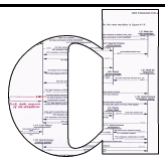
# DEPENDABLE SYSTEMS AND SOFTWARE

Data Networks 2015 Dr. Arnd Hartmanns



## PL3: WebSocket Chat Federation

The soft deadline for this programming lab is July 26. The hard deadline is August 31.

**Submission and group work:** The requirements for the submission and group work are as for PL2.

Your task is to extend your *dnChat* server to allow connections to and from other servers to create a *network* of servers. The clients (from PL1) shall observe no functional difference between being connected to (a) a single server with a set of directly connected users as in PL2 and (b) a server that is part of a network of servers with the same set of users connected to different servers. A server initiates a connection to another server when it reads a line from standard input starting with `connect`, followed by a space, the other server's host name or IPv4 address, and optionally another space followed by a port number. The communication *between servers* shall follow the principles of the *dnChat* protocol as in the previous labs, including the use of WebSockets and default port 42015, but shall use only the following messages:

`SRVR`
Only valid as the first message sent over a server-to-server connection by the server that initiated the connection. Must be sent by that server before sending any other messages. The number must be zero. (This message allows a server to tell whether a new connection is from a client or from another server.)

`ARRV`
As in the previous labs, but with the addition of a fourth line that contains the *hop count*, i.e. the minimum number of *other* servers on the path to the specified user. (The hop count is 0 for users on clients connected directly to this server, 1 for users on clients connected to a server with a direct connection to this one, and so on. When the hop count changes, a server may send new `ARRV` messages.)

`LEFT`
As in the previous labs. Indicates that the specified user is not reachable via this server.

`SEND`
Provides a chat message. The number refers to the chat message. The second line refers to the destination user. If it is `*`, the message shall be broadcast through the server network using controlled flooding. Otherwise, it shall be sent over a path with a minimum number of intermediate servers. The third line is the number of the sending user. The fourth line is the chat message text.

`ACKN`
As in server-initiated communication in the previous labs, but with an extra third line with the number of the user who sent the chat message that is now ack'ed. Shall be unicast like a `SEND` message (see above).

Servers do not send `OKAY` or `FAIL` messages to each other. When an invalid message (i.e. a malformed one, one with an invalid number, ...) is received, the server sends `INVD 0` and closes the connection.

*Hints and details:* Test your implementation with servers connected in cyclic and acyclic topologies of at least four servers. Check that broadcast and unicast messages as well as acknowledgments for these messages are delivered correctly, and that the list of connected users is always correct on all clients (modulo network delays). To achieve the latter, you need to handle the case where a server disconnects from another server without first sending `LEFT` messages. When this happens, in-transit (chat) messages can get lost, but you do not need to implement reliable data transfer mechanisms to handle this case—it is acceptable that the message just gets lost since chat messages are acknowledged end-to-end anyway. A well-behaved server, however, will send `LEFT` messages for all users it knows about and wait a second or two before disconnecting from another server. Your implementation shall be well-behaved. Think about what type of routing algorithm you have to implement, and what information you consequently have to store in addition to your current forwarding table mapping user numbers to connections. You can also assume in server-to-server communication that numbers are uniquely generated, such that, for example, there cannot be two chat messages with the same number in the network at the same time.