Running into an error sending a JSON object to the primary flashbots relay. Typically, the only response I'm getting, is either 400: Bad Request, or 400: Could not parse body as JSON. I only run into this problem while I attempt to send a contract call. Sending a request for user information resolves just fine, with the same format as before. See below:

func EncodeFunction(contractData string, functionCall Function) ([]byte, error) { // Takes a function's signature, it's arguments. and returns it as an encoded bytearray, without the 0x prefix.

contractABI, err := abi.JSON(strings.NewReader(contractData)) if err != nil { return []byte{}, err } data, err := contractABI.Pack(functionCall.FunctionName, functionCall.ArgSlice...) if err != nil { return []byte{}, err } / *var dataString string dataString += "0x" for _, b := range data { dataString += fmt.Sprintf("%02x", b) }* / return data, nil

}

func CreateSignedTx(toAddress common.Address, data []byte, privateKey *ecdsa.PrivateKey, endpoint string) (string, error) { // Create and return a single signed raw transaction, as a string. var err error

publicKey := privateKey.Public() publicKeyECDSA, ok := publicKey.(*ecdsa.PublicKey) if !ok { return "", err } fromAddress := crypto.PubkeyToAddress(*publicKeyECDSA) client, err := ethclient.Dial(endpoint) if err != nil { return "", err }

gasPrice, err := client.SuggestGasPrice(context.Background()) if err != nil { return "", err } nonce, err := client.PendingNonceAt(context.Background(), fromAddress) if err != nil { return "", err } tx := types.NewTransaction(nonce, // nonce toAddress, big.NewInt(0), // value uint64(21000), // gasLimit, may need to be raised. gasPrice, data,) chainID, err := client.NetworkID(context.Background()) if err != nil { return "", err } signedTx, err := types.SignTx(tx, types.NewEIP155Signer(chainID), privateKey) if err != nil { return "", err } ts := types.Transactions{signedTx} b := new(bytes.Buffer) ts.EncodeIndex(0, b) rawTxBytes := b.Bytes() rawTxHex := hex.EncodeToString(rawTxBytes)

return "0x" + rawTxHex, nil

}

func constructPayload(bundle Bundle, privateKey *ecdsa.PrivateKey) (*http.Request, error) { marshaledBundle, err := json.Marshal(bundle) if err != nil { return &http.Request{}, err }

req, err := http.NewRequest("POST", config.FlashbotsMainnetEndpoint, bytes.NewBuffer(marshaledBundle)) if err != nil { return &http.Request{}, err } req.Header.Add("Content-Type", "application/json") req.Header.Add("Accept", "application/json")

bundleString := string(marshaledBundle) hashedBundle := crypto.Keccak256Hash([]byte(bundleString)).Hex() sig, err := crypto.Sign(accounts.TextHash([]byte(hashedBundle)), privateKey) if err != nil { return &http.Request{}, err } signature := crypto.PubkeyToAddress(privateKey.PublicKey).Hex() + ":" + hexutil.Encode(sig) req.Header.Add("X-Flashbots-Signature", signature)

return req, nil

}

func submitBundle(reqPayload *http.Request) (string, error) {

httpclient := &http.Client{} resp, err := httpclient.Do(reqPayload) if err != nil { return "", err } if resp.StatusCode != 200 { respBody, _ := io.ReadAll(resp.Body) errStr := fmt.Sprintf("Error from relay. Status: %s. Body: %s", resp.Status, respBody) return "", errors.New(errStr) } else if resp.StatusCode == 200 { respBody, _ := io.ReadAll(resp.Body) return fmt.Sprintf("Success. Result: %s", respBody), nil } defer resp.Body.Close() return "", nil

}

func SimulateBundle(rawTxes []string, privateKey *ecdsa.PrivateKey, endpoint string) error { client, err := ethclient.Dial(endpoint) if err != nil { return err } blockHead, err := client.HeaderByNumber(context.Background(), nil) if err != nil { return err } // ts := time.Now().Unix() / params := map[string]interface{}{ "txs": rawTxes, "blockNumber": fmt.Sprintf("0x%x", blockHead.Number.Uint64()), "stateBlockNumber": "latest", //"timestamp": int(ts), }

```
bundle := map[string]interface{}{
    "jsonrpc": "2.0",
    "id":      1,
    "method":  "eth_callBundle",
    "params":  params,
}*/
```

params := Params{ Transactions: rawTxes, BlockNumber: fmt.Sprintf("0x%x", blockHead.Number.Uint64()), StateBlockNumber: "latest", }

bundle := Bundle{ Json: "2.0", Id: 1, Method: "eth_callBundle", Params: params, }

payload, err := constructPayload(bundle, privateKey) if err != nil { return err } res, err := submitBundle(payload) if err != nil { return err } fmt.Printf(res, "\n") return nil

}

The classes I'm using should have the correct JSON formatting, and checking in a debugger, they appear to line up correctly.

type Function struct { FunctionName string //Name of the function. ArgSlice []interface{} //A list of the arguments, in order, which the relevant function takes.

}

type Bundle struct { Json string json:"jsonrpc" Id int json:"id" Method string json:"method" Params Params json:"params"

}

type Params struct { GenericInput []interface{} json:" ,omitempty" Transactions []string json:"txs ,omitempty" // List of signed raw transactions. BlockNumber string json:"blockNumber ,omitempty" // Hex-encoded block number for which the bundle is valid. MinTimestamp int json:"minTimestamp ,omitempty" // OPTIONAL Minimum timestamp this bundle can be considered valid, in seconds since Unix epoch MaxTimestamp int json:"maxTimestamp ,omitempty" // OPTIONAL Maximum timestamp this bundle can be considered valid, in seconds since Unix epoch RevertTxHash []string json:"revertingTxHashes ,omitempty" // OPTIONAL A list of transactions which are allowed to revert in the bundle. BundleUUID string json:"replacementUUID ,omitempty" // OPTIONAL A UUID which can be used to cancel/replace the bundle. StateBlockNumber string json:"stateBlockNumber ,omitempty" Timestamp int json:"timestamp ,omitempty"

}

What, exactly, am I missing here?