Lab Class Protocol-Design

P2P-Overlay, Part III

P2P-Protokol, Version 0.1
Optimized Forwarding

- Flooding very inefficient
  - many more message copies than needed
  - additional overhead for detecting duplicates
  - unnecessary high network load
- Ways to optimize forwarding:
  - Layer 2 like (e.g., learning switches)
  - Layer 3 like (routing)

P2P-Protokol, Version 0.1
Optimized Forwarding

- Layer-2-like: similar to smart Ethernet switches
- For every message remember:
  - original sender of message
  - incoming link / neighbour
- Forwarding of messages using this table:
  - if we get messages sent by a node over a connection, then we can reach this node using this connection (at least for some time)!
- Table called '*Forwarding Table*'

P2P-Protokol, Version 0.1
Optimized Forwarding

- Automatically learns paths
- Problems:
  - Stale entries when nodes die
    -> use timeouts to remove/replace old entries
    -> refresh with new packets
  - What to do when learning other paths
    -> store TTLs, higher TTL means nearer

P2P-Protokol, Version 0.1
Optimized Forwarding

- Algorithm - Learning
  - extract Node-ID of originator (FROM) from message
  - Enter new / replace existing entry:
    - using originator Node-ID as key
    - replace if better TTL
    - store neighbour/connection, TTL and timestamp
  - If neighbour dies, remove all entries using this neighbour

P2P-Protokol, Version 0.1
Optimized Forwarding

- Algorithm - Forwarding
  - Update Forwarding Table (learning)!
  - lookup *destination* Node-ID in table (FOR)
  - if found
    - if (now - timestamp) < 120 sec // entry is up-to-date
      - forward over connection/neighbour found in table
    - else // entry too old
      - remove entry
      - flood
  - if not found
    - flood

P2P-Protokol, Version 0.1
Automated Session Setups

- Inconvenient to establish connections manually
- Solution:
  - use NEIGHBOUR info from HELLO-Handshake
  - automatically uphold 4 active connections

P2P-Protokol, Version 0.1
Automated Session Setups

- Send neighbour Node-IDs during HELLO-Handshake
- store received NEIGHBOUR list in queue (FIFO)
- After successful session setup:
  - Store Node-IDs learned during HELLO-Handshake in queue (no duplicates!)
  - while less than 4 active connections, connect to nodes from queue

P2P-Protokol, Version 0.1
Automated Session Setups

- What about failed connection attempts?
  - remove Node-ID from queue, try next one
- What to do if an active connections dies?
  - add Node-ID of neighbour to queue
- How to recognize if an *active* connection has died?
  - mark connections as being active

P2P-Protokol, Version 0.1
Data transfer

- Methods
  - via the P2P overlay
    - no new connections needed, i.e., firewall traversal is built-in by design
  - via direct client-to-client connections
    - more efficient, no load on peers
    - data not exposed to intermediate nodes

P2P-Protokol, Version 0.1
Direct client-to-client connections

- requester
  - send GET request to initiate a transfer
- responder
  - open a new socket for incoming TCP connection
  - reply with a DIRECTCONNECT message
    - specifying PORT to connect to
    - and file SIZE to expect

P2P-Protokol, Version 0.1
Direct client-to-client connections (2)

- requester
  - connects to specified PORT
  - receives file
  - checks if file is complete
- responder
  - sends file
  - closes connection

P2P-Protokol, Version 0.1

Direct client-to-client connections (3)

- message template: P2P/0.1 380 DIRECTCONNECT FOR <node id> FROM <node id> MESSAGE-ID <id> KEY <filename> SIZE <size> PORT <port> TTL <ttl>\r\n
- no protocol for file transfer itself – just send it
- make sure your client does not block during sending or receiving!
- what about PUT?

Questions?