JOZABAD

For version 0.1, 27 September 2013

Michael L. Gran

This manual is for the Jozabad project, (version 0.1, 27 September 2013). It provides a lightweight server for chat-like protocols, and a client library.

Short Contents

Table of Contents

1 The Switched Virtual Circuit Protocol

1.1 Introduction

Jozabad is a project that consists of a custom hub-and-spoke communication protocol called the Switched Virtual Circuit protocol, or "SVC", along with a broker and a client library. Workers connect to a broker and send messages to the broker. The broker joins pairs of workers together in channels and forwards messages between connected workers. Jozabad is thus, basically, like a telephone exchange. It is inspired by the ITU's X.25 protocol, but, is simplified.

The basic goals of the protocol are these:

The broker ensures that

- messages are valid
- data rates are obeyed

A worker can

- connect / disconnect to the broker at will
- get a list of other workers from the broker
- request a channel with any other idle worker

The channel

- is exclusive – a worker can have only one channel at a time
- is persistent – it exists as long as the workers want
- is stateful
- is pseudonymous – the workers only know one-anothers' pseudonym

The future goals of the protocol are these:

- The protocol will use future ZeroMQ security protocols to authenticate workers and encrypt traffic.

The name "Jozabad" refers to a talented archer that fought with the biblical King David. The image the archer is a metaphor for hub-to-node information transmission.

Jozabad is the lowest level of a three layer application stack.

- *Jozabad* handles low-level communication.
- *Johanan* is a packet assembler and disassembler, automatically sending and receiving data from I/O buffers when required.
- *Jahaziel* is an interpretation of the Videotex text protocol with in-band escapes for colors, graphics, and audio. It also has a sample rendering engine.

The Switched Virtual Channel Protocol had two roles: *workers* and a *broker*.

The *workers* are processes that need one-on-one communication channel with other workers. They could be clients, servers, or a combination of the two. The SVC doesn't specify.

The *broker* is a middle-man whose purpose is to protect the anonymity of its workers and to ensure that data rate limits are not exceeded.

2 Worker to broker interface

ZeroMQ is used as the transport mechanism for this protocol. Each worker has a single Dealer socket which connects to a broker's Router socket using a TCP transport. The default port is 516, which the IANA has reserved for Videotex.

Each packet transferred from worker to broker must be in a set of defined packet formats, described herein.

This chapter describes the interactions that a worker may have with a broker. For information on interactions that workers may have with one-another, see Chapter 3 [Worker to worker interface], page 5. Formats are described specifically in [Packet formats], page 2.

Nominally, a worker will connect with the broker with `connect` request, wait for a `connect indication` response, request a list of other workers with a `directory request` message, receive that directory in a `directory` message, possibly communicate with another worker as described in Chapter 3 [Worker to worker interface], page 5, and then disconnect with a `disconnect` message.

See ⟨undefined⟩ [Broker actions], page ⟨undefined⟩ for the actions taken by the broker on receipt of packets.

2.1 The worker registry

In this protocol, all communcation is between the worker and the broker. Before the worker starts sending requests to the broker, it first must properly register itself with the broker.

The broker keeps an internal list of all the workers with which it is currently interacting. Packets received from workers that are not in the registry are ignored by the broker.

2.2 Basic packet structure

Every packet transferred across the worker / broker interface consists of a ZeroMQ message frame that has at least three octets. There is a two octet signature and a single octet message identifier. Other fields and data frames are appended as required (see [Packet formats], page 2).

Packet types and their basic use are given in Table 2.1.

Packet Type	W to B	B to W	W to W
Connect request	X		
Connect indication		X	
Disconnect	X		
Disconnect indication		X	
Directory request	X		
Directory		X	
Data			X
RR (Receive Ready)			X
RNR (Receive Not Ready)			X
Clear request		X	X
Clear confirmation	X		X
Reset request		X	X
Reset confirmation	X		X

Table 2.1: This is a list of packet types and their use in various services. 'W to B' indicates messages sent from workers to the broker. 'B to W' is broker to worker. 'W to W' is messages sent from workers to the broker to be forwarded on to another worker.

2.3 Procedure for connection

The connection procedure is used to register a worker with the broker so that it will not ignore messages from that worker.

A worker, after connecting its ZeroMQ dealer socket to a broker's router socket, requests a connection by sending a `connect` message. In the message, the worker gives its desired pseudonym and its *directionality* – whether it intends to make calls, receive calls, or both.

