# Effizientes Routing in P2P Netzwerken

## Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications

Dennis Schade

# Outline

- Short introduction to Chord
- Experiments

# Introduction to Chord

Data addressed by a Hashtable

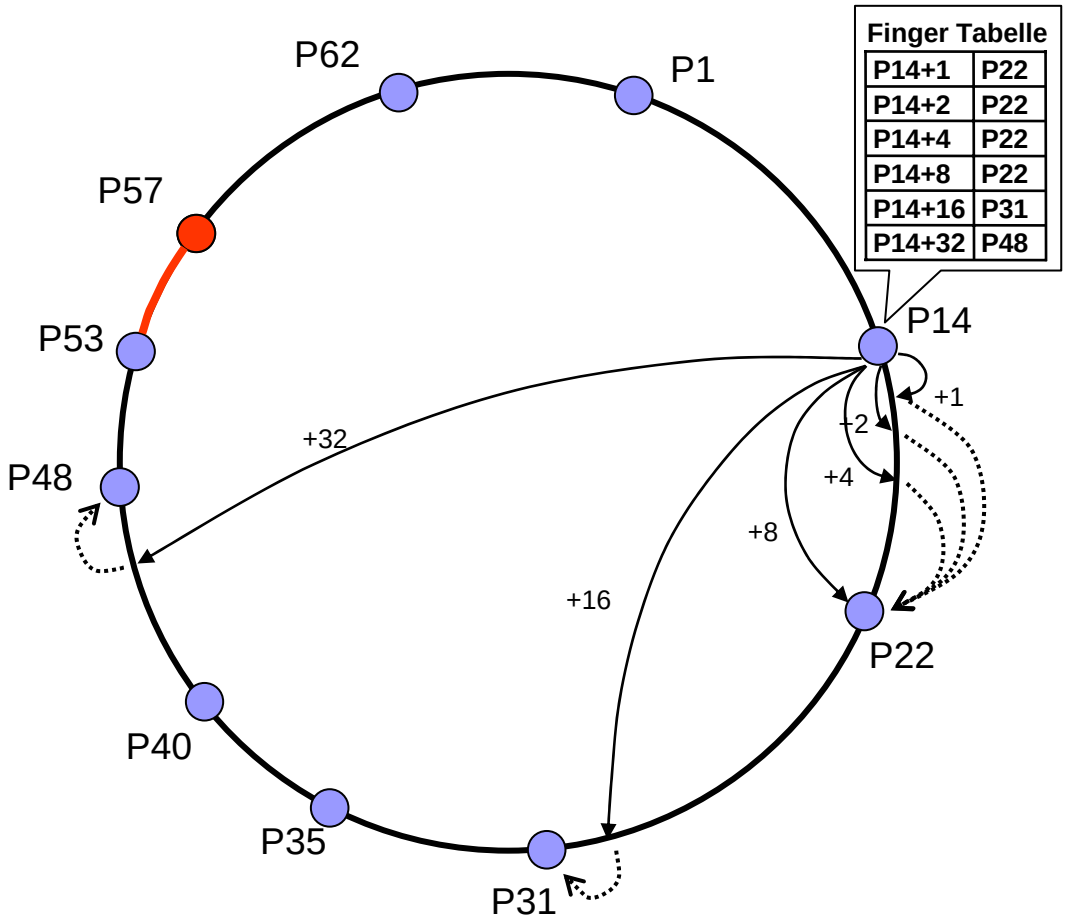
- calculating hashvalues for keys
- storing reference to data identified by hashvalues

