# Yaler – a simple, open and scalable relay infrastructure for the Web of Things

Marc Frei, Thomas Amberg; {frei, amberg}@oberon.ch; Oberon microsystems AG, Zürich, Switzerland

**Abstract**

We demonstrate Web access to embedded computers behind firewalls by showing a system that lets users control an actuator from a standard Web browser. The embedded computer provides a RESTful Web service to control an LED and publishes itself through a simple relay server hosted in the cloud, using a TCP socket and standard HTTP.

**Introduction**

The Web of Things consists of RESTful Web services that measure or manipulate physical properties. In the case leading to our development, a customer who is one of the world's largest manufacturers of transport systems, needed Internet access to each of about 500'000 globally deployed embedded computers. On demand real-time sensor measurements, remote control of operating parameters and being able to push software updates were required to speed up problem diagnosis and repair processes and to enable remote inspection. For a discussion of similar requirements, see [1].

**Architecture**

The following challenges had to be addressed:

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

Addressability - Network address translation (NAT) servers mitigate the shortage in IPv4 addresses by hiding multiple connected devices behind a single public IP address. An embedded computer behind a NAT therefore does not have its own public IPv4 address. It is not individually addressable.

By introducing a simple relay server based on an idea by Nokia [2, 3] and Linden Lab's Reverse HTTP [4] we provide a relatively elegant solution to the above challenges (see Fig. 1).

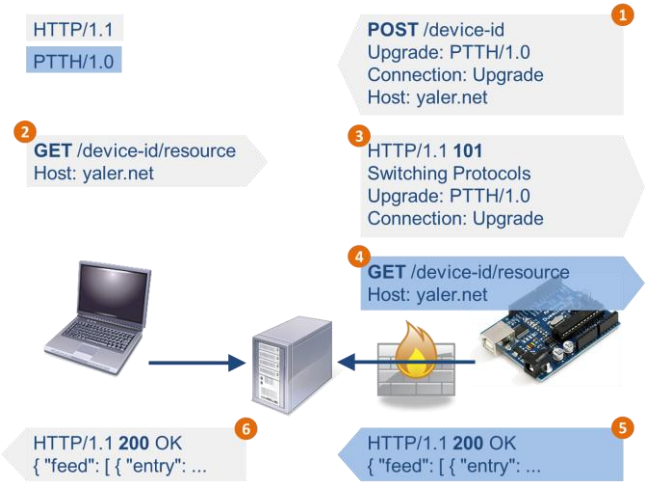


Fig. 1: Protocol to publish (1) and access (2) a device, switch to Reverse HTTP (3), receive (4) and respond to a client request (5, 6)

But using a relay server also introduces two new challenges:

Availability – The relay server becomes a single point of failure, potentially rendering all connected embedded computers inaccessible at once.

Scalability – As the connection from the embedded computer to the relay is left open for a long period of time, a single relay server soon hits the upper limit of concurrently open connections.

**Implementation**

A high performance, single relay server version has been implemented based on Java NIO [5] and hierarchical state machines [6]. It is available for non-commercial use with full source [7]. Currently, we are working on clustering in order to achieve horizontal scalability and high availability. Our design draws heavily on consistent hashing for managing responsibilities within the cluster [8], on a gossip-based group membership service [9], and on a phi-accrual failure detector [10].

**Demo Setup**

We present an Internet-connected Arduino micro controller as a proof of concept device, made accessible through our relay server.

- Arduino Duemilanove (ATmega328) [11] with Ethernet Shield [12] and multicolor LED
- LAN Internet access (RJ45) with DHCP, allowing outbound HTTP(S) connections to standard ports (80, 443) from arbitrary MAC address
- Off-site relay server, hosted on Amazon EC2
- Client computer or mobile phone with Javascript capable browser (Firefox, Safari, IE, ...)
- LAN or Wifi Internet access for client computer or mobile phone

**Evaluation**

The simplicity of our relay server can be qualitatively measured by the lack of external dependencies and the rather small size of the source code. The current version has about 1200 lines of code. Once the planned clustering functionality is included the total will be around 2000 lines of code. This might especially matter as there seems to be a general correlation between code size and security flaws [13].

First tests were made with our relay server deployed on an Amazon EC2 standard instance (large) [14] in Dublin, Ireland and two test programs emulating the client and device side in Zürich, Switzerland. With this setup we reached more than 10'000 concurrently open SSL connections. The duration of the four HTTP transmissions between issuing a client request and the arrival of the corresponding response from the device at the client averaged at about 0.1 seconds.

**References**


[1] http://www.iot2008.org/prg/slides/oelsner.pdf
[2] http://research.nokia.com/research/projects/mobile-web-server
[3] http://research.nokia.com/files/NRC-TR-2006-005.pdf
[4] http://wiki.secondlife.com/wiki/Reverse_HTTP
[5] http://oreilly.com/catalog/9780596002886
[6] http://www.state-machine.com/psicc2/
[7] http://yaler.org/
[8] http://portal.acm.org/citation.cfm?id=258660
[9] http://portal.acm.org/citation.cfm?id=1529974.1529983
[10] http://www.computer.org/portal/web/csdl/doi/10.1109/RELDIS.2004.1353004
[11] http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove
[12] http://www.arduino.cc/en/Main/ArduinoEthernetShield
[13] http://www.daemonology.net/blog/2009-09-04-complexity-is-insecurity.html
[14] http://aws.amazon.com/ec2/#instance