Enhancing Discovery with Liveness

Maarten Bodlaender, Jarno Guidi, Lex Heerink
Philips Research, The Netherlands
Prof. Holstlaan 4, 5656 AA Eindhoven,
{maarten.bodlaender, jarno.guidi, lex.heerink}@philips.com

Abstract – **Discovery protocols like SSDP, SLP, Rendezvous and Jini allow fast detection of devices. They are typically less suitable for fast detection of disappearing devices.**
This paper describes the design of a liveness protocol that quickly detects that devices have left the network. Mechanisms that improve important properties like reliability and efficiency over wireless, scalability and high-dynamics are highlighted. The performance of the liveness protocol is analyzed: it has significantly less message overhead than alternative solutions.

Keywords: discovery, SSDP, Rendezvous, SLP, Jini, liveness, heterogeneity, wireless, dynamics, scalability, failure detection.

I. Introduction

This paper describes how discovery protocols like the SSDP protocol of the UPnP™ 1.0 standard, SLP, Rendezvous and Jini can be enhanced with liveness.

Mobile devices roam between home, office and on-the-move. These environments are characterized by a multitude of devices in the productivity domain (phones, PCs and printers), the entertainment domain (TVs and audio) and home-control domain (lighting and thermostat control). Especially in consumer electronics, when a device leaves, users expect it to be quickly removed from user interfaces of remaining devices.

From this description a number of requirements on the underlying middleware can be derived.

Scalability – The number of devices may range from a few at home to thousands in a corporate network.

Wireless-friendly – Wireless networks usually are less reliable (especially multicast) and have less bandwidth than wired networks.

High-dynamics – In wireless networks with mobile devices, the network topology changes more frequently than in wired networks with stationary devices.

Limited footprint and complexity – The middleware should be implementable on embedded devices that have limited processing capabilities and memory available.

To enable client-applications (“clients”) to keep their view of the network up-to-date, devices can announce when they are about to leave the network. Often, network disconnection occurs before the announcements are sent. Therefore clients cannot rely on announcements of leaving devices.

A common solution is that clients continuously poll devices, for example by using discovery messages. The liveness protocol extends on the notion of polling, but avoids the scaling problems that result from straightforward polling.

A. *Related work: discovery protocols*

SSDP [1] is used in UPnP 1.0. Clients send multicast searches to which devices respond with unicast. Devices send multicast announcements of their presence, with periods that are recommended to be 30 minutes or more.

Rendezvous [2] is based on a version of multicast DNS. It uses multicast searches, together with multicast replies and recommended time-to-live values of resource records of 120 minutes. To scale to larger networks, integration with the normal DNS system is proposed.

SLP [3] uses both multicast search and unicast queries to infrastructure servers. Devices define the lifetime of their own service entries in clients’ caches and that lifetime can be up to 18 hours. Clients cannot choose this lifetime.

Jini [4] is based on Java. Clients use both multicast searches and unicast queries to lookup servers. Devices announce themselves using multicast with recommended intervals of 2 minutes.

In all of the discussed discovery protocols, detecting that a device has suddenly disappeared can take several minutes, unless clients resort to polling. The liveness problem is defined as fast detection by a set of clients that a device has left.

B. *Related work: Failure detection*

Failure detection protocols aim to identify when in a group of nodes, one or more nodes stops executing correctly. Liveness can be modeled as a special case of failure detection by considering that in a dynamic group of nodes, only failure to respond by part of the group (the device) is detected.

In Gossip-like failure detection protocols [5], each member maintains a list of other known members. Periodically the members exchange their lists. This increases the load on the network.

The randomized failure detector algorithm described in [6] assumes that the group is static and known to all members (and thus cannot deal with dynamic environments). Each member periodically pings a random other member. If no response is received, the member does not retry but out-sources this action to a randomly selected subgroup of members. This improves scalability and puts an equal load on all members.

The simple group membership failure detection protocol in [7] targets static groups, and requires each member to periodically broadcast ‘alive’ messages until another member fails. Member failure is detected by a timeout on receiving ‘alive’ messages.

II. LIVENESS

The goal of the liveness protocol is to enable fast detection that a device has disappeared, while guaranteeing that devices are not overloaded by polling messages. It is composed of two mechanisms that reinforce each other: the Liveness Ping protocol and the Proxy-bye protocol.

The Liveness Ping protocol is defined between clients and devices. If a client wants to check a particular subset of devices, it has to initiate the liveness protocol for each device. Clients can change that subset at will. A client is not forced to use liveness.

Liveness Ping – A client regularly checks the presence of a device by sending a Lping message using unicast UDP. The device replies with an Lreply message in a UDP unicast packet. If the client does not receive a reply before a timeout, it retries three times. If these also time out, the client concludes that the device is unreachable and the Proxy-bye protocol is invoked (see below). The use of unicast instead of multicast communication reduces network and device load in larger networks. To support high dynamics in resource-constrained devices, UDP is preferred over TCP, as TCP is state-full and has timeout/retry behavior that is unsuitable for liveness.