A *distributed hashtable (DHT)*

- partitioned
- distributed over peers

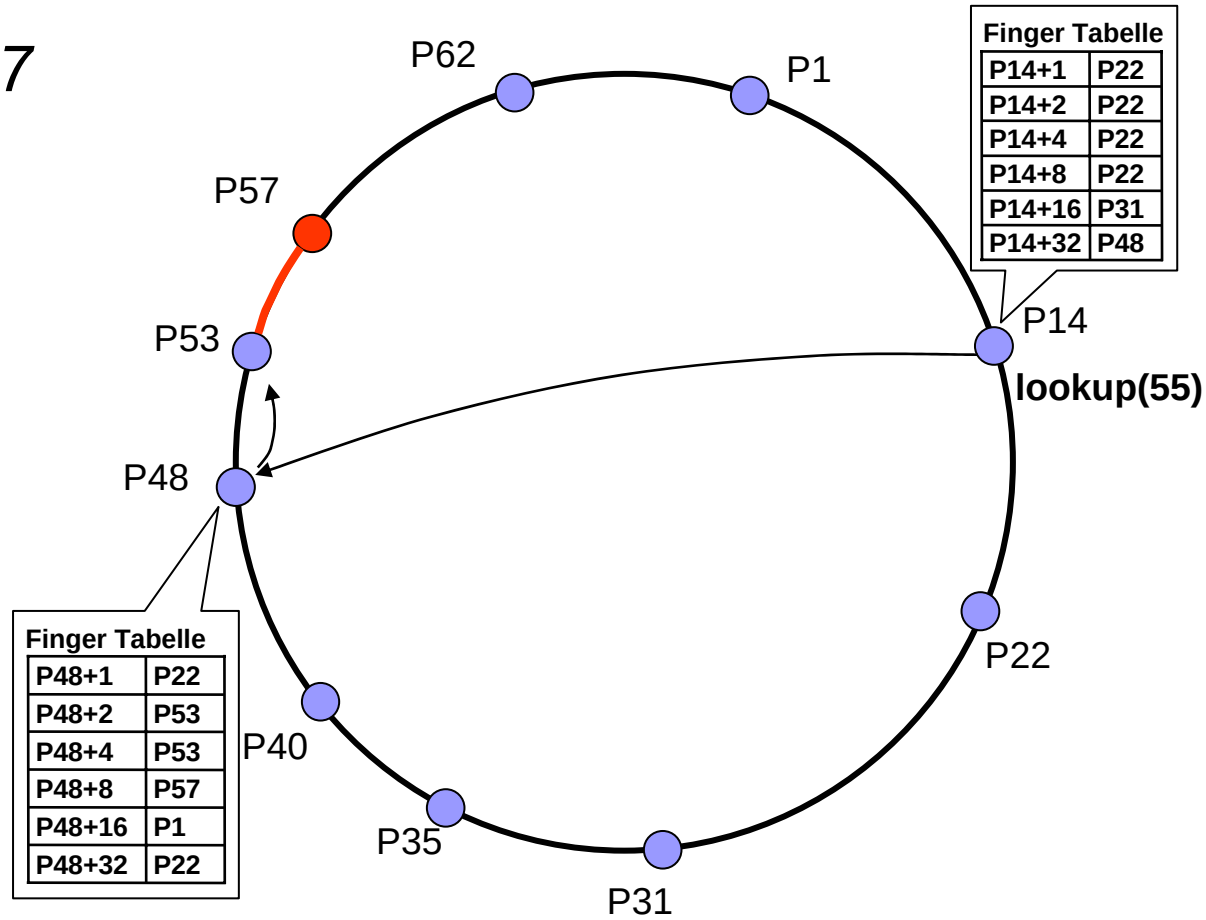
# Introduction to Chord

- *lookup(key) = x, first node x following key with x.ID ≥ key*
- *finger[i] = lookup(x.ID + $2^{i-1}$)*
- *periodic update of neighbours and fingers*



# Introduction to Chord

- *lookup(55) = P57*



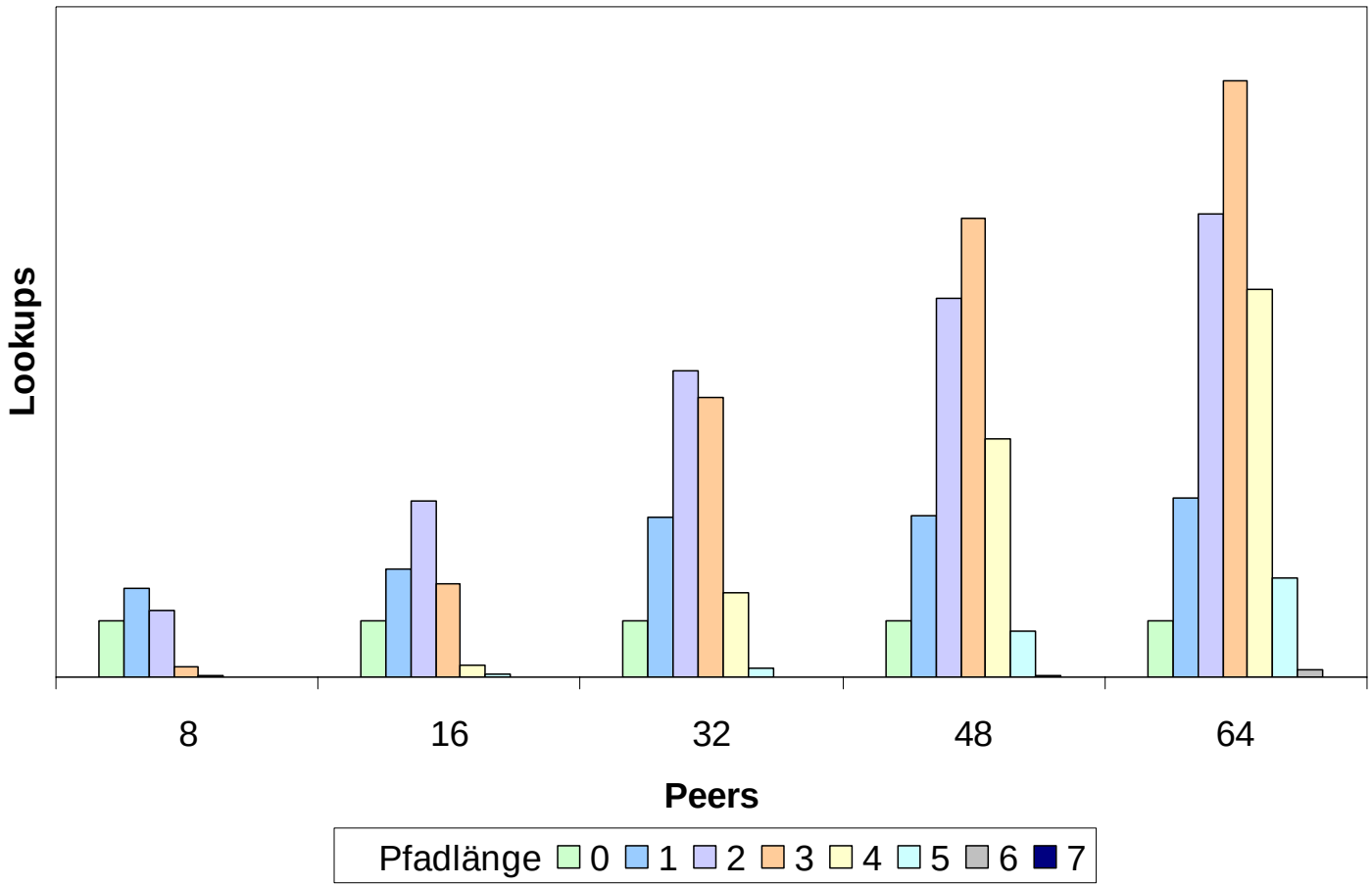
# Experimental Setup

- Java-Implementation of Chord
  - ca. 4500 lines
- Experiments
  - one Java-VM
  - Network device: localhost

