ulm university universität
uulm

VNS Chat Protocol

Echo Meißner, Benjamin Erb

Protocol Specification, Version 1.0

Networked Systems

This document describes a simple chat service protocol used for the lecture *Networked Systems*. The protocol allows multiple users to take part in a chat room situation and exchange messages instantly.

CONTENTS

1 INTRODUCTION

1.1 Purpose

The VNS Chat Protocol (VNSCP) is a simple application-level protocol for client/server-based message exchange between multiple users. It allows users to observe an ongoing chat application passively as well as actively join the chat.

1.2 Requirements

The key words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL** in this document are to be interpreted as described in RFC 2119.

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocols it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be *unconditionally compliant*; one that satisfies all the MUST level requirements but not all the SHOULD level requirements for its protocols is said to be *conditionally compliant.*

1.3 Terminology

This specification uses a number of terms to refer to the roles played by participants in, and objects of, the chat communication.

connection A transport layer virtual circuit established between two programs for the purpose of communication.

command connection A connection used for issuing commands by the user.

pub/sub connection A connection used for notifying the user of new events.

server An application program that accepts command and pub/sub connections in order to provide the chat service.

client An application program that consumes notifications from the pub/sub connection and/or issues commands using the command connection.

message The basic unit of VNSCP communication, consisting of a structured sequence of octets matching the syntax defined in a later section and transmitted via the the command oder pub/sub connection.

request A VNSCP message used by the client for the command connection.

response A VNSCP message used by the server for the command connection.

event A VNSCP message used by the server for the pub/sub connection for notifications.

chat message A text-based payload representing the messages exchanged between clients. A chat message is submitted by a client using a command connection and is then sent to all connected clients using the pub/sub connections. Hence, all chat messages are public for all users. More precisely, there are no private or direct messages.

id Every VNSCP event (i.e., join/leave event and chat messages) is assigned a monotonically increasing integer as a unique identifier. This identifier is included in the event when published via the pub/sub connections, as well as the response to the command that caused this event.

user A human person or a bot using a client and taking part in the chat server by sending commands to the server.

online A user is online when they have established a command connection to the server, has sucessfully executed the `LOGIN` command, has issued at least one command within a certain time span, and has not yet used the `BYE` command.

session As long as a user is online, the command connection represents a session. Once the online status is gone due to timeout or via explicit `BYE`, the sessions ends as well.

username A unique name identifying a user.

1.4 Overall Operation

The VNSCP protocol uses two separate connections for providing a chat service to its clients. The communication of boths connections takes place over TCP/IP connections. The command connection uses a client-initialized request/response protocol. Clients send requests to the server, which in turn sends back an appropriate repsonses, as known from the HTTP protocol. The pub/sub connection uses a publish/subscribe protocol. Clients connect to the service and

implicitly subscribe for notifications of new events. Once a new event is available, the server publishes the event by sending a notification message to the client.

2 MESSAGES

The VNSCP is a text-based protocol, which means that all messages MUST be regarded as UTF-8 encoded strings. Messages MUST NOT contain non-ASCII characters except for the payload field. Messages are structured as follows. The first line contains a single expression describing the type of message and the protocol name/version. The expressions SHOULD always be uppercase single words. Key-value pairs MUST NOT be expected to have a pre-defined order and unknown keys SHOULD be ignored. The following lines contain zero or more message-dependant key-value pairs (also known as fields), each separated by a colon and a whitespace. The number of fields depends on the type of message. In the fashion of HTTP, message lines are terminated by `\r\n`, and an entire message is terminated using an additional empty line. Unlike HTTP, VNSCP message do not have an entity or body.

All VNSCP applications MUST be able to handle all types of messages of a distinct connection (command or pub/sub). An application supporting both connections MUST understand all messages. An application MUST understand all mandatory message fields. It SHOULD also accept and ignore non-default fields that are not shown in this specification. Furthermore, an application SHOULD tear down the connection when it does not recognize the protocol name or protocol version.

2.1 Command Connection Messages

The command connection provides request and response messages. Request messages are always sent by the client and each request is always answered with single response message from the server.

2.1.1 Request Message Structure

```
1 SEND VNSCP/1.0          command expression & protocol/version identifier
2 Text: hi all!           A field with the key "Text" and content "hi all!"
3                         Empty line terminates the message
```

Listing 2.1: A sample request message (here: `SEND` request).

