rceil$ SDUs ankommen. Daher sollte gelten: $n = \lceil \Delta t \cdot R_{max} \rceil \geq \lceil \frac{\Delta t_{min}}{\epsilon_{max}} \cdot R_{max} \rceil$. Ein weiterer Vorteil dieser Methode ist, daß man gleichzeitig Durchsatz und Verzögerung messen kann, da die Messung der Verzögerung immer rahmenorientiert erfolgt und die Uhr somit nur einmal ausgelesen werden muß.

Der Durchsatz $D_{1...n}$ bei einem dynamischen Intervall berechnet sich bei einer Messung nach n Paketen der Länge P_i zu

$$D_{1...n} := \frac{\sum_{i=1}^{n} P_i}{\Delta T} = \frac{\sum_{i=1}^{n} P_i}{(\sum_{i=1}^{n} \Delta t_i)}$$

wobei Δt_i die Zeit zwischen vollständigem Erhalt des Pakets P_{i-1} und Empfang des letzten Bytes des Pakets P_i, sowie ΔT die Gesamtzeit $\sum_{i=1}^{n} \Delta t_i$ bezeichnet (s. Abbildung 2.8). Der Wert von $D_{1...n}$ stimmt übrigens nicht überein mit:

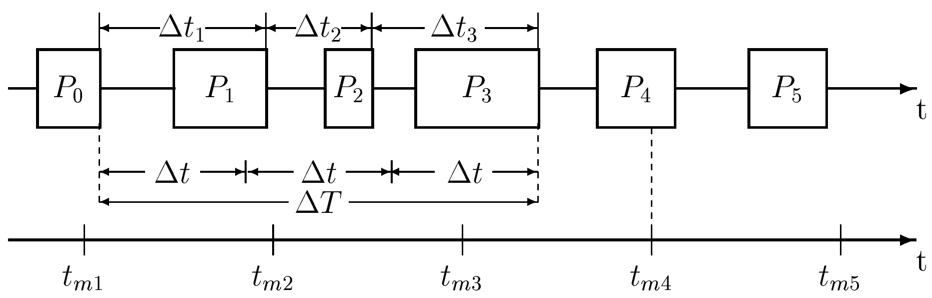


Abbildung 2.8: Durchsatzmessung mittels statischem und dynamischem Intervall

$$\overline{D} := \frac{1}{n} \sum_{i=1}^{n} \frac{P_i}{\Delta t_i} = \frac{1}{n} \sum_{i=1}^{n} D_i$$

also dem arithmetischen Mittel der Durchsätze D_i der einzelnen Pakete über dem Intervall. Dafür ist aber

$$D_{1...n} := \frac{\sum_{i=1}^{n} P_i}{\Delta T} = \frac{\sum_{i=1}^{n} P_i}{n \cdot \frac{\Delta T}{n}} = \frac{1}{n} \sum_{i=1}^{n} \frac{P_i}{\Delta t}$$

also gleich dem arithmetischen Mittel der einzelnen Pakete, wenn innerhalb von ΔT gleichmäßig nach $\Delta t := \frac{\Delta T}{n}$ gemessen wird. Die Differenz $D_{1...n} - \overline{D} = \sum_{i=1}^{n} D_i \frac{\Delta t_i - \Delta t}{\Delta T}$ verschwindet dann in diesem Fall. Man erkennt damit eine Eigenschaft dieser Methode: das Ankunftsmuster der letzten n Pakete wird als gleichmäßig angenommen, auch wenn es extrem schwankt. Dauert es z.B. sehr lange, bis das erste Paket beim

Empfänger eintrifft, dann die nächsten $n-1$ Pakete „burstartig" schnell hintereinander kommen, so wird der Empfänger gleichmäßige Ankunftszeiten innerhalb von ΔT annehmen. Das „Sammeln" von n Paketen hat aber auch Nachteile: es liegt kein Ergebnis der Messung vor, solange nicht alle n Pakete eingetroffen sind, während hingegen bei Messung mit einem statischen Zeitintervall möglicherweise kein Paket innerhalb eines Meßintervalls eintrifft. Beim statischen Zeitintervall (also per Timer-Interrupt) werden allerdings ebenfalls die Ankunftszeiten der Pakete bzw. Bytes innerhalb des Intervalls als gleichverteilt angenommen. Der Nachteil des Wartens auf alle Pakete kann aber durch einen *dynamischen Mechanismus* aufgehoben werden, indem man nach Erhalt eines Pakets P_i die Zeit Δt_i mißt. Falls $\sum_{k=1}^{i} \Delta t_k > \frac{\Delta t_{min}}{\epsilon_{max}}$ (es seien also bisher i Pakete gesammelt worden), reicht in diesem Fall die Uhrauflösung, so daß ab Δt_i die Berechnung des Durchsatzes zulässig ist. Auch im ungünstigsten Fall, wenn einige Pakete sehr schnell hintereinanderkommen, doch dann länger kein Paket eintrifft und so für die schnellen Pakete noch keine Auswertung vorgenommen wird, ist das Verhalten sinnvoll! Denn die Genauigkeit der Messung dieser „schnellen Pakete" würde nicht ausreichen und so kein brauchbares Ergebnis liefern (z.B. Zeitdifferenz zwischen erstem und letztem Paket ist 0 oder der vorgegebene relative Meßfehler ϵ_{max} wird nicht eingehalten). Trifft schließlich die benötigte SDU ein, so sollte man den ersten schnellen SDUs einen anderen Durchsatz zuordnen, als der letzten. Dies wird erreicht, in dem der Zeitpunkt des Eintreffens der vorletzten SDU für die Durchsatzberechnung so vergrößert wird, daß die Zeitdifferenz zur SDU der letzten Messung gerade $\frac{\Delta t_{min}}{\epsilon_{max}}$ beträgt. Mit der vorgestellten Methode erreicht man also eine minimale Reaktionszeit des Monitors bei gleichzeitiger „Burstkompensation" und Meßfehlerberücksichtigung.

Für beide Ansätze (statisches und dynamisches Zeitintervall) gilt, daß die Größe des Meßintervalls kritisch für die Genauigkeit ist: je kleiner das Intervall desto genauer die Überwachung (analog zu Abschnitt 2.2.1). Wenn z.B. das Intervall einige Dateneinheiten (Rahmen) umfaßt, so werden die gemessenen Werte über dem Intervall gemittelt und daher können kurzzeitige Überschreitungen der vorgegebenen Grenzen unbemerkt bleiben. Deswegen sollte *theoretisch* das (variable) Meßintervall genau einer Rahmenlänge entsprechen, um mit jeder empfangenen Einheit (interne Ereignisse nach Abschnitt 1.1.1) die entsprechenden QoS-Werte zu überprüfen. Praktisch ist dies jedoch sogut wie unmöglich, da erstens hochauflösende Zeitnehmer dafür nötig sind und diese derzeit nicht preiswert für Endsysteme zur Verfügung stehen[15] und zweitens durch die hohe Meßfrequenz (Auflösung nach Abschnitt 1.1.1) ein großer Platz- und Rechenzeitbedarf (damit also Systembelastung) entsteht. Daher muß das Intervall zwangsweise mehrere Rahmen umfassen und der zu messende Wert wird über dem Intervall gemittelt. Gilt es lediglich eine Zielvorgabe langfristig einzuhalten, so kann man das Intervall entsprechend vergrößern.

[15] hier stellt man sich die Frage, warum mit der Verfügbarkeit von Hardware für Hochgeschwindigkeitskommunikation nicht auch hochauflösende Zeitnehmer in diese integriert wird.

Auswertung

Die Auswertung der Meßergebnisse besteht aus dem *Vergleichen der Dienstgüte-Vorgabe mit der erbrachten Dienstqualität* im letzten gemessenen Intervall und der Erstellung einer Beurteilung, ob die Dienstgüte verletzt wurde. Die Auswertung kann beim Empfänger oder beim Sender erfolgen und sollte vom Monitor selbst vorgenommen werden (anstatt von der Protokoll-Instanz). Dazu muß der QoS-Monitor Kenntnis der QoS-Parameter-Vorgaben haben. Liegen die gemessenen Werte außerhalb der Vorgaben, so ist dies unter Angabe der verletzten Parameter dem QoS-Manager, bzw. der Protokollinstanz anzuzeigen. Eine „hohe" Dienstgüte ist je nach Parameter zu interpretieren: beispielsweise beim Durchsatz ein hoher Wert, bei der Verzögerung ein niedriger Wert.

- Auswertung beim Sender
  Erfolgt die Auswertung beim Sender, so müssen alle vom Empfänger gemessenen QoS-Werte mittels einer speziellen Protokolldateneinheit (PDU) (vom Transportprotokoll oder vom QoS-Manager) oder einer Monitor-PDU (dazu wäre noch ein Monitor-Protokoll nötig!) über den getrennten Signalisierungskanal zum Sender geschickt werden.

- Auswertung beim Empfänger
  Erfolgt die Auswertung beim Empfänger, so müssen die gemessenen QoS-Werte dort mit den QoS-Vorgaben verglichen werden. Dazu muß der Empfänger Kenntnis über die QoS-Zielvorgaben haben, die ihm ja aber durch den Verbindungsaufbau bekannt sein dürften. Erst bei Verletzung der vereinbarten Dienstgüte braucht eine spezielle Protokoll-Steuerungsnachricht (QoS-Verschlechterungsnachricht, QoS-Alarmmeldung) verschickt werden. Da nicht jedesmal die Meßwerte übertragen werden müssen, ist diese Methode zu bevorzugen.

Zusätzlich zur Beurteilung, ob eine QoS-Vorgabe verletzt wurde, kann praktischerweise die Auslösung einer Aktion anhand von *Schwellwerten* nach Abbildung 2.9 erfolgen. Wird ein Schwellwert über- bzw. unterschritten, so kann bereits eine Reaktion veranlaßt werden, um dem Überschreiten der „harten" Grenzwerte entgegenzuwirken. Es handelt sich also um eine Art „Vorwarn-Mechanismus" der allerdings nur bei entsprechender Tendenz (momentane QoS strebt nahezu monoton auf die Grenze zu, Betrachtung des Gradienten) angewendet werden sollte, um Hysterese-Effekte zu vermeiden. Bei der statistischen Klasse entfällt er möglicherweise, da ja u.U. eine gewisse Anzahl an Überschreitungen der QoS-Grenzen erlaubt ist. In diesem Fall könnte man vor Erreichen der erlaubten Anzahl an Verletzungen ebenfalls schon Korrektur-Maßnahmen einleiten (Schwellwert für die Wahrscheinlichkeit). Ist $p = 1$, so ist ein Überschreiten der Grenzwerte nicht erlaubt und es werden normalerweise die Ressourcen nicht für die Spitzenwerte reserviert, sondern für Werte in der

Nähe der Zielwerte[16] (meist Intervall-Mitte). Ein Schwellwertmechanismus könnte auch hier bereits im voraus die Bereitstellung zusätzlicher Ressourcen auslösen, um Spitzenwerte ebenfalls erfüllen zu können.

Bei deterministischen Parametern kann eine Überschreitung des Schwellwertes in Richtung auf eine „gute" QoS normalerweise ignoriert werden, da ja z.B. Policing-Mechanismen (s. Abschnitt 1.3 Seite 8) dafür sorgen, daß die Anwendung nicht „mehr" QoS nutzt, als ihr zusteht. Solche Mechanismen müßten auch in Netzwerkelementen implementiert sein, denn sonst könnten z.B. durch Zwischenpufferung und Weiterleiten mit einer höheren Ausgangsrate als der Eingangsrate auch die maximale Senderate überschritten werden.

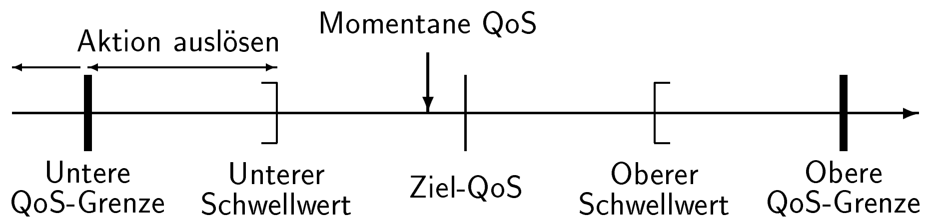


Abbildung 2.9: Schwellwerte und Grenzen der QoS-Parameter

2.2.6 Messung der QoS-Parameter

In diesem Abschnitt wird hergeleitet, wie man die einzelnen QoS-Parameter aus Abschnitt 2.2.1 messen kann.

Durchsatz

Über einem Intervall Δt wird die Anzahl an empfangenen Dateneinheiten (protokollinterner Zähler) gemessen und anschließend der aktuelle Durchsatz errechnet. Wie in Abschnitt 2.2.5 bereits erläutert, ist zu überlegen, ob Δt statisch festgelegt wird oder dynamisch schwankt, definiert durch eine vorgegebene Anzahl der empfangenen Dateneinheiten. Im einen Fall wird Δt festgelegt, im anderen die Anzahl n der empfangenen Rahmen und Δt dann gemessen. Wichtig ist auch die Granularität der Dateneinheiten: In welcher Bezugsgröße wird gemessen (Rahmen oder Bytes)? Hier sollte man sich nach der QoS-Vorgabe richten und die gleiche Einheit wählen. Die *Spitzenrate* wird einfach als Maximum dieses momentanen Durchsatzes mitgeführt, die *mittlere Rate* wird über alle bisher gemessenen Intervalle gemittelt

[16] Ein Überschreiten des Zielwerts in Richtung „höhere QoS" sollte – beispielsweise bis zur Standardabweichung σ – durch entsprechende Reservierung der Ressourcen schon zu Anfang berücksichtigt werden, damit nicht permanent Maßnahmen ergriffen werden müssen.

oder bei Angabe eines Mittelungsintervalls I dieses benutzt. Die *Burstgröße* kann über die maximale Anzahl der zur Spitzenrate gesendeten Dateneinheiten ermittelt werden. Beispiel: Im Zeitraum $\Delta t = 50\mu s$ wurden 6 Rahmen a 125 byte empfangen $\Rightarrow \mu_t = 120$Mbit/s.

