Bridging the World Together

CONTACT@ADAMFURMANEK.PL

HTTP://BLOG.ADAMFURMANEK.PL

FURMANEKADAM

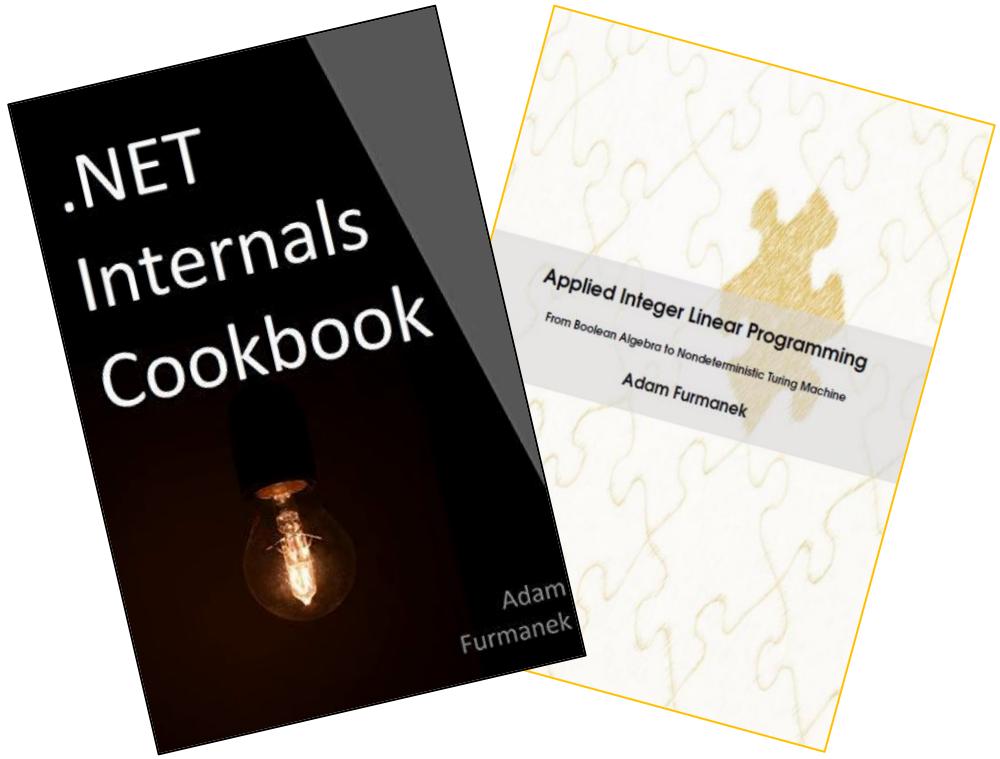
About me

Software Engineer, Blogger, Book Writer, Public Speaker.
Author of ***Applied Integer Linear Programming*** and ***.NET Internals Cookbook***.

http://blog.adamfurmanek.pl

contact@adamfurmanek.pl

furmanekadam



Random IT Utensils

IT, operating systems, maths, and more.

Agenda

System at a glance.

History of IMs.

Architecture.

Food for thought.

Summary.

System at a glance

It all started with…

Initial scope

Jabber support only.

Sending and receiving e-mails.

It must work as a „copy and paste” application. No external components, no database, nothing.

It’s not Sev1 application. If it dies – that’s okay.

It must work on Windows and be written in .NET Framework.

Write it in one evening and don’t touch anymore.

Current Scope

It is my main IM and “web monitoring” tool (for forums, books, discounts etc).

Supports web, desktop, and mobile UI. Has dedicated mobile application.

Can mail me, text me, call me, read messages out loud, transform speech to text.

I can mail it, text it, use mobile, desktop, or web interfaces to send messages.

I can use it on an airplane with messaging WiFi only (and still can contact any network, not just Messenger or Whatsapp).

It can share media between networks so I can “just send” image to any network, no matter if it supports attachments or not (like IRC).

Recognizes when I’m around or unavailable and then sends more notifications.

I can call anyone over Cell Network (i.e. Cell Network -> Messenger).

Can schedule messages for later so I can compose now and get it deliver at a specific time.

Supports recalling messages so I can stop them from getting delivered after hitting enter.

Notifies about deadletters and failures so I’m paged when there is an issue.

Encrypts messages so it’s safe in transit over public channels.

Supports 30+ networks.

Runs on a single box with relatively low resource usage but can also scale on other machines if needed.