2.1.2 List of Request Messages

The following request messages sent by the client MUST be understood for command connections.

`LOGIN` This request initiates a new session for the user. The server either accepts the login with `LOGGEDIN` or it rejects the login attempt with an `ERROR` message (i.e. when the username is already taken). In the latter case, the client MAY retry the login.

`SEND` A `SEND` message submits a new chat message originating the current user session. When accepted, the server returns a `SENT` response. Otherwise, the server sends an `ERROR` message with the reason. Errors can be triggered by too long message, too many message or invalid messages. Note that the server can respond with a `EXPIRED` message, when the user session timed out.

`PING` The use of the `PING` is twofold: On the one hand, this message indicates that the client is still active and prevents timeouts even if the user has not sent new message in the past. On the other hand, this message triggers a `PONG` response which lists all users currently logged in to the server. Note that the server can also respond with a `EXPIRED` message, when the user session timed out.

`BYE` This request ends the current user session. Once acknowledged, both end points can shut down the underlying connection. Note that the server can respond with a `EXPIRED` message, when the user session timed out. In this case, the client can simply tear down the connection, as the session has already ended then.

2.1.3 Response Message Structure

```
1 VNSCP/1.0 PONG                protocol/version identifier & command expression
2 Date: 2023-12-23 15:32:12     A field with "Date" key and current timestamp as value
3 Usernames: foo,bar,bla        A field with "Users" key and comma separated user list
4                               Empty line terminates the message
```

Listing 2.2: A sample response message (here: response to `PING`).

2.1.4 List of Response Messages

The following response messages sent by the server MUST be understood for command connections.

`LOGGEDIN` This message confirms a successful log-in and acknowledges the new session. The server MUST now recognize the current user as logged in until the a timeout occurs or the user logs out using BYE.

`SENT` This message confirms the submission of a new message. The server MUST multiplex the message text to all available pub/sub connections using `MESSAGE`. The server MUST also reset the counter for this client.

`PONG` A pong message verifies that user timer for her time out has been reset. Additionally, the pong messages lists all users currently logged in to the chat room. The server MUST reset the client's counter.

`EXPIRED` This message indicates that the previous request could not be executed because the user session timed out. The client must re-establish a new session first by logging in again.

`ERROR` This message indicates that the previous request could not be fulfilled due to errors such as invalid sesison states or invalid messages. The `Reason` field always contains a textual description of the error.

`BYEBYE` This message responds to a BYE request. Now the endpoints can tear down the connection. Also, the user session MUST be invalidated.

2.2 Pub/Sub Connection Messages

The pub/sub connection is a one-way publish protocol that only allows servers to send notification messages to clients.

2.2.1 Pub/Sub Message Structure

The message structure of pub/sub messages is analogous to that of response messages of the command protocol.

```
1 VNSCP/1.0 MESSAGE              protocol/version identifier & command expression
2 Id: 121                        A unique monotonically incremented integer as an identifier for this event
3 Date: 2023-12-23 18:32:53      "Date" field with current server time as value
4 Username: tom21                A field with "username" key and name as value
5 Text: hi all!                  A field with the key "Text" and content "hi all!"
6                                Empty line terminates the message
```

Listing 2.3: A sample event message (here: message notification).

2.2.2 List of Event Messages

The following messages MUST be understood by the client when connected to the pub/sub connection.

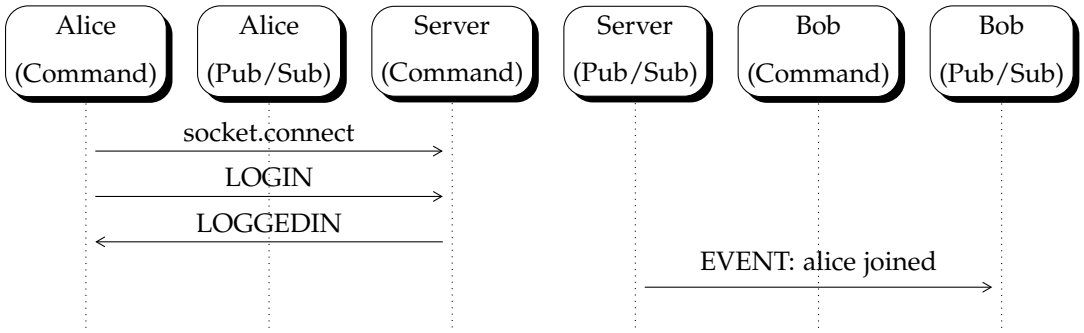
`EVENT` An event message is dispatched for any non-message event that occured. This includes join and leave notifications for users.

