

# A Generic and Modular Accounting and Charging System for Peer-to-Peer Applications

**David Hausheer**

*Computer Engineering and Networks Laboratory, TIK*

*Swiss Federal Institute of Technology, ETH Zürich*

*E-Mail: hausheer@tik.ee.ethz.ch*



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich



# Motivation and Problem Statement

- More and more applications benefit from **scalability** and **performance** of P2P systems
  - **File sharing** (BitTorrent, eMule, KaZaA)
  - Distributed **storage**, distributed **computing** (OceanStore, SETI@home)
  - **Host sharing** (PlanetLab)
  - PeerCast, Skype, Groove, ...
- However, P2P systems are vulnerable against **selfish behavior** (free riding)
- Some P2P applications use specific mechanisms to **balance** contribution and consumption of peers
  - Examples:
    - BitTorrent's tit-for-tat mechanism, eMule's credit system
  - Problems:
    - Purely **file sharing oriented** => difficult to apply to other P2P applications
    - No support for **commercial applications**: missing „charging“ functionality



# Objectives

## □ Goal

- To design a **generic** and **modular accounting and charging (A&C) system** for P2P applications

## □ Functional objectives

- Provide **accountability**, i.e. offer appropriate mechanisms to account and charge for **service usage** within P2P applications

=> Give peers appropriate incentives to contribute own resources

=> Punish selfish behavior

- Support both **non-commercial** and **commercial** applications

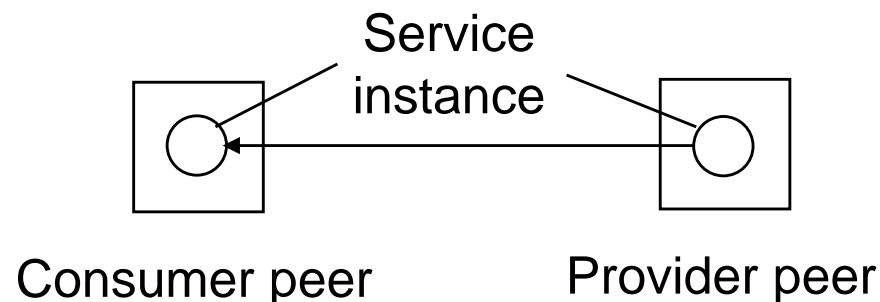
## □ Design objectives

- Can be used for potentially **any P2P application** independently from specific underlying accounting mechanisms (=> **generic**)
- Enables the use of a variety of **accounting mechanisms** (=> **modular**)



# Design Principle 1: Sessions

- A **session** refers to the **use** of a particular **service**
  - E.g. the download of a particular file or the use of computing power
- Every session has two **session partners**
  - A **provider** peer, offering the service
  - A **consumer** peer, using the service
- For every session, there is a **Service Level Agreement (SLA)**
  - Specifies the **basic terms and conditions** of the service usage
  - Agreed upon (**signed**) by both session partners
- On both session partners an **instance** of the **service** is created



## Design Principle 2: Accounting Events and Accounts

- **Accounting events** are generated by the service instances
  - Contain **service-specific** accounting data (e.g. MBs, CPU cycles)
- An **account** is a repository to store and aggregate **accounting data**
- Two types of accounts are distinguished
  - **Session accounts**
    - Used to account for a particular session
    - E.g. amount of MBs downloaded or number of CPU cycles used
  - **Peer accounts**
    - Used to aggregate accounting data from several sessions
    - Keep track of the total amount of service contribution and consumption by a particular peer



## Design Principle 3: Tariffs

- **Tariffs** are part of the SLA and define **how service usage is accounted for**
  - Represented by a specific **tariff formula** which may include several **tariff parameters**
    - Tariff parameters can be **static** or computed **dynamically** during run-time
    - E.g. depending on the **time of day** or the **balance** of a particular **account**
  - Evaluate the **service-specific data** within **accounting events** and transform it into a **generic balance update** which can be stored and aggregated by the appropriate **account**
- Tariffs specify **when** and by **how much** accounts have to be updated
  - E.g. at the **beginning** or **end of a session**, when a particular **threshold** has been **reached** or after a certain **time** has **passed**
  - This enables a variety of **settlement schemes**
    - E.g. pre-payment, post-payment, or more fine-grained schemes



