89 characters of base-11?!

Mobile networking in rural Ethiopia!

Talk: Ben Kuhn (benkuhn.net, @benskuhn)
Code: Lincoln Quirk, Ka-Ping Yee, Fabian Tamp, Sander Latour
Images: Fabian Tamp, Joyce Keeley, Eve Bigaj, Mbalka Malick Kebe, Abdulahi Muhdin, Abdiselam Mohamed

Context: Wave

Mobile money

Ethiopia, 2016: 34% of adults are banked

wave: deposit/withdraw at local shops, send via an app

"uber for bank tellers"

Plus cute penguin mascot

Problem 1

Remote villages

users in rural areas

state-owned monopoly telecom

=> network is 😭

Solution 1

SMS fallback

yup

really

encoding in base64

log_2(64) = 6 bits per char

151 chars base64 = 906 bits = 113.25 bytes

"Just use Twilio!"

HA HA HA

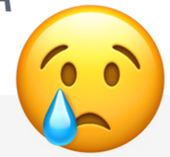
ETHIOPIA

PAY-AS-YOU-GO

SMS pricing is based on the destination and type of message you're sending, as well as the carrier to which the SMS is being sent. Our pay-as-you-go pricing gives you a fair price no matter what you build.

Pay-as-you-go SMS pricing

NUMBER USED	TEXT MESSAGES		PICTURE MESSAGES		MESSAGING SERVICE FEATURES *
	TO SEND † ‡	TO RECEIVE †	TO SEND † ‡	TO RECEIVE †	
LOCAL NUMBERS	N/A	N/A	N/A	N/A	INCLUDED
INTERNATIONAL NUMBERS	$0.0651	N/A	N/A	N/A	INCLUDED



Data center time!

"data center"

Failover

Highly available.

Problem 2

Stuff is slow

even in cities

ping takes 5 sec

app takes 30 sec

what gives?!

Standard protocol stack

Alphabet soup time!

IP: "try to send a blob of data to Abdikadir"

TCP: "reliably send + receive a stream from Abdikadir"

TLS: "make sure only the real Abdikadir can read my stream"

HTTP: "get the object at https://abdikadir.com/a/path"

(or anything else with request / response)

(fig. 1: a stack of unshaved yaks)

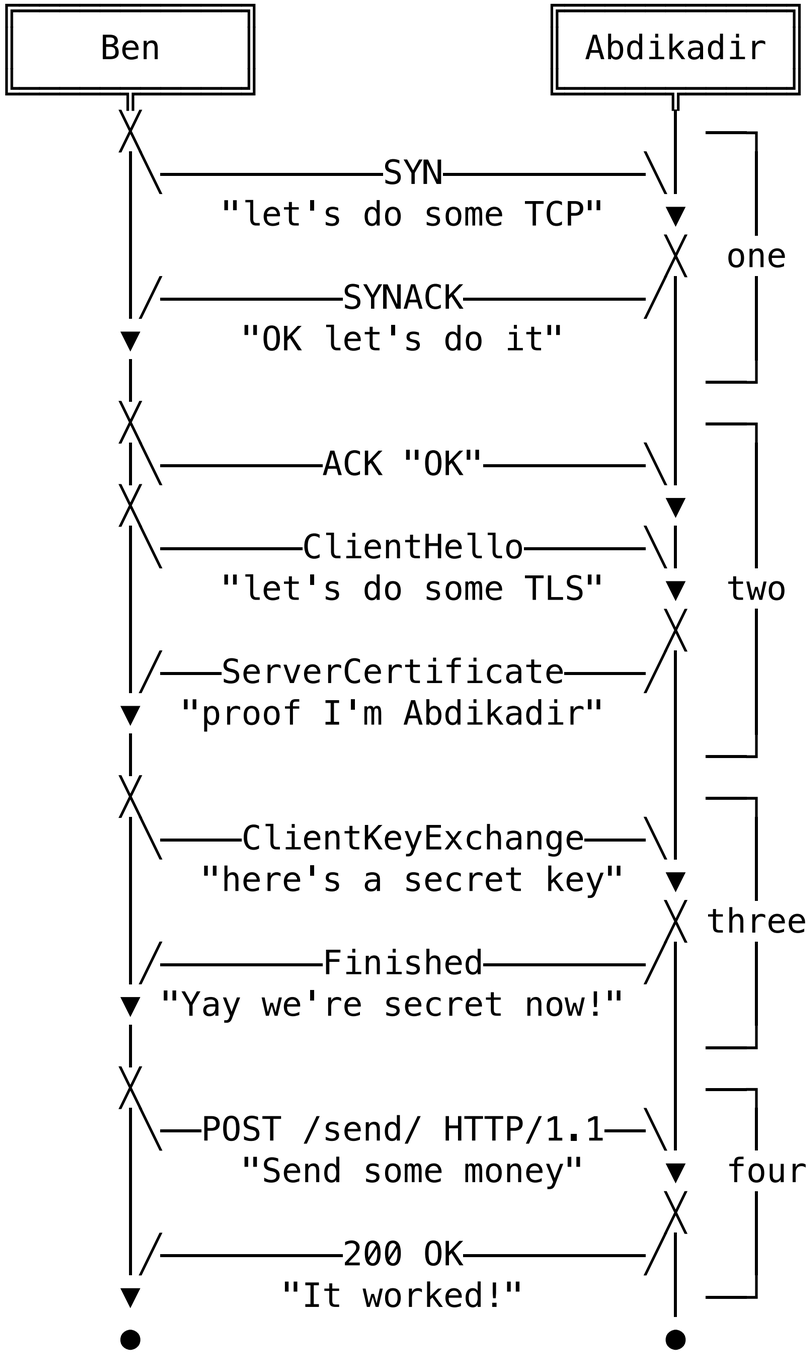
Roundtrips!

