Rechnernetze II

SoSe 2020

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 29. Mai 2020

Rechnernetze II

SoSe 2020

6 Überlastkontrolle und *Quality of Service*

Inhalt

- Überlastkontrolle
- Überlastvermeidung
  - DECbit, RED, quellenbasierte Überlastvermeidung
- *Quality of Service*
  - Anforderungen, *IntServ*, *DiffServ*

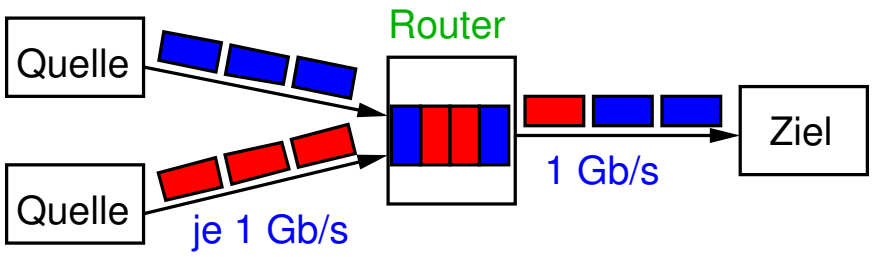
- Peterson, Kap. 6.1-6.5
- Tanenbaum, Kap. 5.3, 5.4

6.1 Überlastkontrolle

Was bedeutet Überlast?

- Pakete konkurrieren um Bandbreite einer Verbindung
- Bei unzureichender Bandbreite:
  - Puffern der Pakete im Router
- Bei Pufferüberlauf:
  - Pakete verwerfen
- Ein Netzwerk mit häufigem Pufferüberlauf heißt **überlastet** (*congested*)

Beispiel einer Überlastsituation



- Sender können das Problem nicht direkt erkennen
- Adaptives Routing löst das Problem nicht, trotz schlechter Link-Metrik für überlastete Leitung
  - verschiebt Problem nur an andere Stelle
  - Umleitung ist nicht immer möglich (evtl. nur ein Weg)
  - im Internet wegen Komplexität derzeit utopisch

Überlastkontrolle

- Erkennen und möglichst schnelles Beenden der Überlast
  - z.B. einige Sender mit hoher Datenrate stoppen
  - in der Regel aber Fairness gewünscht
- Erkennen von drohenden Überlastsituationen und Vermeidung der Überlast (**Überlastvermeidung**, ☞ **6.2**)

Abgrenzung

- **Flußkontrolle** verhindert, daß **ein Sender** seinen **Empfänger** überlastet
- **Überlastkontrolle** (bzw. **-vermeidung**) verhindert, daß **mehrere Sender** einen Teil des **Netzwerks** (= Zwischenknoten) überlasten

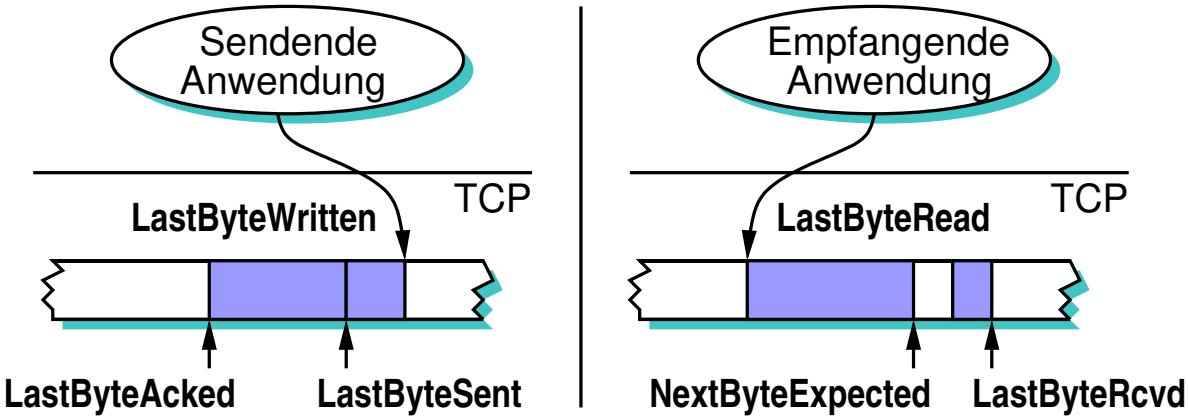
6.1.1 Überlastkontrolle in TCP

- Einführung Ende der 1980'er Jahre (8 Jahre nach Einführung von TCP) zur Behebung akuter Überlastprobleme
  - Überlast ⇒ Paketverlust ⇒ Neuübertragung ⇒ noch mehr Überlast!
- Idee:
  - jeder Sender bestimmt, für wieviele Pakete (Segmente) Platz im Netzwerk ist
  - wenn Netz „gefüllt" ist:
    - Ankunft eines ACKs ⇒ Senden eines neuen Pakets
    - „selbsttaktend"
- Problem: Bestimmung der (momentanen) Kapazität
  - dauernder Auf- und Abbau anderer Verbindungen

6.1.1 Überlastkontrolle in TCP ...

Erinnerung: Flußkontrolle mit *Sliding-Window*-Algorithmus

- Empfänger sendet in ACKs **AdvertisedWindow** (freier Pufferplatz)
- Sender darf dann noch maximal so viele Bytes senden:
  EffectiveWindow =
  AdvertisedWindow − (**LastByteSent** − **LastByteAcked**)



6.1.1 Überlastkontrolle in TCP ...

Erweiterung des *Sliding-Window*-Algorithmus

- Einführung eines **CongestionWindow**
  - Sender kann noch so viele Bytes senden, ohne Netzwerk zu überlasten
- Neue Berechnung für **EffectiveWindow**
  - **MaxWindow** =
    MIN (CongestionWindow, AdvertisedWindow)
  - **EffectiveWindow** =
    MaxWindow − **(LastByteSent** − **LastByteAcked)**
- Damit: weder Empfänger noch Netzwerk überlastet
- Frage: Bestimmung des **CongestionWindow**?

6.1.1 Überlastkontrolle in TCP ...