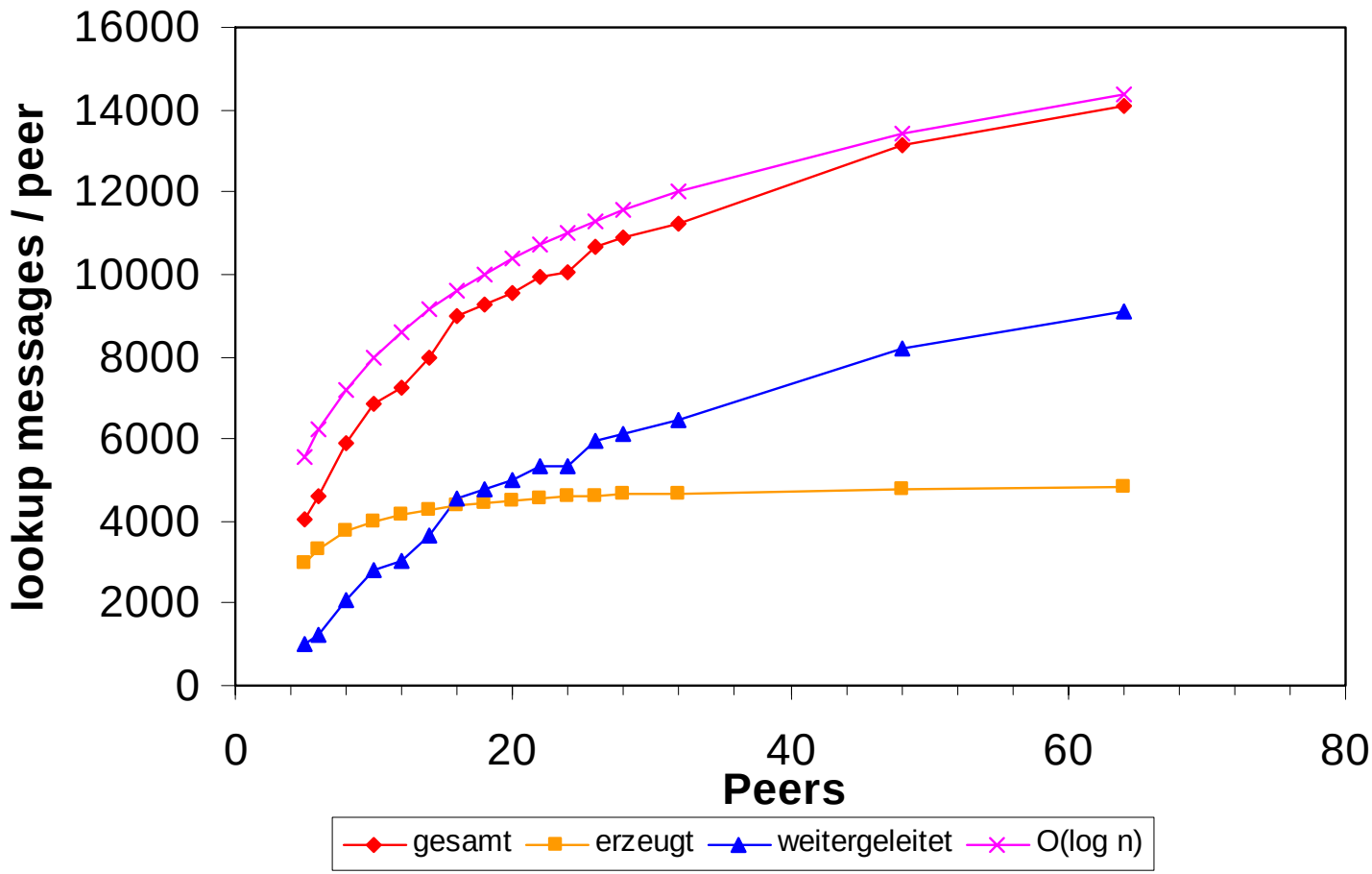
# Experimental Setup

- Path length
- Number of routing messages
- Lookup duration & timeouts
- Bandwidth used
  - Lookups
  - Stabilization