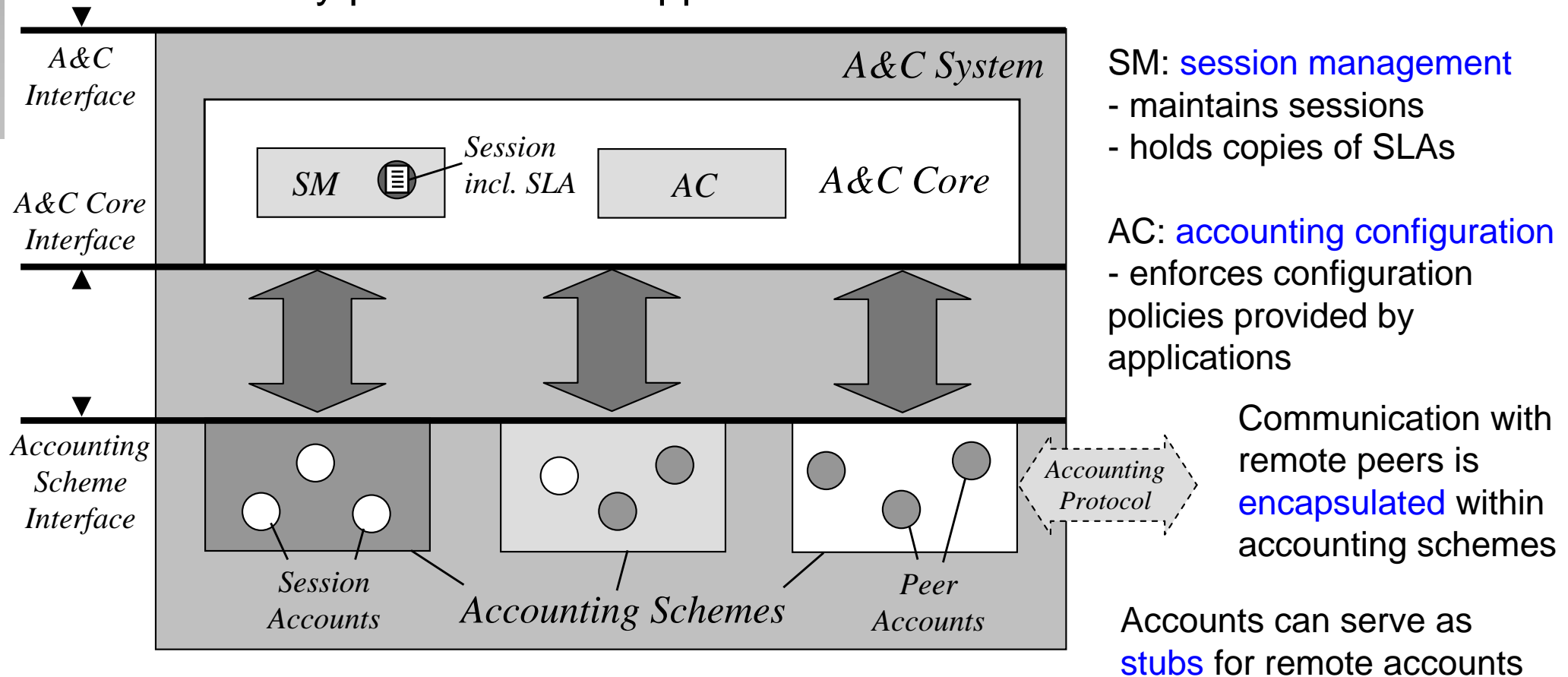
## Design Principle 4: Accounting Schemes

- An **accounting scheme** provides specific functionality for keeping **accounts**
  - Accounting schemes may implement any specific type of accounting
    - E.g. local, centralized, decentralized or token-based accounting
    - They make specific **trade-offs** between efficiency, scalability, and security
  - => Depending on the requirements different schemes may be applied for accounts within an application
- For **compatibility** reasons it is important that all peers within an application use the **same scheme(s)**
  - Every application has to provide a **configuration policy** that specifies which accounting schemes are used to hold what type of accounts