`MESSAGE` A VNSCP message of type `MESSAGE` is dispatched whenever a chat participant has sent a new message to the chat room.

3 MESSAGE FLOWS

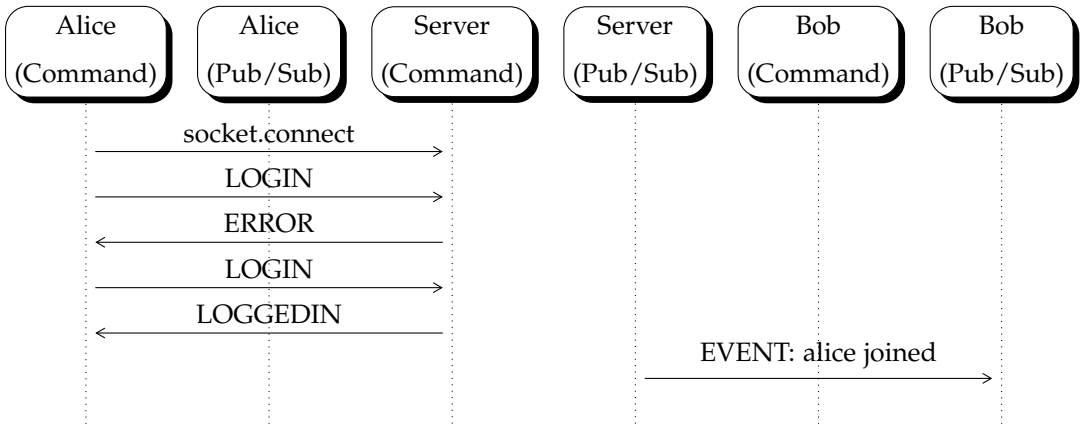
3.1 LOGIN

Alice logs into the chat room. All subscribers receive a notification of the new user. Note that she should also to connect to the pub/sub socket in order to receive chat notifications.



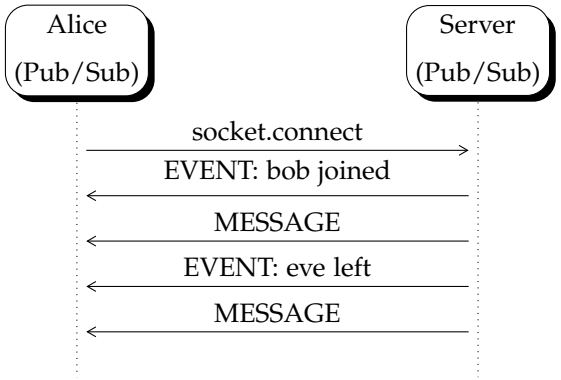
3.2 LOGIN Failure

Alice tries to login to the chat room. However, her username is already in use and she gets an `ERROR` response. After changing the name, she retries and evenutally succeeds to login.



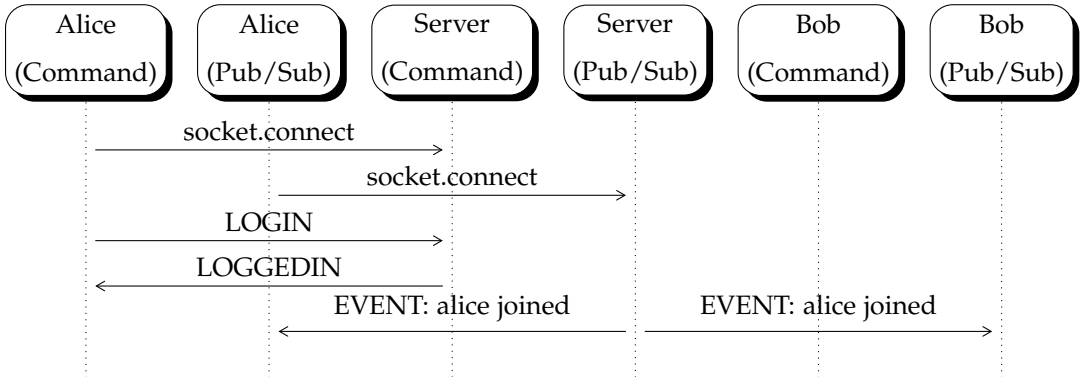
3.3 Subscribing to Events

Alice connects to the pub/sub socket. Thus, she subscribes for notifications immediately. Note that this step is independent of any user session or login procedure.



