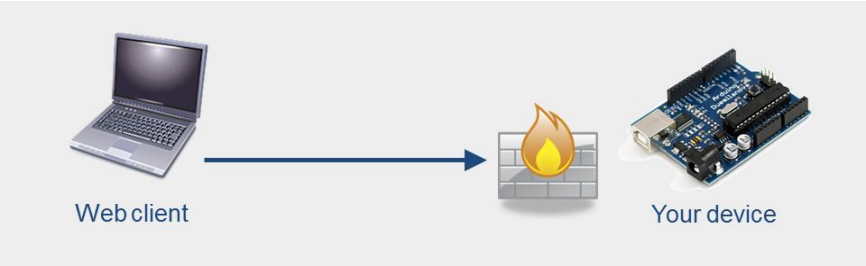
Yaler Protocol Documentation

09.11.2012, tamberg@yaler.net

Overview

The Yaler relay infrastructure enables secure Web access to embedded systems in a mobile network or hidden behind a firewall or NAT. This document describes the Yaler protocol, which a device has to implement in order to become accessible via Yaler.

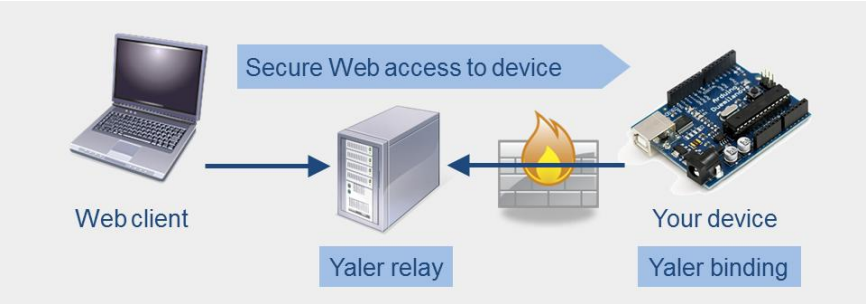
Internet-connected devices face two principal challenges:



Accessibility – Firewalls prevent incoming HTTP connections to the embedded system, especially if the administrator is not known or adding an exception to the firewall policy is not an option.

Addressability – To cope with the increasing shortage of free IPv4 addresses, embedded systems often can only be provided with dynamically changing IP addresses. NAT also mitigates the address depletion problem by hiding multiple connected devices behind a single public IP address. An embedded system behind NAT therefore does not have its own public address. It is not individually addressable.

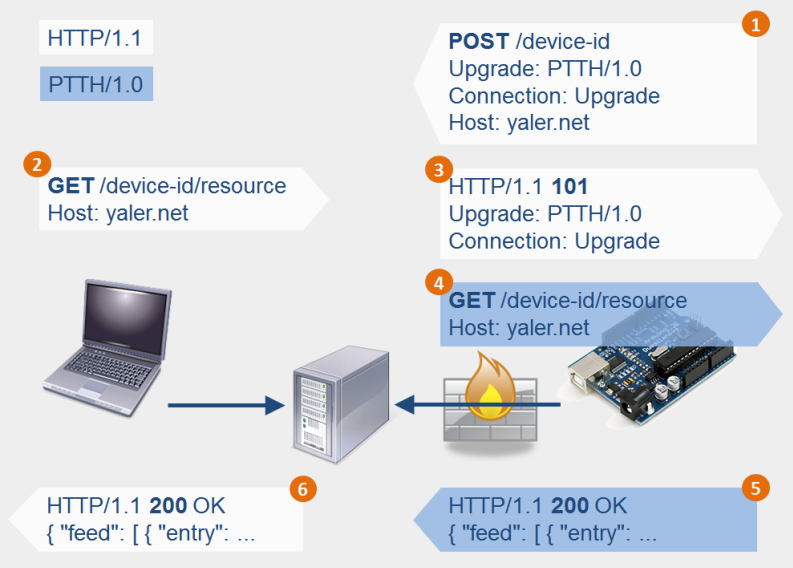
Yaler solves those challenges with a relay:



Introducing a relay enables the device to establish an outbound HTTP connection on a standard port, which is usually allowed by firewalls. Yaler then serves as the public access point for connected devices routing all incoming traffic from clients to the device and back. The code implementing the Yaler protocol on the device is called *Yaler binding* and can be provided as custom code, as a library (e.g. YalerContrib) or as a command line tool (e.g. YalerTunnel).

