

## Multicast

Motivation

Grundlagen

Zustellungsgarantien

Ordnungsgarantien

Paxos



- Fehlertoleranz durch Replikation
    - Redundante Applikationsinstanzen auf unterschiedlichen Rechnern
    - Problemstellung: Replikatzustände müssen konsistent gehalten werden
  - Aktive Replikation
    - Einsatz von gleichberechtigten, deterministischen Replikaten
    - Jedes Replikat führt alle Anfragen aus
    - Konsistenz durch Erstellung einer totalen Ordnung auf alle Anfragen
  - Passive Replikation
    - Bestimmung eines Primärreplikats, das als einziges Anfragen ausführt
    - Aktualisierung der anderen Replikate mittels Zustandsänderungen
    - Konsistenz durch Einhaltung der Reihenfolge der Zustandsänderungen
- ⇒ Unabhängig von der Replikationsart: Mechanismus zum Senden von Nachrichten an die gesamte Replikatgruppe erforderlich



- Multicast-Unterstützung durch das Kommunikationssystem
  - Vorteil: Geringere Komplexität der Replikations-/Anwendungsschicht
  - Beispiel: Teil eines Gruppenkommunikationssystems
  - Implementierungsvarianten
    - Auf Basis spezieller Netzwerkmechanismen (z. B. IP-Multicast) Hier nicht betrachtet
    - Mit Hilfe von Punkt-zu-Punkt-Verbindungen
- Herausforderungen
  - Zuverlässige Zustellung von Nachrichten trotz Fehlern
  - Bereitstellung von Ordnungsgarantien
  - Beweisbarkeit zugesicherter Eigenschaften
- Literatur
  -  Christian Cachin, Rachid Guerraoui, and Luís Rodrigues  
**Introduction to Reliable and Secure Distributed Programming (2nd ed.)**  
Springer, 2011.



- Unterschiede zu Broadcast
  - Sender kennt die Empfänger seiner Nachricht
  - Explizite Anmeldung bei Eintritt in eine Multicast-Gruppe erforderlich
- Zentrale Operationen eines Kommunikationssystems
  - Senden (multicast)
    - Anwendung übergibt Nachricht an das Kommunikationssystem
    - Nachricht enthält Adressen aller designierten Empfänger
  - Ausliefern (deliver)
    - Kommunikationssystem übergibt Nachricht an die Anwendung
    - Einmal ausgelieferte Nachricht kann nicht zurückgenommen werden
- *Knoten*: Teilnehmer an einer Multicast-Gruppe
  - Übliche Synonyme: Prozess, Rechner, Prozessor,...
  - Im Folgenden: Unterscheidung im Kontext von Fehlern
    - *Fehlerhaft*: Knoten, der während der Beobachtungsphase ausfällt
    - *Korrekt*: Knoten, der nicht fehlerhaft ist



- Grundansatz zur Realisierung von Fehlerdetektoren
  - Austausch von Heartbeat-Nachrichten zwischen Knoten
  - Überwachung der Heartbeat-Nachrichten mittels Timeouts
  - Auswertung des Ausbleibens von Heartbeat-Nachrichten
- Problem: Genauigkeit der Fehlererkennung
  - Synchrone Systeme
    - Bekannte obere Schranken für Ausführungs- und Übertragungszeiten
    - Ausbleiben einer Heartbeat-Nachricht ist Beleg für einen Fehler
    - Präzise Fehlererkennung möglich
  - Asynchrone Systeme
    - Keine oberen Schranken für Ausführungs- und Übertragungszeiten
    - Ausbleiben einer Heartbeat-Nachricht ist allenfalls Hinweis auf einen Fehler
    - Keine präzise Fehlererkennung möglich

⇒ Die {Nicht-,}Verfügbarkeit eines Fehlerdetektors hat im Allgemeinen Einfluss auf das Design von Algorithmen [Siehe später...]