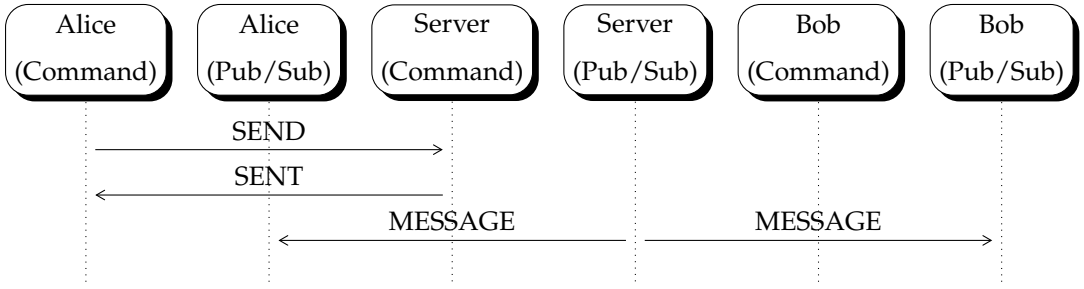
3.4 Full LOGIN

Alice connects both to the pub/sub and the comnand socket. After successfull establishing a session, she gets notified about her own join event.



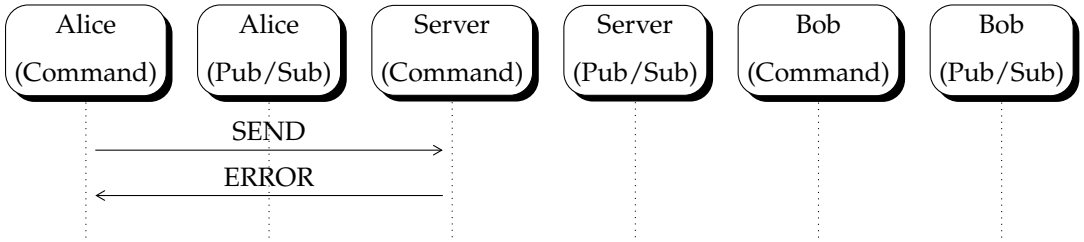
3.5 SENDing a Message

Alice successfully sends a message to the chat room. All subscribers get notified.



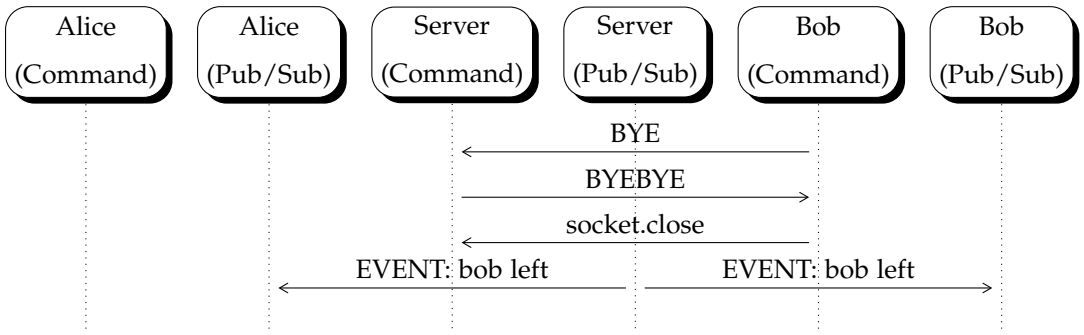
3.6 Failing to SEND a Message

Alice successfully sends a message to the chat room. The message gets rejected due to invalid content.