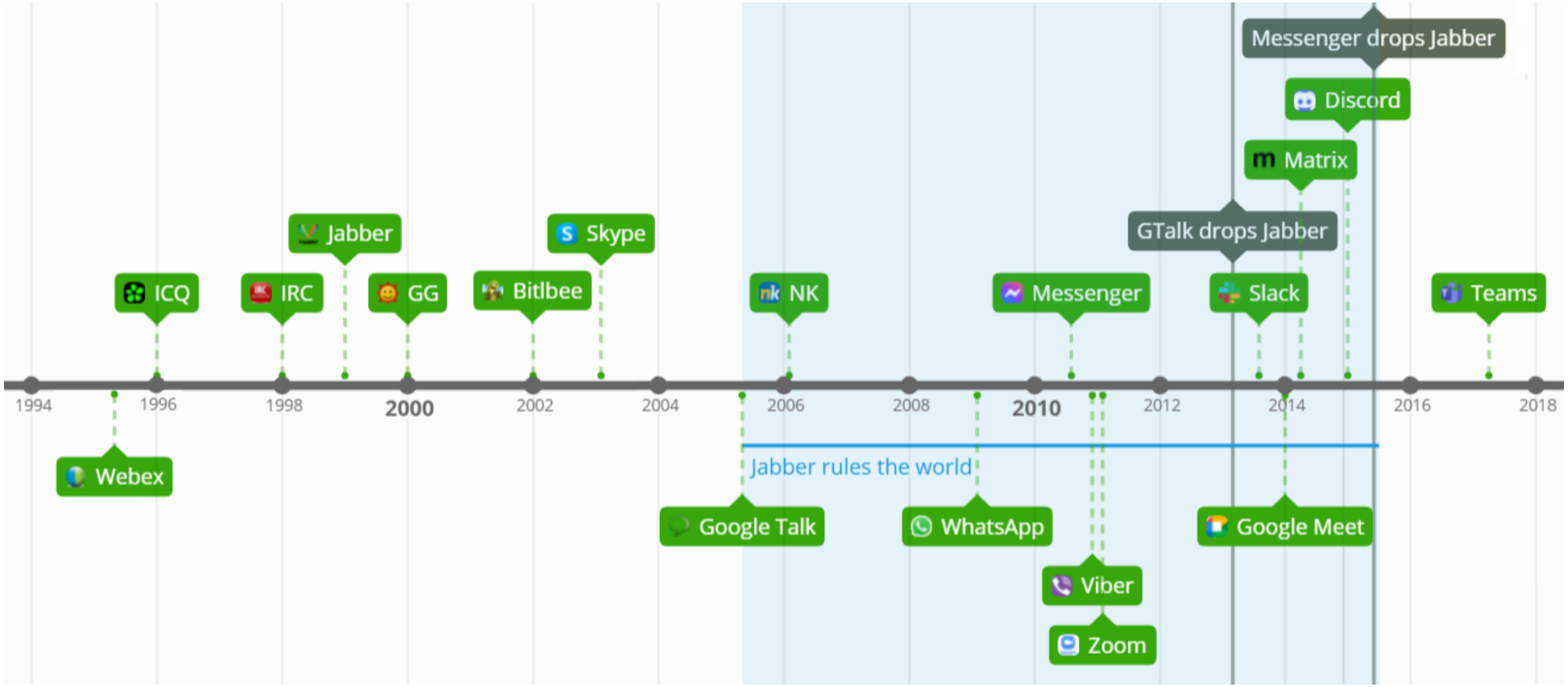
Self heals itself and can survive pretty significant outages (as long as the machine doesn’t die, obviously).

Is free and doesn’t use paid components.

And the most important — it works for years now and I know I can trust it.

History

History


Messenger drops Jabber
Discord
Matrix
GTalk drops Jabber
Jabber
Skype
ICQ
IRC
GG
Bitlbee
NK
Messenger
Slack
Teams
1994
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
Webex
Jabber rules the world
Google Talk
WhatsApp
Google Meet
Viber
Zoom

Integrations

XMPP

- Multiple networks via Gateways which didn’t actually work.
- Poor multi-user chat support.
- Supported in GTalk, Messenger, NK. Dropped at some point because of inconsistencies, lack of video support nad Not Invented Here syndrome.

Bitlbee

- Irc-based application for connecting multiple networks.
- Integration based around the concept of multi-user chat.

Matrix

- Based on bots which are not transparent for the network.

Multicommunicators?