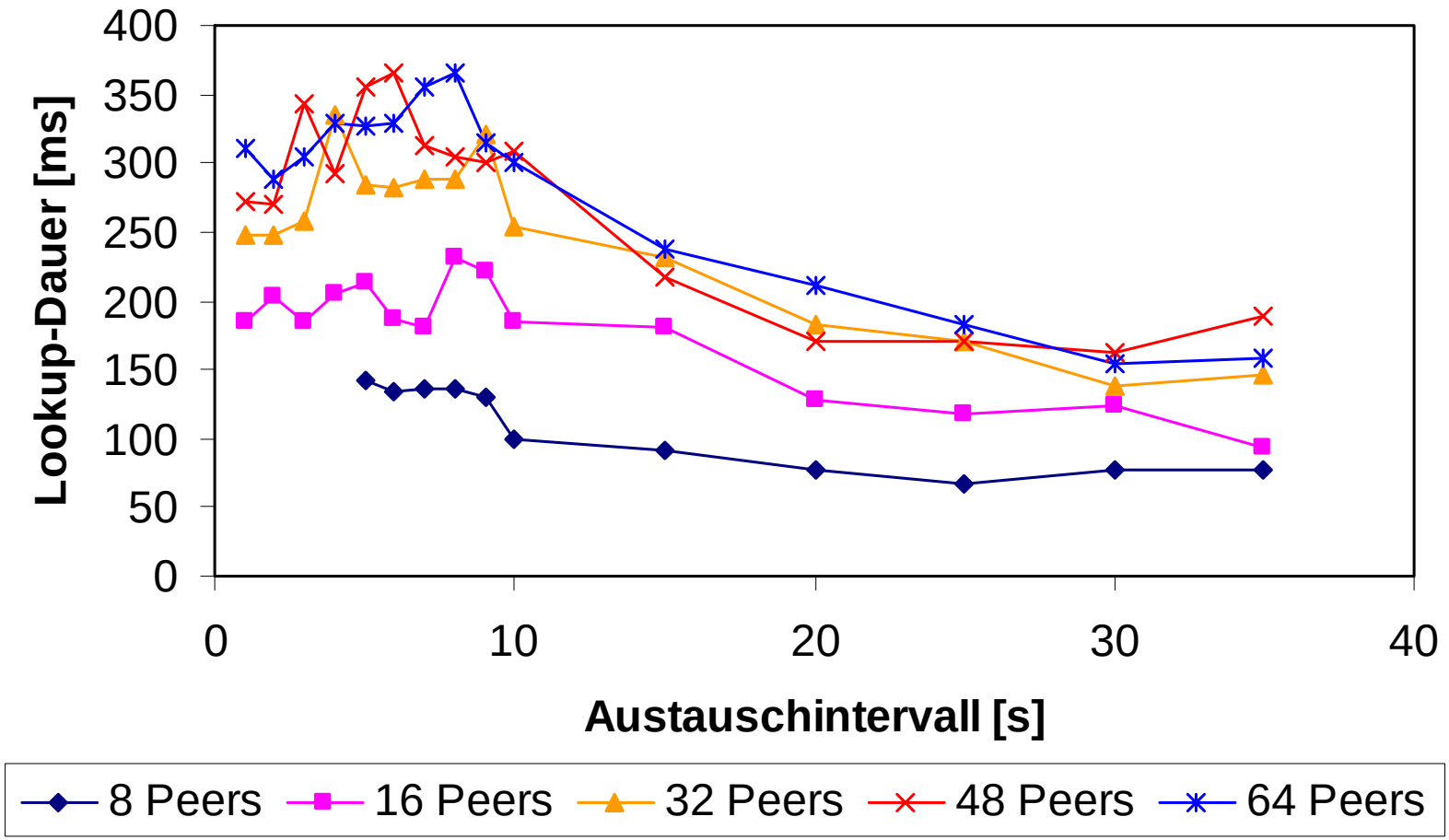
# Path Lengths



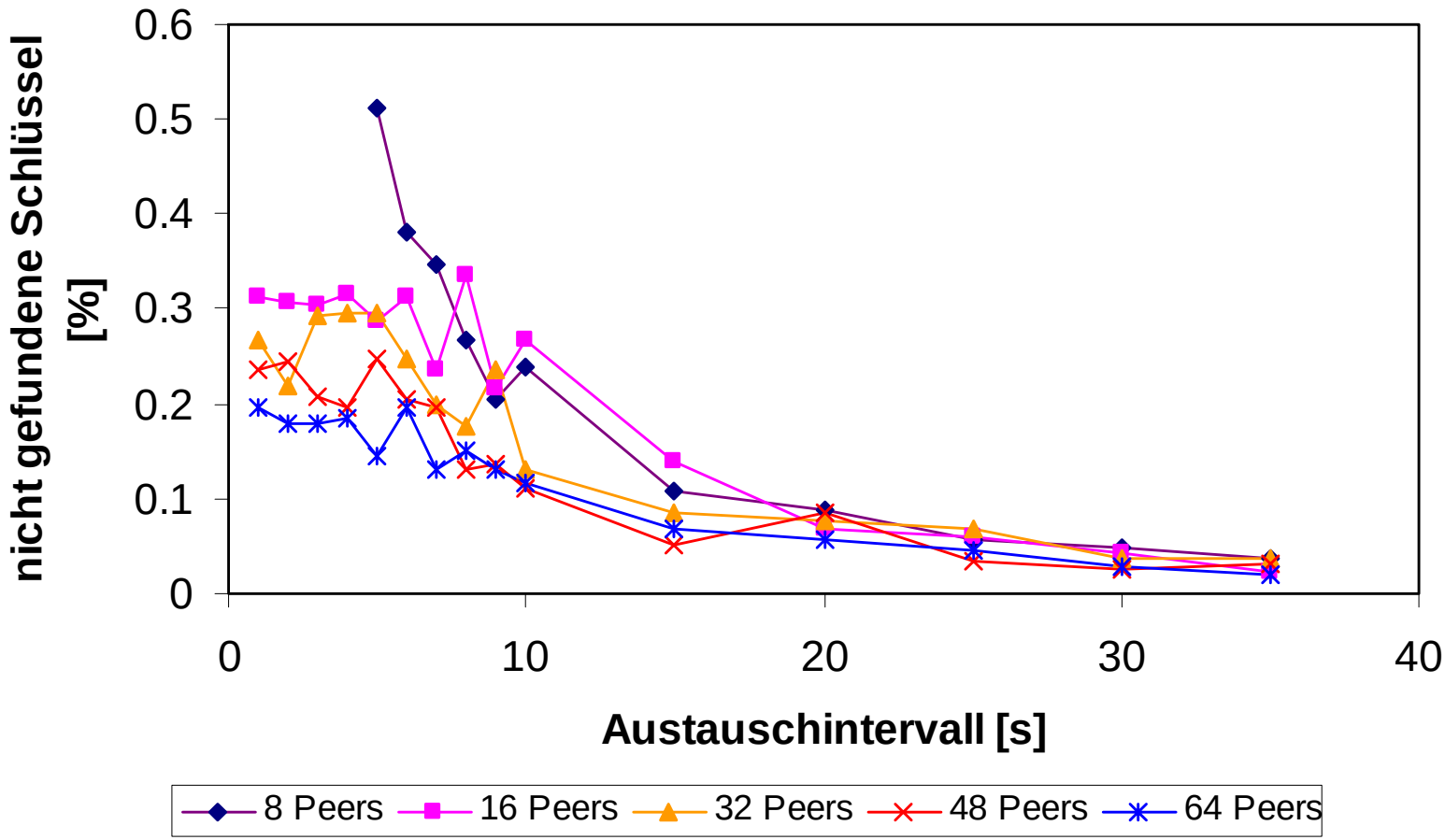
# Routing Messages



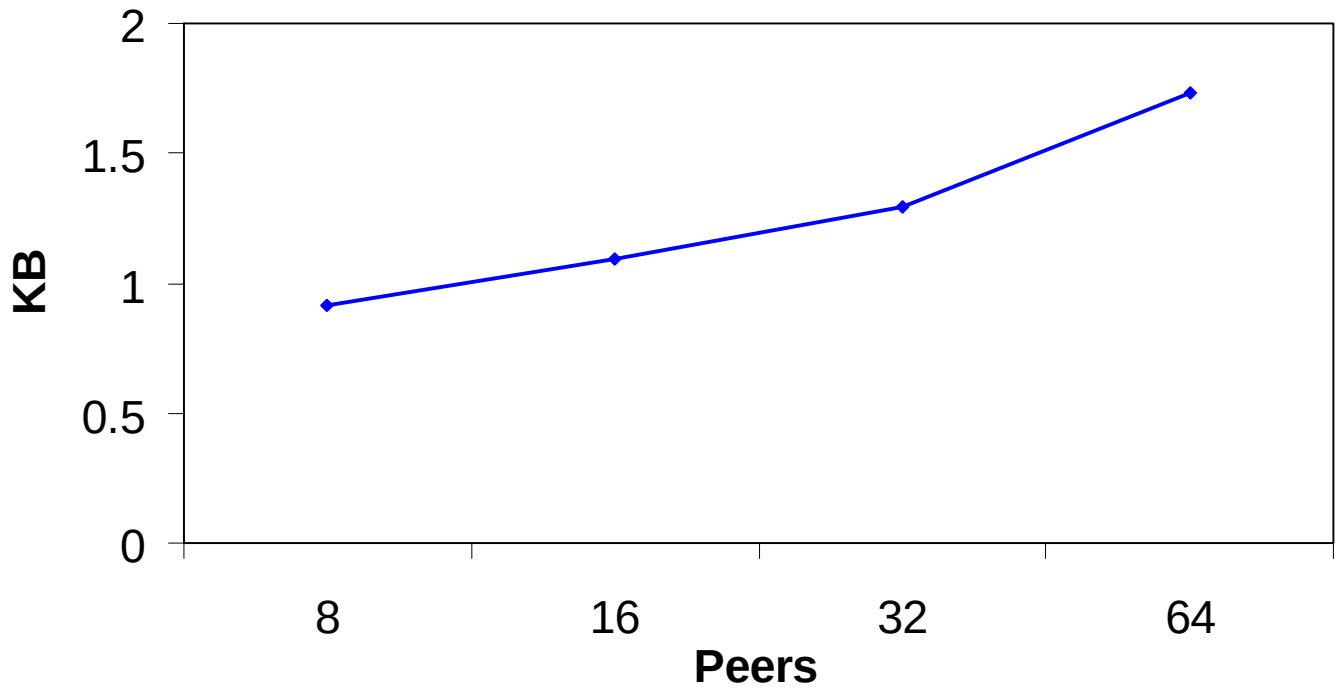