Fraglich ist, ob man z.B. beim Sender den Durchsatz der von der Anwendung empfangenen Dienstdateneinheiten mißt oder den der an den darunterliegenden Dienst übergebenen Protokolldateneinheiten und analog beim Empfänger die empfangenen PDUs oder die an die Anwendung gelieferten SDUs betrachtet. Im ersten Fall wird der effektive Durchsatz gemessen, d.h. es geht die durch das Protokoll und die darunterliegende Netzwerkschicht bewirkte Verzögerung ein, im anderen Fall nur der effektive Durchsatz der darunterliegenden Netzwerkkomponente. Wird nur der effektive Durchsatz des Transportsystems gemessen, so muß aufgrund anderer Meßdaten entschieden werden, ob der Einbruch im Durchsatz eventuell auf Protokollmechanismen selbst zurückgeht (wie etwa Übertragungswiederholungen) oder durch die darunterliegenden Schichten verursacht wurde. Da das Protokoll normalerweise, wie im Abschnitt 2.2.3 erläutert wurde, die Einhaltung der geforderten QoS-Parameter garantiert, kann die Verschlechterung also sonst nur von den darunterliegenden Schichten verursacht worden sein. Wird der effektive Durchsatz der darunterliegenden Netzwerkkomponente gemessen, so muß ein anderer Vergleichswert für diesen gemessenen Durchsatz verwendet werden, der dann die Verzögerung durch das Transportprotokoll mitberücksichtigt. Das wird durch die QoS-Abbildungsfunktionen erledigt. Wird also eine Verschlechterung des effektiven Durchsatzes festgestellt, so muß erst die Ursache dafür festgestellt werden und danach kann der QoS-Manager versuchen, durch Feinjustierung (Fine-Tuning) die momentane Verschlechterung zu kompensieren (z.B. durch Änderung der Dringlichkeiten oder Vergrößerung der Pufferanzahl).

Verzögerung

Die Verzögerung ist die verstrichene Zeit zwischen Empfang einer (N)-SDU an der (N)-Schnittstelle[17] bei der Sendeinstanz und dem Ausliefern dieser (N)-SDU bei der Empfängerinstanz durch die (N)-Schnittstelle. Dazu ist es nötig, daß die Sendeinstanz die (N)-SDU direkt nach Erhalt von der Schnittstelle mit einem *Zeitstempel* versieht. Setzt man *synchronisierte Uhren* bei Sender und Empfänger voraus, so kann dieser Zeitstempel mit der (N)-PDU zum Empfänger verschickt werden. Zum Zeitpunkt der Auslieferung der (N)-SDU an den empfangenden (N)-Dienstnutzer zieht die empfangende Instanz diesen Zeitstempel von der aktuellen Zeit ab und erhält so die Verzögerung. Leider ergibt sich hier das Problem der Uhrensynchronisation und der Auflösung der Uhren, falls die Verzögerung etwa durch direkte räumliche

[17] eigentlich wird an der (N)-Schnittstelle eine (N)-IDU (**I**nterface **D**ata **U**nit) – bestehend aus einer (N)-ICI (**I**nterface **C**ontrol **I**nformation, sozusagen Aufrufparameter) und der (N)-SDU – übergeben.

Nachbarschaft sehr gering ist. Die Uhrensynchronisation könnte durch ein Protokoll wie etwa NTP[18] erfolgen. Es bleibt eine minimale Abweichung der beiden Uhren bei Sender und Empfänger voneinander, die durch die Uhren-Synchronisation aus technischen Gründen nicht unterschritten werden kann. In [RFC 1305] wird diese Abweichung jedoch in der Größenordnung von Nanosekunden angegeben, was für die meisten Anwendungen ausreichen sollte (deren Anforderungen liegen meist im Millisekunden-Bereich) und außerdem auch der Signallaufzeit „zugeschlagen" werden könnte (die Stationen lägen dann geographisch weiter auseinander). Berücksichtigt werden muß allerdings die Tatsache, daß die Meßergebnisse zum Zeitpunkt der Synchronisation verfälscht oder unbrauchbar werden können (wird über die Synchronisation und die sich daraus ergebende Abweichung informiert, so können die Meßergebnisse um den Synchronisationszeitpunkt jedoch wieder brauchbar werden). Ein weiterer Vorteil durch den Einsatz von Zeitstempeln ergibt sich, falls der Sender z.B. die (N)-SDUs mit einer niedrigeren Rate als der Mindestrate, die er selbst verlangt hat, an die (N)-Schnittstelle übergibt. So nutzt der Sender seine verlangte „Mindest-QoS" gar nicht aus (was dem Dienstanbieter ja eigentlich nur recht sein kann) und der empfangsseitige Monitor stellt eine Unterschreitung der verlangten minimalen QoS fest. Der Empfänger kann jedoch aufgrund der mit der (N)-PDU übermittelten Sendezeitstempel unterscheiden, ob die QoS vom Netz verletzt oder bereits beim Sender unterschritten wurde.

Eine andere Möglichkeit zur Messung der Verzögerung wäre, eine Quittung für die empfangene Einheit zu verschicken und so den Sender alleine die Zeit zwischen Abschicken und Ankunft der Quittung messen zu lassen. Dazu muß aber prinzipiell die exakte Laufzeit der Quittung bekannt sein (die zudem ja auf einem getrennten Signalisierungskanal mit einer anderen Übertragungscharakteristik verschickt wird), was hier die Schwierigkeit darstellt.

Verzögerungsschwankung

Die *maximale* Verzögerungsschwankung ist einfach zu messen, indem man nach Messung der Verzögerung deren Minimum und Maximum mitführt und ihre Differenz bildet. Die *momentane* Verzögerungsschwankung erhältlt man durch Differenzbildung der gemessenen Ende-zu-Ende Verzögerung d_i von der Zielverzögerung D. Als Zielverzögerung wurde hier das momentane arithmetische Mittel der Verzögerungswerte gewählt, so daß die Abweichung vom momentanen Mittel als J_i gemessen wird. Zusätzlich wird die Einhaltung der Bedingung $d_{max} - d_{min} \leq J_{max}$ überprüft.

[18]**N**etwork **T**ime **P**rotocol, siehe [RFC 1305].

Fehler

Je nach Protokoll-Mechanismus werden die unterschiedlichsten Fehler erkannt und behandelt. Daher sind die zu messenden Parameter vom jeweiligen aktiven Protokollmodul (z.B. Vorwärtsfehlerkorrektur, Übertragungswiederholung) abhängig.

- Datenfehler
  In Abhängigkeit vom Protokollmechanismus werden bei Feststellen eines Fehlers protokollinterne Zähler erhöht, die die Anzahl der Datenfehler festhalten. Für die unterschiedlichen Fehlerarten sollten auch verschiedene Zähler existieren. So z.B. für Anzahl falscher Bits, Bytes, oder Rahmen. Datenfehler können beispielsweise durch Prüfsummen erkannt oder sogar korrigiert werden.

- Datenverlust
  Auch hier sollte der Verlust eines Rahmens gezählt werden. Der Verlust kann z.B. empfangsseitig durch eindeutige PDU-Folgenummern festgestellt werden oder sendeseitig durch Ablaufen eines Zeitnehmers, der auf ein Ausbleiben einer Quittung hindeutet.

- Datenduplikate
  Prinzipiell wie Datenverlust zu behandeln und ebenfalls mit Folgenummern zu erkennen.

- Angezeigte Fehler
  Manche Fehler können nur erkannt, aber nicht behoben werden. Solch ein Fehler wird durch ein Dienstprimitiv *angezeigt*. Es muß also die Anzahl dieser Fehler gezählt, d.h. ein Zähler jedesmal bei Vorkommen des Primitivs erhöht werden. Die Behandlung angezeigter Fehler darunterliegender Schichten erfolgt meist durch einen Austausch von Statusinformation (=Synchronisation) beider Transportdienstinstanzen.

2.2.7 Aspekte des Multicasting

Wie bereits in Abschnitt 2.2.5 erwähnt, verschärft sich das Problem einer Auswertung beim Sender hier erheblich. Die Anzahl der Meßdaten wird noch mit der Zahl der Empfänger multipliziert! Diese Datenmenge und Auswertung trifft dann eine einzelne QoS-Monitorinstanz beim Sender, die dadurch schnell überlastet werden könnte. Daher ist eine Auswertung bei den Empfängern zu bevorzugen, die sich nur im Bedarfsfall an die sendende Instanz wenden.

2.2.8 Monitor-Schnittstellen

Wie in Abbildung 2.5 zu erkennen ist, sind für eine QoS-Monitorinstanz mindestens zwei Schnittstellen zu definieren: Einmal die Schnittstelle zwischen QoS-Monitor und dem Protokoll, bzw. dem Monitor-Modul des Protokolls, die mit *MP-Schnittstelle* bezeichnet werden soll. Zum zweiten die Schnittstelle zwischen QoS-Monitor und QoS-Manager, die mit *MM-Schnittstelle* bezeichnet wird. Für beide Schnittstellen wird eine beispielhafte Mindest-Funktionalität vorgeschlagen, die selbstverständlich *nicht* so implementiert werden muß oder sogar nicht so implementiert werden sollte (Ineffizienz durch viele Prozeduraufrufe).

MP-Schnittstelle

Richtung Protokoll → Monitor

1. `NewConnectionInd(ConnectionID,QoSRequirements)`
   Damit zeigt das Protokoll dem QoS-Monitor die Eröffnung einer neuen Verbindung an. Das könnte auch der QoS-Manager mitteilen, da er die entsprechenden QoS-Vorgaben hat. Über die Vorgaben muß jedoch erst mit den Partnern verhandelt werden und diejenige Instanz, die zuerst die endgültige Vorgabe kennt, sollte auch den Monitor benachrichtigen. `ConnectionID` ist dabei eine eindeutige Verbindungskennung und `QoSRequirements` eine QoS-Parametervorgabe, die dem Protokoll bekannt ist.

2. `CloseConnectionInd(ConnectionID)`
   Damit wird das Ende einer Verbindung signalisiert.

3. `EventInd(ConnectionID,TimeStamp,Event)`
   Zeigt der Monitorinstanz ein Eintreten eines Ereignisses an. Der `TimeStamp` ist die Systemzeit zum Zeitpunkt des Ereignisses. `Event` charakterisiert das Ereignis, so daß der Monitor noch zugehörige weitere Daten bekommt oder auslesen kann.

Richtung Monitor → Protokoll

1. `GetParameter(ConnectionID,ParameterList,ValueList)`
   Liest den Wert eines oder mehrerer Parameter einer angegebenen Verbindung aus. Diese Funktion ermöglicht eine periodische Abfrage (Polling) der Werte (ist zu vermeiden aufgrund hoher Systembelastung), bzw. ein Auslesen weiterer Parameter nach einem `EventInd()`.

2. `GetParameterAddress(ConnectionID,ParameterName,Address)`
   Gibt die Speicheradresse eines Parameters für eine angegebene Verbindung zurück. Absicht dieses Befehls ist, dem QoS-Monitor *direkten* Zugriff auf protokollinterne Variablen zu erlauben, um zum Beispiel Monitoring mit Hilfe des Netzwerk-Managements zu ermöglichen. Auf Schreib-/Lesekonflikte zwischen Protokoll und Monitor muß hierbei im Fall von nicht-atomaren Variablen geachtet werden. Der hierfür relevante Adreßraum des Protokolls sollte natürlich für die Monitorinstanz zugänglich sein.

MM-Schnittstelle

Richtung Manager → Monitor

1. `StartMonitoring(ConnectionID,ParameterSet,Policy)`
   Mit dieser Funktion kann der QoS-Manager das Monitoring einzelner Parameter einschalten. `Policy` bezeichnet dabei das gewünschte Verhalten des Monitors: sollen permanent Meßergebnisse ausgeliefert werden (weil Meßergebnisse z.B. einer anderen Instanz oder dem Dienstnutzer zugänglich gemacht werden sollen), oder nur eine Benachrichtigung im Falle einer QoS-Verletzung erfolgen?

2. `StopMonitoring(ConnectionID,ParameterSet)`
   Damit kann das Beobachten einzelner Parameter ausgeschaltet werden.

3. `ReportPolicy(ConnectionID,Policy)`
   Mit diesem Befehl wird die augenblickliche Policy geändert.

4. `QoSRequirementsChange(ConnectionID,NewRequirements)`
   Dem Monitor wird eine Änderung der QoS-Vorgabe mitgeteilt.

Richtung Monitor → Manager

1. `GetQoSRequirements(ConnectionID,RequirementList)`
   Der Monitor kann damit für eine Verbindung die QoS-Vorgabe erfragen.

2. `QoSAlertInd(ConnectionID,DegradedParameterList)`
   Der Monitor teilt die Verletzung der QoS-Vorgabe mit. Die betroffenen Parameter werden in der `DegradedParameterList` mitgegeben. Der Monitor muß in diesem Fall dem Manager und den Protokollinstanzen genügend Zeit einräumen, damit diese geeignete Maßnahmen treffen können (d.h. feststellen, ob lokale Ressourcen betroffen sind. Falls ja: Protokoll-Ressourcen anpassen, falls nein: anderen QoS-Manager (Sendeseite) benachrichtigen). Im Falle der Benachrichtigung der Sendeseite sollte diese Nachricht möglichst bestätigt

übertragen werden. Erst nach Abwarten eines Intervalls oder Empfang einer Nachricht vom QoS-Manager sollten Alarmmeldungen wieder verschickt werden.