More protocols, more problems

TCP handshake = 1 roundtrips

TLS handshake = 2 roundtrips

HTTP = 1 last roundtrip



A slimmer protocol

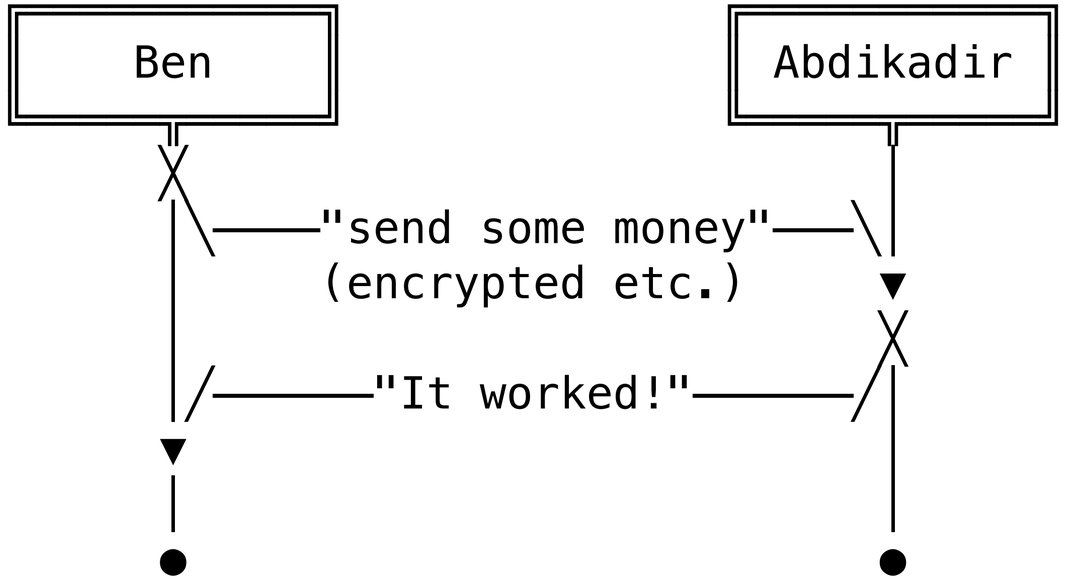
```
talking to single, pre-known server
   -> no need for key exchange
   -> no cipher suite negotiation

all our data fits in 1 packet
   -> no streams
```