# Lookup Duration



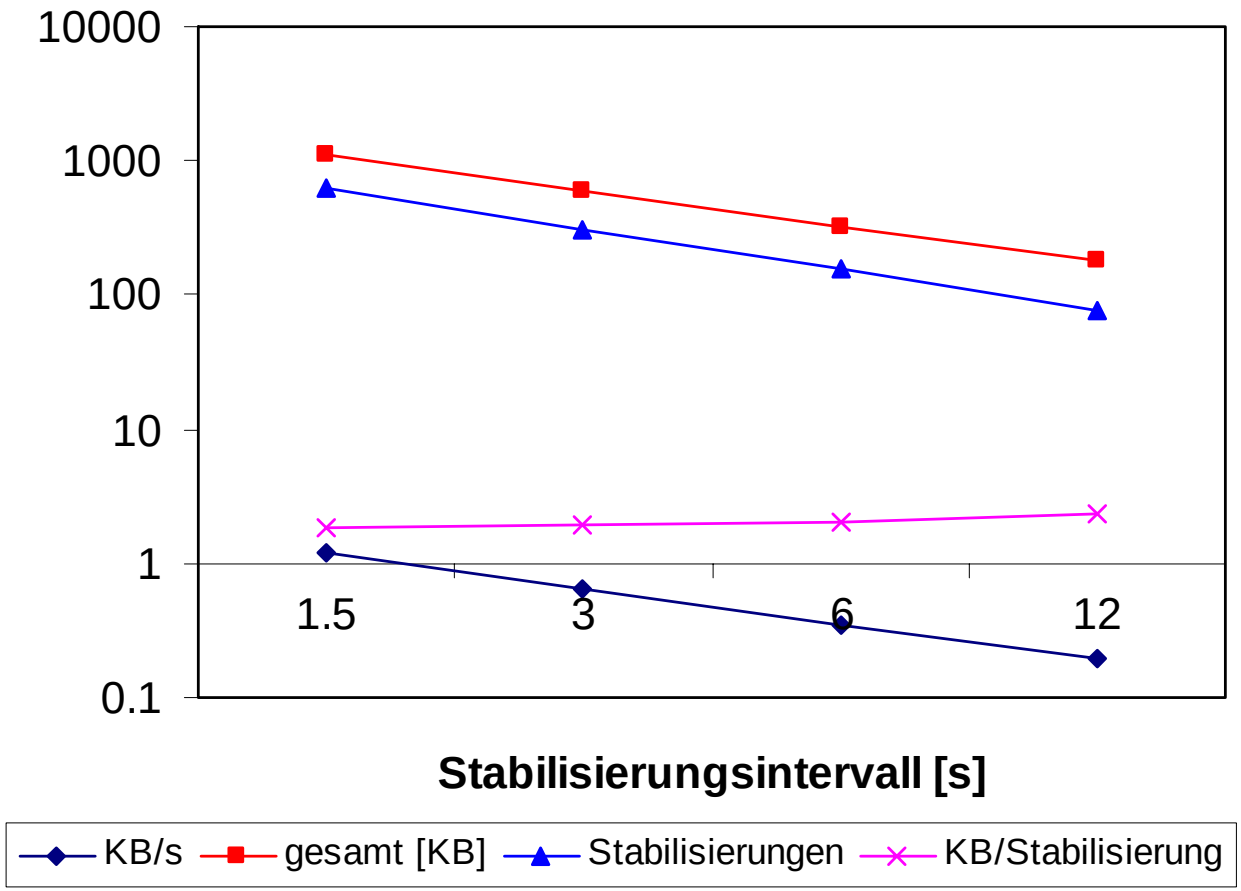
# Lookup Timeouts



# Data per Lookup



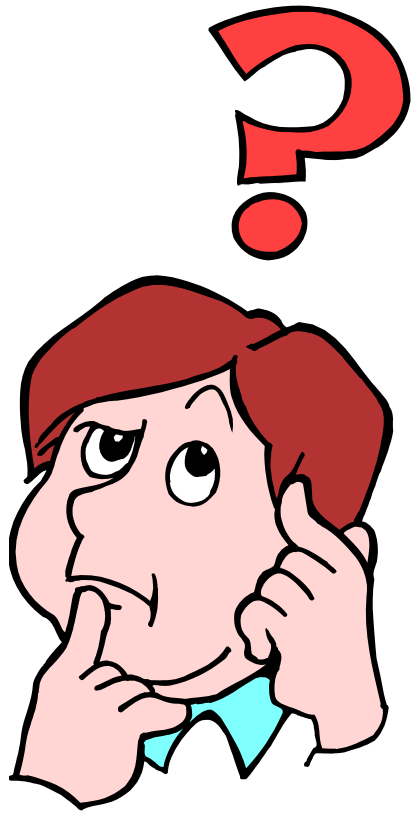