3. `MeasuredQoSDataReq(Destination,ConnectionID, MeasuredParameterList)`
   Damit werden die gemessenen Werte entweder an den *lokalen* QoS-Manager übermittelt oder den *entfernten* der zugehörigen Partnerinstanz (Angabe des Ziels über `Destination`).

2.3 Zusammenfassung der Merkmale

In der nachstehenden Tabelle 2.1 sind die wichtigsten Merkmale des QoS-Monitors aufgeführt:

Merkmal	Begründung
Monitoring von Simplex-Kanälen	größere Flexibilität, Ressourcenschonung
Kontrollnachrichten über separaten Signalisierungskanal	Simplex-Kanal für Datenstrom
zu überwachende QoS-Parameter: Durchsatz, Ende-zu-Ende Verzögerung, Delay-Jitter, Datenfehler	variable Parameter
3 Parameterklassen: deterministisch, statistisch, bestmöglich	Flexibilität, bessere Ressourcennutzung
selbständige QoS-Monitorinstanz	Asynchrone Auswertung, Parallelität
Messung an der Dienstschnittstelle (Messung der SDUs)	Einfacher Vergleich mit den Dienstnutzer-Vorgaben
Durchsatzmessung mit dynamischem Zeitintervall, Sammeln von SDUs	geringere Systembelastung, Uhrauflösung u. Meßfehler werden berücksichtigt
Messung der Verzögerung über Zeitstempel	fehlender Rückkanal mit identischer Übertragungscharakteristik
Auswertung beim Empfänger	weniger Kommunikationsaufwand
Schwellwerte zur Vorwarnung	Vermeidung von QoS-Verletzungen

Tabelle 2.1: Merkmale des QoS-Monitors

3. QoS-Monitoring und Netzwerkmanagement

Die Nähe der Aufgaben des Netzwerkmanagements (NM) und des QoS-Monitoring legt es nahe, das QoS-Monitoring mit Hilfe des Netzwerkmanagements zu realisieren. Zuerst wird die OSI-Sicht des Netzwerkmanagements dargestellt und erläutert, wie man eine QoS-Architektur in das Netzwerkmanagement integrieren könnte. Schließlich wird das Simple Network Management Protocol (SNMP) kurz vorgestellt und eine Realisierung des QoS-Monitoring mit Hilfe von SNMP skizziert.

3.1 OSI Netzwerkmanagement

In [Blac 92] werden folgende Aufgaben des Managements aufgezählt: Planung, Organisation, Monitoring, Abrechnung und Steuerung von Aktivitäten und Ressourcen. Die OSI- und Internetstandards sehen die letzten drei Aufgaben als wesentlich für das *Netzwerkmanagement* an. Im einzelnen sind das die Funktionsbereiche: Fehlerbehandlung (Fault Management), Leistungsüberwachung und Tuning (Performance Management), Konfigurationsverwaltung (Configuration Management), Kostenabrechung (Accounting Management) und Sicherheitsaufgaben (Security Management). Logische oder physikalische Objekte im Netzwerk sollen verwaltet werden. Diese Objekte werden als *Managed Objects* (kurz MO) bezeichnet. Der Zugriff auf diese Objekte erfolgt mittels eines *Agenten Prozesses* im *Agenten*, der eine Netzwerkkomponente darstellt, die einige der zu verwaltenden Objekte enthält. Ein *Manager* kommuniziert über einen *Manager Prozeß* mit den Agenten-Prozessen mittels eines Managementprotokolls (CMIP–Common Management Information Protocol) und steuert dadurch die entsprechenden Operationen in den Agenten, wodurch letztendlich auf die zu verwaltenden Objekte zugegriffen wird. Die Managed Objects, die ja auch logischer Art sein können, werden durch sogenannte *Management Information Bases* (kurz *MIB*) repräsentiert. Hierbei handelt es sich in der Regel um eine Datenbank, die die Attribute der MO enthält. Der Zugriff auf ein MO, bzw. ein Attribut desselben erfolgt über einen eindeutigen Objektnamen, der durch eine genormte Namenshierarchie (Management Information Tree) festgelegt ist. Auf den

Managed Objects können folgende Operationen durchgeführt werden: Create (Erzeugen), Delete (Löschen), Action (Ausführen einer Operation), Get Value (Wert lesen), Add Value (Wert zu einem Attribut hinzufügen), Remove Value (Wert aus einer Menge von Werten löschen), Replace Value (Wert ersetzen), Set Value (Wert setzen). Zusätzlich besitzen MO die Fähigkeit sogenannte Benachrichtigungen (Notifications) über interne Ereignisse an den Agenten abzugeben, der dann wiederum den zuständigen Manager benachrichtigen kann. Ansonsten läuft die Kommunikation nach dem Anfrage-Verfahren (Polling) ab: Der Manager stellt eine Anfrage an den Agenten (Wert lesen oder setzen, etc.) und der Agent teilt dem Manager das Ergebnis der Anfrage in einer Rückantwort mit. Abbildung 3.1 verdeutlicht den Zusammenhang:

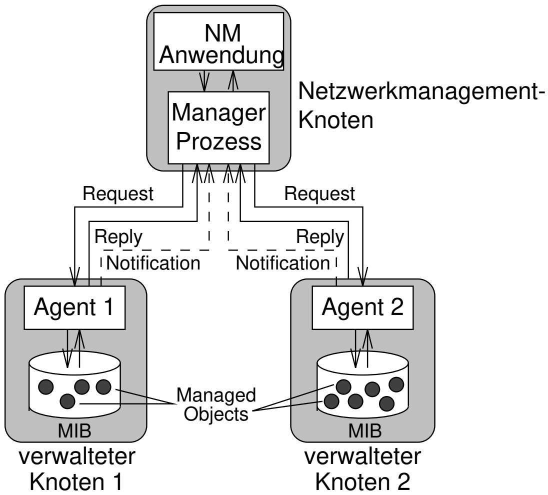


Abbildung 3.1: Netzwerkmanagement - Agenten und Manager

Ein Netzwerkmanagement sollte beispielsweise ermöglichen, Drucker, Router und Rechner zu überwachen und zu konfigurieren. Da ein Netzwerk hauptsächlich aus heterogenen Komponenten – also Komponenten verschiedener Hersteller – besteht, ist es wichtig, daß das Management-Framework (also die Management-Architektur) eine optimale Offenheit ermöglicht. Das Netzwerkmanagement nach OSI setzt auf Diensten der Schicht 7 auf, so daß der gesamte Protokollturm des ISO/OSI-Basisreferenzmodells auch in einfachsten Geräten implementiert werden müßte.

3.2 Integration einer QoS-Architektur

Da die Dienstqualität durch Parameter festgelegt werden kann, spricht nichts dagegen, diese QoS-Parameter in einer MIB zu verwalten. Aber nicht nur die ausgehandelten Parameter für eine Verbindung könnten über das Netzwerkmanagement verwaltet werden, sondern auch die aktuellen, bzw. gemessenen QoS-Parameter. Im

Internet-Standard SNMP z.B. stehen auch protokollinterne Parameter von IP (Internet Protokoll) und TCP (Transport Control Protocol) in einer MIB, es wäre also nur eine logische Fortsetzung, die QoS-Parameter eines Hochgeschwindigkeits-Transportprotokolls ebenfalls in einer MIB zugänglich zu machen.

In [JuSP 92] wird die Beziehung zwischen Netzwerkmanagement und der Dienstqualität in B-ISDN untersucht. Dort wird die Architektur aus Abbildung 3.2 vorgeschlagen: Ein QoS-Element in der Anwendungsschicht interagiert mit dem Netzwerkmanagement, der Anwendung und dem Benutzer. Der Benutzer kann durch ein QoS-Element hindurch auf eine Anwendung in der Anwendungsschicht zugreifen. Bei Verbindungsaufbau werden die Anforderungen des Benutzers in einer ihm verständlichen Sprache durch das QoS-Element in entsprechende Dienstattribute umgesetzt. Die kommunizierenden QoS-Elemente verhandeln nun über diese Attribute durch Benutzung des Netzwerkmanagements. Nach erfolgreicher Verhandlung werden die ausgehandelten Attribute an die Anwendung weitergereicht. Für jede Schicht werden QoS-bezogene Daten in einer MIB gehalten und durch das NM verwaltet. Während der Datenaustauschphase hat das QoS-Element die Aufgabe den momentanen QoS-Zustand mittels des NM zu beobachten (Monitoring) und zu erhalten (Maintenance). Eine Änderung in der Dienstgüte kann durch Benachrichtigungen (Notifications) angezeigt werden.

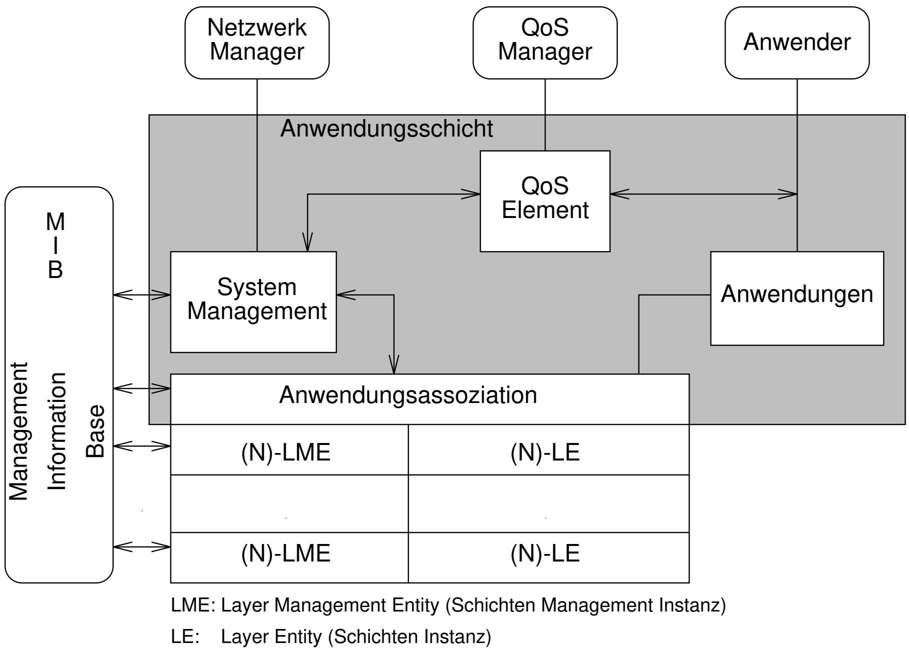


Abbildung 3.2: Eine QoS-Architektur basierend auf Netzwerkmanagement

3.3 SNMP

Das „Simple Network Management Protocol“ stellt eine Implementierung eines Netzwerkmanagementprotokolls für das Internet dar und liegt inzwischen in einer zweiten Version (SNMPv2) vor. Primäre Ziele beim Entwurf des Protokolls waren *Einfachheit* und *Erweiterbarkeit*. Im Gegensatz zum OSI-Ansatz benötigt SNMP lediglich einen verbindungslosen Transportdienst (konkret den Datagrammdienst UDP[1], obwohl beliebige andere verbindungslose Transportprotokolle denkbar sind). Die rasche Verfügbarkeit und Einfachheit von Implementierungen des Protokolls begründen seinen Erfolg gegenüber dem OSI-Management-Framework. Alle zu verwaltenden Objekte werden durch die sogenannte *Structure and Identification of Management Information* (SMI) festgelegt, die mit Hilfe der abstrakten Beschreibungssprache ASN.1 (Abstract Syntax Notation Number One) beschrieben wird. Jedes Managed Object (MO) wird durch einen eindeutigen Bezeichner identifiziert. Zur Übertragung der Managementdaten muß jedoch eine konkrete Transfersyntax benutzt werden, die bei SNMP durch die *Basic Encoding Rules* (BER) festgelegt ist. Als Datentypen für die MOs stehen skalare Datentypen, sowie konzeptionelle Tabellen zur Verfügung, die aus Zeilen skalarer Datentypen bestehen. SNMPv2 stellt zur Abfrage von MO-Instanzwerten die drei Operationen *Get-request, Get-next-request,* und *Get-bulk-request* zur Verfügung. Zur Änderung von MO-Instanzwerten wird die Operation *Set-request* benutzt, die auch zur Ausführung von Aktionen benutzt werden kann. Unaufgeforderte Meldungen vom Agenten zum Manager sind durch *Traps* möglich, der *Inform-request* bietet die Möglichkeit zum Nachrichtenaustausch zwischen Managern. Eine *Management Information Base* (MIB) wird mittels der SMI definiert und stellt die logischen Ressourcen dar, die vom Netzwerkmanagement verwaltet werden. Diese logischen Ressourcen müssen letztendlich auf reale Ressourcen abgebildet werden, was die Aufgabe der *Anbindung* ist (s. [Jens 94]).

Ein Beispiel für eine konzeptionelle Tabelle, die die Zustände von TCP-Verbindungen wiedergibt und mittels SMI in einer MIB (SNMP MIB-II, RFC1213) vereinbart wurde, ist in Abbildung 3.3 zu sehen.