The broker will respond with either a `connect indication` message, if the connection request is accepted, or with a `diagnostic` message, if the connection request is rejected.

The maximum time a worker should have to wait for a response from the broker should not exceed time-limit T20 (see ⟨undefined⟩ [Time limits], page ⟨undefined⟩).

2.4 Error handling

⟨undefined⟩ [table:errorhandling], page ⟨undefined⟩ specifies the reaction of the broker when special error conditions are encountered. Every message that a worker sends may be rejected by the broker. In that case, the broker will send a `diagnostic` message back to the worker. The `diagnostic` message contains information about the reason for which a message was rejected.

2.5 Directory requests

Once connected, a worker will probably wish to know what other workers are currently connected. For this, it sends a `directory request` request to the broker.

The broker will respond with a `directory` message which contains a list of all the currently connected workers.

The `directory` message can be quite large, so it may not be requested more frequently than time-limit T2 (see ⟨undefined⟩ [Time limits], page ⟨undefined⟩) under penalty of disconnection.

The maximum time a worker should have to wait for a response from the broker should not exceed time-limit T21 (see ⟨undefined⟩ [Time limits], page ⟨undefined⟩).

2.6 Disconnection requests

2.6.1 Disconnection requests from the worker

Well behaved workers will send `disconnect` requests before they disconnect, so that they broker knows that it should remove them from the directory.

The broker will respond with a `disconnect indication` message, but, the worker is under no obligation to wait for the message and may ignore it.

2.6.2 Disconnection by the broker

The broker may send a `disconnect indication` message to a worker at any time. This may occur if the broker is shutting down, or if the broker is punishing the worker for using too much bandwidth or for being idle for too long.

The worker need not respond. Its responses would be ignored anyway.

3 Worker to worker interface

Once a worker is connected to the broker, it may set up a two-way communication channel with another worker, aka a *logical channel*. Nominally a worker will send a `call request` message to establish a logical channel, wait for a `call accepted` response, send and receive `data` messages, controlling flow using the `RR` (receive ready) messages, and then shut down the logical channel using a `clear request`.

All message traffic passes through the broker, which is necessary to keep each worker as anonymous as possible. The broker will validate worker-to-worker messages and then foward them along.

When discussing the state, the two workers on a logical channel are labelled X and Y. The worker X is the worker that send the `call request`. The worker Y is the worker that accepts the `call request`.

Figure 3.1 shows the state diagram that defines the allowed actions for worker X on a virtual call. For worker Y, the same state machine applies if you swap 'send' and 'receive'.