Bestimmung des CongestionWindow

- Hosts bestimmen **CongestionWindow** durch Beobachtung des Paketverlusts
- Basismechanismus:
  - *Additive Increase / Multiplicative Decrease*
- Erweiterungen:
  - *Slow Start*
  - *Fast Retransmit / Fast Recovery*

Vorgehensweise

- **CongestionWindow** sollte
  - groß sein ohne / bei wenig Überlast
  - klein sein bei viel Überlast
- Überlast wird erkannt durch Paketverlust
- Bei Empfang eines ACK:
  - **Increment** = **MSS** · **(MSS** / **CongestionWindow)**
  - **CongestionWindow** += **Increment**

  im Mittel: Erhöhung um MSS Bytes pro RTT
  (MSS = *Maximum Segment Size* von TCP)
- Bei Timeout: **CongestionWindow** halbieren
  - höchstens, bis MSS erreicht ist

Anmerkungen zu Folie 199:

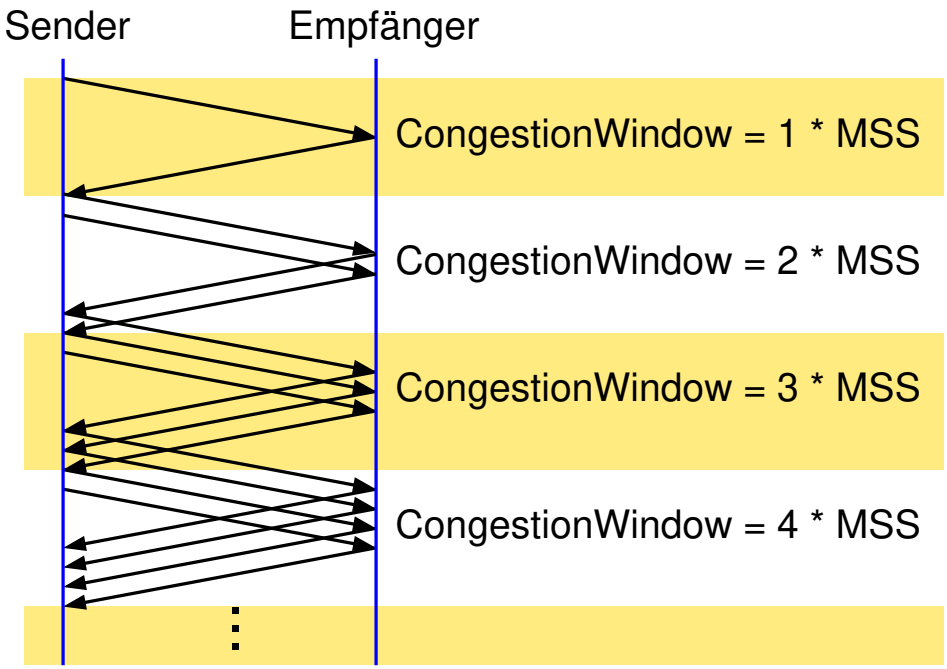
Die Grundidee beim *Additive Increase / Multiplicative Decrease* ist folgende:

- Wenn das Überlastfenster vollständig übertragen wurde, vergrößere das Fenster um ein Paket (additiv)
- Wenn ein Paket verloren geht, halbiere das Fenster (multiplikativ)

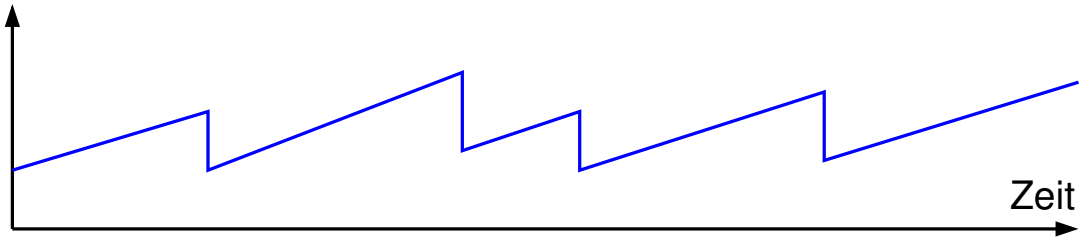
Da in TCP die Fenstergröße aber nicht in Paketen (bzw. Segmenten), sondern in Bytes gemessen wird, wird das **Increment** gemäß folgender Überlegung berechnet:

- Pro RTT wird immer ein **CongestionWindow** an Daten versendet.
- Im Mittel soll daher das **CongestionWindow** um ein (maximal grosses) Segment, also 1 **MSS**, pro RTT erhöht werden.
- Wenn maximal grosse Segmente versendet werden, bestätigt somit jedes ACK den Anteil **MSS** / **CongestionWindow** der Gesamtdatenmenge.
- Also wird das Überlastfenster mit jedem ACK um diesen Anteil der **MSS** erhöht, d.h. um **(MSS** / **CongestionWindow)** · **MSS**.

Additive Increase



Typischer Zeitverlauf des CongestionWindow



- Vorsichtige Erhöhung bei erfolgreicher Übertragung, drastische Reduzierung bei Erkennung einer Überlast
  - ausschlaggebend für Stabilität bei hoher Überlast
- Wichtig: gut angepaßte Timeout-Werte
  - Jacobson/Karels Algorithmus

6.1.3 *Slow Start*

Hintergrund

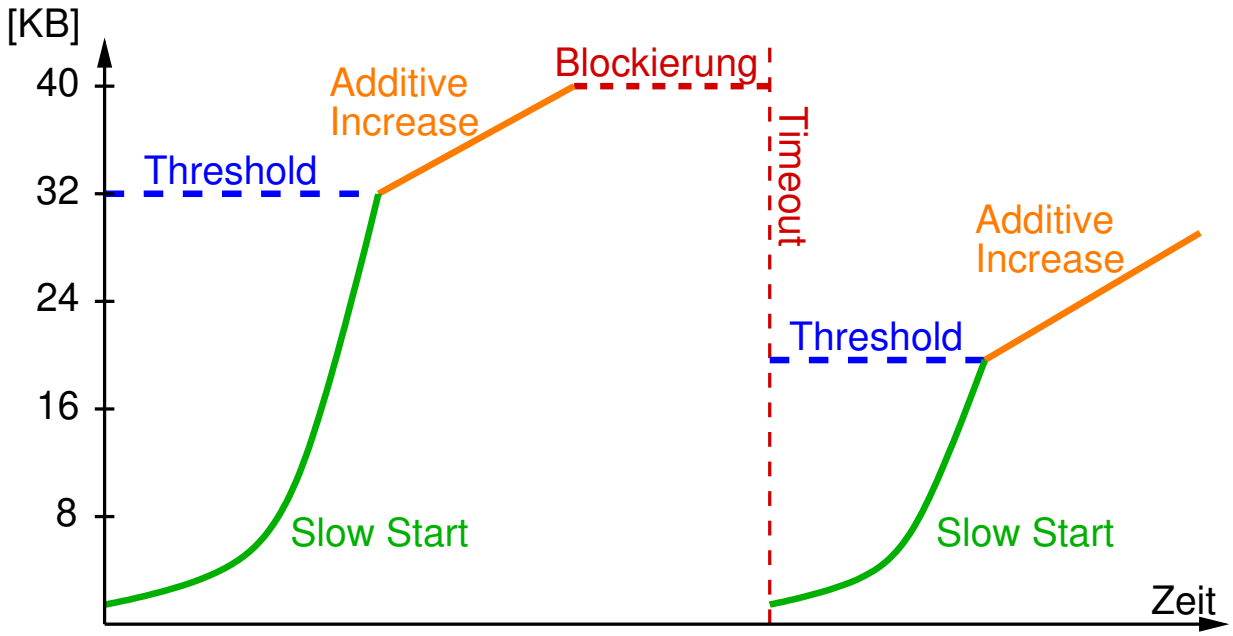
- Verhalten des ürsprünglichen TCP beim Start (bzw. bei Wiederanlauf nach Timeout):
  - sende **AdvertisedWindow** an Daten (ohne auf ACKs zu warten)
  - d.h. Start mit maximalem **CongestionWindow**
- Zu aggressiv, kann zu hoher Überlast führen
- Andererseits: Start mit **CongestionWindow** = MSS und *Additive Increase* dauert zu lange
- Daher Mittelweg:
  - Start mit **CongestionWindow** = MSS
  - Verdopplung bis zum ersten Timeout

6.1.3 *Slow Start* ...