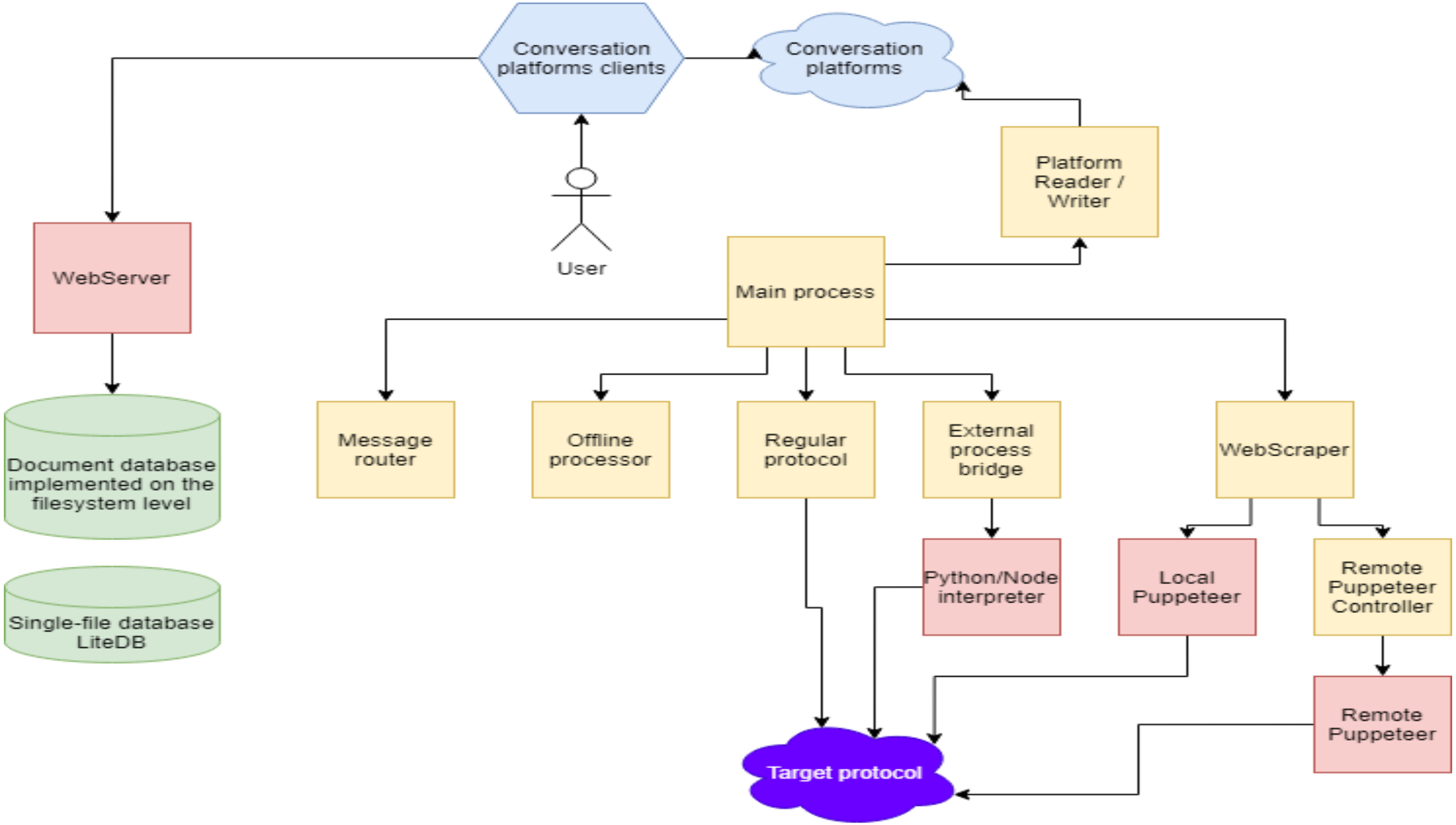
Architecture

General idea

There is some backend which connects to multiple networks, downloads messages and processes them.

Some networks are supported natively, some via gateways, some via webscraping.

Messages are then pushed to some chat platform like Slack.


Conversation platforms clients
Conversation platforms
Platform Reader / Writer
User
WebServer
Main process
Document database implemented on the filesystem level
Single-file database LiteDB
Message router
Offline processor
Regular protocol
External process bridge
WebScraper
Python/Node interpreter
Local Puppeteer
Remote Puppeteer Controller
Target protocol
Remote Puppeteer

Why processes instead of threads?

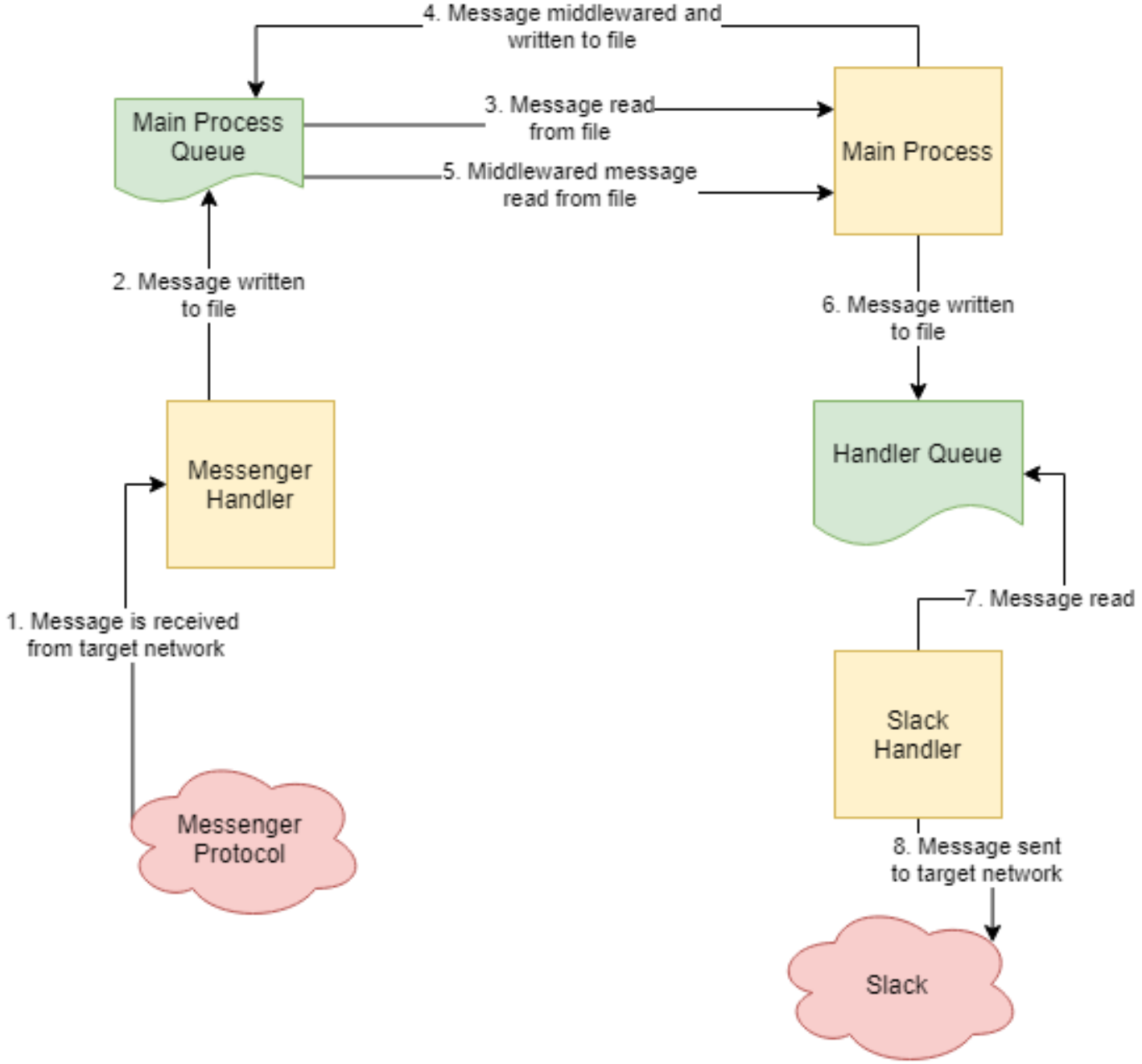
Unhandled exception on a thread can take the proces down (and there may be no way to stop it).