# Fair-Loss-Punkt-zu-Punkt-Verbindung

- Grundidee
  - Modell einer unzuverlässigen Netzwerkverbindung zwischen zwei Knoten
  - Ausgangsbasis für die Modellierung komplexerer Verbindungen
- Eigenschaften
  - **Faire Verluste:** Eine von einem korrekten Knoten unendlich oft gesendete Nachricht wird vom Empfänger unendlich oft ausgeliefert
  - **Endliche Vervielfachung:** Eine von einem korrekten Knoten endlich oft gesendete Nachricht wird vom Empfänger endlich oft ausgeliefert
  - **Keine Erzeugung:** Nur gesendete Nachrichten werden ausgeliefert
- Bemerkungen
  - Einschränkende, aber realistische Annahmen über Unzuverlässigkeit
  - Betrachtung temporärer, gutmütiger Netzwerkfehler
    - Keine dauerhafte Netzwerkpartitionierung zwischen Sender und Empfänger
    - Wahrscheinlichkeit, dass eine gesendete Nachricht ausgeliefert wird, ist  $> 0$



# Perfekte Punkt-zu-Punkt-Verbindung

- Eigenschaften
  - **Zuverlässige Auslieferung:** Falls Sender und Empfänger einer Nachricht korrekt sind, liefert der Empfänger die Nachricht letztendlich aus
  - **Keine Vervielfachung:** Eine Nachricht wird maximal einmal ausgeliefert
  - **Keine Erzeugung:** Nur gesendete Nachrichten werden ausgeliefert
- Realisierung
  - Ausgangsbasis: Fair-Loss-Punkt-zu-Punkt-Verbindung
  - Sendeseite
    - Wiederholte Übertragung einer zu sendenden Nachricht
    - Pausen zwischen einzelnen Übertragungen
  - Empfangsseite
    - Protokoll über bereits ausgelieferte Nachrichten
    - Erkennung und Unterdrückung von Duplikaten

⇒ **Baustein für Multicast-Implementierungen**

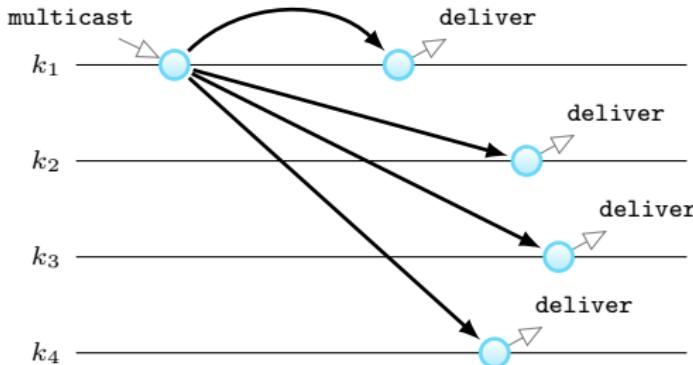


- Eigenschaften
    - **Gültigkeit:** Falls ein korrekter Knoten eine Nachricht  $n$  sendet, dann liefert jeder korrekte Knoten die Nachricht  $n$  letztendlich aus
    - **Keine Vervielfachung:** Eine Nachricht wird maximal einmal ausgeliefert
    - **Keine Erzeugung:** Falls ein Knoten die Nachricht  $n$  eines Senders  $k$  ausliefert, wurde  $n$  auch zuvor von Knoten  $k$  gesendet
  - Bemerkungen
    - Vergleiche: Eigenschaften einer perfekten Punkt-zu-Punkt-Verbindung
    - Keine Aussagen über
      - Nachrichten, die von fehlerhaften Knoten gesendet wurden
      - das Ausliefern von Nachrichten auf fehlerhaften Knoten
    - Es kann von fehlerhaften Knoten gesendete Nachrichten geben, die nur von einem Teil der korrekten Knoten ausgeliefert werden
- ⇒ Im Fehlerfall: Inkonsistenz zwischen korrekten Knoten möglich



## ■ Realisierung

- Einsatz perfekter Punkt-zu-Punkt-Verbindungen
- Sender verschickt Nachricht an alle Knoten
- Ein Knoten liefert die Nachricht aus, sobald er sie bekommen hat



## ■ Problemszenarien (Beispiele)

- Sender fällt aus, bevor er die Nachricht an alle Knoten gesendet hat
- Empfänger fällt aus, bevor er die Nachricht ausliefern konnte