3.4 QoS-Monitoring mit SNMP

Da auch Performance-Monitoring zu den Aufgaben des Netzwerkmanagements gehört und QoS-Parameter gut in einer MIB abgelegt werden können liegt es nahe, das QoS-Monitoring mit Unterstützung des Netzwerkmanagements zu realisieren. Eine praktische Hilfe ist auch der *Trap-Mechanismus*, der Werte überwacht und automatisch bei Überschreiten von Grenzen eine Nachricht (den *Trap*) an einen Manager verschickt. So könnte man die Einhaltung der QoS-Parametervorgaben ein-

[1] **U**ser **D**atagram **P**rotocol

fach prüfen, indem man die durch den QoS-Monitor gemessenen QoS-Werte in einer MIB ablegt und bei einer Verletzung automatisch einen Trap als QoS-Alarmmeldung verschickt. Eine Architektur nach Abbildung 2.5 (Abschnitt 2.2.3, S. 23) wird also folgendermaßen erweitert: Der QoS-Manager erhält einen SNMP-Manager-Teil (Netzwerkmanagement-Anwendung) und in den QoS-Monitor wird ein SNMP-Agent integriert. Die Kommunikation zwischen QoS-Monitor und QoS-Manager, sowie zwischen QoS-Manager und QoS-Manager wird also über das SNMP abgewickelt. Nach Abschnitt 2.1 müßte das SNMP auf dem getrennten Signalisierungskanal realisiert werden, da bei einer Simplex-Verbindung keine Rückrichtung zur Verfügung steht.

Leider bedeutet ein ständiges Überprüfen der QoS-Werte in der MIB eine hohe (oft überflüssige) Systembelastung, zudem würde momentan die erforderliche Auflösung nicht erreicht (in [Jens 94] wird das minimale Überprüfungsintervall mit $\frac{1}{100}$s angegeben). Daher ist ein *ereignisgesteuerter* Ansatz zu wählen. Dadurch wird die Implementierung nicht mehr so einfach wie oben dargestellt, denn die Auswertung wird jetzt doch wieder von der eigentlichen Monitor-Instanz vorgenommen und bei Verletzung eines Werts, kann diese dann einen Trap initiieren. Damit übernimmt das SNMP nur noch die Kommunikationsfunktionen und bietet mehr Flexibilität, da die QoS-Werte auch von außerhalb des Systems abgerufen werden können. Die interne (lokale) Kommunikation zwischen QoS-Monitor und QoS-Manager auch über SNMP abzuwickeln (wie in Abbildung 3.4 dargestellt), bedeutet aber erhöhten Aufwand, da z.B. Parameter nur mittels der *Set-Request* Funktion *einzeln* übermittelt werden können. Weil es sich bei den auszutauschenden Daten meistens um Tabellenzeilen mit mehreren Parameterwerten handelt, ist diese *Set-Request* Methode zu umständlich und auch zu langsam (man bedenke, daß die Daten ja auch noch gemäß BER kodiert werden!). SNMPv2 ist also auch noch zu unflexibel und nicht mächtig genug, um etwa einen gemeinsamen Agenten für QoS-Manager und QoS-Monitor zu nutzen, denn dazu müßte man auch Traps von Agenten (QoS-Manager Empfangsseite) zu Agenten (QoS-Manager Sendeseite) schicken können. Was also bleibt, ist die Idee, die QoS-Parameter und die aktuellen Meßwerte in einer QoS-MIB verfügbar zu machen, die mittels eines zugehörigen Agenten von außerhalb abgefragt (nur Lesen) werden kann.

```
-- the TCP Connection table
-- The TCP connection table contains information about this
-- entity's existing TCP connections.

tcpConnTable OBJECT-TYPE
    SYNTAX  SEQUENCE OF TcpConnEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
            "A table containing TCP connection-specific
            information."
    ::= { tcp 13 }

tcpConnEntry OBJECT-TYPE
    SYNTAX  TcpConnEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
            "Information about a particular current TCP
            connection.  An object of this type is transient,
            in that it ceases to exist when (or soon after)
            the connection makes the transition to the CLOSED
            state."
    INDEX   { tcpConnLocalAddress,
              tcpConnLocalPort,
              tcpConnRemAddress,
              tcpConnRemPort }
    ::= { tcpConnTable 1 }
```

Abbildung 3.3: konzeptionelle Tabelle für TCP-Verbindungen

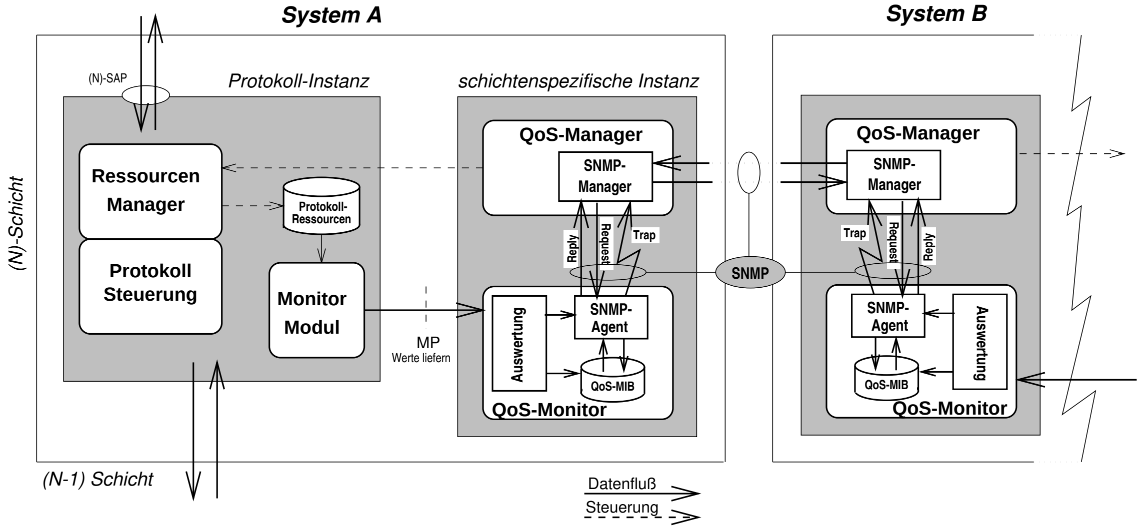


Abbildung 3.4: QoS-Monitoring mit SNMP

4. Implementierung

Dieses Kapitel beschäftigt sich mit der Implementierung des QoS-Monitors für ein Transportprotokoll. Zuerst werden *allgemein* implementierungstechnische Details untersucht und eine plattformübergreifende Implementierung entwickelt. Die *konkrete* Implementierung ist dann die Abbildung auf eine bestimmte Zielarchitektur (hier: OSF/1 auf einer DEC Alpha-Workstation) und ein bestimmtes Transportprotokoll (hier: PATROCLOS).

4.1 Generische Implementierung

„Generisch" ist hier im Sinne von plattform- oder systemunabhängig gemeint. Zunächst ist zu überlegen, welche Änderungen am Transportprotokoll vorgenommen werden müssen, um die Meßdaten bereitzustellen. Die Architektur nach Abbildung 2.5 sieht den *QoS-Monitor* als eigenständige Instanz vor. Im folgenden wird angenommen, daß diese als Prozeß (oder auch Thread, s. Fußnote 11 auf S. 23) realisiert wird und somit *nebenläufig* zum Protokoll agiert. Ein weiterer Vorteil dieser Realisierung ist die Nutzung des gemeinsamen Adreßraums zum Nachrichtenaustausch (*Kooperation*, s. [Wett 93]), was Kopieroperationen weitgehend vermeidet. Oberstes Ziel der Implementierung ist es, das Transportprotokoll so wenig wie möglich durch das Monitoring zu beeinflussen.

4.1.1 Protokollerweiterung

Die Betrachtungen aus Kapitel 2 lassen zunächst darauf schließen, daß das Protokoll um *Zeitstempel* erweitert werden muß, falls diese protokollseitig noch nicht vorgesehen oder zu ungenau für unsere Zwecke sind. Die Zeitstempel werden zur Messung der Verzögerung und der Verzögerungsschwankung benötigt.

Zeitstempel

Es wird *pro SDU* ein Zeitstempel benötigt. Dieser Zeitstempel muß vom Sender zum Empfänger (oder den Empfängern) *übertragen* werden, um die Verzögerung beim Empfänger zu ermitteln. Der Zeitstempel muß die „globale Zeit“ (durch Uhrensynchronisation) kodieren. Falls das Protokoll die SDU segmentiert, ist entweder *allen* zugehörigen PDUs dieser Zeitstempel im PCI-Header[1] mitzugeben oder nur *einem* PCI-Header (was weniger Redundanz bedeutet und daher zu bevorzugen ist), je nach Protokoll-Mechanismus. Falls das Protokoll letzteres nicht vorsieht (aufgrund konstanter Header-Länge und damit einfacherer Verarbeitung), kann man dazu übergehen, den Zeitstempel noch vor die Nutzdaten zu setzen, denn andernfalls kann bei vielen TPDUs pro TSDU die Redundanz groß werden und schließlich wird man einmal für die genutzten Bandbreiten zahlen müssen. Dabei sollte ein unnötiges Kopieren der Daten vermieden und der Zeitstempel erst dort eingesetzt werden, wo der PCI-Header mit der TSDU zusammengefügt wird. Sollte das Transportprotokoll auch unvollständige TSDUs ausliefern, d.h. die TSDU wird trotz verlorengegangener TPDUs wieder reassembliert und ausgeliefert, so sollte aus Redundanzgründen – entgegen obigem Argument – der Zeitstempel doch in jeder TPDU enthalten sein, falls die erste TPDU verloren geht. Meines Erachtens ist dieses unvollständige Ausliefern von TSDUs nur mäßig sinnvoll, da eine Anwendung in den seltensten Fällen darauf angemessen reagieren kann. Die TSDUs haben eine *von der Anwendung festgelegte Semantik* und diese läßt sich auch so festlegen, daß die Anwendung sinnvoll auf den Verlust einer *ganzen* TSDU reagieren kann, was im Vergleich zum Verlust von Daten *innerhalb* einer TSDU einfacher zu handhaben ist. Als Beispiel dazu läßt sich eine Videoanwendung betrachten: Die Anwendung wird die TSDUs wahrscheinlich so wählen, daß durch eine TSDU ein gesamtes (u.U. komprimiertes) Einzelbild übertragen wird. Der Verlust von Teilen dieses Bildes wird wahrscheinlich größere Störungen beim Betrachten verursachen, als der Verlust eines kompletten Einzelbildes. Mechanismen um auf alle möglichen Fälle von Verlust innerhalb einer TSDU zu reagieren, dürften zu komplex sein, als sie sinnvoll einzusetzen. Stattdessen sollte sich die Anwendung darauf beschränken, den Verlust einer „semantischen Einheit“ (TSDU) kompensieren zu können. Eine Anwendung die z.B. unkomprimierte Sprachdaten überträgt, könnte jedoch u.U. auch einen Verlust von Daten innerhalb einer TSDU tolerieren, nur bekommt man hier auch schon Schwierigkeiten, sobald man nicht reine PCM-Technik[2] verwendet, sondern z.B. DPCM, ADPCM oder Predictive Encoding. Diese Verfahren kodieren nämlich differentiell, weshalb das Fehlen von Zwischenstücken fatal wäre.

Es stellt sich für das weitere Vorgehen mit Zeitstempeln die Frage nach der Genauigkeit, dem Zeitsystem und dem Datenformat für den Datenaustausch.

- Genauigkeit
  Wie in Abschnitt 2.2.6 erläutert wurde, reicht prinzipiell eine Genauigkeit im Nanosekundenbereich ($1\text{ns} = 10^{-9}\text{s}$), da momentan weder die Uhrensynchronisation, noch die meisten Systemuhren feiner auflösen. Mit 1ns lassen sich

[1] **P**rotocol **C**ontrol **I**nformation – protokollinterne Information
[2] **P**ulse **C**ode **M**odulation

Raten von maximal 10^9 SDUs/s auflösen. Da Verzögerungen schonmal im Sekundenbereich (oder sogar Minutenbereich) liegen können, sollte praktisch eine Auflösung „nach oben" in mehreren Stunden (oder ein Tag – entsprechend 86400 Sekunden) ausreichen. Dann müßten allerdings noch die Situationen berücksichtigt werden, falls ein System bereits wieder auf 0 umschaltet.

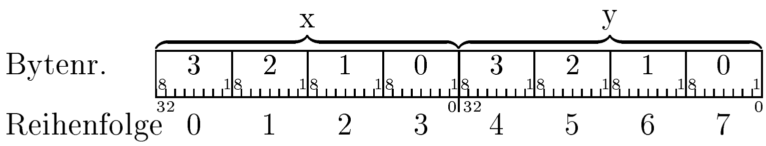
- Zeitsystem
  Die Anzahl der Sekunden bezieht sich auf die Epoche des *1. Januar 1970 00:00:00 UTC*. Die Festlegung des Zeitpunkts „0" erfolgte willkürlich und vereinfacht die Implementierung, da viele UNIX-Systeme die Zeit schon in diesem Format liefern.

  In UTC (Universal Coordinated Time) wird die heutige Standardzeit ausgedrückt, deshalb soll sie auch hier zugrundegelegt und benutzt werden. UTC basiert auf der Internationalen Atomzeit (International Atomic Time, abgekürzt TAI), die durch das Bureau International de l'Heure (BIH) in Paris festgelegt wird. Sie errechnet sich aus den Messungen einiger Cäsium Atomuhren. Eine Sekunde der Atomzeit ist festgelegt durch 9.192.631.770 Zustandsübergänge eines (Cäsium) Cs-133 Atoms. Die Anzahl der Übergänge entsprachen am 1. Januar 1958 00:00:00 Uhr genau einer *mittleren Sonnensekunde*. Diese (astronomische) mittlere Sonnensekunde berücksichtigt, daß die Tage mit der Zeit immer länger werden (aufgrund einer immer langsamer werdenden Erddrehung). Leider weichen die 86400 TAI Sekunden von der mittleren Sonnensekunde inzwischen wieder ab, so daß Atomzeit mit der astronomischen Zeit synchronisiert werden muß. Dazu fügt das BIH alle paar Jahre sogenannte *Schaltsekunden* ein, wenn die Abweichung 800ms beträgt. Das UTC Zeitsystem unterscheidet sich damit von der bisherigen zivilen Standardzeit GMT (Greenwich Mean Time), die astronomischen Ursprungs war. Eine gute Einführung zu diesem Thema findet man in [Tane 94] (S. 124-127) und [RFC 1305].