StackOverflowException may do the same.

Deadlock may stop whole process if you make a mistake.

There may be no way to recover the connection.

Native errors kill you and there is no use in fighting that.



Queuing is the key!

Do not couple components, you want your workflow to be as transparent as possible.

Emit events and messages.

Self-healing is the most important!

UI

It should look nice if that’s your primary application.

Timestamps need to be formatted.

Link previews are great.

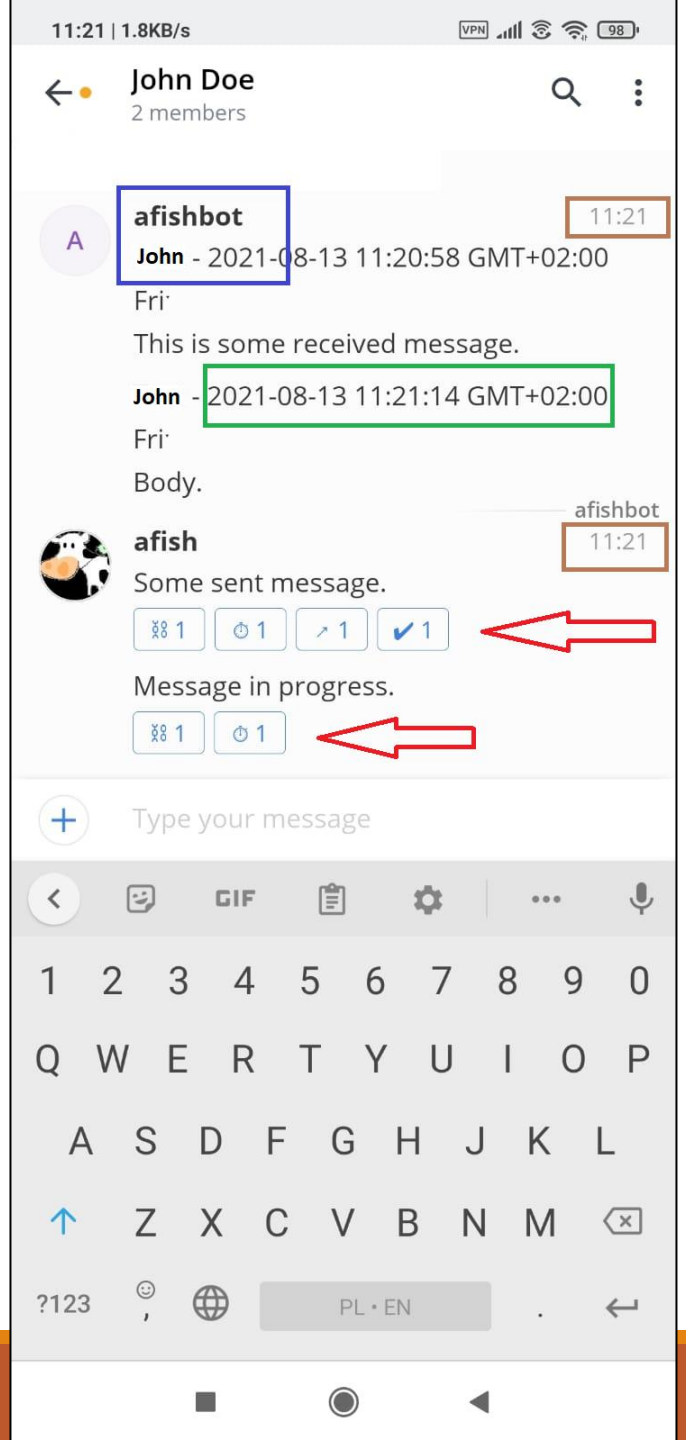
Message markers are useful – was the message sent, delivered, displayed etc.

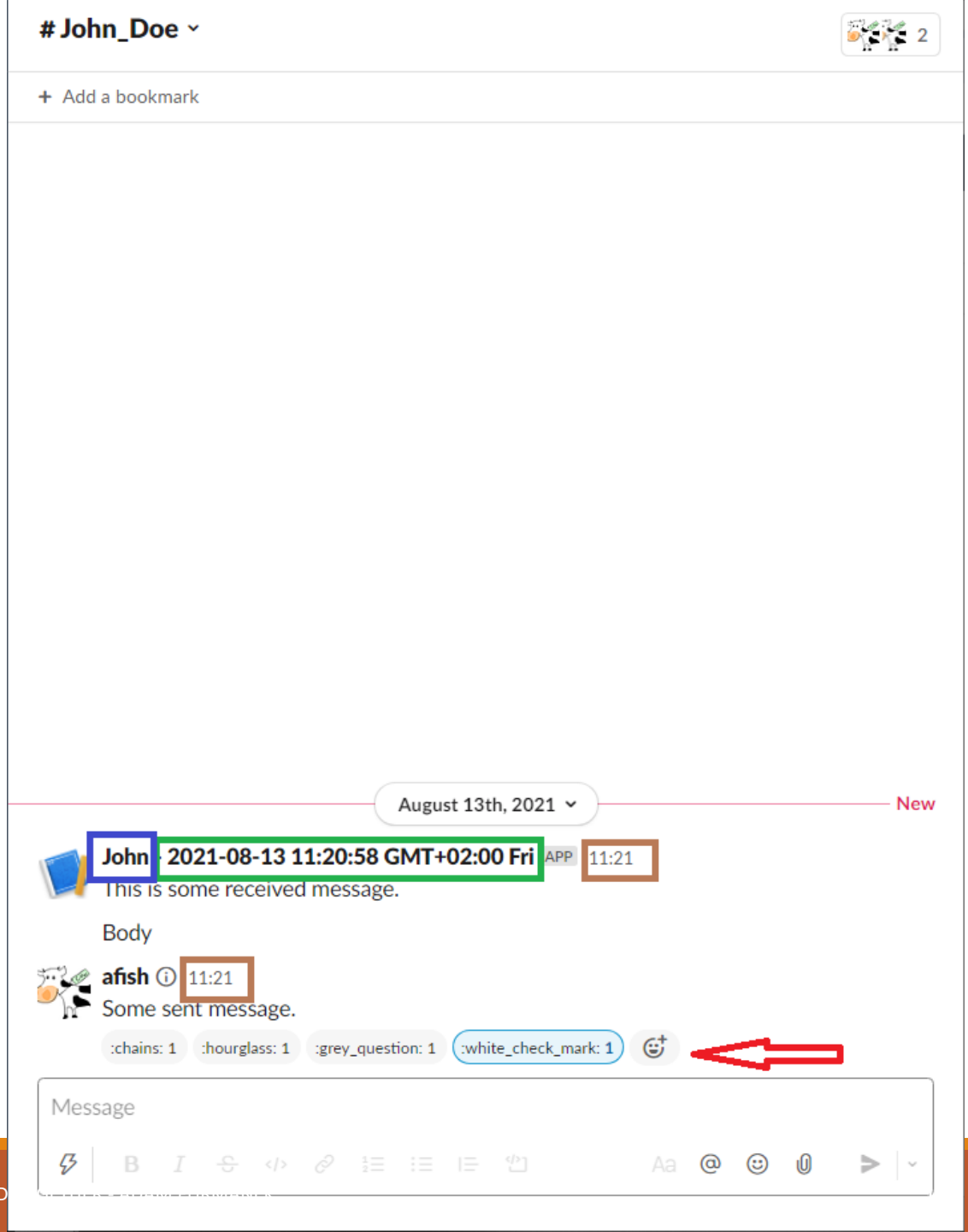
Highlight new messages is cool. Showing popups and using sounds is great.

