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 extract 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 0.2**

- Protocol in version 0.1 too limited
  - cannot transport user data
    - => no downloads/uploads possible
  - doesn't support additional message parameters
  - doesn't support multiple applications (aka. port numbers)
P2P-Protokol, Version 0.2

- Solution: do it like HTTP :-)
  - Separate message header and body
    => allows user data transfer
  - allow multiline headers
    => allows additional parameters
    => can distinguish different applications,
      e.g., file transfer, routing protocol, ...
P2P-Protokol, Version 0.2

- Message format: Header
  - multiline
  - first line like version 0.1, but `P2P/0.2`
  - contains one or more option lines:
    - `<parameter> : <value> \r\n`
    - e.g. `Content-Length: 0`
  - header ends with empty line (`\r\n`)
**P2P-Protokol, Version 0.2**

- Message format: Body
  - up to 2048 bytes (2K) in size
  - may be empty

- Size of body as message option!
  - `Content-Length: 2048`

- If empty (= no body)
  - `Content-Length: 0`
Handshake Messages:
- needs empty line...
- but no parameters or body

Mandatory header parameters for non-handshake messages:
- Content-Length
- Application

For all non-handshake messages so far:
- No body: Content-Length: 0
- No Application: Application: none
**P2P-Protokol, Version 0.2**

- Implementation Issues
  - before reacting to messages, first need to read messages *completely*!
  - Must not intersperse message parts! First process and forward one message before looking at next one!
Reading messages (assumes non-blocking I/O)

check $sock->eof() and $sock->error()
# might take multiple entries into while() loop to read
# complete message -> keep per-Connection array to
# store header lines
while ($line = <$sock>) {
    # check header end
    if($line eq "\r\n") {
        # now have complete message header!!
        # lookup 'Content-Length: <size>' in @msg
        # read size bytes of body

        # NOW have complete message!!
        # process message in @msg
        # delete message
        @msg = ();
    } else {
        # append $line at end of @msg
    }
}