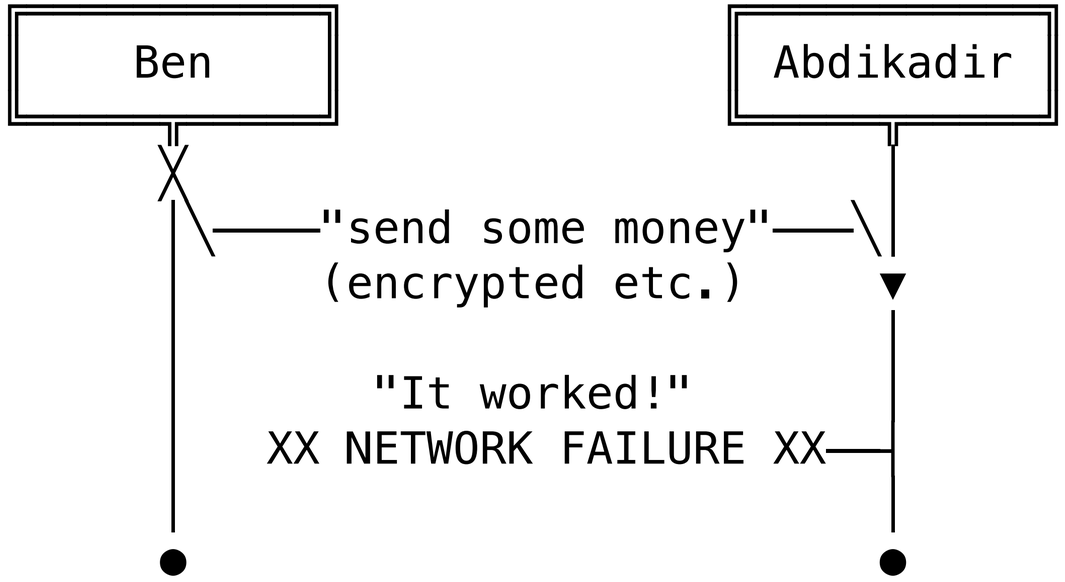
UDP!

