Zeroties: a Publish/Subscribe Service for Applications in Local Ad-Hoc Networks

Chris Satterfield Felix Grund
University of British Columbia
Vancouver, BC

ABSTRACT

1 INTRODUCTION

Throughout the last decade we have seen the Internet become the most commonly used infrastructure for communication: regardless of physical location, people and their devices communicate via messengers and VoIP, and collaborate via live editing tools like, for example, Google Docs, Sheets, and, Slides. At the same time, we have seen some applications communicating over local area networks become increasingly common for certain scenarios like home entertainment (e.g. Google Chromecast, Apple Bonjour, Spotify Connect) or Wi-Fi printers. With the advancements of "smart" devices and the "Internet of Things" (IoT) it is likely that this trend will grow beyond these currently still narrowly scoped application domains.

The adoption of standards that eliminate the burden of manual configuration of network devices further contributes to this movement. One such standard that has received widespread usage is Zero-configuration networking [4] and its protocols mDNS [2] for service advertisement and DNS-SD for service discovery [1]. Using the *Zeroconf* protocols, devices can publish named services in the local network and discover such services automatically in an ad-hoc fashion. While most applications for Zeroconf networks are shipped with specific hardware (e.g. Google Chromecast dongle) there have recently been attempts to provide software environments for developers to enable them write their own applications on already existing hardware infrastructure. One such application was Mozilla FlyWeb[1], an addon for the Firefox browser that made it possible to advertise and discover services from within Web applications through a JavaScript API. In a previous project, we created *Successorships* [2], a JavaScript library exposing an easy-to-use API to build fault-tolerant Zeroconf web applications. Successorships was built on top of Mozilla Flyweb as one main part of its architecture. This decision confronted us with significant problems:

- FlyWeb had been declared abandoned by Mozilla even before we finished our work on our library.
- The implementation of the Zeroconf protocols in FlyWeb was slow to a degree that made our library fairly unusable in practice.
- FlyWeb contained a bug that made our library usable on MacOS only.

To overcome our troubles in Successorships, we introduce Zeroties, a platform-independent asynchronous publish/subscribe service for Zeroconf advertisement and discovery. We carefully reviewed different publish/subscribe designs [3] and implemented a communication scheme based on *asynchronous notifications*. The operations exposed by Zeroties are as follows:

- **Publish**: publish a Zeroties service and advertise it in the local network
- **Subscribe**: listen for updates on the list of available Zeroties services

Zeroties ships with two components: (1) a standalone OS-level application, and (2) addons for Chrome and Firefox that connect to this application. With our addon implementations for Chrome and Firefox we aim to show that our approach translates well between different browsers and does not share the restrictions of FlyWeb. Our browser addons expose an API to web applications that comprises the full service/discovery functionality of Zeroconf. As a result, we have successfully eliminated the ties that prevented Successorships from usage in practice.

To evaluate Zeroties, we first created the webapp-based presentation for this project using Successorships in combination with the Zeroties app and the addon for Google Chrome. Running the presentation on Chrome proved that we successfully broke ties with FlyWeb and Mozilla Firefox. Furthermore, we could see in this example application that recovery from server failures was significantly faster than with the previous version based on FlyWeb. To evaluate these findings empirically, we repeated the performance measurements from the Successorships project, both with the previous version based on FlyWeb and the new version based on Zeroties. We simulated XXX server failures and subsequent recoveries in a small mobile network and found that the system recovered in average about three times as fast with Zeroties than with FlyWeb.

In summary, we make the following contributions:

- Zeroties, a asynchronous standalone publish/subscribe service for Zeroconf applications.
- Addons for Chrome and Firefox that make this service available to web applications.
- An empirical evaluation based on a Zeroties sample application indicating significant performance improvements.

The remainder of this paper is organized as follows: Section 2 provides some background and motivation on why the idea for Zeroties came to be. Section 3 presents the system model and design goals of Zeroties, before Section 4 describes its implementation. We evaluate Zeroties in Section 5 and suggest limitations and future work in Section 6. Section 7 situates our work in the context of related research and Section 8 concludes the paper.

2 BACKGROUND AND MOTIVATION

We have been compelled by the idea to make the Zeroconf protocols available to web applications. For example, consider the scenario of a group of people, each with their own device, collaborating on a Google Doc document. Every keystroke in this document will need

[1] https://wiki.mozilla.org/FlyWeb
[2] https://github.com/ataraxie/successorships

to be sent to a Google web server and then be "pushed" too all other group members. Depending on geological location, this pattern of communication can impose roundtrips around the globe and lead to significant delays, despite the physical proximity of the devices. The fact that not everybody around the globe is equipped with the newest network infrastructure makes this even more proplematic.