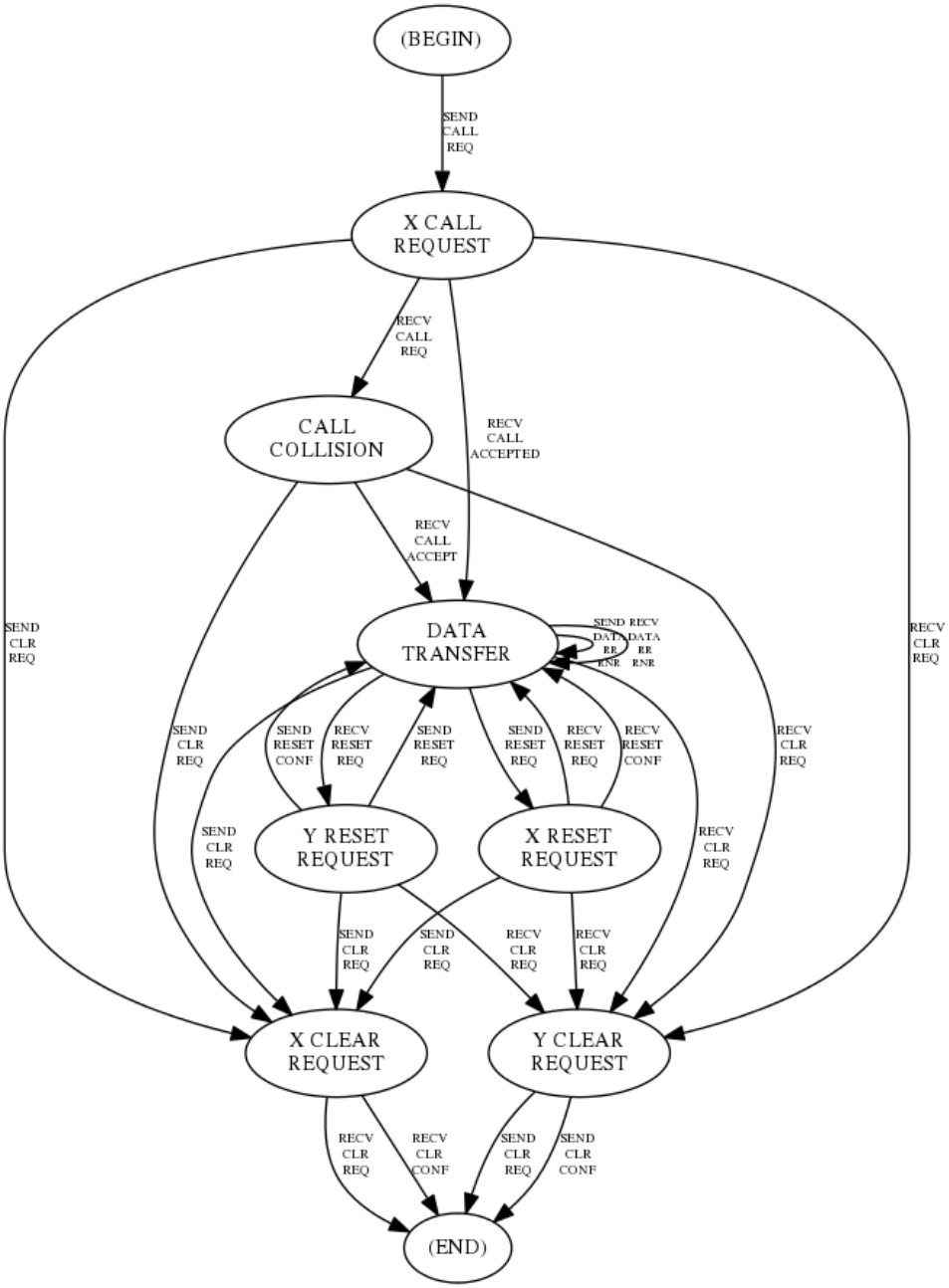

Figure 3.1: This is the state diagram that worker X is expected to follow. The circles represent states and the lines show how sending or receiving messages will modify the state. Messages sent to the broker in violation of the state machine will be rejected by the broker.

See ⟨undefined⟩ [Broker actions], page ⟨undefined⟩ for the actions taken by the broker on receipt of packets.

3.1 Call request packet

The calling worker indicates a call request by transferring a `call request` packet to the broker. The call request packet includes the called worker's pseudonym.

In the `call request` the worker states its pseudonym, the pseudonym of the worker that it wishes to contact, the maximum packet size, the maximum data rate, and a flow control parameters.

The broker will validate the `call request` packet and forward it along. The broker might reduce the packet size or data rate parameters before forwarding it.

After sending out a call request packet, the worker should consider itself to be in the `X Call Request` state.

3.2 Call accepted packet

If the called worker approves of the connection request, the calling worker will either receive a `call accepted` packet. This packet will repeat back the called and calling pseudonyms, max data rate, max packet size, and flow control parameters. They may have been modified from the values that the calling worker sent out.

After receiving a `call accepted` packet, the worker should consider itself to be in the `Data Transfer` state.

3.3 Call collision

Call collision occurs when two workers send one another `call request` packets simultaneously. When a calling worker receives a `diagnostic` packet informing it of a call collision in lieu of receiving a `call accepted` packet, it will know that call collision has occurred. It is now worker Y, the callee, and not the worker X, the caller. When that occurrs, worker X should continue to wait for a `call accepted` packet from worker Y.

3.4 Clearing by a worker

At any time, a worker may indicate that it is "clearing" the channel. Clearing the channel is ending this worker-to-worker connection. It will send a `clear request`. The broker will foward this along to the other worker.

The worker will eventually receive a response of `clear confirmation` from the broker. After receiving `clear confirmation`, the logical channel is closed by the broker and will no longer forward packets between these two workers.

3.5 Clearing by the broker

If a channel is misbehaving, the broker itself may send a `clear request` to both workers on the channel.

3.6 Clear collision

If both workers simultaneously send `clear request` messages, this is a *clear collision*. If a worker receives a `clear request` when it is expecting a `clear confirmation`, it should treat that request as a confirmation and then assume that the channel is closed.

3.7 Unsuccessful call

If the broker can't establish a logical channel between workers after receiving a `call request` from a worker, the will respond with a `clear request`, specifying the reason for the failure.

3.8 Data, RR, and RNR packets

Once a call has been established and has reached the `Data Transfer` state, the two workers are finally ready to send data back and forth.

Index