The Lping message contains the client address. Lreply messages contain the PINGCOUNT and addresses of two other clients (see below for semantics).

Pingload Control – To avoid that the number of Lping messages per second (the 'pingload') overloads the device, a mechanism for bounding the pingload is added.

When a device receives an Lping it increases an internal counter (PINGCOUNT) by a quantity (PINGINCREASE). The resulting value of PINGCOUNT is returned in the Lreply. Clients maintain the last pingcount value they received from the device (LASTPINGCOUNT). Furthermore, they time the interval between consecutive liveness pings (PERIOD). Using these values, when a client receives an Lreply, it calculates the pingload of the device over the past period:

$$PINGLOAD = \frac{PINGCOUNT - LASTPINGCOUNT}{PERIOD}$$

To limit the pingload of a device, clients are required to have at least a certain DELAY between two consecutive liveness pings. The value of DELAY is specified by a set of rules, but clients are allowed to wait longer: the DELAY is a lower bound for PERIOD. When a client detects that the pingload on a device exceeds threshold HT, it has to increase its delay, to lower the pingload. When a client detects that the pingload falls below threshold LT, it is allowed to ping more often. This is captured by the following adaptation rules:

(R1) If $PINGLOAD > HT$ then $DELAY = 2 \times DELAY$

(R2) If $PINGLOAD < LT$ then $DELAY = 2 \times DELAY / 3$

Figure 1 shows an example with instantiated values for the different thresholds. Client CP1 is pinging the device once every second. After CP2 starts checking the same device, CP1 detects that the high threshold HT has been exceeded and consequently doubles its DELAY.

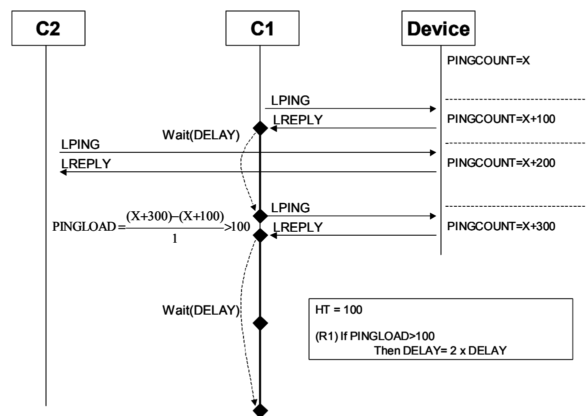

Figure 1 - C1 adapts its DELAY

Self-healing – In dynamic environments, a sudden reduction of clients can lead to very low pingloads. It can take a long time until the remaining clients ping again. To limit this effect, the maximum delay is bounded (MaxD). The final adaptation rule R1 becomes:

(R1) If $PINGLOAD > HT$ then $DELAY = \min(MaxD, 2 \times DELAY)$

Similarly, a sudden influx of new clients can temporarily push the pingload above the high threshold. To limit this effect, a minimum delay (MinD) is introduced. Constants MaxD and MinD are expressed in seconds. The final adaptation rule R2 becomes:

(R2) If $PINGLOAD < LT$ then $DELAY = \max(MinD, 2 \times DELAY / 3)$

PingIncrease – Devices can tune their pingload by choosing the value of the variable PINGINCREASE. When the protocol stabilizes, the maximum number of Lping messages per second that the device serves is:

$$MaxPPS = \frac{HT}{PINGINCREASE}$$

In Figure 1, the device receives no more than 1 Lping per second, since HT is 100 and PINGINCREASE is 100.

Zero-message overhead membership – Piggybacked on the Lreply messages, address information (IP address and UDP port) is exchanged between clients (used in Proxy-Bye, see below). To enable this exchange, each device maintains information about the last two distinct clients that sent an Lping, and returns this information in the Lreply. No direct communication amongst group members is required.

Device behavior – The behavior of the device is described by the following pseudo-code.

```
1  For each incoming Lping from client C Do
2  PINGCOUNT = PINGCOUNT + PINGINCREASE
3  Send Lreply containing PINGCOUNT and addresses of last 2 clients
4  If C is not one of the last 2 clients Then
5      Remove information about the oldest client
6      Store information about client C
```

The simplicity of computation and data structures fulfills the limited footprint and complexity requirement.

Proxy-bye – Whenever a client detects the device is unreachable, it notifies other clients by sending proxy-bye

messages. A proxy-bye contains the address of the device and the LASTPINGCOUNT received by the client that generates the proxy-bye. This information identifies the proxy-bye, thus enabling the discard of duplicate proxy-byes.

Spreading effect – Clients send proxy-byes to all known clients that are checking the same device. If at least one client is on the local link, the proxy-bye is multicast. In addition, off-link clients receive a unicast.

With high probability the forwarding graph has a depth of log(#clients), allowing fast propagation, even across the Internet. After each liveness ping, the forwarding links between clients are automatically updated to reflect the latest set of interested clients.