- Datenformat
  Die Bruchteile einer Sekunde (in Nanosekunden ausgedrückt), lassen sich mit mindestens 30 Bit kodieren. Daher wählt man einfachhalber ein 32-Bit-Wort für die Kodierung der Sekundenbruchteile. Die Anzahl der Sekunden sollte mindestens einen Tag umfassen können, deswegen werden hier mindestens 17 Bit benötigt. Auch hier bietet sich ein 32-Bit Wort als Darstellung für die Anzahl der Sekunden an. Eine Darstellung als Festkommazahl ist ungünstig, da eine Umrechnung zu teuer wäre. Es ist noch die Reihenfolge der beiden 32-Bit-Worte und der Bytes innerhalb der Worte festzulegen[3]. Dabei sollte man sich an den *Basic Encoding Rules* für ASN.1 orientieren ([ISO 8825]). Die Bitreihenfolge für ein Byte ist dort so festgelegt, daß das höchstwertige Bit zuerst übertragen wird, bei ganzzahligen Datentypen wird das höchstwertige

[3] Die Reihenfolge der Bits soll hier nicht interessieren, da angenommen wird, daß der vom Transportdienst in Anspruch genommene Dienst Bytes entgegennimmt.

Byte zuerst übertragen (Big Endian). Somit kann ein Zeitstempel $t = x + y \cdot 10^{-9}$s in 64-Bit wie folgt kodiert werden:



Mit der gewählten Wortbreite und dem Anfangszeitpunkt lassen sich Zeitstempel bis zum Jahre 2106 kodieren, was man nach einem kurzen „Reality Check“ als ausreichend bezeichnen kann.

4.1.2 Sendeseite

Auf Sendeseite sind folgende Schritte im Protokoll nötig: Nach Empfang einer TSDU vom Dienstnehmer wird die Uhr ausgelesen und der Zeitstempel generiert. Dazu muß die systeminterne (aber synchronisierte) Uhrzeit möglicherweise erst in das UTC Format gebracht werden. Das Protokoll hat eventuell noch die Übertragungssyntax der Zeitstempel herzustellen (Byteordering). Der Zeitstempel wird (zusammen mit der SDU-Nummer und der SDU-Größe) in ein Feld von Zeitstempeln der Größe n geschrieben. n kann man durch n_{max} (siehe S. 27) festlegen. Verwendet man den dynamischen Mechanismus für n (Abschnitt 2.2.5, S. 29), so wird nach Ankunft einer SDU die Zeitdifferenz zur *letzten SDU der vorigen Messung*[4] gebildet. Ist diese Zeitdifferenz für die Genauigkeit der Durchsatzmessung bereits ausreichend (beträgt sie z.B. $c \cdot \frac{\Delta t_{min}}{\epsilon_{max}}, c \geq 1$), so kann die Durchsatzberechnung bereits erfolgen und n schwankt damit zwischen 1 und n_{max}. Ein größeres c senkt die Systembelastung, erhöht andererseits die Reaktionszeit des Monitors. Das Sammeln von n TSDUs ist besonders im Falle eines Bursts sinnvoll, da sonst mit jeder empfangenen TSDU ein Signal an den Monitor geschickt werden müßte. Obwohl ein Signal von Thread zu Thread sicherlich nur wenige Befehle des Betriebssystems erfordert, würde dies trotzdem (bei einem Burst) zu einem nicht zu vernachlässigendem „Overhead“ führen. Sind n TSDUs empfangen worden, signalisiert der Protokoll-Thread dem Monitor-Thread *asynchron*, daß nun n Meßwerte vorliegen. Die asynchrone Signalisierung ist wichtig, da das Protokoll in seiner Aktivität durch den Monitor nicht aufgehalten werden darf! Der Monitor beginnt daraufhin, die n Werte aus dem Feld zu kopieren und berechnet nun aus dem ersten und letzten Zeitstempel den gemittelten Durchsatz der letzten n TSDUs. Diese n TSDUs bekommen alle den gleichen mittleren Durchsatz als Momentanwert für den Durchsatz zugewiesen, denn bei dieser Meßmethode wird das Ankunftsmuster der n TSDUs als gleichmäßig angenommen

[4]Bei Verbindungsaufbau sollte ebenfalls ein Zeitstempel übermittelt werden, damit auch der Durchsatz für die erste SDU ermittelt werden kann!

(s. Abschnitt 2.2.5 auf Seite 28). Das für die Meßmethode ungünstigste Ankunftsmuster ist das folgende: wenige sehr schnelle SDUs kommen an und dann folgt eine längere Pause bis zum Eintreffen der nächsten SDU. Ist die Zeitspanne nach den schnellen SDUs für die Meßgenauigkeit noch zu gering (also kleiner als $\frac{\Delta t_{min}}{\epsilon_{max}}$), wird auf die nächste SDU gewartet. Wenn nun alle SDUs den mittleren Durchsatz zugeordnet bekämen, entspräche das nicht mehr der vorliegenden Situation. Abhilfe laßt sich dadurch schaffen, in dem man den ersten $n-1$ SDUs einen anderen Durchsatz zuordnet als der letzten SDU. Da für die ersten $n-1$ SDUs die Genauigkeit noch nicht ausreicht, wird für die Durchsatzberechnung die Ankunftszeit der vorletzten SDU einfach auf die nötige Zeitdifferenz $\frac{\Delta t_{min}}{\epsilon_{max}}$ aufgerundet. Der dadurch entstandene Fehler entspricht dann wieder der Vorgabe. Zu beachten ist außerdem, daß bei Beenden der Verbindung eine noch unvollständige Tabelle zur Auswertung geschickt wird. Auf Sendeseite sind weder Verzögerung noch Jitter oder Datenfehler (höchstens die Anzahl an Übertragungswiederholungen) zu messen, dafür kann aber überwacht werden, ob der Transportdienstnutzer (bzw. die Anwendung) seine Spezifikation der maximalen und mittleren Senderate einhält („Policing" an der Transportdienstschnittstelle, s.S. 8). Es sollten u.U. Plausibilitätsprüfungen der errechneten Werte folgen. Danach folgt die eigentliche Auswertung, d.h. das Überprüfen auf Schwell- bzw. Grenzwerte. Das Protokoll sollte währenddessen in einem zweiten Feld die neuen Zeitstempel sammeln (Prinzip des *Double-Buffering*). Die Umschaltstrategie (scheduling policy) sollte so gewählt werden, daß zwar der Protokoll-Thread mit höherer Priorität als der Monitor-Thread läuft, dieser jedoch auch zyklisch Rechenzeit erhält.

4.1.3 Empfangsseite

Auf der Empfangsseite wird im Protokoll nach Erhalt der TSDU i (wenn die TSDU aus den TPDUs rekonstruiert wurde und an den Transportdienstnutzer weitergereicht werden soll) ebenfalls die Systemuhr ausgelesen (Empfangszeitpunkt T_i^E) und der Sendezeitpunkt T_i^S aus dem übertragenem Zeitstempel gelesen. Beides wird zusammen mit der SDU-Folgenummer, der SDU-Größe und Zählern für Datenfehler ebenfalls in ein Feld der Größe n geschrieben. Wurden n TSDUs empfangen oder reicht die verstrichene Zeit für die Genauigkeit bereits vor Erreichen von n, so wird dem (empfangsseitigen) Monitor-Prozeß signalisiert, daß die n Meßwerte abzuholen sind. Dieser kopiert dann die Daten in eine eigene Meßwerttabelle (zur Entkopplung vom Protokoll) und errechnet (nach einer Plausibilitätsprüfung) den Durchsatz sowie Verzögerung und Jitter für jede TSDU und aktualisiert die QoS-Meßwerte der gesamten Verbindung. Die errechneten Werte werden danach mit den Vorgaben verglichen (Auswertungsteil). Dabei ist zu beachten, daß möglicherweise keine Meßwerte errechnet werden können, falls die Plausibilitätsprüfung negativ ausfällt. Diese ungültigen oder undefinierten Werte dürfen natürlich nicht an der Auswertung teilnehmen und sind daher entsprechend zu kennzeichnen. Die empfangsseitigen Threads sind in Abbildung 4.1 dargestellt. Beim (nicht nur lesenden)

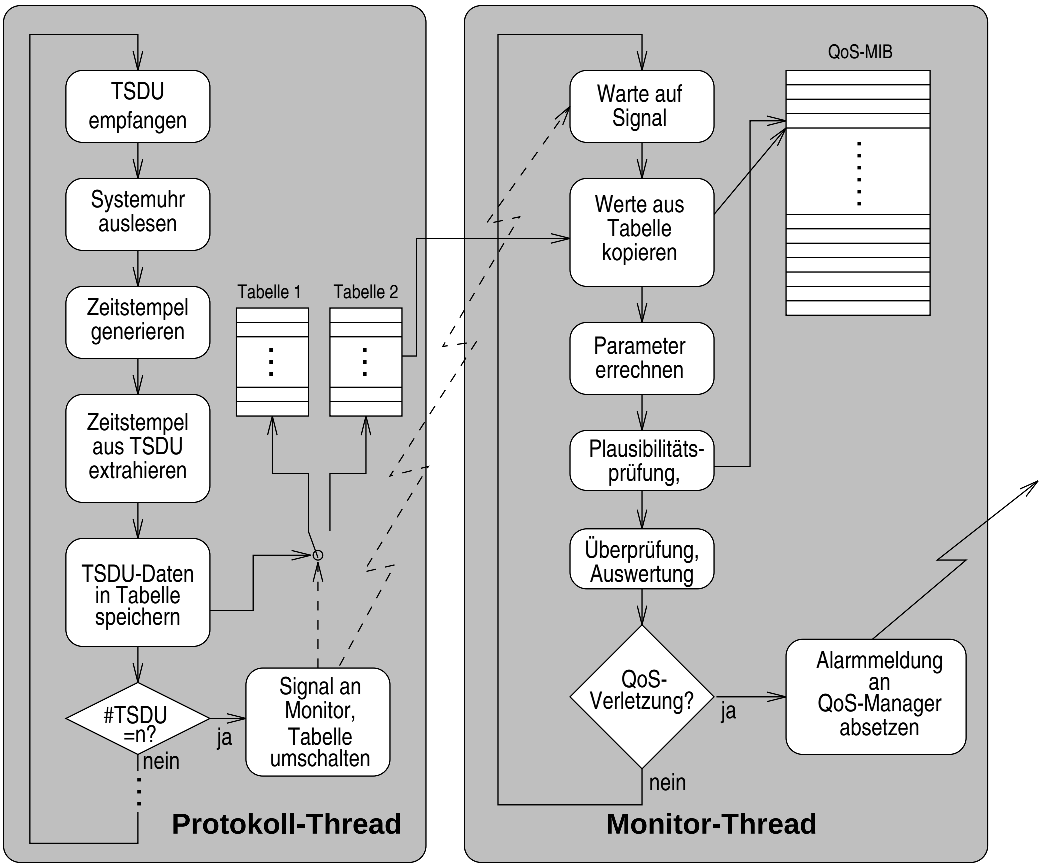


Abbildung 4.1: Ablaufschema der Protokoll- und Monitor-Threads

Zugriff auf gemeinsame Daten müssen die Prozesse jedoch synchronisiert werden (Semaphor-Konzept). So ergibt sich hier z.B. ein Problem, wenn der Monitor mit der Auswertung nicht mehr hinterherkommt und noch eine Tabelle ausliest, die das Protokoll bereits wieder beschreiben möchte. Das Protokoll sollte aber nicht unnötig durch den Monitor aufgehalten werden. Denkbar wäre etwa ein umlenkendes Empfangen des Signals beim Monitor, d.h. der Monitor wird in seiner augenblicklichen Tätigkeit unterbrochen und so beispielsweise zum Umschalten auf die nächste Tabelle gezwungen. Dabei gingen allerdings Meßwerte verloren. Eleganter ist die Methode, den Monitor als *Fließband* (Pipeline) zu realisieren, wobei die erste Stufe lediglich die Aufgabe übernimmt, die n Werte aus der Tabelle zu kopieren. Die zweite Stufe führt die Berechnungen und Auswertung durch, solange noch nicht alle Aufträge der ersten Stufe bearbeitet wurden. Der Monitor besteht in diesem Fall also aus zwei Threads (siehe Abbildung 4.2). Dieser „Mehraufwand" (insbesondere das Kopieren) ist sinnvoll, um die Wartezeiten, die durch die Synchronisation beim Zugriff auf gemeinsame Daten entstehen können, so kurz wie möglich zu halten und auf diese Weise mehr Nebenläufigkeit durch eine losere Kopplung zu erzeugen. Eine weitere Alternative wäre die Benutzung der Interprozeß-Kommunikation (Inter Process Communication – IPC), die Kanäle, bzw. Tore (Ports) zur Verfügung stellt (s. [Zimm 95]). Hier geschieht das Kopieren der Daten implizit und es wird u.U. ein Wechsel in den Kernadreßraum nötig, da die Kommunikationsbefehle Kernaufrufe darstellen. Der Monitor kann dann wie in Abbildung 4.1 realisiert werden, nur daß statt der Signalisierung nun ein asynchrones *Versenden einer Nachricht* erfolgt. Programmiertechnisch fällt der Aufwand aufgrund der Benutzung des mächtigeren Mechanismus sicher geringer aus, die Effizienz hängt jedoch weitgehend von der angebotenen Implementierung der Kommunikationsmechanismen ab. Der zusätzliche Thread zum Kopieren der Werte wird gegenüber der Fließband-Methode gespart. Welche Implementierung nun in der Praxis schneller ist, läßt sich a priori nicht beurteilen. Der oben dargestellte Ablauf bezieht sich auf nur eine Verbindung, daher sind