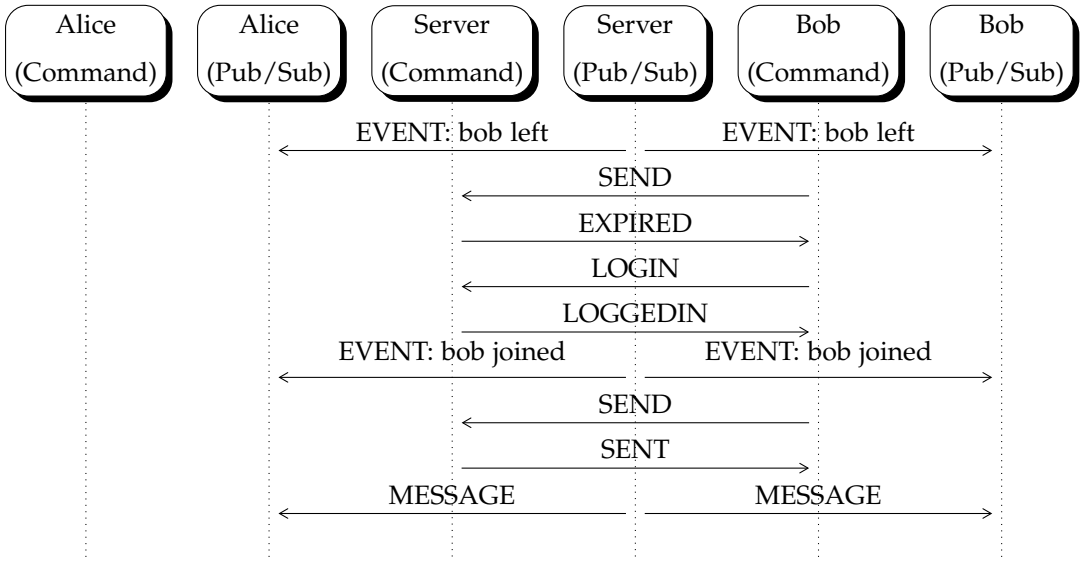
3.7 LOGOUT

Bob ends his session by sending a `BYE` message. Bob must also independently tear down the pub/sub connection before or after the `LOGOUT` command by closing the socket.



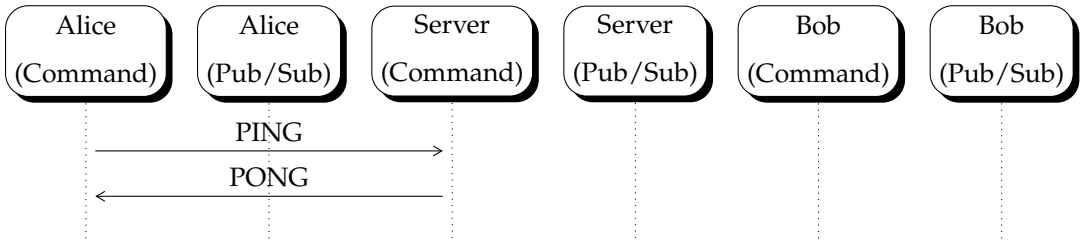
3.8 Unexpected Timeout and Re-LOGIN

In this scenario, Bob's session expires and his client does not detect it. Hence, when he sends the next message, he has to handle the `EXPIRED` response first and re-login, before being able to resend the original message submission.



3.9 PING-PONG

Alice refreshes her status by sending a `PING` request. She also receives a list of all users currently online.



4 Example Message Exchanges

4.1 Login Procedure

Alice has connected to the command socket of the server and triggers the login procedure. The login is successful.

```
1  LOGIN VNSCP/1.0              –> from ALICE to SERVER
2  Username: alice23            –>
3                               –>
4  VNSCP/1.0 LOGGEDIN           <– from SERVER to ALICE
5  Id: 123                      <–
6  Date: 2023-12-23 18:29:12    <–
7                               <–
```

This time, the login fails and Alice needs to retry, because her initial username is already in use.

```
1  LOGIN VNSCP/1.0              –> from ALICE to SERVER
2  Username: alice23            –>
3                               –>
4  VNSCP/1.0 ERROR              <– from SERVER to ALICE
5  Date: 2023-12-23 18:29:12    <–
6  Reason: The selected username is already in use.
7                               <–
8  LOGIN VNSCP/1.0              –> from ALICE to SERVER
9  Username: alice23            –>
10                              –>
11 VNSCP/1.0 LOGGEDIN           <– from SERVER to ALICE
12 Id: 123                      <–
13 Date: 2023-12-23 18:30:01    <–
14                              <–
```

Once a user has connected, a new message is dispatched to all subscribers of pub/sub connections:

```
1  VNSCP/1.0 EVENT              <– from SERVER to subscriber(s)
2  Id: 123                      <–
3  Date: 2023-12-23 18:32:53    <–
4  Description: alice23 has joined
5                               <–
```