Verhalten bei Timeout

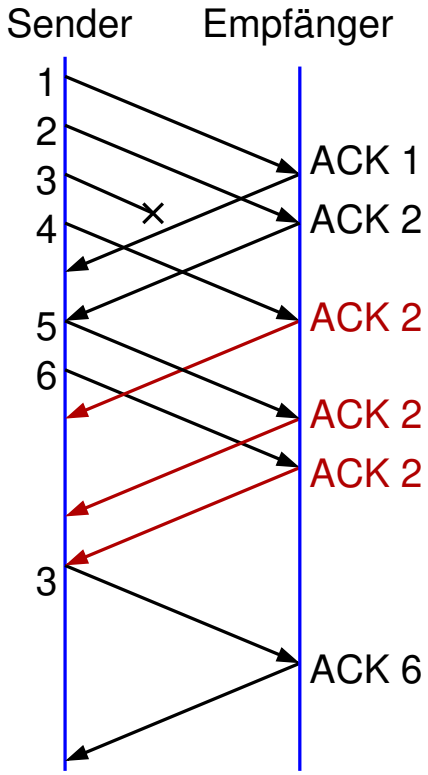
- *Slow Start* wird auch verwendet, wenn eine Verbindung bis zu einem Timeout blockiert:
  - Paket X geht verloren
  - Sendefenster ist ausgeschöpft, keine weiteren Pakete
  - nach Timeout: X wird neu übertragen, ein kumulatives ACK öffnet Sendefenster wieder
- In diesem Fall beim Timeout:
  - **CongestionThreshold** = **CongestionWindow** / 2
  - *Slow Start*, bis **CongestionThreshold** erreicht ist, danach *Additive Increase*

Typischer Verlauf des CongestionWindow



6.1.4 *Fast Retransmit / Fast Recovery*

- Lange Timeouts führen oft zum Blockieren der Verbindung
- Idee: Paketverlust kann auch durch Duplikat-ACKs erkannt werden
- Nach dem dritten Duplikat-ACK:
  - Paket erneut übertragen
  - **CongestionWindow** halbieren ohne *Slow Start*
- *Slow Start* nur noch am Anfang und bei wirklichem Timeout



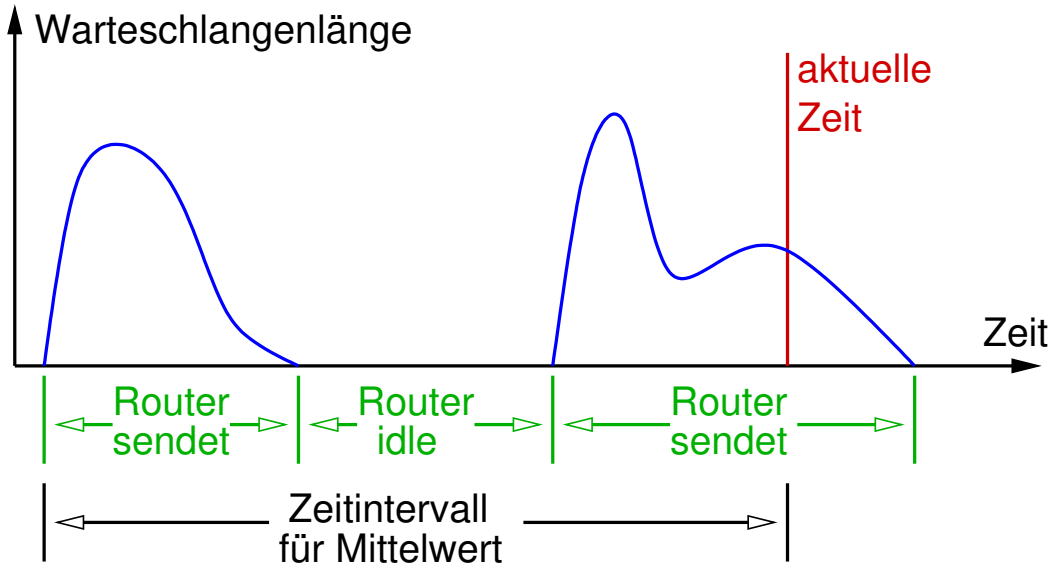
6.2 Überlastvermeidung

- Ziel der **Überlastvermeidung**: reduziere Senderate, **bevor** Überlast (d.h. Paketverlust) entsteht
  - d.h. bei ersten Anzeichen einer drohenden Überlast
- Alternativen zur Erkennung drohender Überlast:
  - Router-zentrisch
    - Router melden an Hosts, wenn sie die Senderate reduzieren sollen
    - Basis: mittlere Länge der Paket-Warteschlange im Router
    - Beispiele: DECbit, RED
  - Host-zentrisch (quellenbasierte Überlastvermeidung)
    - Hosts beobachten Anzeichen drohender Überlast selbst
    - z.B.: steigende Latenz, sinkender Durchsatz
    - Beispiel: TCP Vegas

6.2.1 DECbit

- Verwendet in DECs *Digital Network Architecture*
  - verbindungsloses Netz mit verbindungsorientiertem Transportprotokoll (analog zu TCP über IP)
- Router überwacht mittlere Länge der Warteschlange (Puffer)
- Bei Überschreiten eines Grenzwerts:
  - Router setzt ein Überlastbit im Paketheader
  - Empfänger kopiert dieses Überlastbit in sein ACK
  - Sender reduziert seine Senderate

Mittlere Warteschlangenlänge



- Überlastbit wird gesetzt, wenn Mittelwert ≥ 1

Reduktion der Senderate

- Überlastfenster mit *Additive Increase / Multiplicative Decrease*
- Host bobachtet, wieviele Pakete im letzten Fenster zu ACKs mit gesetztem Überlastbit führten
  - mehr als 50%: Fenster auf 7/8 der Größe reduzieren
  - weniger als 50%: Fenstergröße um 1 erhöhen