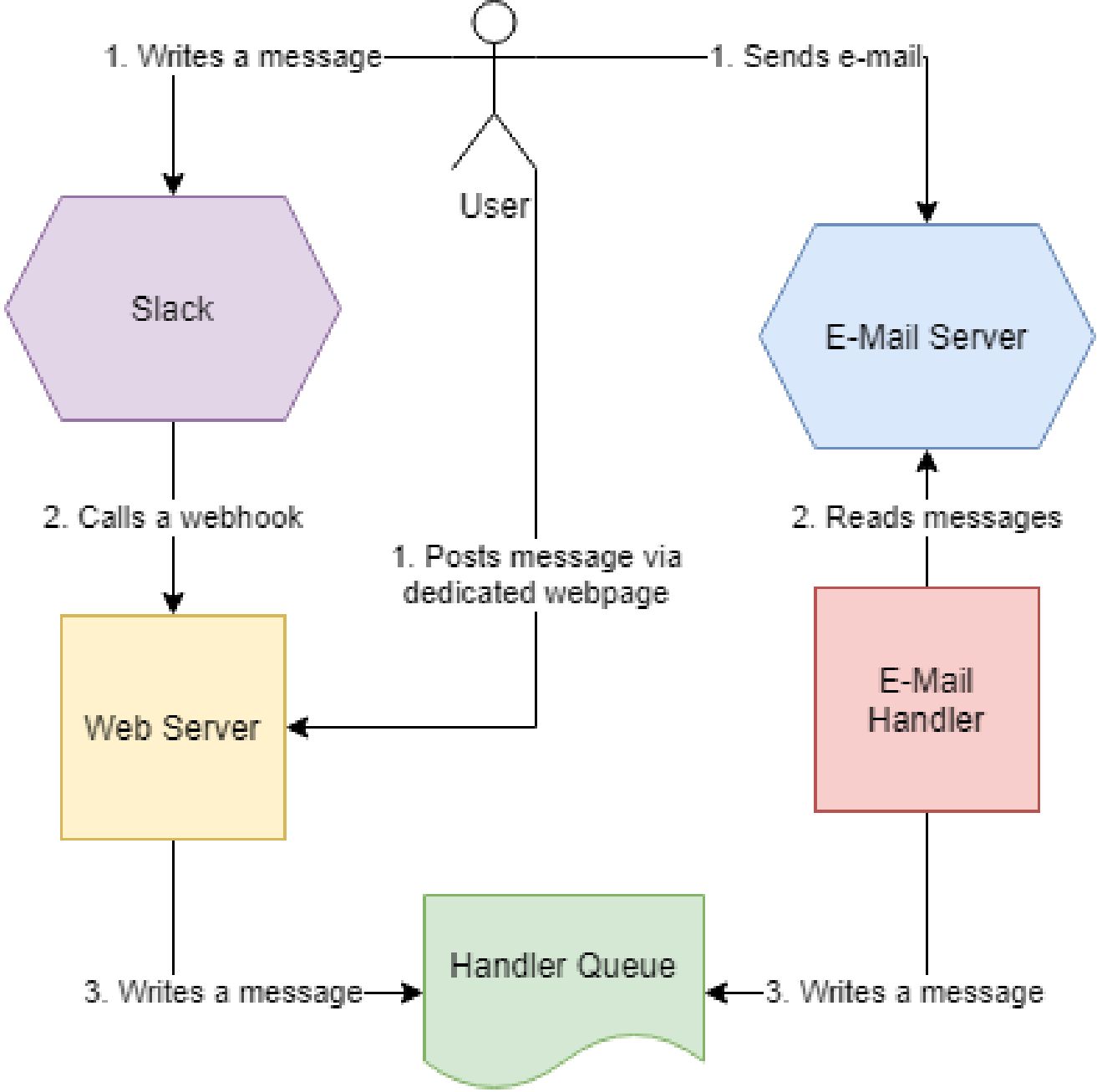
It should be fast.