Zeroconf web applications provide a solution to this problem. In the described Google Doc scenario, one person in the group would access the the Google Doc on the Internet and subsequently become a server in the local network. Her device would then serve the resources it fetched from the actual web server to local clients who maintain a channel to this local server. Effectively, only one Internet connection is required for the whole group, rather than one connection for every device[3], because all clients except the locally serving client access the original application (in this case Google Docs). Figure 1 illustrates this shift in architecture.

Evidently, this pattern of communication has one major problem: it has a single point of failure, namely the client acting as a local server. Solving this problem was our motivation behind Successorships; whenever the currently serving client failed, a new client in the network was elected as the new server and connectivity in the network was re-established automatically. Successorship thus provided a framework to build fault-tolerant Zeroconf web applications with an easy-to-use JavaScript API. However, our decision to built it on Mozilla FlyWeb proved to be a major limitation. Not only did we limit usage of our library to one specific browser vendor; we found out during the course of the project that Mozilla had already deprecated FlyWeb in favor of other priorities. To make things worse, FlyWeb had a dependency to a Firefox core component (the module responsible for launching a web server within the browser) that was only shipped in a range of versions of the developer edition of Firefox. In addition to this platform dependency that would, in essence, prevent our framework to be used in practice, the implementation of the Zeroconf protols in FlyWeb was incredibly slow. According to our evaluation, recovery from failure of the currently serving client took more than 30 seconds in many cases. With a continuously growing 'Limitations and Future Work' section, we were finally struck by the discovery of an unresolved issue in FlyWeb that made our framework only work on MacOS. Despite our thoughtful and eager motivations, we had built a tool that nobody would ever use in practice. Hence the call to get rid of this dependency entirely and provide our own layer below Successorships.

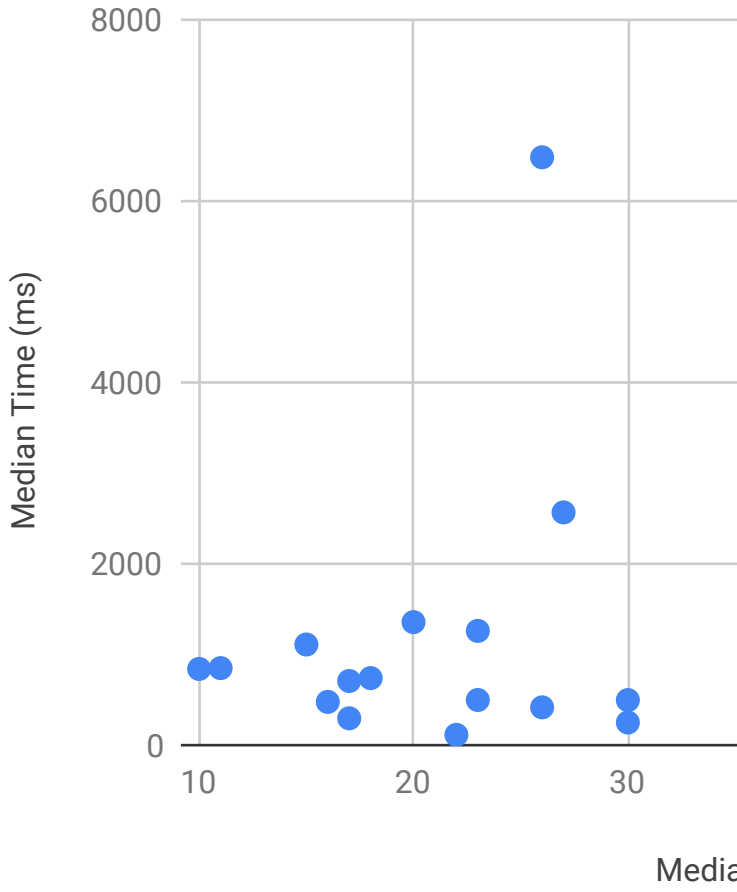

Figure 1: Architecture shift: on the left side is the traditional web application architecture where all devices are connected to a web server through the Internet. On the right is the Zeroconf architecture where only one client requires such a connection and becomes a server to all other clients in the network.

3 APPROACH

3.1 Environment and System Architecture

Figure 2 illustrates the environment, placement of Zeroties, and the communication between components. We describe the parts in the figure refering to the underscored numbers as follows.