U is for Unreliable

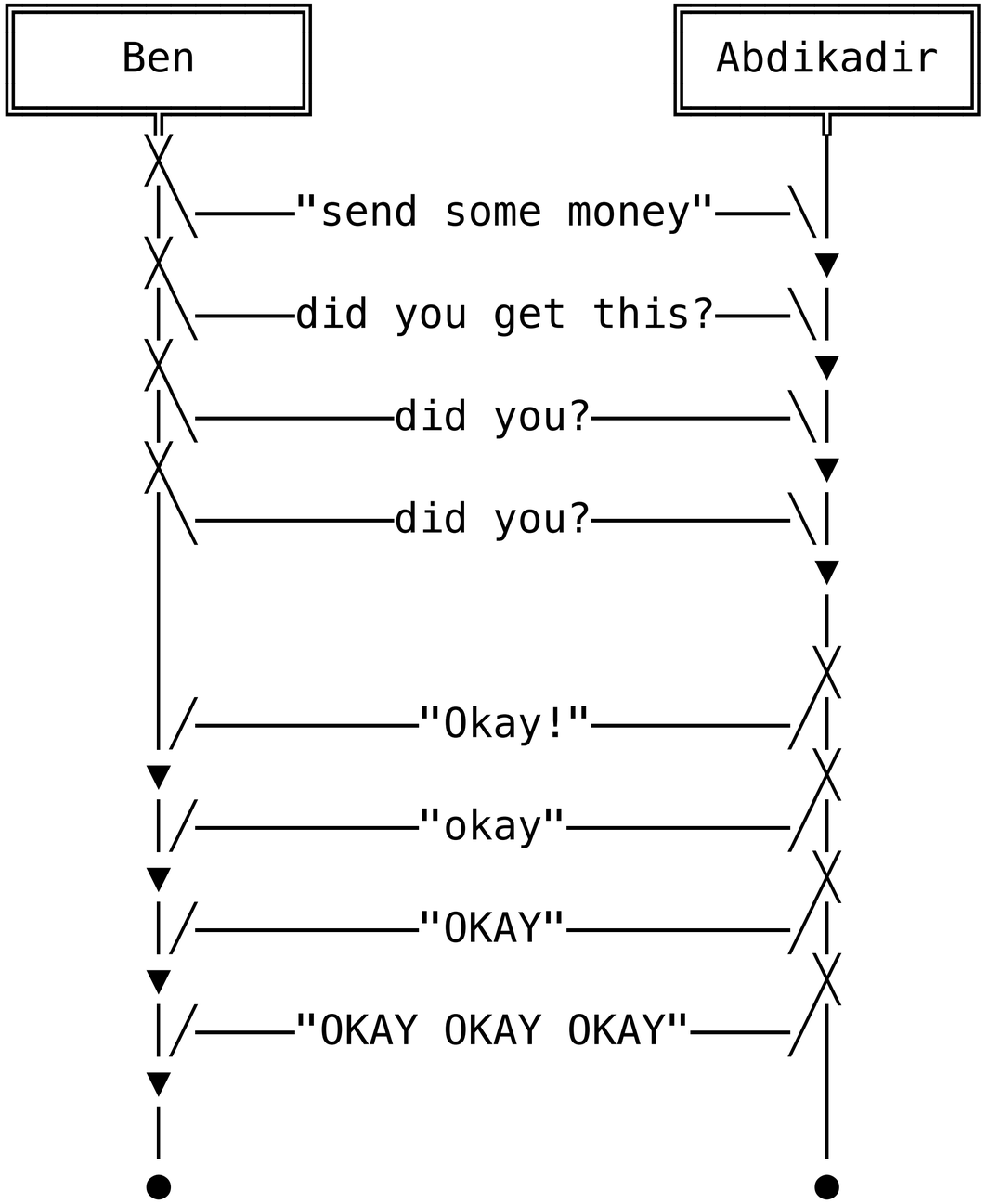
thin wrapper around IP

packets might get dropped 😱





Just UDP Harder



Problem 3

No internet in city center 🤦

Solution 3

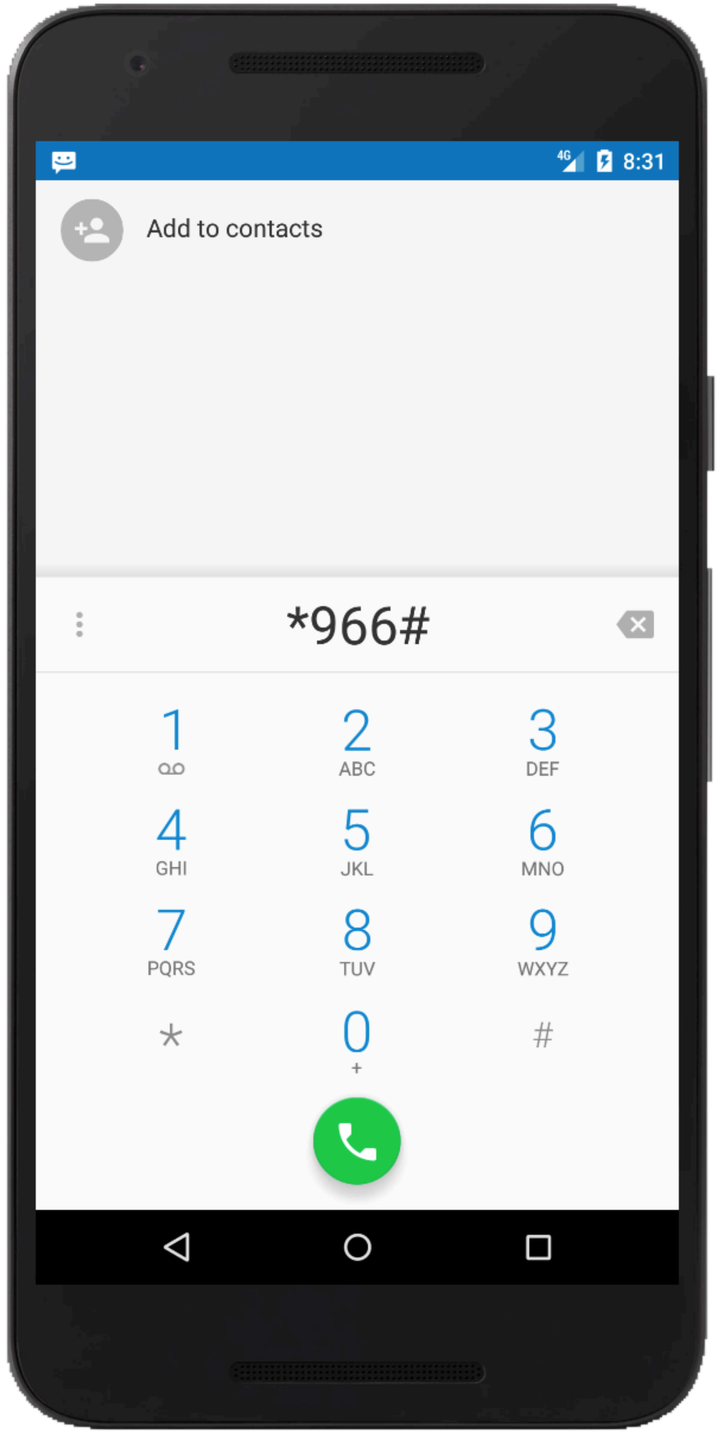
USSD!