Text to Speech and Speech to Text are great.

Security is important.


11:21 | 1.8KB/s
John Doe
2 members
afishbot
11:21
John - 2021-08-13 11:20:58 GMT+02:00
Fri
This is some received message.
John - 2021-08-13 11:21:14 GMT+02:00
Fri
Body.
afishbot
afish
11:21
Some sent message.
1
1
1
1
Message in progress.
1
1
Type your message
GIF
1 2 3 4 5 6 7 8 9 0
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
?123
PL • EN


John_Doe
2
+ Add a bookmark
August 13th, 2021
New
John - 2021-08-13 11:20:58 GMT+02:00 Fri
APP
11:21
This is some received message.
Body
afish
11:21
Some sent message.
:chains: 1
:hourglass: 1
:grey_question: 1
:white_check_mark: 1
Message



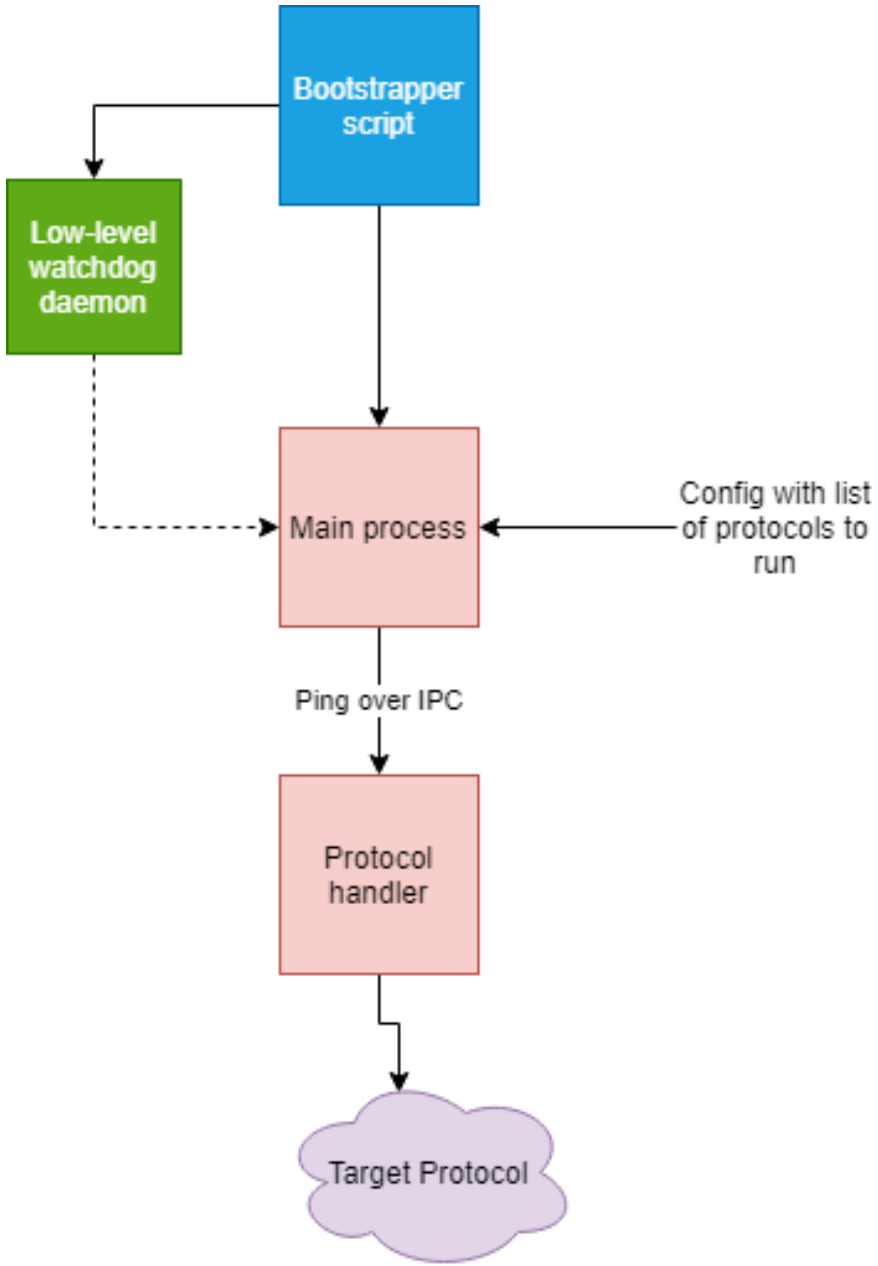
Messages are encrypted in every channel.

They are decrypted in multiple ways:

- Modified open source SMS application for Android
- Hacked IM application for Android
- Injected JS to webpage hosted in browser and in desktop app
- Static webpage used with emails.

Statuses are shown with reactions and custom CSS.

Timestamps are fixed in apps and with custom JS.



Watchdog:

- How do you know it died? Maybe it's just slow? Maybe it's waiting for mutex? Maybe it's working hard on some Infinite loop? **Just add ping**.
- How do you handle ping? If it runs on a separate thread then how do you know it's not the only one thread? What if it's slow? How do you call it?

Let's kill it!

- Stopping the right way won't work but if we kill it then it won't release resources.
- What if it's debugged? What if OS stops us from killing it?
- What if there is some error reporting?
- What if it's a zombie proces and we can't kill it at all?
- What if it's running remotely and we lost the connection?

Restart!

- What if resources are still locked?
- What if it owned critical section? How do we know data is correct?
- What if it dies repeatedly because of a poison message? How many times do we restart it? Backoff? Jitter?
- What if our watchdog dies?

Watchdog solution

Observe proces via named pipe.

Use deep ping on a separate thread.