(1) **Router**: we use the protocols provided by Zeroconf (??) for Zeroties. In essence, this means that an authoritative list of currently available Zeroties services (as a subset of all Zeroconf services) can be obtained from the router at any given time.
(2) **Hosts**: there can be an arbitrary number of hosts in our local network. A host can be any device in the network with an instance of Zeroties running (i.e. any device with an OS capable of Zeroconf/DNS-SD protocols).
(3) **Zeroties App**: the instance of Zeroties running on this device. This is an OS-level application maintaining a list of services. This list is a mirror of the service list obtained from the router by means of the Zeroconf protocols. The list of services constitutes the state of the distributed system [4]

[3]Obviously, this also has the advantage that only one device needs to have Internet connection in the first place. This is can be very interesting especially in scenarios where only dedicated participants in a network have such a connection, for example due to security restrictions.

[4]Note that there is a clear line between Successorships and Zeroties: Zeroties does not know about the arbitrarily complex state of Successorships or any other app that builds on Zeroties.

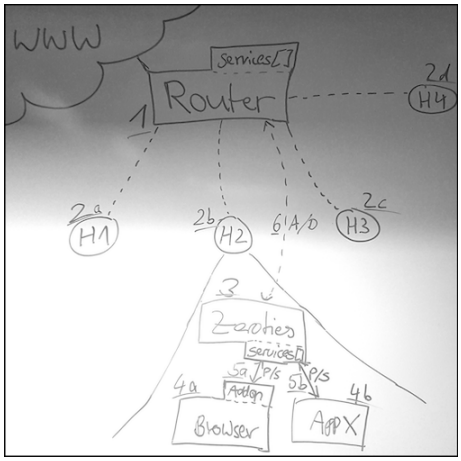

Figure 2: System architecture and environment in Zeroties.

(4) **Browser Addons**: we require browser addons that mediate between web applications and the Zeroties app. These addons expose a JavaScript API to apps built on top of Zeroties. We have implemented addons for Google Chrome and Mozilla Firefox that constitute such a mediator and we have run Successorships apps with these addons (see Section 5). These addons use Zeroties as a publish/subscribe service. Note that the Zeroties application itself is not restricted to these addons. They are only required for web applications running in browsers. Different applications can use the Zeroties service without these components.
(5) **Publish/Subscribe operations**: applications built on top of Zeroties interact with Zeroties using the common publish/-subscribe operations. *Publish* is the publishing of a Zeroties service and *subscribe* makes the application receive changes to the list of available Zeroties services.
(6) **Advertisements and service discovery**: the Zeroties application maintains a list of Zeroties services that is obtained from the router. In a sense, Zeroties acts as a middleware between the router and applications built on top of Zeroties (applications using Successorships as framework being one example). Whenever a Zeroties application publishes a service using the Zeroties publish/subscribe API, this will result in the publishing of a service in the network (service advertisement). The second part of this communication channel is service discovery. We describe both service advertisement and service discovery in more detail in Section 3.2

3.2 Design Goals

Zeroties is on system models as in the example described in Section 2: local ad-hoc network applications. We claim that there is a dedicated set of such applications, in addition to the document collaboration example, e.g.

- A project presentation at a meetup where the audience can connect to the presentation and interact with it.
- An application for printer control in an office
- An application for the heating system of a hotel
- Multiplayer mode for browser-based games

Such applications have in common a limited number of nodes (generally < 100) and comparably lax requirements for time-to-recovery from system failures: certain downtimes can tolerated as soon as a consistent state is reached eventually. For example, a running application converging to a consistent state within a time frame of multiple seconds after failures is acceptable. Our particular goals behind Zeroties are described in the following paragraphs.

Replication.

In the face of published and removed (or failed) services we want the system to converge to a consistent list of services within 10 seconds.

3.3 Zeroties API

4 IMPLEMENTATION

5 EVALUATION

6 LIMITATIONS AND FUTURE WORK

7 RELATED WORK

[5]

8 CONCLUSION

REFERENCES

[1] S. Cheshire and M. Krochmal. Dns-based service discovery. RFC 6763, RFC Editor, February 2013.
[2] S. Cheshire and M. Krochmal. Multicast dns. Technical report, 2013.
[3] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.
[4] E. Guttman. Autoconfiguration for ip networking: enabling local communication. *IEEE Internet Computing*, 5(3):81–86, May 2001.
[5] S. G. Hong, S. Srinivasan, and H. Schulzrinne. Accelerating service discovery in ad-hoc zero configuration networking. In *Global Telecommunications Conference, 2007. GLOBECOM'07. IEEE*, pages 961–965. IEEE, 2007.