## ■ Eigenschaften

[Grau: Identisch zum Best-Effort-Multicast]

- **Gültigkeit:** Falls ein korrekter Knoten  $k$  eine Nachricht  $n$  sendet, dann liefert Knoten  $k$  die Nachricht  $n$  letztendlich auch aus
- **Keine Vervielfachung:** Eine Nachricht wird maximal einmal ausgeliefert
- **Keine Erzeugung:** Falls ein Knoten die Nachricht  $n$  eines Senders  $k$  ausliefert, wurde  $n$  auch zuvor von Knoten  $k$  gesendet
- **Einigung:** Falls eine Nachricht  $n$  von einem korrekten Knoten ausgeliefert wird, dann liefern letztendlich alle korrekten Knoten  $n$  aus

## ■ Bemerkungen

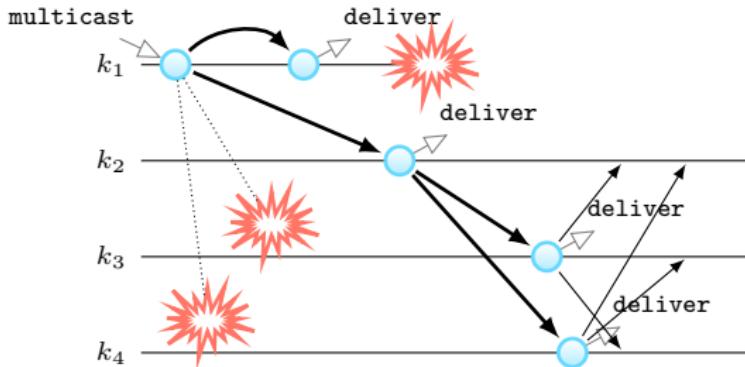
- Konsistenz zwischen korrekten Knoten ist sichergestellt
- Keine Aussage über fehlerhafte Knoten (vgl. Best-Effort-Multicast)

⇒ **Im Fehlerfall: Inkonsistenz zwischen korrekten Knoten und fehlerhaften Knoten möglich**



## ■ Realisierung

- Grundansatz wie bei Best-Effort-Multicast
- Ein Empfänger leitet die Nachricht an alle anderen Knoten weiter

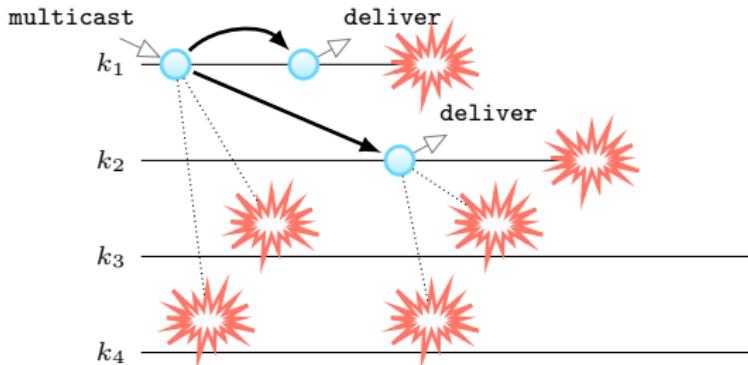


- ## ■ Varianten: Weiterleitung der Nachrichten an andere Knoten
- unmittelbar nach ihrem Empfang (*Eager*)
  - nach der Erkennung eines Knotenausfalls (*Lazy*)



### Beispielszenario

- Fehlerhafter Knoten sendet Nachricht
- Nachricht erreicht keinen korrekten Knoten
- Nachricht wird von mindestens einem fehlerhaften Knoten ausgeliefert



### Problem

- deliver könnte zu einer Aktion (z. B. Antwort an Client) geführt haben
- Korrekte Knoten werden diese Aktion niemals durchführen



### ■ Eigenschaften

[Grau: Identisch zum zuverlässigen Multicast]