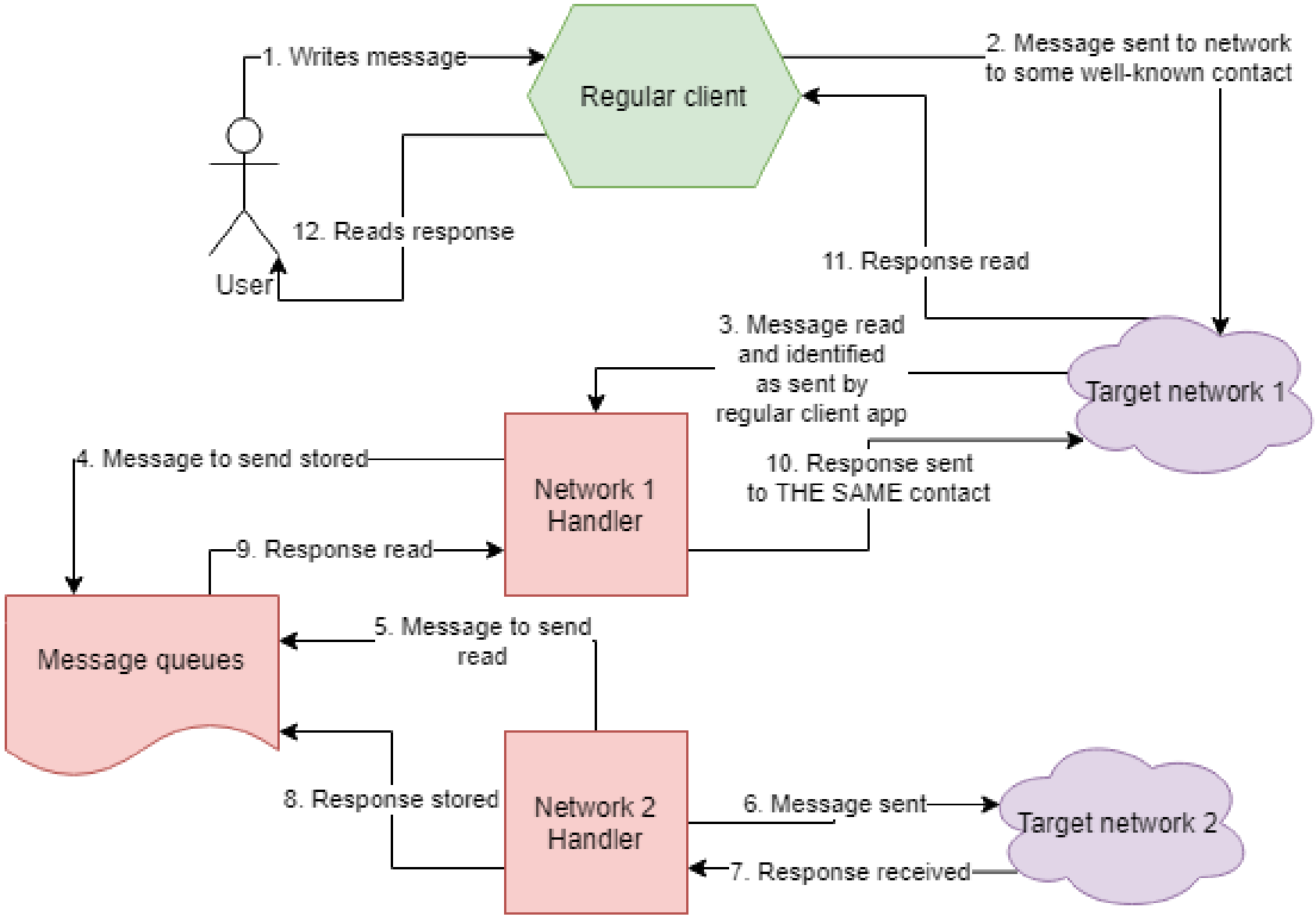
If ping fails 3 times – kill the proces and restart it.

Take memory dump before killing it.

Processes observe watchdog and kill themselves if it dies.

Never wait indefinitely for a mutex!

Use custom mutex to track ownership. If proces A detects that B holds mutex for too long, A takes memory dump of B and kills it to retake the lock.


1. Writes message
Regular client
2. Message sent to network
to some well-known contact
User
12. Reads response
11. Response read
3. Message read
and identified
as sent by
regular client app
Target network 1
4. Message to send stored
Network 1
Handler
10. Response sent
to THE SAME contact
9. Response read
Message queues
5. Message to send
read
8. Response stored
Network 2
Handler
6. Message sent
Target network 2
7. Response received

Food for thought

Security

You HAVE TO encrypt. Your data will leak and you don’t want it to be plain-text.

You will HAVE TO be able to rotate keys sooner or later.

You probably want to support multiple encryption schemes for backward compatibility.

Base64 sounds great but it’s not. Avoid special characters, especially for SMSs.

Go with Base62.

Keep in mind your devices needs to keep up and you don’t want to consume too much of a CPU power.

Protocol quirks

How to run:

- Native support in your beloved language.
- Native support in some other language you can use (like IKVM or Jython).
- Native support in external proces.

How to connect:

- Stateful connection
- Polling
- CLI
- Transports via gateways
- Web scraping

Libraries are buggy:

- They cause deadlocks, exceptions, crashes
- They hang on errors
- If you have an error – immediately kill the process

Protocols differ a lot:

- Some represent groupchats as a chatroom (Jabber)
- Some send messages to multiple users (GG)
- Some create „fake” ID to represent groupchat (Messenger)
- Similarly for images and videos

External clients

Some networks actively support 3rd party clients

- Jabber, IRC

Some networks do not support them but do not fight them

- Gadu-Gadu, Slack

Some of them require manual activity once in a while

- Messenger

Some of them actively fight 3rd party apps

- Discord, WhatsApp

Often web scraping is a solution!