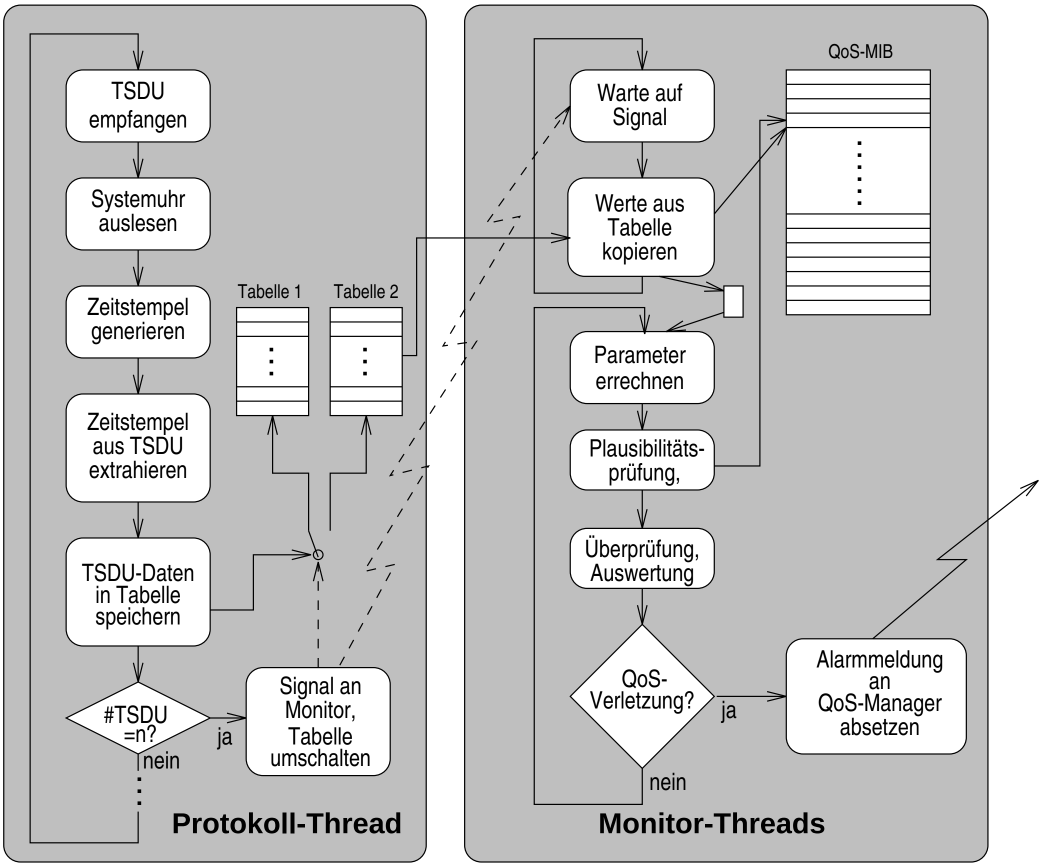


Abbildung 4.2: Monitor als Fließband organisiert

noch folgende Aspekte zu berücksichtigen: Im Normalfall wird jeder Transportprotokollinstanz eine Monitorinstanz zugeordnet, da durch die Realisierung als Threads ein gemeinsamer Adreßraum benötigt wird. Weil jede Transportprotokollinstanz normalerweise mehrere Verbindungen gleichzeitig verwalten kann (Multiplexing), sollte auch die Monitorinstanz mehrere Verbindungen verwalten können. Deshalb muß zusätzlich zur Signalisierung noch der Verbindungskontext (hauptsächlich die zur Verbindung gehörende Dienstgüte-Vorgabe) übermittelt werden, was durch Auslesen einer gemeinsamen Variable geschehen kann. Es ist z.B. denkbar, daß jede Anwendung eine eigene Transportprotokollinstanz in ihrem Adreßraum besitzt und mehrere Verbindungen gleichzeitig verwaltet. Bei Geschwindigkeitsengpässen kann man auch mit mehr als zwei Puffern *pro Verbindung* arbeiten (dadurch werden die oben erwähnten Wartezeiten vermieden). Problematisch wird es, wenn der Monitor mit der Auswertung nicht schnell genug ist und somit die nötige Synchronisation zwischen Protokoll und Monitor verloren geht. Man kann zwar den Monitor so konstruieren, daß keine Meßdaten verloren gehen, aber dann wächst die Menge der zu zwischenspeichernden Daten unaufhörlich und die Auswertung dieser Daten wird wahrscheinlich überflüssig, da sie „zu alt“ sind. Daher muß in irgendeiner Form ein Mechanismus vorgesehen werden, um die Synchronisation zwischen Dienstgütemonitor und Protokoll wiederherzustellen (zum Beispiel durch Überschreiben von Daten in einem Ringpuffer o.ä.).

4.2 Konkrete Implementierung

Die konkrete Implementierung soll auf einer DEC Alpha-Workstation unter dem Betriebssystem OSF/1 Version 3.0 durchgeführt werden. OSF/1 benutzt *Mach* als Betriebssystemkern und stellt Threads (nach POSIX 1003.4a[5]) zur Verfügung. Zur leichteren Portierung sollen so weit wie möglich POSIX-Funktionen benutzt werden (s. [Baum 95]).

4.2.1 OSF/1

OSF/1 stellt u.a. folgende Mechanismen für Threads zur Verfügung:

- Mutexes
  Mutexes (Mutex leitet sich von **Mut**ual **Ex**clusion=gegenseitiger Ausschluß ab) sind Objekte, die den Zugriff zwischen mehreren Threads auf gemeinsame Daten regeln. Jedem Datenobjekt, daß theoretisch von mehreren Threads gleichzeitig manipuliert oder gelesen werden kann, sollte also ein Mutex zugeordnet werden. Damit stellen Mutexes eine Realisierung der „klassischen“ *Semaphoren* oder *Sperren* (nach [Wett 93]) dar. Man „klammert“ einen „kritischen Abschnitt“, in dem auf das gemeinsame Datenobjekt zugegriffen wird, zwischen den Befehlen „Sperren“ (`pthread_mutex_lock()`) und „Entsperren“

[5] **P**ortable **O**perating **S**ystem **I**nterface (based on Uni**x**), der Zusatz Unix in Klammern ist allerdings obsolet.

(`pthread_mutex_unlock()`) ein. Somit wird sichergestellt, daß zu jedem Zeitpunkt nur *ein* Prozeß auf die gemeinsamen Daten zugreift (andere Prozesse, die auch zugreifen wollen, werden in den Wartezustand versetzt und beim Entsperren fortgesetzt). Es gibt drei verschiedene Arten von Mutexes:

  - fast mutex
    eine binäre Sperre, d.h. wenn derselbe Prozeß nach einem Sperren nochmals sperrt, ohne vorher zu Entsperren, verklemmt sich der Prozeß.
  - recursive mutex
    eine Zählsperre, d.h. ein Prozeß kann auch selbst mehrfach hintereinander sperren. Die Realisierung ist allerdings langsamer als bei der *fast*-Variante.
  - nonrecursive mutex
    eine binäre Sperre, die keine Verklemmung bei nochmaligem Sperren produziert, sondern eine Fehlermeldung. Bei der Programmentwicklung zur Fehlersuche gut einzusetzen und später durch *fast mutexes* zu ersetzen, jedoch langsamer als diese.

- Signalisierung mit Condition Variables
  Zu jeder Condition Variable gehört ein Mutex und ein Prädikat. Ein asynchrones Signalisieren und synchrones, sowie versuchendes Empfangen, und ein Signalisierungsrundruf (signal broadcast) werden zur Verfügung gestellt. Durch Condition Variables kann ein Prozeß *warten*, bis gemeinsame Daten einen bestimmten Zustand erreicht haben, in welchem das Prädikat den Wert „wahr“ annimmt. Nach der Warteanweisung muß der Wert des Prädikats erneut abgefragt werden, um sicherzugehen, daß fortgesetzt werden kann. Es handelt sich bei diesem Mechanismus also um ein „grobkörnigere“ Synchronisierung, als bei den Mutexes.

Diese Synchronisationsmechanismen sollten nun so eingesetzt werden, daß ein möglichst großer Grad an Nebenläufigkeit erzielt wird, trotz Operationen auf gemeinsamen Daten. Insbesondere das Protokoll darf nicht durch den Monitor aufgehalten werden. Daher müssen Daten kopiert werden (allerdings innerhalb eines Adreßraums), um mehr Nebenläufigkeit zu erzielen. Hier ist fein abzuwägen, ob ein Kopieren der Daten lohnt, weil das Bearbeiten der Daten u.U. länger dauert (und so ein Thread unnötigerweise warten muß), oder aber das Kopieren der Daten länger als deren Auswertung dauert. Durch die Anwendung der Fließband-Architektur aus Abbildung 4.2 soll eine möglichst große Parallelität erzielt werden.

4.2.2 Verwendete Objekte

Hier sollen kurz die bei der Implementierung verwendeten Datenstrukturen und Operationen erläutert werden.

Kommunikationsobjekte

Die Kommunikation zwischen den drei Threads wurde durch zwei elementare Warteschlangen (FIFO-Queues) realisiert. Die Warteschlange arbeitet intern mit Mutexes und Condition Variables zur Signalisierung. Elemente der Warteschlange sind Befehle mit Argumenten, wie etwa die Mitteilung eines Verbindungsaufbaus oder das Kopieren, bzw. Auswerten einer Tabelle. Die Implementierung vermeidet häufiges Aufrufen einer Speicheranforderung (`malloc()`) und Messungen ergaben, daß die Kommunikation über Ports einiges langsamer ist (Faktor 5–10).

QoS-Parameterobjekte

Es sind zwei Objekte zu entwerfen: einmal die *QoS-Spezifikation*, d.h. eine Spezifikationsmöglichkeit für QoS-Parameter, sowie ein Objekt, um die *QoS-Meßdaten* abzulegen. Da ein QoS-Manager für PATROCLOS noch nicht existiert, sowie PATROCLOS selbst keine quantitativen Dienstqualitätsparameter vorsieht, mußten diese Objekte neu entworfen werden.

Zuerst die QoS-Spezifikation (in C-Notation): für jeden Parameter, der zu einer der drei Klassen *statistisch, deterministisch, bestmöglich* (siehe Abschnitt 2.2.2, S. 20) gehört, gibt es die gleiche Datenstruktur. Sie enthält minimale und maximale Werte, einen oberen und unteren Grenzwert, sowie einen mittleren Wert, zu dem ein Intervall angegeben wird, auf das er sich bezieht. Außerdem kann für statistische Parameter der Anteil `prob_a` der SDUs, die die minimalen und maximalen Grenzwerte unter- bzw. überschreiten dürfen, unter einer Anzahl `prob_b` unmittelbar aufeinanderfolgender SDUs, angegeben werden. Damit ist sehr genau der Prozentsatz `prob_a/prob_b` der SDUs festgelegt, die die QoS-Vorgabe verletzen dürfen und stellt damit eine Realisierung der „gebrochenen Grenze“ dar (Abschnitt 2.2.2, S. 21). Als Überprüfungsmethode wird das „gleitende Fenster“ verwendet.

```
#define qos_gclass_t(x_type,x_name)
 struct {
  x_type min,        /* minimum value */
         max,        /* maximum value */
         average,    /* average according to the interval avg_int */
         thresh_min, /* threshold minimum */
         thresh_max; /* threshold maximum */
  int    average_interval;/* interval in number of SDUs, */
                     /* -1 means infinity */
  int    prob_a,     /* Nr of SDUs that could violate QoS */
         prob_b;     /* Nr of immediately following SDUs */
  commitmentclass_t
```

```
         commitment; /* which commitment for this parameter? */
 } x_name

typedef struct qos_eclass_struct {
  int max,       /* how many errors are allowed in the interval? */
      interval;  /* counting interval for errors, -1 == infinity */
  commitmentclass_t commitment;
} qos_eclass_t;

typedef enum {best_effort,
              statistical,
              deterministic} commitmentclass_t;
```

Für die deterministische Klasse sind die Minimal- und Maximalwerte strikt einzuhalten. Als Parameter wurden zunächst die folgenden spezifiziert (jeweils minimale, maximale, Durchschnitts- und Grenzwerte wie oben): SDU-Rate, SDU-Größe in Bytes, Durchsatz in Bytes/s, Ende-zu-Ende Verzögerung in Sekunden, Jitter in Sekunden, die Anzahlen der verlorengegangenen, duplizierten und verfälschten SDUs. Weitere Parameter wie z.B. Kosten, Schutz, Priorität, Neuverhandlungsoptionen wurden noch nicht spezifiziert, da sie für das Monitoring nicht benötigt werden. Die Angabe der Ende-zu-Ende Verzögerung und des Jitters in Sekunden erfordert eine Gleitkommadarstellung und man könnte aus Geschwindigkeitsgründen auch eine ganzzahlige Angabe in z.B. Nanosekunden wählen, darauf wurde jedoch bisher verzichtet.

```
/** QoS-specification for a simplex connection **/
typedef struct qos_spec_struct {
 qos_gclass_t(rate_t,rate);      /* min,max and average SDU rate */
 qos_gclass_t(framesize_t,framesize);  /* "     " framesize      */
 qos_gclass_t(throughput_t,throughput);/* "     " throughput     */
 qos_gclass_t(delay_t,delay);     /*  "   " end-to-end delay in s */
 qos_gclass_t(jitter_t,jitter);  /*  "   " jitter in s           */
 qos_eclass_t err_lost;     /*  "   " lost SDUs             */
 qos_eclass_t err_dupl;     /*  "   " duplicated SDUs       */
 qos_eclass_t err_chksum;   /*  "   " biterror rate in SDUs */
} qos_spec_t;
```