# Bandwidth Usage - Stabilization



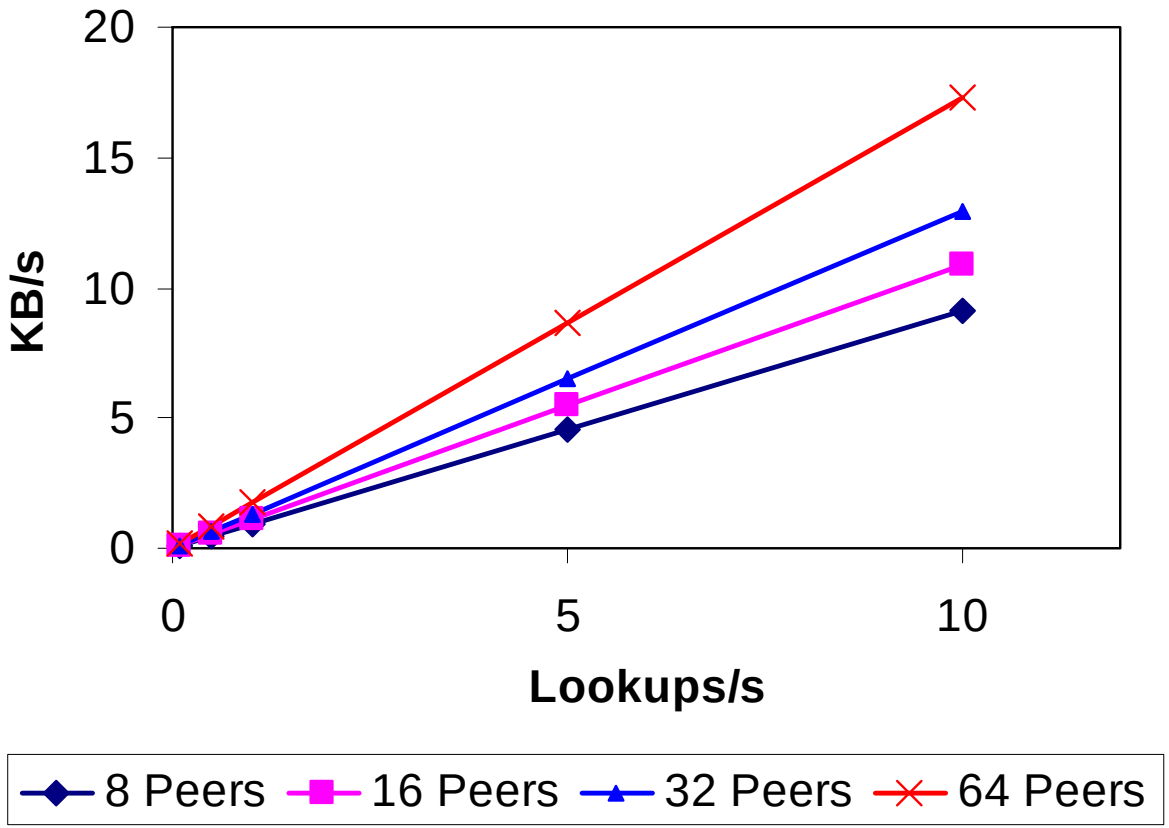
# Questions & Discussion

Thank you for attention!