used for things like entering airtime scratch-cards

dial some wacky looking digits+symbols like *966*<scratchcard number>#

get a synchronous response "your phone has been topped up!"

it's how telcos get paid so it tends to be reliable and fast 😀



Problem 3a: message size

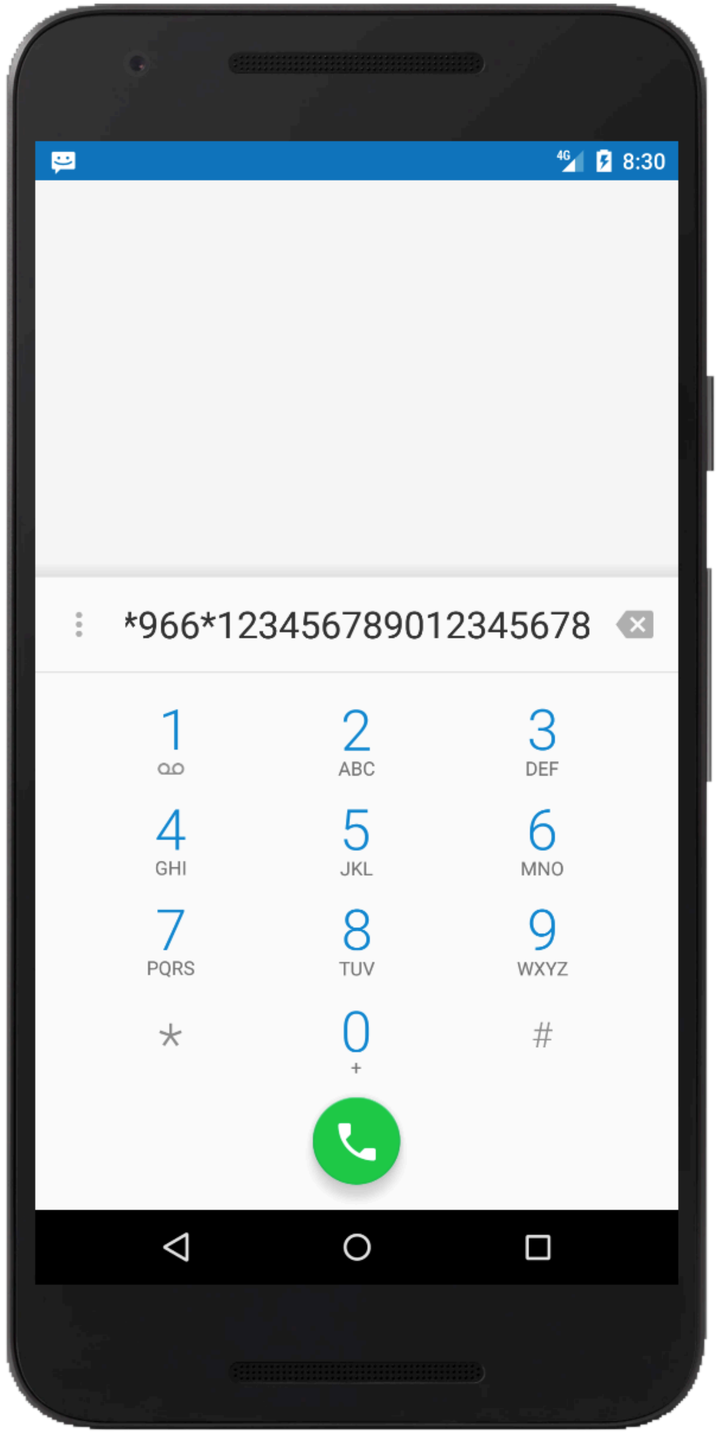
Yup it's even worse than SMS

Dial string limited to 98 chars

Encode bits in [0-9]+ by converting to int

log2(10) = 3.32 bits/digit -> 295 bits = 36.875 bytes

yikes!