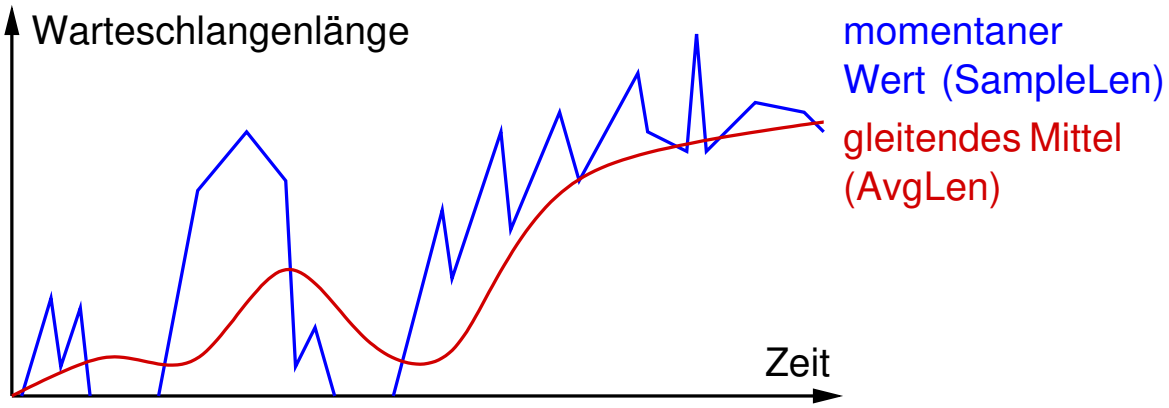
6.2.2 *Random Early Detection* (RED)

- Floyd/Jacobson 1993, entwickelt für den Einsatz mit TCP
- Gemeinsamkeit mit DECbit
  - Router überwachen mittlere Warteschlangenlänge, Quellen passen Überlastfenster an
- Unterschiede:
  - kein explizites Feedback, sondern Verwerfen eines Pakets
    - Zusammenarbeit mit TCP-Überlastkontrolle: TCP halbiert bei Paketverlust das **CongestionWindow**!
    - Router verwirft also (einzelne) Pakete, bevor Puffer wirklich voll ist (und alle verworfen werden müssten)
  - *Random Early Drop* statt ausschließlich *Tail Drop*
    - ab einer bestimmten Warteschlangenlänge, werden ankommende Pakete mit gewisser Wahrscheinlichkeit verworfen

6.2.2 *Random Early Detection* (RED) ...

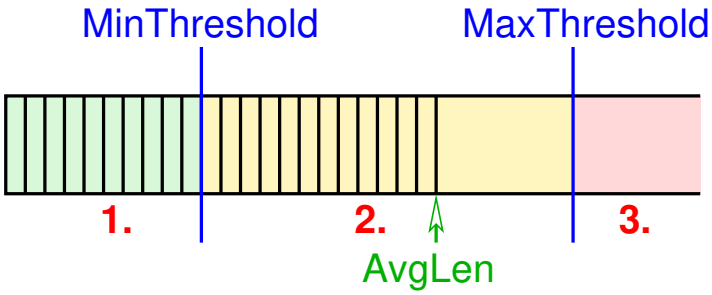
Gewichtetes, gleitendes Mittel der Warteschlangenlänge

- **AvgLen** = $(1-$**Weight**$)$ · **AvgLen** + **Weight** · **SampleLen**
  - **SampleLen** z.B. immer bei Ankunft eines Pakets messen



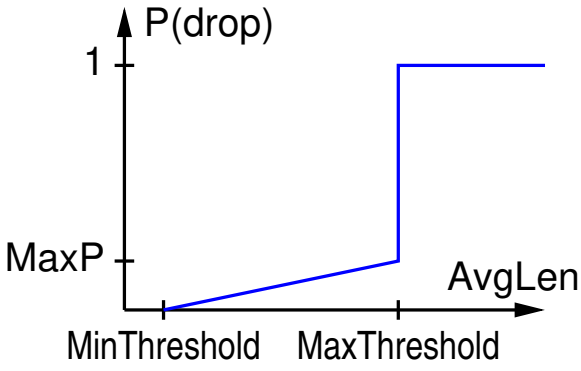
- Reaktion nur auf langlebige Überlast, nicht auf kurze *Bursts*
  - **Weight** so wählen, daß über ≥ 1 RTT gemittelt wird

Grenzwerte für mittlere Warteschlangenlänge



1. Falls **AvgLen** $\leq$ **MinThreshold**: stelle Paket in Warteschlange
2. Falls **MinThreshold** $<$ **AvgLen** $<$ **MaxThreshold**:
   - ➥ berechne Wahrscheinlichkeit **P**
   - ➥ verwirf ankommendes Paket mit Wahrscheinlichkeit **P**
3. Falls **AvgLen** $\geq$ **MaxThreshold**: verwirf ankommendes Paket

Berechnung der *Drop*-Wahrscheinlichkeit



$$\mathbf{TempP = MaxP \cdot \frac{AvgLen - MinThreshold}{MaxThreshold - MinThreshold}}$$

$$\mathbf{P = \frac{TempP}{1 - count \cdot TempP}}$$

- ➥ **count**: Anzahl der Pakete, die gepuffert (also nicht verworfen) wurden, seit **AvgLen** $>$ **MinThreshold**
  - ➥ bewirkt gleichmäßige zeitliche Verteilung verworfener Pakete

Bemerkungen

- RED ist näherungsweise fair
  - Anzahl verworfener Pakete in etwa proportional zum Bandbreitenanteil eines Flusses
- Zur Wahl der Grenzwerte:
  - **MinThreshold** sollte groß genug sein, um Leitung auszulasten
  - oberhalb von **MaxThreshold** sollte noch genügend Speicherplatz für *Bursts* bleiben
  - Differenz größer als typische Erhöhung von **AvgLen** während einer RTT
    - im Internet typisch **MaxThreshold** = 2 · **MinThreshold**

6.2.3 Quellenbasierte Überlastvermeidung

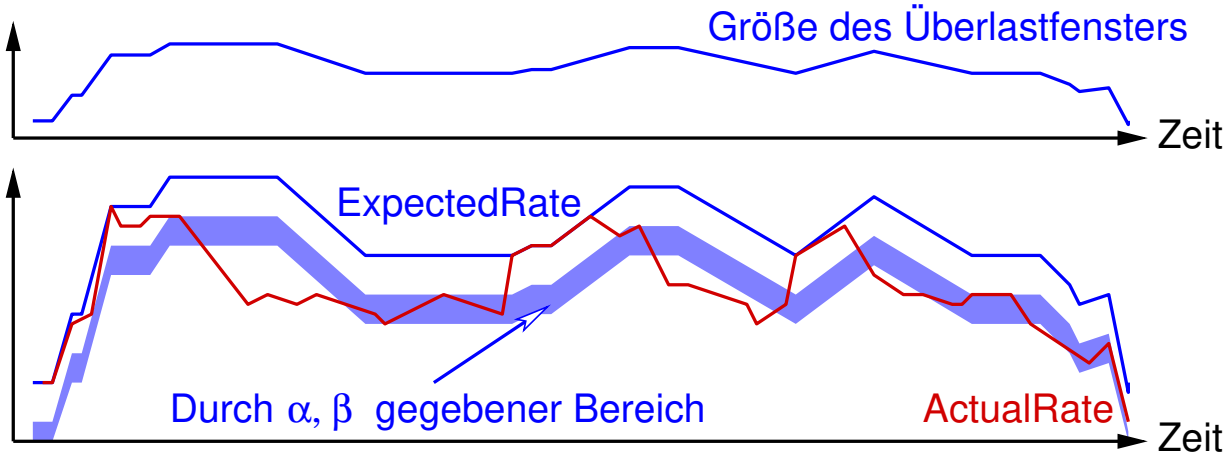
Wie kann eine Quelle drohende Überlast erkennen?