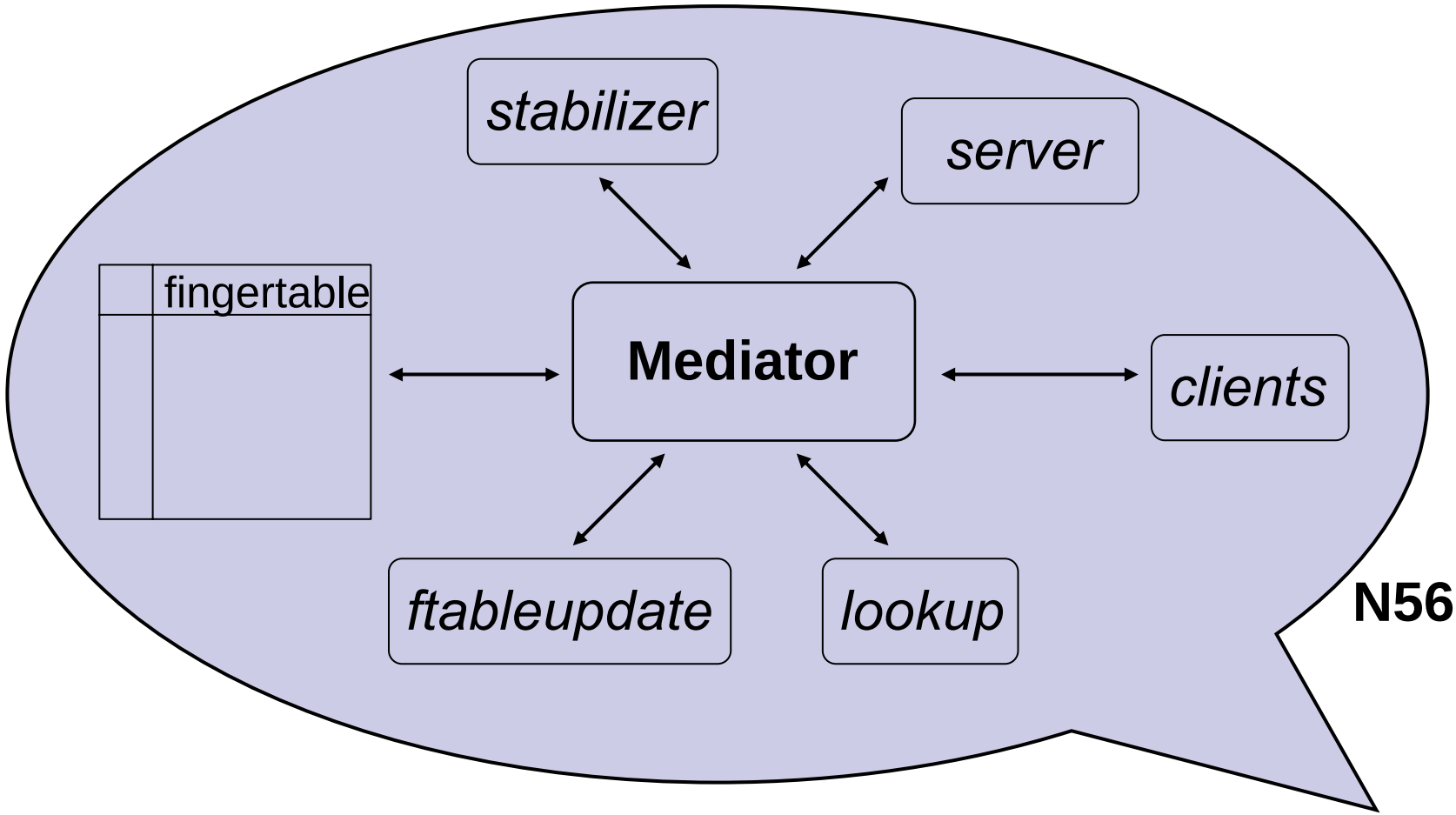
**Questions?**



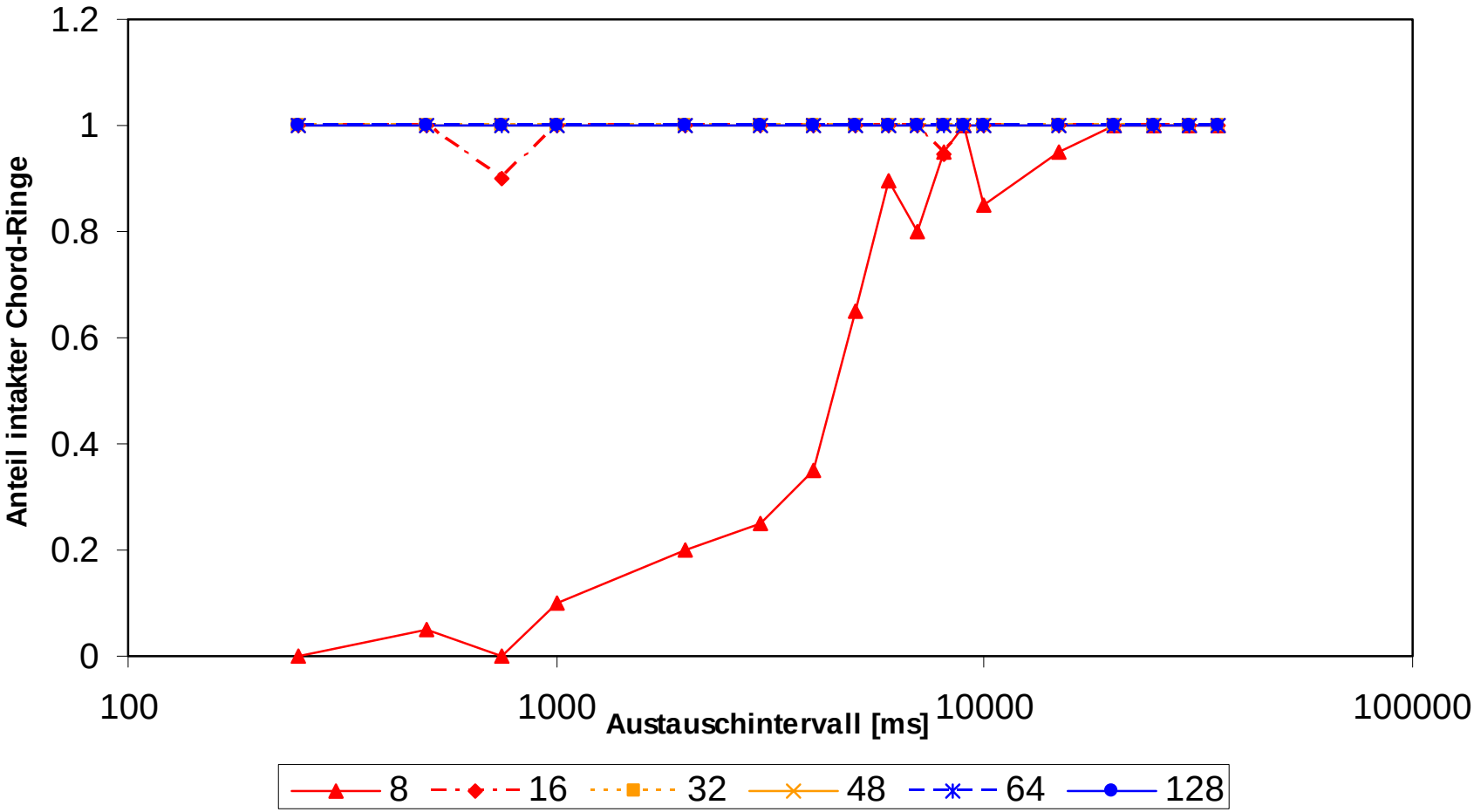
# Bandwidth Usage - Lookups



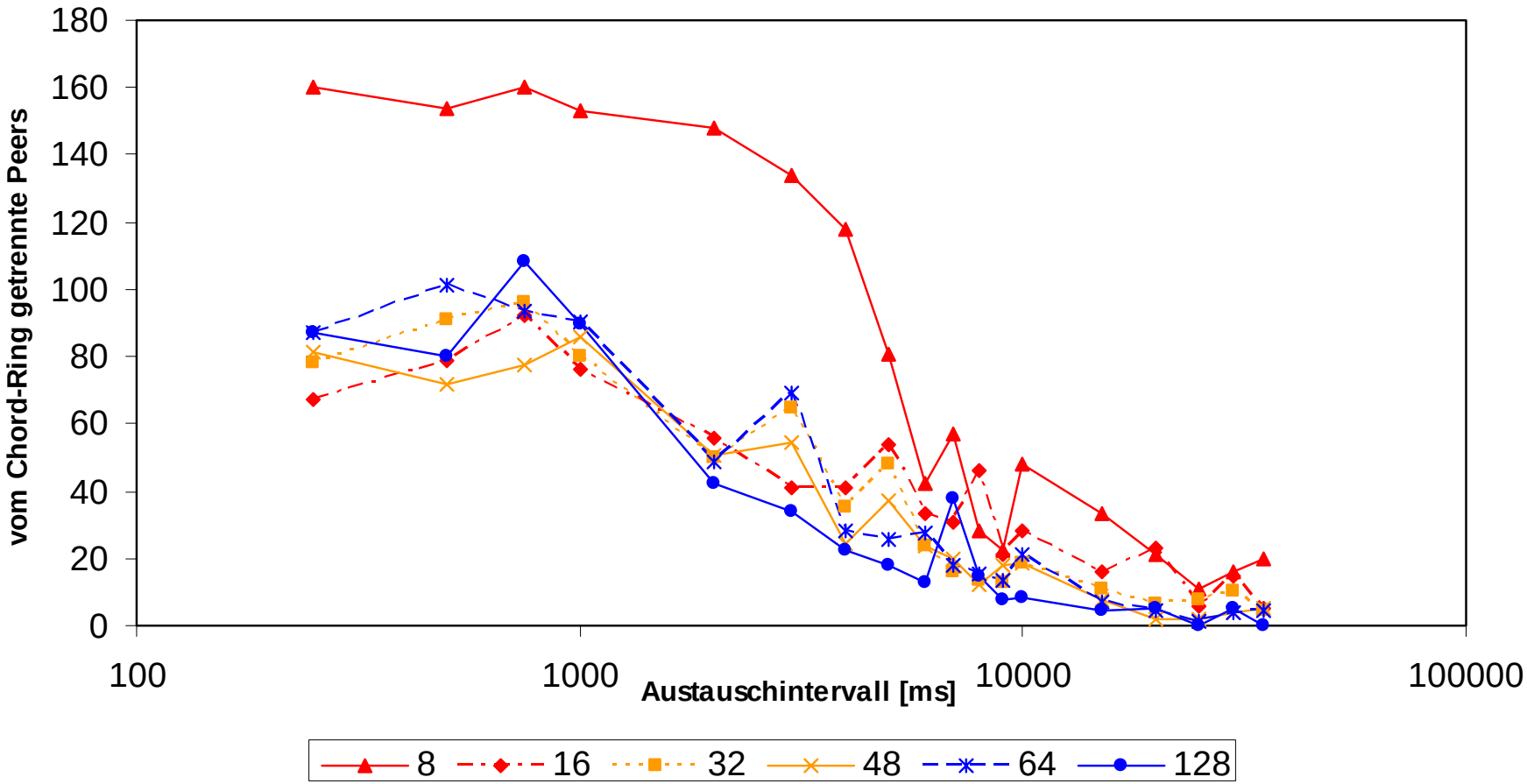
# Implementation Overview

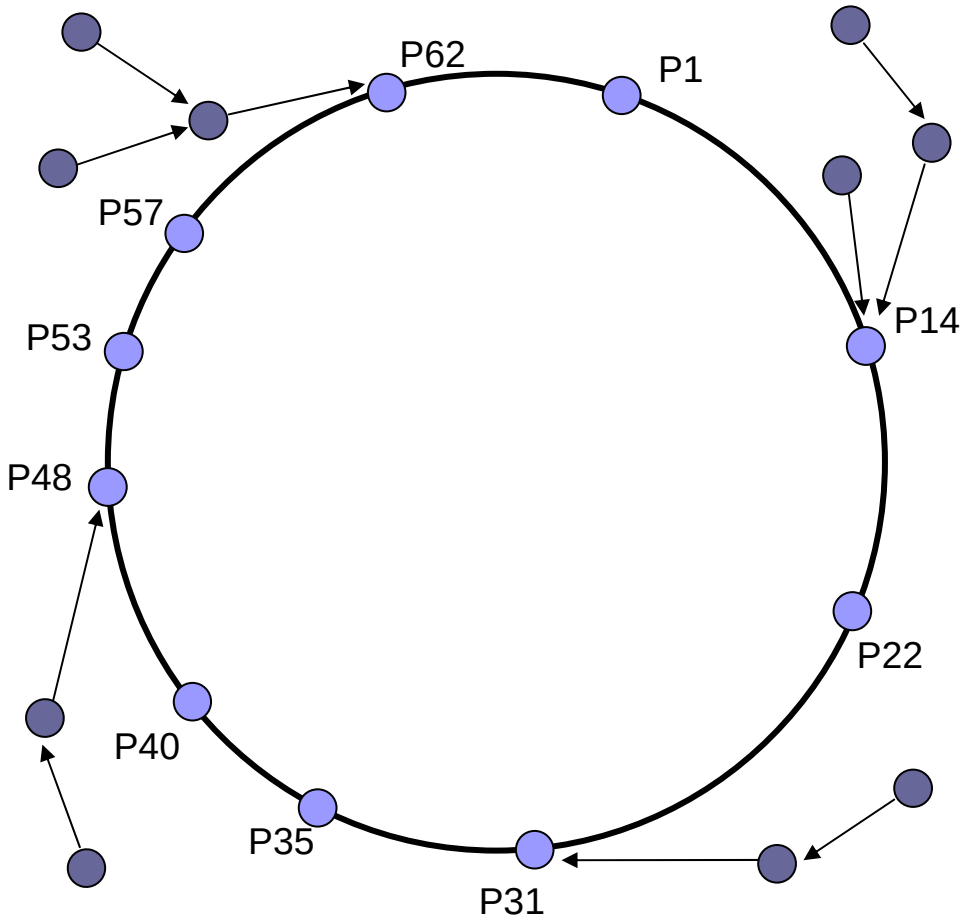


# Robustness


1.2
1
0.8
0.6
0.4
0.2
0
Anteil intakter Chord-Ringe
100
1000
10000
100000
Austauschintervall [ms]
8
16
32
48
64
128

# Robustness (2)




P62
P1
P57
P14
P53
P48
P22
P40
P35
P31

# The Chord algorithm – Node joins and stabilization

Stabilization protocol for a node *x*:

- x.stabilize():
  - ask successor *y* for its predecessor *p*
  - if $p \in (x; y]$ then *p* is *x*‘s new successor
- x.notify():
  - notify *x*‘s successor *p* of *x*‘s existence
  - notified node may change predecessor to *x*