Android source diving
iOS devs, eat your heart out ❤️

TelephonyManager.sendUssdRequest()

```
8737        @RequiresPermission(android.Manifest.permission.CALL_PHONE)
8738        public void sendUssdRequest(String ussdRequest,
8739                                    final UssdResponseCallback callback, Handler handler) {
8740            checkNotNull(callback, "UssdResponseCallback cannot be null.");
8741            final TelephonyManager telephonyManager = this;
8742
8743            ResultReceiver wrappedCallback = new ResultReceiver(handler) {
8744                @Override
8745                protected void onReceiveResult(int resultCode, Bundle ussdResponse) {
8746                    Rlog.d(TAG, "USSD:" + resultCode);
8747                    checkNotNull(ussdResponse, "ussdResponse cannot be null.");
8748                    UssdResponse response = ussdResponse.getParcelable(USSD_RESPONSE);
8749
8750                    if (resultCode == USSD_RETURN_SUCCESS) {
8751                        callback.onReceiveUssdResponse(telephonyManager, response.getUssdRequest(),
8752                                response.getReturnMessage());
8753                    } else {
8754                        callback.onReceiveUssdResponseFailed(telephonyManager,
8755                                response.getUssdRequest(), resultCode);
8756                    }
8757                }
8758            };
8759
8760            try {
8761                ITelephony telephony = getITelephony();
8762                if (telephony != null) {
8763                    telephony.handleUssdRequest(getSubId(), ussdRequest, wrappedCallback);
8764                }
```

PhoneInterfaceManager.handleUssdRequest()

```
1645      public void handleUssdRequest(int subId, String ussdRequest, ResultReceiver wrappedCallback) {
1646        enforceCallPermission();
1647        if (!SubscriptionManager.isValidSubscriptionId(subId)) {
1648            return;
1649        }
1650        Pair<String, ResultReceiver> ussdObject = new Pair(ussdRequest, wrappedCallback);
1651        sendRequest(CMD_HANDLE_USSD_REQUEST, ussdObject, subId);
1652      };
```

```
328                 case CMD_HANDLE_USSD_REQUEST: {
329                     request = (MainThreadRequest) msg.obj;
330                     final Phone phone = getPhoneFromRequest(request);
331                     Pair<String, ResultReceiver> ussdObject = (Pair) request.argument;
332                     String ussdRequest =  ussdObject.first;
333                     ResultReceiver wrappedCallback = ussdObject.second;
334
335                     if (!isUssdApiAllowed(request.subId)) {
336                         // Carrier does not support use of this API, return failure.
337                         Rlog.w(LOG_TAG, "handleUssdRequest: carrier does not support USSD apis.");
338                         UssdResponse response = new UssdResponse(ussdRequest, null);
339                         Bundle returnData = new Bundle();
340                         returnData.putParcelable(TelephonyManager.USSD_RESPONSE, response);
341                         wrappedCallback.send(TelephonyManager.USSD_RETURN_FAILURE, returnData);
342
343                         request.result = true;
344                         synchronized (request) {
345                             request.notifyAll();
346                         }
347                         return;
348                     }
349
350                     try {
351                         request.result = phone != null ?
352                                 phone.handleUssdRequest(ussdRequest, wrappedCallback)
353                                 : false;
354                     } catch (CallStateException cse) {
355                         request.result = false;
356                     }
```