- Messung der RTT zwischen Paket und ACK
  - volle Warteschlangen in Routern bewirken höhere RTT
  - z.B.: verkleinere Überlastfenster auf 7/8, wenn gemessene RTT $\geq$ Mittelwert von bisheriger min. und max. RTT
- Vergleich von erreichtem mit erwartetem Durchsatz
  - erwarteter Durchsatz: Fenstergröße / RTT
    - pro RTT wird ein Fenster an Daten verschickt
    - maximal erreichbarer Durchsatz
  - bei hoher Auslastung:
    - bei größerem Fenster werden lediglich mehr Pakete in Routern gepuffert, aber nicht mehr übertragen

Überlastvermeidung in TCP Vegas

- Berechne **ExpectedRate** = **CongestionWindow** / **BaseRTT**
  - **BaseRTT**: Minimum der bisher gemessenen RTTs
- Berechne **ActualRate** einmal pro RTT
  - Datenmenge, die zwischen dem Abschicken eines Pakets und dem Empfang des zugehörigen ACK übertragen wurde
- Setze **Diff** = **ExpectedRate** - **ActualRate**
- Stelle Überlastfenster so ein, daß $\alpha \leq$ **Diff** $\leq \beta$:
  - falls **Diff** $< \alpha$: vergrößere Überlastfenster linear
  - falls **Diff** $> \beta$: verkleinere Überlastfenster linear
- α, β bestimmen min./max. zu belegenden Pufferplatz

Beispiel zu TCP Vegas



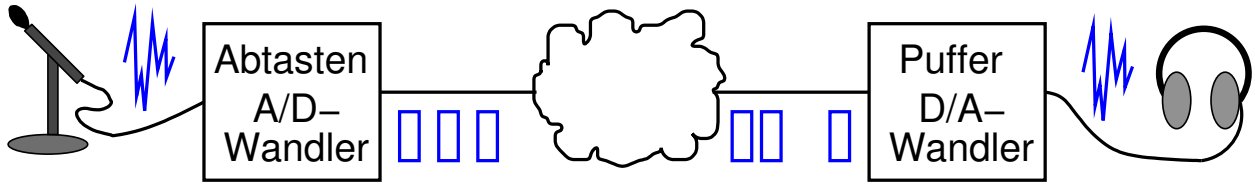
- **ActualRate** unterbalb des Bereichs: Gefahr, daß zu viele Pakete zwischengespeichert werden müssen
- **ActualRate** oberhalb: zu geringe Netzauslastung

6.3 *Quality of Service* (QoS)

- Paketvermittelnde Netze sollen neben traditionellen („elastischen“) Anwendungen auch Echtzeitanwendungen unterstützen, z.B.:
  - Audio- und Video, z.B. Videokonferenz
  - industrielle Steueraufgaben, z.B. Roboter
  - Datenbank-Aktualisierung über Nacht
- Dazu: Daten müssen **rechtzeitig** ankommen
- Netzwerk muß neben *Best Effort* bessere Dienste bieten
  - Zusicherung der zeitgerechten Übertragung
    - i.d.R. ohne Paketverlust und Neuübertragung
- **Dienstgüte** (***Quality of Service***)
  - verschiedene Dienstklassen mit verschiedenen Garantien

6.3 *Quality of Service* (QoS) ...

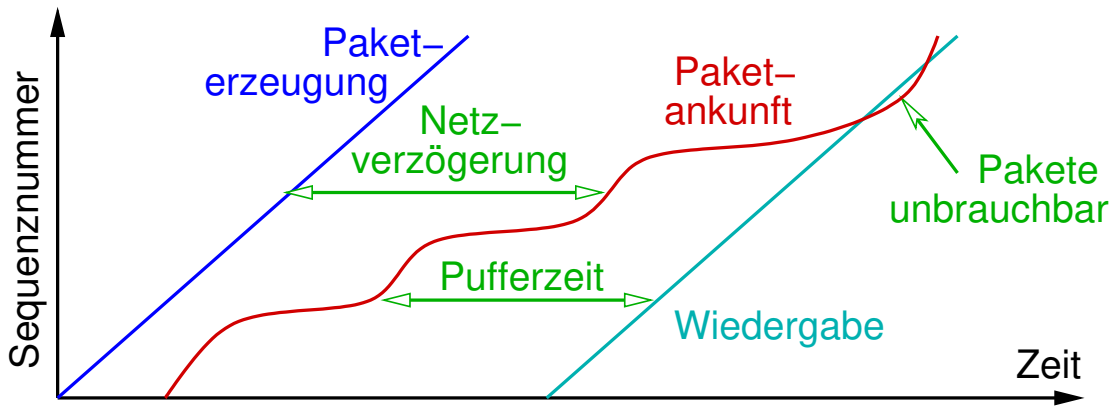
Beispiel: Echtzeit-Audioübertragung



- Sender tastet alle 125 μs ab, sendet regelmäßig Pakete
- Empfänger muß Signal mit der selben Rate wiedergeben
- Pakete treffen beim Empfänger unregelmäßig ein
  - zu spät kommende Pakete i.W. nutzlos!
  - auch verworfene und danach neu übertragene Pakete

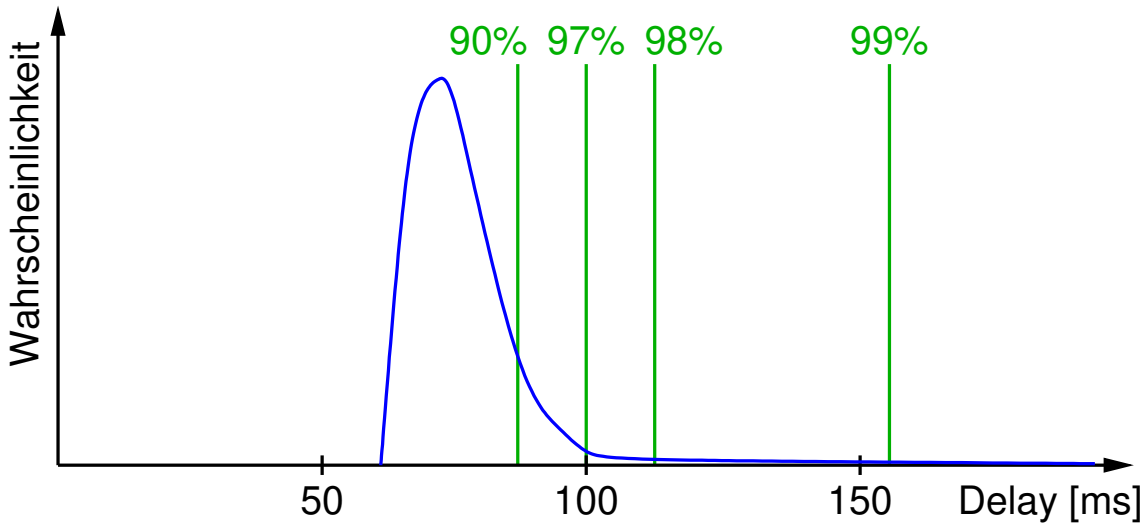
Beispiel: Echtzeit-Audioübertragung ...