4.2 Messaging and Errors

After login, Alice sends a message to the chat room.

```
1 SEND VNSCP/1.0                -> from ALICE to SERVER
2 Text: hi guys!                ->
3                               ->
4 VNSCP/1.0 SENT                <- from SERVER to ALICE
5 Id: 124                       <-
6 Date: 2023-12-23 18:32:53     <-
7                               <-
```

This additionally causes a new message on all pub/sub connections:

```
1 VNSCP/1.0 MESSAGE             <- from SERVER to subscriber(s)
2 Id: 124                       <-
3 Date: 2023-12-23 18:32:53     <-
4 Username: alice23             <-
5 Text: hi guys!                <-
6                               <-
```

Invalid messages will be discarded and also not forwarded to the chat room:

```
1 SEND VNSCP/1.0                -> from ALICE to SERVER
2 Text: heeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeyyyy  guys!
3                               ->
4 VNSCP/1.0 ERROR               <- from SERVER to ALICE
5 Date: 2023-12-23 18:29:12     <-
6 Reason: Message too long.     <-
7                               <-
```

4.3 Logout

Alice is online and wants to logout. She sends a BYE request successfully and closes the socket afterwards.

```
1 BYE VNSCP/1.0                 -> from ALICE to SERVER
2                               ->
3 VNSCP/1.0 BYEBYE              <- from SERVER to ALICE
4 Id: 125                       <-
5 Date: 2023-12-23 19:02:49     <-
6                               <-
```

All subscribing users will be notified using the pub/sub connections:

```
1 VNSCP/1.0 EVENT               <- from SERVER to subscriber(s)
2 Id: 125                       <-
```

```
3  Date: 2023-12-23 19:02:49    <-
4  Description: alice23 has left
5                               <-
```

4.4 Ping/Pong and Timeouts

Alice issues a PING request, either because she wants to prevent a timeout, or because she wants to receive a list of all active users.

```
1  PING VNSCP/1.0               -> from ALICE to SERVER
2                               ->
3  VNSCP/1.0 PONG               <- from SERVER to ALICE
4  Date: 2023-12-23 19:02:49    <-
5  Usernames: alice23,bob16,eve24
6                               <-
```

In case Alice does not adhere to the timeout limits, her session will be disabled by the server after a certain amount of time. When she tries to execute session-bound commands (i.e. `PING`, `SEND` and `BYE`, she will receive an `EXPIRED` response.

```
1  SEND VNSCP/1.0               -> from ALICE to SERVER
2  Text: are you gone?          ->
3                               ->
4  VNSCP/1.0 EXPIRED            <- from SERVER to ALICE
5  Date: 2023-12-23 19:12:31    <-
6                               <-
```

The timeout triggers a "has left" `EVENT`, similar to `BYE` commands. When Alice wants to keep chatting, she needs to redo the `LOGIN` first:

```
1  SEND VNSCP/1.0               -> from ALICE to SERVER
2  Text: are you gone?          ->
3                               ->
4  VNSCP/1.0 EXPIRED            <- from SERVER to ALICE
5  Date: 2023-12-23 19:12:31    <-
6                               <-
7  LOGIN VNSCP/1.0              -> from ALICE to SERVER
8  Username: alice23            ->
9                               ->
10 VNSCP/1.0 LOGGEDIN           <- from SERVER to ALICE
11 Id: 129                      <-
12 Date: 2023-12-23 19:12:49    <-
13                              <-
14 PING VNSCP/1.0               -> from ALICE to SERVER
15                              ->
```

```
16 VNSCP/1.0 PONG                <– from SERVER to ALICE
17 Date: 2023-12-23 19:13:02     <–
18 Usernames: alice23,bob16,eve24
19                               <–
```

4.5 Invalid Requests and Wrong Protocol

If the server receives a request it does not support it should respond with an error to inform the client about the unsupported request.

```
1 WRITE VNSCP/1.0                –> from ALICE to SERVER
2 Text: hello world              –>
3                                –>
4 VNSCP/1.0 ERROR                <– from SERVER to ALICE
5 Date: 2023-12-23 18:39:15      <–
6 Reason: Invalid message format or version.
7                                <–
```

The same applies if the client uses a different protocol or a different version of the VNSCP protocol

```
1 SEND VNSCP/2.0                 –> from ALICE to SERVER
2 Text: hello world              –>
3                                –>
4 VNSCP/1.0 ERROR                <– from SERVER to ALICE
5 Date: 2023-12-23 18:39:15      <–
6 Reason: Invalid message format or version.
7                                <–
```

5 PROTOCOL DETAILS

5.1 Chat Message and Username Format

5.1.1 Chat Messages

A chat message MUST be a non-empty, UTF-8 encoded string, terminated by a line break. Therefore, a chat message MUST NOT contain any line breaks (\r, \n, \r\n) itself. The size of a chat message MUST not exceed 512 bytes. It is OPTIONAL that larger chat messages are supported by clients by transparently splitting up the original text message into multiple smaller chat messages.