GsmCdmaPhone.handleUssdRequest()

```
1252    @Override
1253    public boolean handleUssdRequest(String ussdRequest, ResultReceiver wrappedCallback) {
1254        if (!isPhoneTypeGsm() || mPendingMMIs.size() > 0) {
1255            //todo: replace the generic failure with specific error code.
1256            sendUssdResponse(ussdRequest, null, TelephonyManager.USSD_RETURN_FAILURE,
1257                    wrappedCallback );
1258            return true;
1259        }
1260
1261        // Try over IMS if possible.
1262        Phone imsPhone = mImsPhone;
1263        if ((imsPhone != null)
1264                && ((imsPhone.getServiceState().getState() == ServiceState.STATE_IN_SERVICE)
1265                || imsPhone.isUtEnabled())) {
1266            try {
1267                logd("handleUssdRequest: attempting over IMS");
1268                return imsPhone.handleUssdRequest(ussdRequest, wrappedCallback);
1269            } catch (CallStateException cse) {
1270                if (!CS_FALLBACK.equals(cse.getMessage())) {
1271                    return false;
1272                }
1273                // At this point we've tried over IMS but have been informed we need to handover
1274                // back to GSM.
1275                logd("handleUssdRequest: fallback to CS required");
1276            }
1277        }
1278
1279        // Try USSD over GSM.
1280        try {
1281            dialInternal(ussdRequest, null, VideoProfile.STATE_AUDIO_ONLY, null,
1282                    wrappedCallback);
1283        } catch (Exception e) {
1284            logd("handleUssdRequest: exception" + e);
1285            return false;
1286        }
1287        return true;
1288    }
```

```
1179    @Override
1180    protected Connection dialInternal(String dialString, UUSInfo uusInfo, int videoState,
1181                                      Bundle intentExtras)
1182            throws CallStateException {
1183        return dialInternal(dialString, uusInfo, videoState, intentExtras, null);
1184    }
1185
1186    protected Connection dialInternal(String dialString, UUSInfo uusInfo, int videoState,
1187                                      Bundle intentExtras, ResultReceiver wrappedCallback)
1188            throws CallStateException {
1189
1190        // Need to make sure dialString gets parsed properly
1191        String newDialString = PhoneNumberUtils.stripSeparators(dialString);
1192
1193        if (isPhoneTypeGsm()) {
1194            // handle in-call MMI first if applicable
1195            if (handleInCallMmiCommands(newDialString)) {
1196                return null;
1197            }
1198
1199            // Only look at the Network portion for mmi
1200            String networkPortion = PhoneNumberUtils.extractNetworkPortionAlt(newDialString);
1201            GsmMmiCode mmi = GsmMmiCode.newFromDialString(networkPortion, this,
1202                    mUiccApplication.get(), wrappedCallback);
1203            if (DBG) logd("dialInternal: dialing w/ mmi '" + mmi + "'...");
1204
1205            if (mmi == null) {
1206                return mCT.dial(newDialString, uusInfo, intentExtras);
1207            } else if (mmi.isTemporaryModeCLIR()) {
1208                return mCT.dial(mmi.mDialingNumber, mmi.getCLIRMode(), uusInfo, intentExtras);
1209            } else {
1210                mPendingMMIs.add(mmi);
1211                mMmiRegistrants.notifyRegistrants(new AsyncResult(null, mmi, null));
1212                mmi.processCode();
1213                return null;
1214            }
1215        } else {
1216            return mCT.dial(newDialString);
1217        }
```

PhoneNumberUtils.extractNetworkPortionAlt()

```java
    /**
     * Extracts the network address portion and canonicalize.
     *
     * This function is equivalent to extractNetworkPortion(), except
     * for allowing the PLUS character to occur at arbitrary positions
     * in the address portion, not just the first position.
     *
     * @hide
     */
    public static String extractNetworkPortionAlt(String phoneNumber) {
        if (phoneNumber == null) {
            return null;
        }

        int len = phoneNumber.length();
        StringBuilder ret = new StringBuilder(len);
        boolean haveSeenPlus = false;

        for (int i = 0; i < len; i++) {
            char c = phoneNumber.charAt(i);
            if (c == '+') {
                if (haveSeenPlus) {
                    continue;
                }
                haveSeenPlus = true;
            }
            if (isDialable(c)) {
                ret.append(c);
            } else if (isStartsPostDial (c)) {
                break;
            }
        }

        return ret.toString();
    }
```

PhoneNumberUtils.isDialable()

```
102     /** True if c is ISO-LATIN characters 0-9, *, # , +, WILD  */
103     public final static boolean
104     isDialable(char c) {
105         return (c >= '0' && c <= '9') || c == '*' || c == '#' || c == '+' || c == WILD;
106     }
 54     /*
 55      * Special characters
 56      *
 57      * (See "What is a phone number?" doc)
 58      * 'p' --- GSM pause character, same as comma
 59      * 'n' --- GSM wild character
 60      * 'w' --- GSM wait character
 61      */
 62     public static final char PAUSE = ',';
 63     public static final char WAIT = ';';
 64     public static final char WILD = 'N';
```

The final alphabet

dun dun dunnn

'*' splits up requests

'#' ends the request

'+' made requests fail, no one remembers why 😂

but 'N' works!!

log2(11) = 3.45 bits/char
  -> 38 bytes

2 extra bytes!! 🙌 🙌 🙌

N

Epilogue

I Can't Believe This Crap Is Still In Prod

```
no ussd--requires separate deals w/ each telco
no sms--in Senegal, internet is "good enough"
still use UDP, but excited to replace it with HTTP/3 (aka "QUIC")
```

Thanks for listening ❤️

benkuhn.net/base11/ - @benskuhn

Ben Abeba Scottish restaurant, Lalibela, Ethiopia