Die QoS-Meßdaten für eine Verbindung werden pro Parameter in ähnlicher Weise abgelegt: minimale, maximale und aktuelle Meßwerte, sowie ein arithmetisches Mittel über einem angegebenem Intervall. Die augenblickliche Intervallgröße mitzuführen ist notwendig, damit entschieden werden kann, ob ein Vergleich mit der Parameter-Vorgabe stattfinden darf. Man denke etwa an ungültige Parameterwerte, die durch falsche Zeitstempel entstehen können. Diese Werte dürfen nicht in die Auswertung mit einbezogen werden und deshalb kann die augenblickliche Intervallgröße sinken.

```
#define measured_class_t(x_type,x_name)
 struct {
  x_type min,      /* minimum value                         */
         max,      /* maximum value                         */
         average,  /* average according to                 */
                   /* an interval (see QoS-spec)            */
         last,     /* last measured value                   */
         sum;      /* cumulated value over                  */
                   /* last sum_int used for avg             */
  int    sum_int;  /* actual interval for sum, average      */
  int    violated; /* number of SDUs that violated QoS-spec */
  int    violated_int; /* intervall of SDUs over            */
                       /* which violated SDUs are counted   */
 } x_name

typedef struct measured_error_struct {
 int sum,
     interval;
} measured_error_t;
```

Die gemessenen QoS-Parameter sind im einzelnen: SDU-Rate, SDU-Größe, Durchsatz, Ende-zu-Ende Verzögerung, Jitter, verlorene, duplizierte und verfälschte SDUs. Die SDU-Fehler sind eine besondere Klasse: mit der Zeit kann die Anzahl an Fehlern nur zunehmen, es macht daher keinen Sinn minimale, maximale oder Durchschnittswerte zu errechnen.

```
/** These qos-parameters are per simplex connection   **/
/** delay, jitter and error_rate can only be measured **/
/** for incoming data streams.                        **/
/** NOTE: The initialization of the min and max values
    are dependent on the type **/
typedef struct qos_table_struct {
 measured_class_t(rate_t,rate);  /*  SDUs/s   */
 measured_class_t(framesize_t,framesize);/*  framesize in bytes */
 measured_class_t(throughput_t,throughput);/*throughput in Bytes/s */
 measured_class_t(delay_t,delay);  /* end-to-end delay in s */
 measured_class_t(jitter_t,jitter);/* jitter in s          */
 measured_error_t err_lost;        /* # of lost SDUs       */
 measured_error_t err_dupl;        /* # of duplicated SDUs  */
```

```
 measured_error_t err_chksum;      /* # biterrors in SDUs   */
} qos_table_t;
```

Ablauf

Zuerst werden für jede SDU die SDU-Nummer, Zeitstempel, SDU-Größe und Fehlertyp im Protokoll registriert. Diese Werte werden an den Monitor übergeben, der mittels eines Datenreduktionsverfahrens (Abschnitt 1.1.1, S. 3) daraus die Parameter Durchsatz, Delay und Jitter pro SDU errechnet. Diese Werte werden dann im Auswertungsteil entsprechend ihrer Verpflichtungsklasse gegen ihre Grenzen geprüft und die QoS-Parameter für die Verbindung aktualisiert (s. `qos_table_t`).

4.2.3 PATROCLOS

Bei dem Transportsubsystem PATROCLOS (**Pa**rallel **Tr**ansp**o**rt Subsystem for **C**e**l**l Based High Speed Netw**o**rk**s**) handelt es sich um eine parallele, modulare Protokollarchitektur, die speziell im Hinblick auf die Eigenschaften von Hochgeschwindigkeitsnetzen entworfen wurde (s. [Brau 93]). Wie in Abbildung 4.3 dargestellt, besteht es aus mehreren nebenläufigen Protokollautomaten, die jeweils unterschiedliche Protokollfunktionen übernehmen. Der Begriff Transport*sub*system erklärt sich dadurch, daß PATROCLOS eine mit den OSI-Schichten 2b-4 vergleichbare Funktionalität besitzt. Es bildet daher mit einem entsprechenden Medium (Schichten 1-2a) ein vollständiges Transportsystem.

Notwendige Änderungen

Zunächst mußte PATROCLOS um Zeitstempel erweitert werden. Dabei war die Entscheidung zu treffen, ob die Zeitstempel in jeder TPDU oder nur einmal pro TSDU mitgeschickt werden (s. Abschnitt 4.1.1). Da PATROCLOS die Möglichkeit bietet, TSDUs unvollständig auszuliefern, wurde entschieden, die Zeitstempel in den Protokollkopf des Benutzerdatenrahmens ([Brau 93], S. 89) und damit in jede Daten-TPDU zu integrieren. Das noch unbenutzte Feld *reserved* der Länge 4 Bytes reichte für das in Abschnitt 4.1.1 gewählte Format nicht aus. Daher wurde der Zeitstempel noch hinter dieses *reserved*-Feld gesetzt:

```
typedef struct dataframe_struct {
 seq        sequence;    /* Sequenznummer der TPDU                 */
 dword      length;      /* Laenge Benutzerdaten dieses Pakets     */
 seq        TSDU_seq;    /* Sequenznummer der TSDU                 */
 dword      total_length;/* Laenge der TSDU vor dem Segmentieren */
```

```
 dword      reserved;    /* noch nicht genutzt                    */
 timestamp_ext_t
            utc_timestamp;/* UTC-Zeitstempel 2 x 4 byte(sec,nsec)*/
} Data_frame;
```

Damit ging leider die besondere Eigenschaft verloren, daß der vollständige P-Rahmenkopf in die 44 Byte Nutzlast einer ATM-Zelle paßt.

Eigentlich sollten alle weiteren Änderungen nur noch im Transportdienstschnittstellen-Automaten (TI) (s. Abb. 4.3) erfolgen. Beim Verbindungsauf- und -abbau wird der QoS-Monitor benachrichtigt und beim Verbindungsaufbau zusätzlich ein initialer Zeitstempel *lokal* dem QoS-Monitor übermittelt, der für die Durchsatzberechnung der ersten TSDU benötigt wird. Durch die Struktur von PATROCLOS wurde es nötig den Zeitstempel auch an den Segmentierautomat (SSM) weiterzureichen, der intern eine TSDU in mehrere TPDUs zerlegen kann. Danach werden die TPDUs an den Daten-Sender-Automat (DS) übergeben, wo die konkreten Protokollköpfe erzeugt werden. Dort wird also der Zeitstempel in den Benutzerdatenrahmen eingetragen. Beim Empfangen werden die Automaten in umgekehrter Reihenfolge durchlaufen: vom Daten-Empfänger-Automaten (DR) zum Reassemblierautomaten (RSM) und von dort wieder zum TI. Da im Verbindungsverwaltungsautomaten (CM) fast alle übrigen Automaten als Threads gestartet werden, wurden auch die beiden Threads des QoS-Monitors dort gestartet. Die Kommunikation zwischen TI und QoS-Monitor erfolgt über die in Abschnitt 4.2.2 beschriebene FIFO-Warteschlange. Damit wird eine von der übrigen Implementierung abweichende Kommunikationsform benutzt (Mach-Ports s. [Zimm 95]). Der Kopier-Thread im QoS-Monitor fordert für die neu zu kopierende Tabelle Speicher an und übergibt sie dem Auswertungs-Thread zum Lesen. Dadurch gehen keine Meßwerte verloren und der Kopier-Thread muß nicht auf dem Auswertungs-Thread warten, es kann auf eine weitere Synchronisation verzichtet werden (sonst könnte man auch den Kopier-Thread vollständig einsparen). Normalerweise sind Anforderungen des Systemspeichers (durch `malloc()`) recht zeitaufwendige Operationen, die bei Optimierung durch eine geeignete eigene Speicherverwaltung ersetzt werden sollten (z.B. den Vorteil nutzend, daß ein Thread immer Speicher anfordert, der andere ihn jeweils wieder freigibt). Wie in Abschnitt 4.1.3 erwähnt, darf der Auswertungsteil nicht zu sehr dem Protokoll hinterher laufen. Abzulesen ist dies, wenn die Warteschlangenlänge zwischen Kopier-Thread und Auswertungs-Thread eine gewisse Größe überschreitet. Daraufhin sollte der Kopier-Thread die Warteschlange durchlaufen und alle Anweisungen zum Auswerten der Tabelle in solche umschreiben, die den Auswertungs-Thread veranlassen, die Tabelle auszulassen (es befinden sich möglicherweise noch Nachrichten für Verbindungsauf- oder -abbau in der Warteschlange). Dieser Mechanismus wurde noch nicht implementiert.

Da die Prüfsummen-Berechnung in PATROCLOS noch nicht implementiert war, konnten keine Übertragungsfehler in PDUs und damit in SDUs festgestellt werden.

Fehler des Transportprotokolls werden nicht angezeigt und duplizierte SDUs werden vom Monitor ebenfalls nicht registriert.

4.3 Mögliche Erweiterungen

- PATROCLOS stellt noch keine Mechanismen zur Verfügung, um mit den Dienstnutzern über die Dienstgüte zu verhandeln. Vorhandene Dienstgüteparameter bezogen sich lediglich auf funktionale bzw. qualitative Parameter und dienen dazu, verschiedene Protokollfunktionen zu selektieren. Die Überprüfung auf Einhaltung der Dienstgüte-Vorgaben ist zwar im Monitor implementiert, kann aber noch nicht für Korrekturmaßnahmen verwendet werden. Ebenso fehlt die übergeordnete Instanz, die die Meldungen des QoS-Monitors entgegennimmt und auswertet (in Abschnitt 2.2.3 als „QoS-Manager" bezeichnet).

- Der QoS-Monitor besitzt noch keine Mechanismen zur Regelung der Berichterstattung (Report-Policy) oder zur *Anpassung* an neue Dienstgüte-Vorgaben.

- Die *SNMP-Schnittstelle*, die die vom QoS-Monitor gemessenen Daten in einer MIB zur Verfügung stellt, wurde noch nicht implementiert.

- Der Monitor kann leicht um eine *graphische Anzeige der Meßwerte* erweitert werden.

- Als Testumgebung stand leider keine Anwendung mit hohen Dienstqualitäts-Anforderungen (z.B. Multimedia-Anwendung) zur Verfügung, sondern lediglich eine Testbenutzerumgebung, die manuelle Eingaben entgegennimmt.

- Die *momentane Burstgröße* (Anzahl der unmittelbar hintereinander zur Spitzenrate gesendeten Dateneinheiten) wird nicht ermittelt.

- Die Minima und Maxima werden momentan noch über die gesamte Verbindungsdauer mitgeführt anstatt nur über das Mittelungsintervall `average_int`.

- Es werden alle Parameter gemessen, ein richtiges „Ausschalten" einzelner Parameter wurde nicht realisiert. Die Auswertung der Parameter kann momentan nur durch Zuordnung zu einer `best_effort`-Klasse unterbunden werden.

- Das Registrieren von Fehlern in SDUs (Prüfsumme, Duplikate, angezeigte Fehler) muß noch genauer untersucht werden. Auch hier lassen sich maximale und Durchschnittswerte ermitteln, falls ein endliches Intervall spezifiziert wurde, da dann die Fehleranzahlen auch wieder abnehmen können.