- **Gültigkeit:** Falls ein korrekter Knoten  $k$  eine Nachricht  $n$  sendet, dann liefert Knoten  $k$  die Nachricht  $n$  letztendlich auch aus
- **Keine Vervielfachung:** Eine Nachricht wird maximal einmal ausgeliefert
- **Keine Erzeugung:** Falls ein Knoten die Nachricht  $n$  eines Senders  $k$  ausliefert, wurde  $n$  auch zuvor von Knoten  $k$  gesendet
- **Uniforme Einigung:** Falls eine Nachricht  $n$  von **irgendeinem** Knoten ausgeliefert wird, dann liefern letztendlich alle korrekten Knoten  $n$  aus

### ■ Bemerkungen

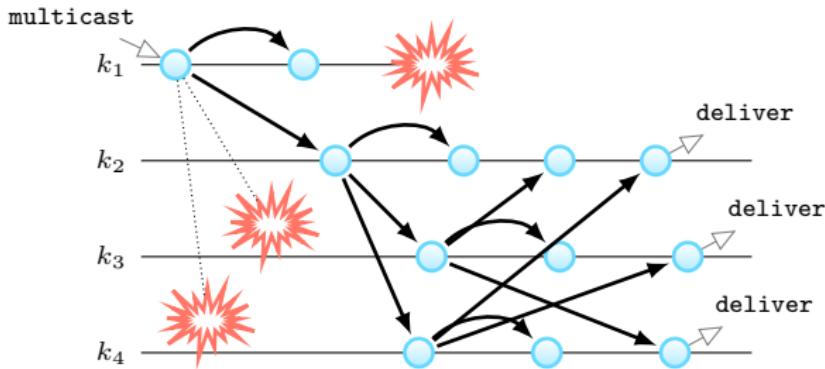
- Einschränkung der im Fehlerfall möglichen Inkonsistenz
- Keine Aussage über Nachrichten, die von fehlerhaften Knoten stammen

⇒ **Eine durch die Auslieferung einer Nachricht ausgelöste Aktion wird letztendlich von allen korrekten Knoten ausgeführt**



## ■ Realisierung

- Grundansatz wie beim zuverlässigen Multicast
- Ein Knoten wartet auf Bestätigungen, bevor er eine Nachricht ausliefert



## ■ Varianten

- Warten auf Bestätigungen aller korrekten Knoten [→ erfordert Fehlerdetektor]
- Warten auf  $\lceil \frac{K+1}{2} \rceil$  Bestätigungen [Annahme:  $K$  Knoten insgesamt, max.  $\lfloor \frac{K-1}{2} \rfloor$  fehlerhaft]



- Eigenschaft
  - *FIFO: First In, First Out*
  - **FIFO-Auslieferung:** Falls ein Knoten eine Nachricht  $n_1$  vor einer Nachricht  $n_2$  sendet, dann liefert ein korrekter Knoten nur dann  $n_2$  aus, wenn er zuvor bereits  $n_1$  ausgeliefert hat
- Bemerkungen
  - FIFO-Eigenschaft bezieht sich jeweils auf die Nachrichten eines Knotens
  - Keine Aussage über die Reihenfolge, in der von unterschiedlichen Knoten gesendete Nachrichten ausgeliefert werden
- Beispiel für Realisierung
  - Nachrichtenversand per (uniformem) zuverlässigem Multicast
  - Nachrichten tragen senderspezifische Sequenznummern
  - Einhaltung der Reihenfolge bei Auslieferung der Nachrichten



- Eigenschaft
  - Annahmen
    - $n_1$  und  $n_2$  sind zwei beliebige Nachrichten
    - $k_1$  und  $k_2$  sind zwei beliebige korrekte Knoten, die  $n_1$  und  $n_2$  ausliefern
  - **Totale Ordnung:** Falls Knoten  $k_1$  Nachricht  $n_1$  vor Nachricht  $n_2$  ausliefert, dann liefert Knoten  $k_2$  auch  $n_1$  vor  $n_2$  aus
- Bemerkungen
  - Totale Ordnung bezieht sich auf sämtliche ausgelieferten Nachrichten
  - Zentraler Baustein replizierter Systeme
  - Problemstellung orthogonal zu FIFO-Ordnung
- Mögliche Realisierung
  - (Uniformer) zuverlässiger FIFO-Multicast als Basis
  - Wahl eines Anführerknotens [Neuwahl nach Ausfall des aktuellen Anführers]
  - Anführer bestimmt die globale Auslieferungsreihenfolge aller Nachrichten