The Yaler Protocol

The core of the Yaler protocol consists of the following handshake:



The Yaler relay allows a device to publish[1] itself under a unique name, e.g. *device-id*. Whenever a Web client tries to access[2] a resource on the device, Yaler switches the connection to the device to Reverse HTTP[3], and forwards[4] the client request. After processing the request, the device responds[5] and Yaler returns[6] the response back to the client.

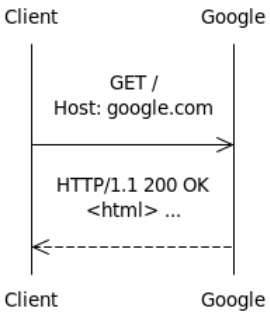
The Reverse HTTP protocol can be thought of as "HTTP with reversed roles". A Web service running on the device does not necessarily have to be aware of the fact that it is accessed via Yaler. Once the initial handshake is done, everything works as expected.

Sequence Diagram Notation

To have a closer look at the Yaler protocol we will use the sequence diagram notation shown on the right.

Normal arrows indicate HTTP requests (e.g. a Web client such as a browser requesting the Google Web page). Dashed arrows denote HTTP responses (e.g. Google replying with a 200 OK response and some HTML in the response body).

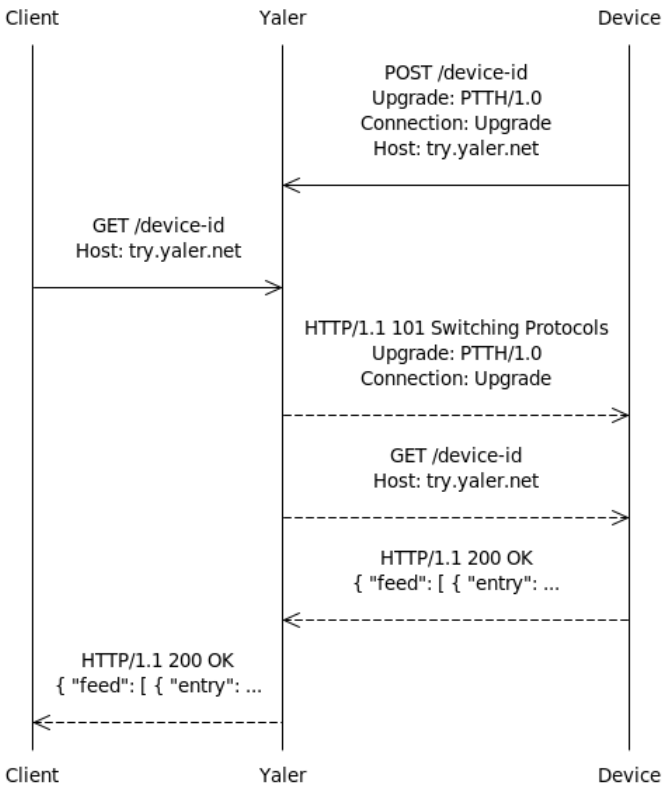
All sequence diagrams in this document have been generated with http://www.websequencediagrams.com/



The following messages together form the Yaler handshake.

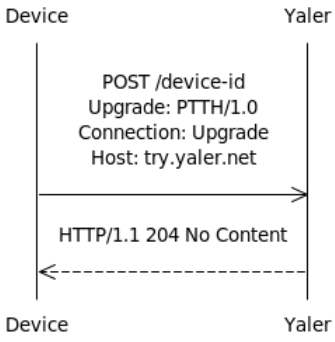
101 Switching Protocols

As described above, the Yaler protocol is initiated by a HTTP POST request over a connection from the device to Yaler, which is kept open until a client request comes in, and answered with status code *101 (Switching Protocols)* as well as the client's request in the response body. The device handles this request and replies to Yaler using the still open connection. Finally, Yaler returns the device's response to the client as a normal HTTP response.