- Einführung eines Wiedergabepuffers:



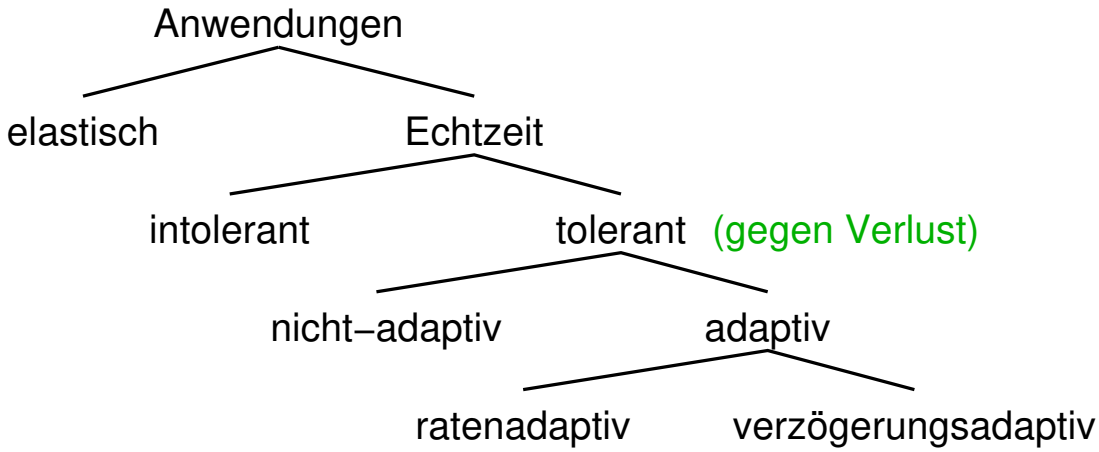
- Je größer der Puffer, desto toleranter gegen Verzögerung
- Verzögerung durch Puffer darf nicht zu groß werden
  - z.B. für Konversation max. ca. 300 ms

Typ. Verteilung der Verzögerungen einer Internet-Verbindung



- 97% aller Pakete kommen nach max. 100 ms an
- Einige Pakete brauchen aber auch mehr als 200 ms

Taxonomie von Anwendungen



- Ratenadaptiv: z.B. Anpassung der Auflösung bei Video
- Verzögerungsadaptiv: z.B. Anpassung Puffergröße bei Audio

(Animierte Folie)

Methoden zur Unterstützung von QoS

- Feingranulare Ansätze
  - stellen Dienstgüte für einzelne Datenflüsse bereit
  - im Internet: *Integrated Services*
  - benötigt viel „Intelligenz“ in den Routern
- Grobgranulare Ansätze
  - stellen Dienstgüte nur für aggregierten Verkehr bereit
  - im Internet: *Differentiated Services*
  - einfacher in den Routern, skalierbarer
  - keine wirklichen Ende-zu-Ende Garantien

6.3.1 *Integrated Services*

IntServ (IETF RFC 2205-2215)

- Motivation: Unterstützung von Multimedia-Streaming
- Basis: zusätzliche Dienstklassen
  - *Guaranteed Service*
    - für intolerante Anwendungen
    - Netz garantiert maximale Verzögerung
  - *Controlled Load*
    - für tolerante, adaptive Anwendungen
    - emuliert wenig belastetes Netz
- *Resource Reservation Protocol* (RSVP) erlaubt Reservierungen

6.3.1 *Integrated Services* ...

Mechanismen zur Erbringung der Dienste

- *FlowSpecs*
  - Beschreibung des angeforderten Dienstes
  - Beschreibung der Datenflüsse
- Zugangskontrolle
  - kann der Dienst bereitgestellt werden?
- Ressourcen-Reservierung
  - Protokoll zwischen Netzwerk-Komponenten
- Paket-Scheduling
  - um Dienstgüte-Anforderungen in Routern zu erfüllen

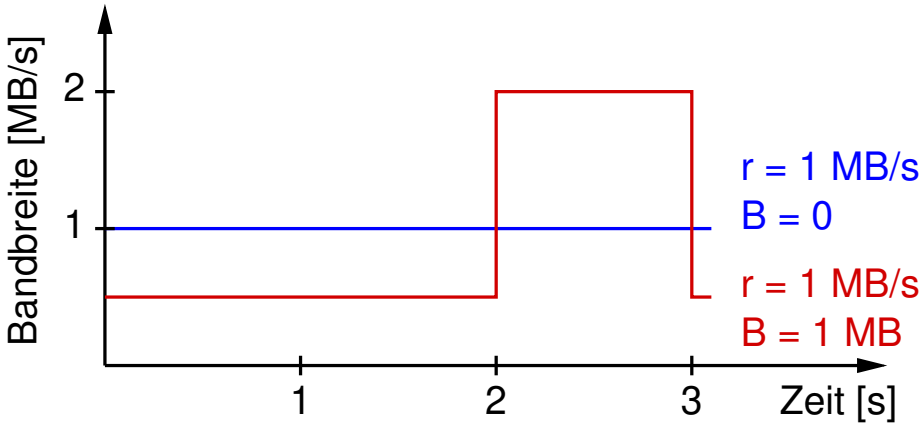
FlowSpec

- *RSpec* (*Request*): Beschreibung des angeforderten Dienstes
  - *Controlled Load Service* (CLS)
  - *Guaranteed Service* (GS) + Spezifikation der maximalen Verzögerung
- *TSpec* (*Traffic*): Beschreibung des Datenflusses
  - maximale mittlere Datenrate
  - Varianz der Datenrate (*Bursts*)
  - Spezifiziert durch ***Token Bucket***

Token Bucket Filter

- Wandelt Datenstrom so um, daß
  - die mittlere Datenrate höchstens r ist
  - *Bursts* mit höherer Rate auf eine Größe von B Bytes beschränkt sind
    - d.h. Router mit Datenrate r am Ausgang braucht höchstens B Bytes an Puffer
- Idee des Filters:
  - Sender benötigt für jedes gesendete Byte ein Token
  - Sender startet mit 0 Token,
    bekommt r Token pro Sekunde,
    kann maximal B Token akkumulieren

Token Bucket Filter ...