- Paxos-Protokoll
  - Entwickelt Ende der 1980er, Papierveröffentlichung 1998
  - Einigungsprotokoll zur Replikation deterministischer Zustandsmaschinen
  - Tolerierung von Replikatausfällen
- Varianten (Beispiele)
  - *Cheap Paxos*      Einsparung von Ressourcen
  - *Fast Paxos*        Reduzierung der Latenz
  - *Byzantine Paxos*    Tolerierung willkürlicher Fehler
- Literatur



Leslie Lamport

**The Part-time Parliament**

*ACM Transactions on Computer Systems, 16(2):133–169, 1998.*



Jonathan Kirsch and Yair Amir

**Paxos for System Builders: An Overview**

*Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware (LADIS '08), S. 14–19, 2008.*

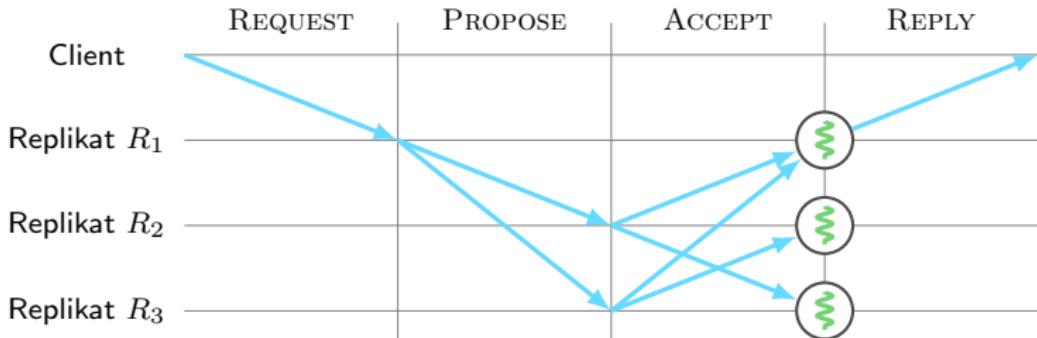


- Architektur

- $2f + 1$  Replikate zur Tolerierung von bis zu  $f$  Replikatausfällen
  - Vorbereitung: Wahl eines Anführerreplikats (hier: Replikat  $R_1$ )

- Protokollinstanz: Zuweisung einer Sequenznummer zu einer Anfrage

- REQUEST Client sendet Anfrage an Anführerreplikat
  - PROPOSE Anführer schlägt {Sequenznummer, Anfrage} vor
  - ACCEPT Andere Replikate nehmen Vorschlag an
  - REPLY Beantwortung der Anfrage nach Erhalt von  $f$  ACCEPTS



- Anführerwechsel
  - Anlass: Trotz neuer Anfragen werden keine neuen Instanzen gestartet
  - Mögliche Implementierung
    - Neuer Anführer ( $\rightarrow$  Replikat mit nächsthöherer ID) schlägt Wechsel vor
    - Andere Replikate stimmen dem Vorschlag zu
      - \* Bericht über Anfragen, für die sie ein PROPOSE/ACCEPT geschickt haben
      - \* Ab sofort: Ignorieren von PROPOSE-Nachrichten des alten Anführers
    - Wechsel beendet, sobald  $f$  andere Replikate dem neuen Anführer folgen
    - Neuer Anführer startet Instanzen für bekannte Anfragen neu
- Sicherungspunkte
  - Probleme
    - Ohne zusätzliche Maßnahmen müssten beim Wechsel des Anführers sämtliche bisherigen Protokollinstanzen wiederholt werden
    - Langsame Replikate haben keine Chance aufzuholen
  - Ansatz: Regelmäßige Erzeugung von Anwendungssicherungspunkten
    - Verwerfen von Informationen über ältere Protokollinstanzen
    - Bereitstellung des Sicherungspunkts auf Anfrage eines anderen Replikats