5.1.2 Usernames

A username MUST only contain alphanumerical chars (a-z, A-Z, 0-9), MUST have a minimal length of 3 chars and a maximum length of 15 chars. It MUST NOT contain any special characters and non-ASCII characters.

5.2 Timeouts

Servers MUST manage timeouts for each client by denoting the time of inacticity of its session. Once a client submits a `SEND` or `PING` request, the timer is reset. If a timer causes a timeout, the server MUST disable the session and answer subsequent requests with `EXPIRED`, as long as the client has not re-established a new session via `LOGIN`.

Clients SHOULD generally prevent timeouts by using periodic `PING` requests or appropriate timer managements.

As of this writing, the timeout lease for the VNSCP protocol is set to **10 minutes** of inacticity.

5.3 Connection Handling

The VNSCP uses persistent connections for both command and pub/sub connections. Once established, a socket can be used for multiple request/response cycles resp. multiple dispatched notifications. The command connection MUST be explicitly closed after using the BYE command,

or it SHALL be closed when not yet online. The pub/sub connection of a client SHOULD be closed once the client is about to quit.

Clients MUST NOT use pipelining. That is, a new request MUST NOT issued unless the previous request has been answered by the server. In case of faulty requests, servers MAY shut down the connection after sending the `ERROR` response. Clients MUST expect that the connection can go down for this reason at any time.

6 HELPFUL STATE MACHINES

6.1 Command Connection: Client (simplified)

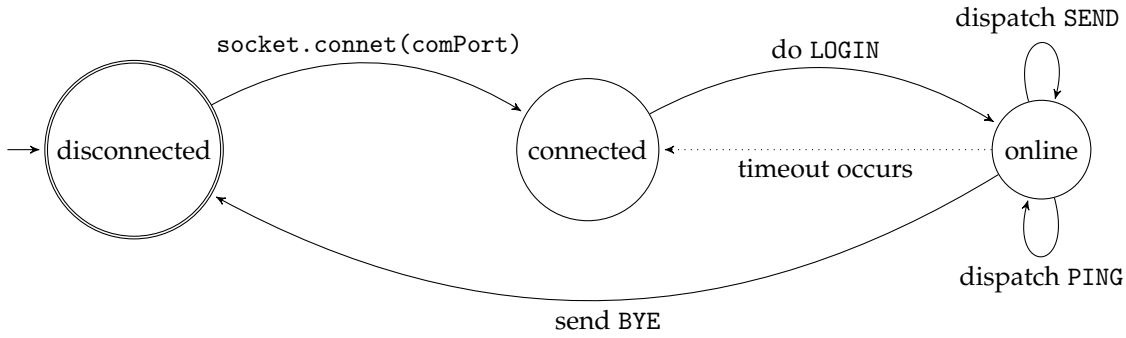


Figure 6.1: A simplified automaton for the command connection of a client.

6.2 Pub/Sub Connection: Client (simplified)

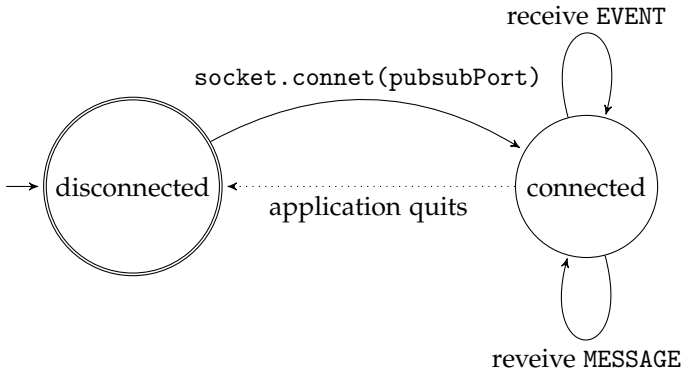


Figure 6.2: A simplified automaton for the pub/sub connection of a client.