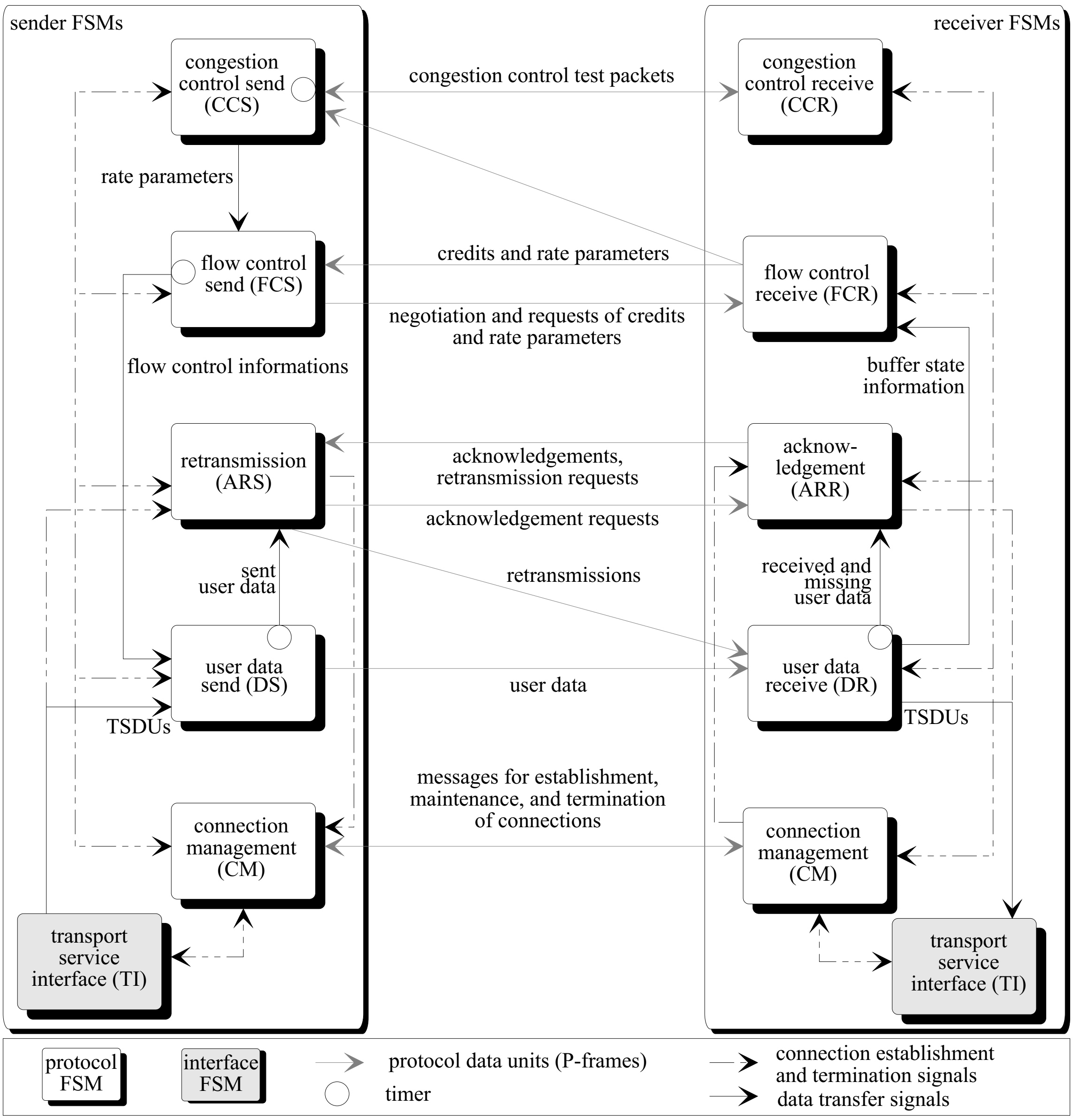


Abbildung 4.3: PATROCLOS Architektur

5. Zusammenfassung und Ausblick

Die Aufgabe bestand darin, einen Dienstgüte-Monitor für ein Transportsystem zu entwerfen und in ein bestehendes Transportsubsystem zu integrieren. Die Untersuchung der Anforderungen an einen Dienstgüte-Monitor ergab, daß der Monitor das Kommunikationssystem so wenig wie möglich belasten darf und folgende Dienstgüte-Parameter mißt: Durchsatz, Ende-zu-Ende Verzögerung, Verzögerungsschwankung, Datenfehler. Die Parameter wurden genau definiert und es wurde beschrieben, *wie* sie zu Messen sind. Ferner wurden die drei Verpflichtungsklassen *deterministisch*, *statistisch*, *bestmöglich* definiert. *Jeder Parameter* wird genau einer der drei Klassen zugeordnet. Der Monitor mißt Dienstqualitäts-Parameter an der Transportdienstschnittstelle und betrachtet TSDUs als zu messende Objekte. Eine zum Transport-Protokoll *nebenläufige, eigenständige Monitor-Instanz* soll höherer Leistung und geringer Beeinflussung des Protokolls Rechnung tragen. Um die Systembelastung gering zu halten, wurde ein *ereignisorientiertes* Meßverfahren gewählt, bei dem die Ankunft einer Dateneinheit (hier TSDU) als Ereignis gewertet und Zwischenankunftszeiten der Dateneinheiten gemessen werden. Für die Durchsatzberechnung wurde die unzureichende *Uhrauflösung* und der dadurch eingeführte *Meßfehler* berücksichtigt. Bei hohen Datenraten werden deshalb Meßdaten von mehreren Dateneinheiten gesammelt, bevor sie vom Monitor ausgewertet werden können. Ist die Zwischenankunftszeit jedoch für die Meßgenauigkeit groß genug, so können auch Meßdaten einer einzelnen Dateneinheit sofort ausgewertet werden. Somit reagiert der Monitor last-adaptiv und schnellstmöglich, Unterbrechungsauslösung und -behandlung werden komplett vermieden und der Kommunikationsaufwand zwischen Protokoll und Monitor minimiert. Um die Ende-zu-Ende Verzögerung zu messen, wurden *synchronisierte Uhren* bei Sender und Empfänger vorausgesetzt und die Dateneinheiten mit *Zeitstempeln* versehen, die mit übertragen werden, da von Simplex-Datenströmen ausgegangen wurde und in diesem Fall kein Rückkanal zur Verfügung steht.

Eine Integration in bzw. eine Realisierung durch das *Netzwerkmanagement* wurde untersucht. Die derzeit verfügbaren Mechanismen erwiesen sich jedoch als noch nicht mächtig genug für eine Realisierung des Monitors als Netzwerkmanagement-Agenten. Die Meßdaten des Dienstgüte-Monitors könnten aber problemlos über das Netzwerkmanagement verfügbar gemacht werden.

Schließlich wurden bei der konkreten Implementierung zunächst plattformunabhängige Aspekte untersucht, eine „generische Implementierung“ unter OSF/1 erstellt und mit dem Transportsubsystem PATROCLOS verbunden.

Da der Dienstgüte-Monitor nur eine Komponente einer ganzen Dienstqualitäts-Architektur darstellt, könnten weitere Arbeiten das nötige Rahmenwerk, sowie die Komponenten für diese Architektur schaffen und das Zusammenspiel der Komponenten realisieren, um schließlich das Ziel von flexiblen Diensten mit *garantierter Dienstqualität* zu realisieren.

Literaturverzeichnis

[Baum 95] Baumgärtel, Hans Georg
„Wie geschmiert – POSIX.1: Programmierschnittstelle für offene Systeme“
iX Januar 1995, S.140-145, Heise Verlag Hannover

[Blac 92] Black, Uyless
„Network Management Standards“
McGraw-Hill Inc., 1992

[Brau 93] Braun, Torsten
„Ein paralleles Transportsubsystem für zellenbasierte Hochgeschwindigkeitsnetze“
Dissertation, Fakultät für Informatik, Universität Karlsruhe
VDI Fortschritt-Berichte, Reihe 10, Nr. 264, VDI-Verlag Düsseldorf 1993

[CCGH 93] Campbell, Andrew; Coulson, Geoff; García, Francisco; Hutchison, David; Leopold, Helmut
„Integrated Quality of Service for Multimedia Commmunications“
Proc. IEEE INFOCOM'93, S.732-739, San Francisco, USA, April 1993
Internal Report No. MPG-93-17, Departement of Computing, Lancaster University, Lancaster LA1 4YR
ftp: ftp.lancs.ac.uk /pub/mpg

[CaCH 93] Campbell, Andrew; Coulson, Geoff; Hutchison, David
„A Multimedia enhanced Transport Service in a Quality of Service Architecture“
Internal Report No. MPG-93-22, Departement of Computing, Lancaster University, Lancaster LA1 4YR, Oktober 1993

[CaCH 94] Campbell, Andrew; Coulson, Geoff; Hutchison, David
„A Quality of Service Architecture“
Internal Report No. MPG-94-08, Departement of Computing, Lancaster University, Lancaster LA1 4YR, April 1994

[CaSa 92] Cavanaugh, John David; Salo, Timothy J.
„Internetworking with ATM WANs“

Minnesota Supercomputer Center, Inc.
14. Dezember 1992

[Ferr 78] Ferrari, Domenico
„Computer Systems Performance Evaluation"
Prentice Hall Inc., Englewood Cliffs, New Jersey 07632, 1978

[FeSZ 83] Ferrari, Domenico; Serazzi, Guiseppe; Zeigner, Alessandro
„Measurement and Tuning of Computer Systems"
Prentice Hall Inc., Englewood Cliffs, New Jersey 07632, 1983

[Ferr 90] Ferrari, Domenico
„Client Requirements for Real-Time Communication Services"
IEEE Communications Magazine, Vol. 28, Nr. 11, November 1990, S. 65-72

[HeGr 94] Hein, Mathias; Griffiths, David
„SNMP - Simple Network Management Protocol Version 2"
International Thomson Publishing GmbH, 1994

[ISO 8825] ISO/IEC
„Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)"
Information technology - Open Systems Interconnection
International Standard, Reference number ISO/IEC: 1990 (E)

[ISO 93] ISO/IEC JTC1/SC6/WG4
„From Best Effort to Enhanced QoS"
Telecommunications and information exchange between systems – Transport Layer
N 827, IBN - Belgian National Body, 14.7.1993

[ISO-QoS-WD 94] ISO/IEC JTC/SC21
„Quality of Service – Basic Framework — Working Draft #4"
Project JTC1.21.57.2/ITU-T Q2/7;
Open Systems Interconnection, data management and Open Distributed Processing
Meeting of ISO/IEC JTC1/SC21, Southampon, Juli 1994

[Jens 94] Jensen, Peter-Nikolaus
„Flexible Anbindung von Ressourcen an einen Managementagenten"
Diplomarbeit am Institut für Telematik, Fakultät Informatik, Universität Karlsruhe (TH), August 1994

[JuSP 92] Jung, Jae-il; Seret, Dominique; Picard, Robert
„Quality of Service in B-ISDN and Relation with Network-Management"
Networks Departement, ENST Paris, 46 rue Barrault, 75634 Paris Cedex 13,

Frankreich
IEEE ICC'92, Chicago, Juni 1992

[KaMR 93] Kanakia, Hemant; Mishra, Partho P.; Reibman, Amy
„An Adaptive Congestion Control Scheme for Real-Time Packet Video Transport"
SIGCOMM'93, Ithaca, N.Y., USA, September 93

[LoKK 93] Lockemann, Peter; Krüger, Gerhard; Krumm, Heiko
„Telekommunikation und Datenhaltung"
Carl Hanser Verlag, 1993

[OiTo 93] Oikawa, Shuichi.; Tokuda, H.
„User-Level Real-Time Threads: An Approach Towards High Performance Multimedia Threads"
Proc. Fourth International Workshop on Network on Operating System Support for Digital Audio and Video (NOSSDAV'93)
Lancaster University, Lancaster LA1 4YR, UK, November 1993
Lecture Notes in Computer Science 846, Springer-Verlag 1994, S.66-76

[RFC 1305] Mills, David L.
„Network Time Protocol (Version3)
Specification, Implementation and Analysis"
Network Working Group, David L. Mills, University of Delaware
Request For Comments 1305, IETF, per ftp auf RFC-Servern verfügbar

[RCCB 94] Robin, Philippe; Coulson, Geoff; Campbell, Andrew; Blair, Gordon; Papathomas, Michael
„Implementing a QoS Controlled ATM Based Communications System in Chorus"
Internal Report No. MPG-94-05, Departement of Computing, Lancaster University, Lancaster LA1 4YR, März 1994

[Tane 94] Tanenbaum, Andrew S.
„Distributed Operating Systems"
Prentice Hall, 1994-95

[ToKi 93] Tokuda, Hideyuki; Kitayama, Takuro
„Dynamic QOS Control Based on Real-Time Threads"
Proc. Fourth International Workshop on Network on Operating System Support for Digital Audio and Video (NOSSDAV'93)
Lancaster University, Lancaster LA1 4YR, UK, November 1993
Lecture Notes in Computer Science 846, Springer-Verlag 1994, S.114-123

[WaCr 93] Wang, Zheng; Crowcroft, Jon
„Analysis of Burstiness and Jitter in Real-Time Communication"

Departement of Computer Science, University College London,
London WC1 6BT, U.K.
SIGCOMM'93, Ithaca, N.Y., USA, September 93

[Wett 93] Wettstein, Horst
„Systemarchitektur“
Hanser Verlag, 1993

[ZhFe 93] Zhang, Hui; Ferrari, Domenico
„Rate-Controlled Static-Priority Queueing“
Proceedings INFOCOM'93, San Francisco, März 1993
Computer Science Division, Univ. of California at Berkeley,
Berkeley CA 94720, ftp: tenet.berkeley.edu

[Zimm 95] Zimmermann, Peter
„Implementierung des Transportsystems PATROCLOS unter OSF/1“
Diplomarbeit am Institut für Telematik, Fakultät Informatik, Universität Karlsruhe (TH), März 1995

[Zitt 91] Zitterbart, Martina
„High Speed Transport Components“
IEEE Network, Vol. 5, No. 1, Januar 1991, S.54-63

[Zitt 95] Zitterbart, Martina
„Hochleistungskommunikation“
Band 1: Technologie und Netze
Oldenbourg Verlag, München/Wien, 1995

Index

Agent, 39
Agenten Prozeß, 39
ASN.1, 42
ATM, 7
Auflösung, 4
Auswertung, 30

BER, 42, 49
bestmöglich, 22
Betriebsmittel, 3
Burst, 15

CMIP, 39

Dienstgüte, 1
Dienstqualität, 1
Dienstqualitätsanforderung, 7
Double-Buffering, 51
Duplex-Verbindung, 13
Durchsatz
  Definition, 15
  Messung, 28

Ende-zu-Ende Verzögerung, 16
Ereignis, 2
externes Ereignis, 5

Fehler, 19
Fenster
  gleitendes, 21
  springendes, 21

garantiert, 20
gebrochene Grenze, 21
Genauigkeit, 4, 48

Hardware-Ereignis, 3
Header, 48

Implementierung
  generisch, 47
  konkret, 54
internes Ereignis, 5

Jitter, 16

Leistung, 2

Mach, 54
Managed Object, 39
Meßintervall, 26
Messung
  der Fehler, 34
  der Verzögerung, 32
  der Verzögerungsschwankung, 33
  des Durchsatzes, 31
  über dynamischem Intervall, 26
  über statischem Intervall, 26
Meßwerkzeuge, 2
MIB, 39
MO, 39
Monitor, 4
Multicasting, 34

Netzwerkmanagement, 39
NM, 39

OSF/1, 54
Overhead, 50

PATROCLOS, 59
PCI, 48
Policing, 8
Polling, 40

QoS, 1
QoS-Anforderung, 7, 12

QoS-Monitor, 1
QoS-Vorgabe, 7, 12
Quality of Service, 1

Ressourcen, 3

Schwellwertmechanismus, 30
Sensor, 3
Simplex-Verbindung, 13
SMI, 42
SNMP, 42
Software-Ereignis, 3
statistisch, 13, 20
Stichprobe, 5
Systemressourcen, 3

Thread, 23, 47

UDP, 42
Uhrauflösung, 27
Umformer, 3
UTC, 49

Verzögerung, 16
Verzögerungsschwankung, 16

Zeitstempel, 47
    Datenformat, 49