Upon receiving a proxy-bye, a client first decides whether to ignore it by checking whether it is a duplicate. If the client does not ignore the message, it sends a single Lping to the device. If the device does not react, the client considers it unreachable and forwards the proxy-bye. This protects against out-of-order delivery and malicious proxy-bye messages.

```
1  If device already considered unreachable then ignore message
2  If LASTPINGCOUNT already received then ignore message
3  Send an Lping to the device
4  If an Lreply is received before timeout then ignore message
5  Consider the device unreachable
6  If the proxy-bye was received through unicast Then
7      If other clients on the same link are known Then
8          Multicast proxy-bye local link
9  Unicast proxy-bye to off-link clients
```

Forgetting old clients – In case client C1 receives information about client C2 at time T, it has to keep the information about C2 until T+MaxD. Afterwards, C2 is old and can be removed.

III. Performance Evaluation

The scalability of the liveness protocol is analyzed and compared with two alternative solutions: UDP pinging (similar to unicast SSDP searches) that requires 2 packets, and pinging with a TCP connection (similar to unicast Jini searches) that requires 10 or more packets.

The analysis assumes that #C clients check a device once a second. Furthermore, it is assumed that there is no message loss and the set of clients is static, no clients join or leave. The total number of packets per second is calculated.

The overall number of packets exchanged in the UDP and TCP solutions are given by the following formulas:

$$\#UDP=2\times\#C$$

$$\#TCP=10\times\#C$$

The liveness protocol bounds the pingload of a device to MaxPPS. To ensure self-healing in dynamic systems, clients ping at least every MaxD secs. The final formula is:

$$\#Liveness=2\times\max\left(\frac{\#C}{MaxD},\min\left(\frac{\#C}{MinD},MaxPPS\right)\right)$$

In Figure 2, MinD=1, MaxD=30 and MaxPPS=4. On the horizontal axis is the number of clients, on the vertical axis is the number of packets per second.

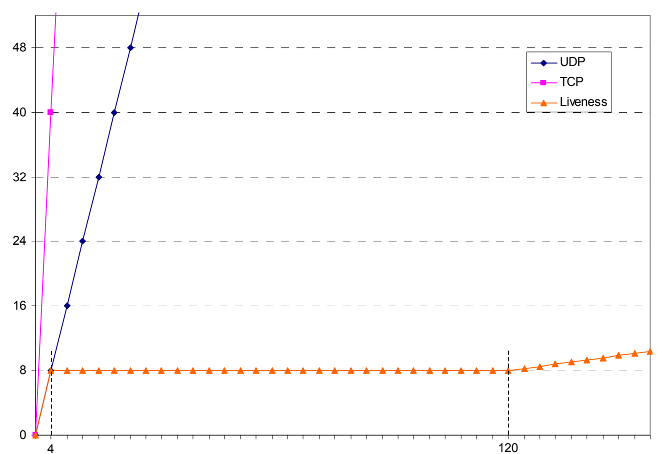

Figure 2 – Scaling the number of clients

Both the UDP and TCP solution grow linearly (factor 2 and 10) with #C. The liveness protocol has a linear growth until MaxPPS×MinD (#C=4 in Figure 2), then a constant load until MaxPPS×MaxD (#C=120 in Figure 2). Finally, it shows a linear growth (factor 0.067 in Figure 2) that is MaxD times smaller than UDP.

IV. Conclusions

A Liveness protocol has been proposed that can enhance discovery. It deals with fast detection of disappearing devices. It consists of two parts: a liveness ping, and a proxy-bye.

The paper has shown that the liveness ping has good scaling characteristics compared to straightforward heartbeat solutions, achieving packet reductions of a factor 30 and more. This scaling is achieved by a self-healing pingload control mechanism that gives average load guarantees on devices.

Piggybacked on the liveness ping is a zero-message overhead membership mechanism, which enables proxy-byes between clients that are on different links.

The Proxy-bye protocol combines the desire to have an effective notification on a single link with the need to scale to larger networks. It uses a mix of multicast and unicast that quickly spreads proxy-byes to all interested clients. Finally, re-checking liveness before propagating the proxy-bye protects against false or old proxy-byes.

References

[1] *UpnP device architecture 1.0*, www.upnp.org.

[2] *The Rendezvous protocol*, www.apple.com/macosx/pdfs/Rendezvous_TB.pdf.

[3] E.Guttman, C.Perkins, J.Veizades, M.Day, *Service Location Protocol, Version 2*, IETF, RFC2608, 1999, www.rfc-editor.org/rfc/rfc2608.txt.

[4] W.K. Edwards, *Core Jini*, Prentice Hall, June 1999.

[5] R. van Renesse, Y. Minsky, M. Hayden, *A Gossip-style failure detection service*. Middleware '98: IFIP Int. Conference on Distributed Systems Platforms and Open Distributed Processing, pp. 55-70, Springer Verlag, 1998.

[6] I. Gupta, T.D. Chandra, G.S. Goldszmidt, *On scalable and efficient distributed failure detectors*. Proc. 20th ACM Symp. on Principles of Distributed Computing, pp.170-179, 2001.

[7] M. Raynal, F. Tronel, *Group membership failure detection: a simple protocol and its probabilistic analysis,* Distributed Systems Engineering 6 (3), pp. 95-102, 1999.