# Implementation

- Based on the presented design principles an **A&C system** has been implemented within the MMAPPS P2P middleware
  - In scope of the middleware an instance of the A&C system is installed on every peer within an application

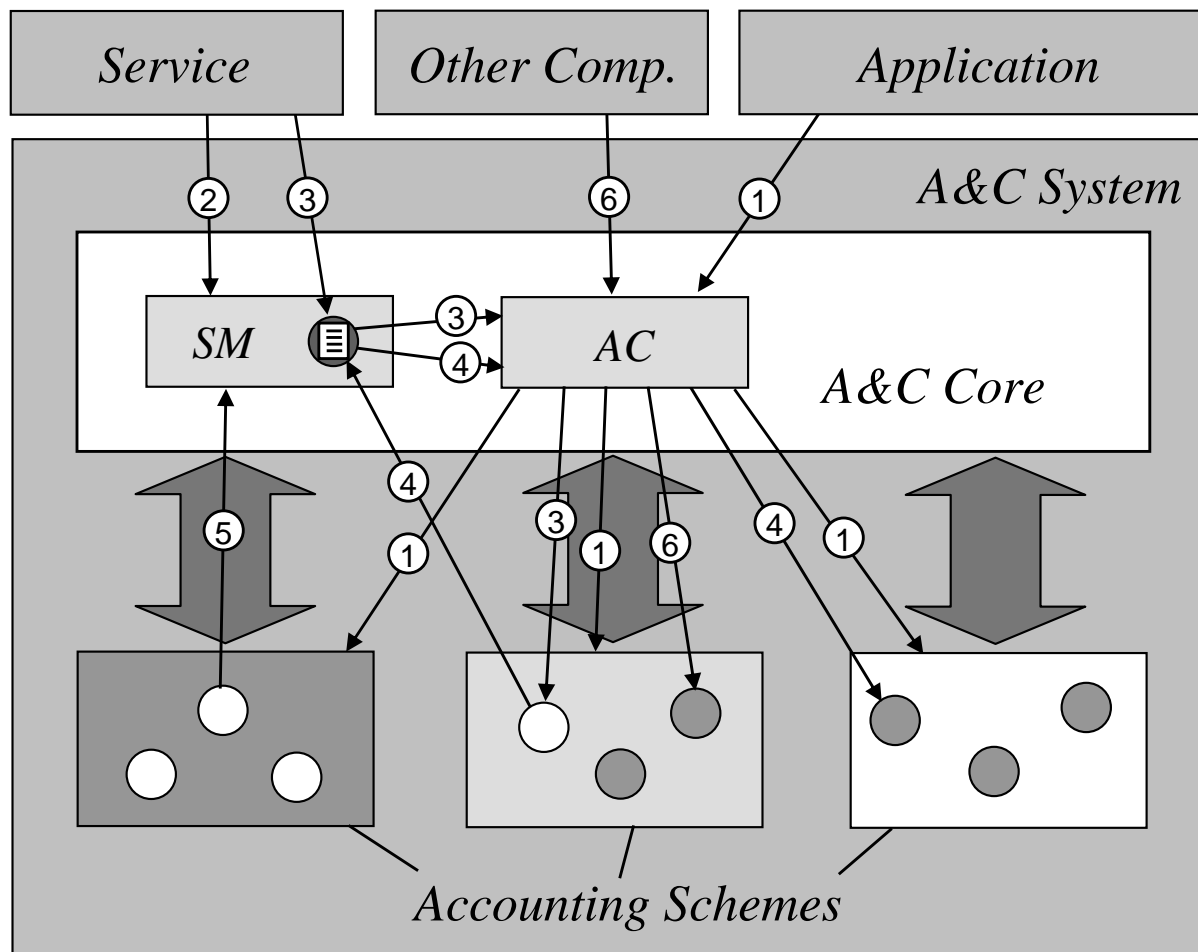


# Interfaces

- **A&C interface**
  - Can be accessed from the application and from other middleware components to
    - Configure the A&C system
    - Notify new sessions
    - Send accounting events
    - Query accounts
- **A&C core interface**
  - Can be used by accounting schemes to
    - Report back the balance of an account
    - Notify a remote peer about a new session
- **Accounting scheme interface**
  - Needs to be implemented by every individual accounting scheme and offers the following methods:
    - `configureScheme`: initializes the scheme with generic and scheme-specific parameters
    - `createPeerAccount`, `createSessionAccount`: create new accounts
    - `notifyBalanceUpdate`: sends balance update to an account
    - `queryAccount`