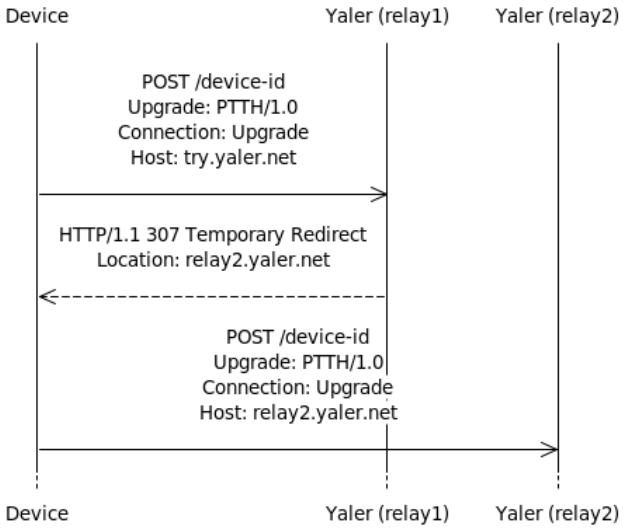
204 No Content

If Yaler did not receive a client request for a published device after a certain amount of time, Yaler responds to the device with a *204 No Content* response. The device then re-publishes itself with the initial HTTP POST request. The connection can stay open during this round-trip.



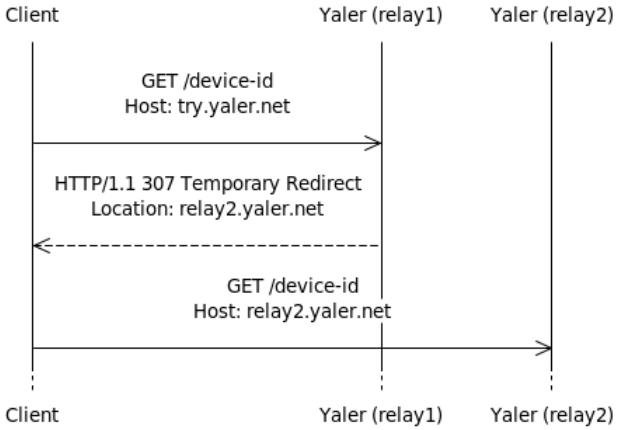
307 Temporary Redirect

To be more scalable and guarantee high availability, Yaler can be organized in clusters of multiple relay instances splitting the load among each other. This means the device have to be able to follow HTTP redirects, in cases where a request was routed to the wrong relay node:



Note that the initial request should always be addressed to the cluster's DNS name (here: try.yaler.net) because individual relay instances might not always be reachable.

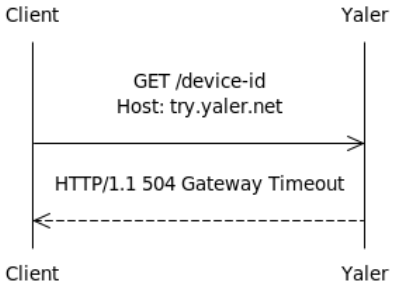
The Web client also has to follow such redirects in order to find the Yaler instance where the desired device is waiting for a connection:



Most existing Web clients do this by default or have at least an option to enable redirects.

504 Gateway Timeout

If a client tries to send a request via Yaler, but no device is present to handle it, Yaler responds with a *504 Gateway Timeout* response.



Connection Timeout

If a connection between the client and the device (via Yaler) has been idle for at least 30 seconds, Yaler closes the connection. This mechanism is necessary to prevent stale connections from taking up too many resources on the relay.

During normal operation this should not be an issue, as most Web services will probably serve requests in less than 30 seconds anyway. But if the connection is used to stream data, precautions have to be taken. A simple way to keep such connections alive is to use message framing and insert null packages into the stream.

Recovering From Abnormal Closure

Abnormal closures can happen for any number of reasons. If the cause is transient, reconnecting might fix the problem. However, if an error affects a large number of connected devices, reconnecting “en masse” might amount to a denial of service (DoS) attack on the relay.

To prevent this, clients should use some form of back off when trying to reconnect after abnormal closures. The first reconnect attempt should be delayed by a random amount of time between 0 and 5 seconds. If the first reconnect fails, subsequent attempts should be delayed by increasingly longer amounts of time, using a method such as exponential back off.