- Beide Datenströme haben dieselbe mittlere Datenrate, aber unterschiedliche Bucket-Tiefe

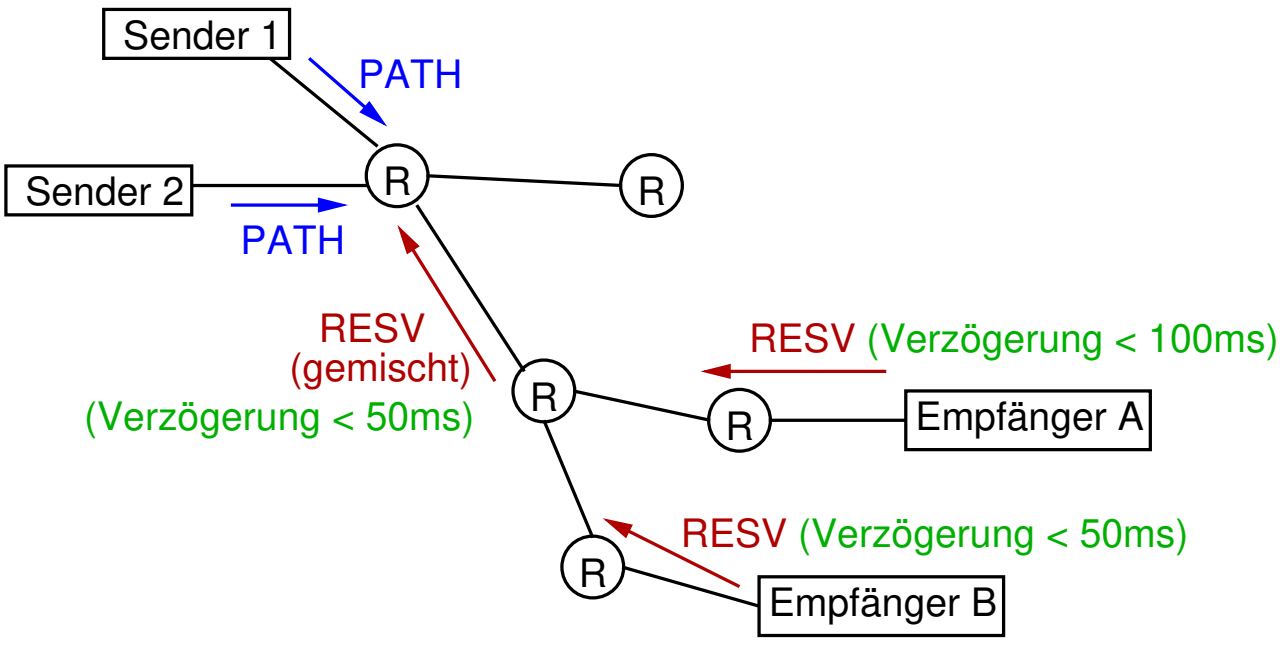
Zugangskontrolle

- Wenn neuer Datenfluß bestimmte Dienstgüte anfordert:
  - betrachte *RSpec* und *TSpec*
  - entscheide, ob Dienst bereitgestellt werden kann
    - bereits zugestandene Dienste dürfen nicht beeinträchtigt werden
- Entscheidung ggf. auf Basis von Benutzerklassen
- Nicht verwechseln mit *Policing*:
  - Prüfung, ob Datenstrom die *TSpec* einhält
  - regelverletzende Pakete werden verworfen bzw. als Wegwerfkandidaten markiert

RSVP Reservierungsprotokoll

- Deutlich verschieden von Signalisierungsprotokollen in verbindungsorientierten Netzen
  - Ziel: Robustheit verbindungsloser Netze erhalten
  - Daher: *Soft State* mit periodischem Auffrischen ($\sim$ alle 30 s)
- RSVP unterstützt auch Multicast-Flüsse
  - Reservierung erfolgt durch Empfänger
- Sender schicht PATH-Nachricht mit *TSpec*
- Empfänger schickt RESV-Nachricht mit seinen Anforderungen (*RSpec*) entlang derselben Route zurück
  - Router teilen erforderliche Ressourcen zu
  - ggf. Zusammenfassung von Anforderungen möglich

RSVP Reservierungsprotokoll ...



Paketklassifizierung und Scheduling

- Klassifizierung: Zuordnung von Paketen zu Reservierungen
  - IPv4: über Quell-/Zieladresse, Quell-/Zielport, Protokoll
  - IPv6: über **FlowLabel**
- Einordnung in Klasse bestimmt Bearbeitung des Pakets in Router-Warteschlange (Scheduling):
  - bei GS: eine Warteschlange pro Fluß, *Weighted Fair Queueing* (WFQ)
    - *Fair Queueing* mit Gewichtung: jede Warteschlange erhält Bandbreitenanteil entsprechend ihrer Gewichtung
    - Ende-zu-Ende-Verzögerung leicht berechenbar
  - bei CLS: eine gemeinsame Warteschlange mit WFQ

Anmerkungen zu Folie 232:

Beim *Fair Queueing* erhält jeder Fluß seine eigene Warteschlage. Idealerweise würden diese Warteschlangen in einer bitweisen Round-Robin-Strategie (RR) abgearbeitet, d.h., aus jeder Warteschlange wird abwechselnd ein Bit gesendet. Dann erhalten klarerweise alle Flüsse exakt denselben Bandbreitenanteil. Beim *Weighted Fair Queueing* erhält jede Warteschlange ein Gewicht n, wobei aus dieser Warteschlange im idealisierten Modell dann n Bits gesendet werden. Die Warteschlange erhält dann im Vergleich zu den anderen mehr Bandbreite.

Natürlich kann man die Warteschlange in Wirklichkeit nicht bitweise abarbeiten: es muß ja immer ein komplettes Paket entnommen und gesendet werden. Daher nähert man das obige Ideal wie folgt an: man berechnet für jedes Paket, wann es gemäß dem idealisierten Modell vollständig gesendet wäre und wählt die Pakete dann in der Reihenfolge der so berechneten Zeiten zum Senden aus.

Näheres dazu im Buch von Peterson und Davie (Kap. 6.2.2. in der 3. Auflage).

Zur Skalierbarkeit

- Router müssen Information für einzelne Datenströme speichern
- Potentiell sehr viel Information:
  - z.B. Audioströme (64 Kb/s) über OC-48-Leitung (2.5 Gb/s): bis zu 39000 Flüsse!
- Für jeden Fluß muß Klassifizierung, *Policing* und Warteschlangenverwaltung durchgeführt werden
- Zusätzlich: Mechanismen, um große Reservierungen für lange Perioden zu vermeiden
- Wegen Skalierbarkeitsproblemen:
  - bisher keine breite Anwendung von *IntServ*

Anmerkungen zu Folie 233:

Bei Messungen wurden sogar 256000 Flüsse über eine OC-3 Leitung festgestellt [Kurose/Ross, S. 538].