# Procedures and Interactions



(1) **configure** A&C system, instantiate **accounting schemes**, create **peer account**

(2) notify **new session**, store **SLA**, create new **session account**

(5) create new session on **remote peer** (depending on scheme)

(3) send **accounting events**, evaluate **tariff**, create **balance updates**, update accounts

(4) **report back** account balance, generate further balance updates

(6) **query** accounts, forward query to accounting scheme

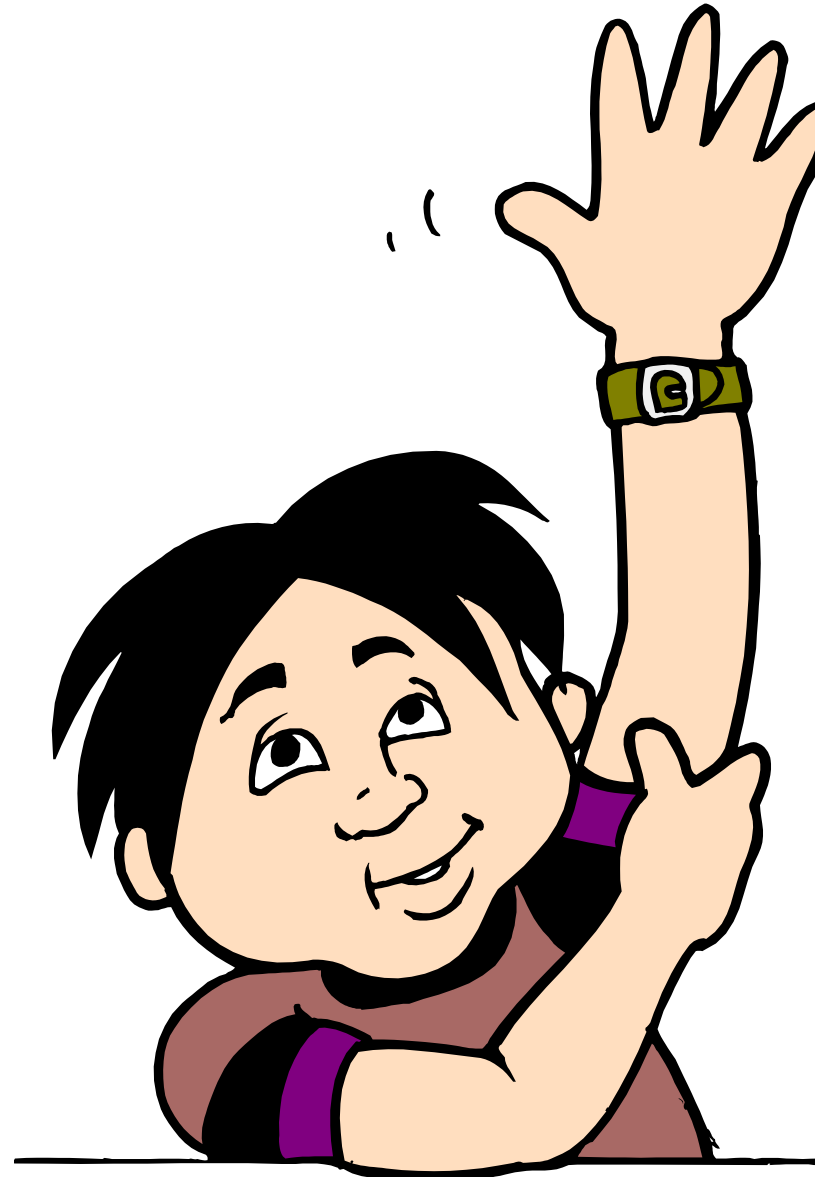


# Conclusions

- The presented A&C system is **modular**...
  - It thus enables a **flexible** use of different **accounting schemes**
- ... and handles service-specific events in a **generic** fashion
  - It can thus be used for a **variety** of **P2P applications**
- A **prototype** has been implemented
  - And successfully used with **different applications**...
    - A file-sharing application and a collaboration system
  - ... and **accounting schemes**
    - Local accounting, token-based accounting, remote accounting
  - Further accounting schemes are envisaged to be implemented in future



Thank you! – Any Questions?