Webscraping

```
// Based on https://stackoverflow.com/a/39165137/1543037
window.FindReact = function FindReact(dom, traverseUp = 0) {
    const key = Object.keys(dom).find(key=>key.startsWith("__reactInternalInstance$"));
    const domFiber = dom[key];
    if (domFiber == null) return null;

    // react less than16
    if (domFiber._currentElement) {
        let compFiber = domFiber._currentElement._owner;
        for (let i = 0; traverseUp > i; i++) {
            compFiber = compFiber._currentElement._owner;
        }
        return compFiber._instance;
    }

    // react 16+
    const GetCompFiber = fiber=>{
        //return fiber._debugOwner; // this also works, but is __DEV__ only
        let parentFiber = fiber.return;
        while (typeof parentFiber.type == "string") {
            parentFiber = parentFiber.return;
        }
        return parentFiber;
    };
    let compFiber = GetCompFiber(domFiber);
    for (let i = 0; traverseUp > i; i++) {
        compFiber = GetCompFiber(compFiber);
    }
    return compFiber.stateNode;
}
```

Read model if possible (React, Knockout.JS, Angular).

Scrape network calls (directly or via Chrome extensions).

Extract data from DOM data- keys.

Just parse DOM.

Keep in mind there may be multiple interfaces for one application

- No JS mode
- Mobile View
- Different web page (Skype)

Mutexes

Windows Mutex does not expose ownership. You don’t know who holds it.

There is Windows Chain Traversal API which is not super helpful.

Solution? Implement your custom Mutex!

Store <PID and TID> in 64 bits.

Keep in mind PID and TID may be reused but that’s very unlikely.

Use memory mapped file.

Initialize timeout and spinTimeout. Calculate when we need to stop trying to acquire the lock.

Loop indefinitely. Start with checking the time.

Use CAS to replace 0 with identifier. It it worked – we’re done.

Otherwise extract current holder and check if it exists. Also, check if it’s the same holder as before (to avoid killing cascade).

If it doesn’t exist then clear the lock.

If it does and time is up then take memory dump, log command line and finally kill the process.

Semaphores

Semaphores track ownership in different way.

When thread holding semaphore dies it doesn’t decrease the semaphore count.

You don’t get information that the semaphore was abandoned.

You can’t use semaphores. You need to emulate them with mutexes.

Implementing simple file system-based database

Pros:

- You don’t need indexes as file system takes care of that (which MAY become slow).
- You have locks implemented out-of-the-box.
- You support transactions (they are not super useful, though).
- You can with filtering easily.
- It can be accessed from multiple machines with NFS.

Cons:

- NFS is slow and makes some of features unavailable.
- Scan is not serializable (more of read committed).
- **Document update is tricky.**

How do you overwrite files?

If you try something naive (like just overwriting the file) then you end up with atomicity and transactions which do not work (especially when working with big files).

Two files and pointer:

- Requires atomic deletion of the file

Reading:

- Check if POINTER exists.
- If it does – return A.
- Otherwise return B.

Writing

- Check if POINTER exists.
- If it does – write B and delete POINTER
- Otherwise write A and create empty POINTER

Three files:

- No requirements

Reading:

- Read A, Read B, compare
- If they are the same – write A to C, return A
- Otherwise write garbage to A, garbage to B, write C to A, write C to B, return C

Writing:

- Take first character of new A and old A, write something different to B
- Write A, write B, write C.

SMS

They replace content.

They get lost.

They get duplicated.

They are limited to 160 characters (or even fewer).

They get out of order.

GoogleVoice.

Zadarma.

Sending via E-Mail.

Delivering to E-Mail.

Github for 2FA.

Let’s add some voice

TEXT TO SPEECH

```
function say(text, voice){
    var speak = new SpeechSynthesisUtterance();
    speak.text = text;
    speak.voice = speechSynthesis.getVoices()[voice];
    speechSynthesis.speak(speak);
}
```

SPEECH TO TEXT

You can use Google API in Chrome (not in Chromium).

It recognizes 95% of the sentence (which IMO is not enough).

For translations you can use Yandex (it’s free).

You can call via IFTTT, 2FA or something paid likie Twilio.

Random thoughts

Self-healing is the most important!

Name your threads. Take memory dumps. Log PID and TID.

Don’t exit application deep inside your code.

Watch for deadletters because of deadletters.

Always generate new message based on some context – use factories and cloning instead of constructors.

Have sanity checks and assertions in place. If you send 10 messages per second – is it okay? 100? 1000?

If something is suspicious then stop instead of risking spamming others.

Summary

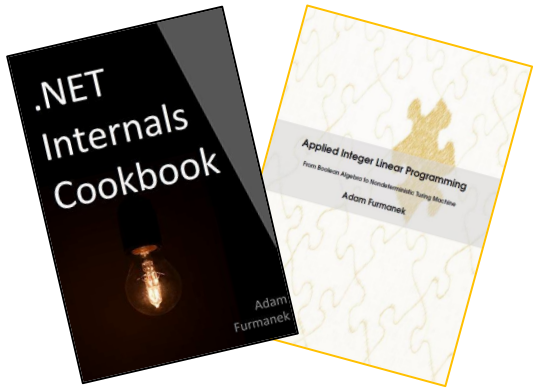
Reimplementing computer science is cool.

Data persistence and self-healing is crucial.

Once you enter wide-Internet you should trust no one.

I’m yet to implement Paxos ;)

Q&A



Random IT Utensils

IT, operating systems, maths, and more.

Thanks!

CONTACT@ADAMFURMANEK.PL

HTTP://BLOG.ADAMFURMANEK.PL

FURMANEKADAM