6.3.2 Differentiated Services

DiffServ (IETF RFC 2474/2475)

- Ziel: bessere Skalierbarkeit
- Einteilung des Verkehrs in wenige Verkehrsklassen
  - Zuteilung von Ressourcen an Verkehrsklassen statt einzelne Ströme
- Arbeitsaufteilung:
  - Router an der Peripherie (z.B. Eingangsrouter eines ISP)
    - Zuweisung einer Verkehrsklasse an Pakete
      - z.B. je nach Kunde eines ISP
    - Policing: Einhaltung eines Verkehrsprofils
  - innere Router:
    - Weiterleitung der Pakete entsprechend Verkehrsklasse

6.3.2 Differentiated Services ...

Kennzeichnung von Verkehrsklassen

- **TOS**-Feld in IPv4 bzw. **TrafficClass**-Feld in IPv6
  - wird vom Eingangsrouter gesetzt
  - z.B. aufgrund von Quell-/Ziel-Adresse, Quell-/Zielport
  - einfachster Fall: betrachte nur Quell-IP-Adresse
    - d.h. z.B. normaler und „Premium“ Internetzugang
- Feld legt in Routern das Weiterleitungsverhalten (*Per Hop Behavior*, PHB) fest
  - derzeit spezifiziert:
    - *Expedited Forwarding*, EF (RFC 2598)
    - *Assured Forwarding*, AF (RFC 2597)

Expedited Forwarding

- Ziel: Pakete mit EF-Kennzeichung werden mit
  - minimaler Verzögerung
  - geringstmöglichem Paketverlust

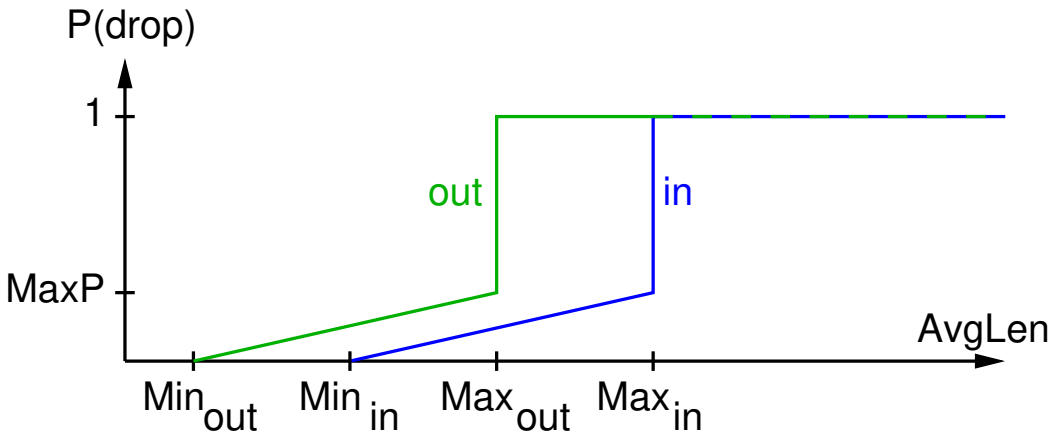
  weitergeleitet, z.B. für *Voice over IP* (VoIP)
- Router kann dies nur garantieren, wenn Ankunftsrate von EF-Paketen limitiert ist
  - Sicherstellung durch Router am Rand der Domäne
- Mögliche Implementierungen:
  - EF-Pakete erhalten strikte Priorität
  - WFQ mit entsprechend hoher Gewichtung für EF-Pakete

Assured Forwarding

- Ziel: gute Bandbreite bei geringer Verlustrate (z.B. für Video)
- 4 Klassen, jeweils mit Mindestanteil an Bandbreite und Pufferplatz
  - realisierbar durch mehrere Warteschlangen und WFQ mit entsprechenden Gewichtungen
  - Beispiel mit zwei Klassen (*Premium*, *Best Effort*):
    - Gewichtung 1 für *Premium*, 4 für *Best Effort* führt zu 20% reservierter Bandbreite für *Premium*
    - Verzögerung für *Premium* aber nicht notwendigerweise geringer als bei *Best Effort*
- Innerhalb jeder Klasse drei Drop-Prioritäten
  - für die Überlastvermeidung innerhalb der Klassen
  - realisierbar durch *Weighted RED*

Weighted RED

- Beispiel: RED mit zwei verschiedenen Drop-Wahrscheinlichkeiten



  - RIO: *RED with In and Out*
  - *In*: innerhalb der Verkehrsprofils, *Out*: außerhalb

6.3.3 Diskussion

- QoS im Internet motiviert durch Multimedia-Anwendungen
- *IntServ*
  - *Guaranteed Service* kann max. Verzögerung garantieren
  - getrennte Behandlung der einzelnen Flüsse
  - explizite, dynamische Ressourcenreservierung durch RSVP
  - Probleme: größere Änderungen in Routern, Skalierbarkeit
- *DiffServ*
  - Skalierbarer Ansatz
  - Klassifikation von Paketen am Rand des Netzwerks, Paketklasse bestimmt Behandlung in Routern
  - Frage: ausreichend für Anwendungen?

6.4 Zusammenfassung / Wiederholung

Überlastkontrolle

- Überlast: häufiger Paketverlust im Netz, da Pakete nicht weitergeleitet und auch nicht mehr gepuffert werden können
- Überlastkontrolle in TCP
  - *Additive Increase / Multiplicative Decrease*
    - verkleinere Fenster bei Überlast drastisch
  - Erweiterungen: *Slow Start*, *Fast Retransmit / Fast Recovery*

Überlastvermeidung

- Senderate reduzieren, bevor Überlast auftritt
- Erkennung drohender Überlast:
  - Router: mittlere Warteschlangenlänge
  - Host: Latenz bzw. Durchsatz

6.4 Zusammenfassung / Wiederholung ...

Überlastvermeidung ...

- DECbit
  - falls mittlere Warteschlangenlänge über Grenzwert:
    - Router setzt Überlastbit im Paketheader
  - Empfänger kopiert Überlastbit in ACK
  - Sender reduziert Überlastfenster, wenn die Mehrzahl der ACK's Überlastbit gesetzt hat
- RED (*Random Early Detection*)
  - für Zusammenarbeit mit TCP konzipiert
  - falls mittlere Warteschlangenlänge über Grenzwert:
    - Router verwirft zufällig einige Pakete
  - TCP reduziert bei Paketverlust das Sendefenster

Überlastvermeidung ...

- Quellenbasierte Überlastvermeidung
  - TCP Vegas: Vergleich von tatsächlichem Durchsatz und erwartetem (maximalem) Durchsatz
    - Differenz ist Maß für beanspruchten Pufferplatz
  - Sendefenster so regeln, daß immer (nur) eine kleine Zahl von Puffern belegt wird

QoS (*Quality of Service*)

- Unterstützung verschiedener Dienstklassen mit Garantien bzgl. Bandbreite, Latenz, Jitter, etc.
- Für Realzeit- bzw. Multimedia-Anwendungen

QoS (*Quality of Service*) ...

- *Integrated Services*: feingranularer Ansatz
  - fluß-spezifische Dienstgüte-Garantien
  - Mechanismen: *Flow Specs*, Zugangskontrolle, Reservierung, Paket-Scheduling
    - Spezifikation des Flusses über *Token Bucket*
  - zwei Dienste: *Guaranteed Service*, *Controlled Load*
  - Problem: Skalierbarkeit in den Routern
- *Differentiated Services*: grobgranularer Ansatz
  - Einteilung der Pakete in Verkehrsklassen
  - Zuteilung von Ressourcen an Verkehrsklassen