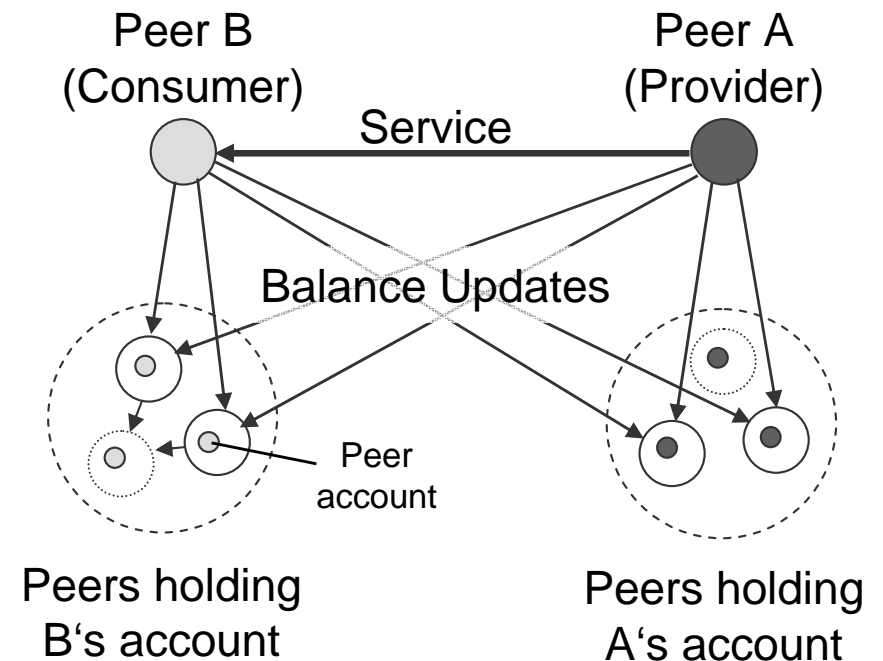
# Remote Accounting Concept

## □ Key Idea

- **Accounts** are **held remotely** on other peers, i.e. third party peers
- Provider / consumer send **balance updates** to remote peers
- Enables to **distribute** and **replicate** accounts over several peers
- Increases **reliability** and **availability** of the accounting data
- A higher **credibility** or **trustworthiness** can be achieved

## □ Problems

- **Collusion** among peers
  - Update only provider account
- Either peer may **cheat**
  - Provider doesn't deliver service
  - Consumer sends incorrect balance update
- **Consistency** of accounts



# PeerMint Approach

- Distinguishes between **peer accounts** and **session accounts**
  - **Session** = use of a **service**, e.g. file download
  - **Session accounts** keep accounting data within a session
  - **Peer accounts** aggregate information from several sessions
  - **Tariffs** specify when and by how much accounts are **updated**
- Uses a redundant set of  $m$  **session mediation peers**
  - Mediation peers keep a session account
  - **Verify** if the two peers agree and update session account accordingly
  - Update peer accounts as specified in **tariff** (e.g. at end of a session)
- Enables **macro-payments**
  - A peer with a high negative credit can pay a peer with a high positive credit
  - Payments are dealt as services
- Technical Design
  - Uses **majority decisions** to overcome byzantine failures
  - Implemented on top of **Pastry** (uses keys and nodelds from PeerMart)
  - Implements generic accounting API of **MMAPPS A&C system**



# Example Session in PeerMint

## □ Session Setup

- Session partners create **signed SLA** with tariff, peerIds and sessionId
- SLA is used by session mediation peers to prove **eligibility**

## □ Account Maintenance

- When **leaf-set changes** new peer becomes responsible
- Obtains current **balance** from account holders

## □ Account Queries

- **No privacy**  
=> any peer can query a peer account

