Are You Still There?
— A Lightweight Algorithm To Monitor Node Presence in Self-Configuring Networks —

Henrik Bohnenkamp[a], Johan Gorter[a], Jarno Guidi[b] and Joost-Pieter Katoen[*,a,c]

[a] University of Twente, 7500 AE Enschede, the Netherlands

[b] Philips Research, Prof. Holstlaan 4, 5656 AA Eindhoven, the Netherlands

[c] Aachen University of Technology, 52056 Aachen, Germany

[*] Contact author. E-mail: katoen@cs.utwente.nl, tel: +31-53-4895675

Abstract

This paper is concerned with the design of a distributed algorithm to monitor the availability of nodes in self-configuring networks. It is intended as extension to service discovery protocols such as SSDP, SLP, Rendezvous and Jini that allow for fast node detection. The simple scheme to regularly probe a node—"are you still there?"—may easily lead to over- or underloading. The essence of our algorithm is therefore to automatically adapt the probing frequency. We show that a self-adaptive scheme to control the probe load, originally proposed as an extension to the UPnP^{TM} (Universal Plug and Play) standard, leads to an unfair treatment of nodes: some nodes probe fast while others almost starve. An alternative distributed algorithm is proposed that overcomes this problem and that tolerates highly dynamic network topology changes. The algorithm is very simple and can be implemented on large networks of small computing devices such as mobile phones, PDAs, and so on. The distributed algorithms are modeled using the formal modeling language MODEST *and are analyzed by means of the discrete-event simulator of the* MÖBIUS *tool.*

Keywords: discrete-event simulation, distributed algorithms, formal specification, performability evaluation, plug-and-play networks, self-configuring networks

1. Introduction

Home and office environments typically consist of a multitude of devices, ranging from desktop PCs and televisions to remote controls, thermostat controls and mobile phones. Such devices no longer act as stand-alone entities, but interact with each other via so-called plug-and-play networks. These networks are highly dynamic (devices can join and leave at a high frequency), typically contain hundreds of devices, and are self-configuring—there is no centralized device that manages the network. For economic reasons, the protocols running these networks should be simple to implement and should impose low memory and processing requirements. The distributed management and control of such networks is a challenging issue that currently is faced by, amongst others, the UPnP^{TM} Forum (Universal Plug and Play, `www.upnp.org`) in which over 700 companies world wide (such as consumer electronics, home automation, and IT companies) are involved.

This paper is considered with the *membership management* of these highly dynamic, self-configuring networks. In particular, we are concerned with the design and analysis of a distributed algorithm that is aimed to maintain (and to some extent disseminate) *up-to-date* information about the presence (or absence) of devices (also called nodes). That is to say, the distributed algorithm allows for the monitoring of the availability of a node by other nodes. Normally, when a node goes off-line, it informs other nodes by sending a bye-message, but if it suddenly becomes unavailable, no such indication is sent, and the studied algorithm comes into play. An important requirement is that the absence of nodes should be detected quickly (e.g., in the order of one second) while avoiding to overload nodes. As the basic mechanism of our membership management algorithm is to simply regularly check (i.e., probe) whether a node is still present, we refer to it as the probe protocol.

Related protocols are failure detection and monitoring protocols. Failure detection protocols [15, 17] aim to identify whether in a group of nodes, one or more nodes stop executing correctly. In our setting there are two types of nodes, *devices* and *control points* and only the failure of a single type of node is relevant (devices). Monitoring protocols involve the aggregation of various sorts of data such as finding aggregated statistics (like average, sum and the like) that are distributed among nodes in the network [14, 16]. The probe protocol considered in this paper shares many aspects with the newscast computing approach [14]: it maintains up-to-date (membership) information in a self-organizing way, without any central intervention, in a dynamically changing and large-scale distributed environment. In particular, it continues to operate properly without manual intervention under the—according to varying patterns—joining and (un)intentional leaves of devices.

This paper describes two probe protocols for membership management, presents the essential ingredients of their formal models, and discusses analysis results obtained by means of discrete-event simulation. The simplest scheme one could consider is to regularly probe a device—"are you still there?". This scheme, however, easily leads to over- or underloading of devices. The essence of our algorithms is therefore to automatically adapt the probing frequency. We show that a self-adaptive scheme to control the probe load of a probed device leads to an unfair treatment of control points (CPs): some CPs probe fast while others almost starve. This protocol [3] can be implemented as an (proprietary) extension of the UPnP^{TM} (Universal Plug and Play) standard.

The main contribution of this paper is an alternative distributed algorithm that overcomes this problem and that tolerates highly dynamic network topology changes. The simplicity of the algorithm allows for the implementation on large networks of small computing devices such as mobile phones, PDAs, and so on. Although the presented distributed algorithm is intended as an extension to service discovery protocols (such as SSDP [19], SLP [13], Rendezvous and Jini [11]) in home and office environments, its principles can also be used for neighbor unreachability detection in IPv6 and for membership management in other dynamically changing distributed environments.

The paper is further organized as follows. Section 2 describes the probe protocol as proposed by Bodlaender *et al.* [3] and focuses in particular on the self-adaptive scheme to regulate the probe frequency. Section 3 presents the main details of the model of this probe protocol and discusses its analysis results. Section 4 and 5 describe the alternative probe protocol, its formal model and analysis results, respectively. Finally, Section 8 concludes.

2. The Self-Adaptive Probe Protocol

Two types of nodes are distinguished: simple nodes (*devices*) and somewhat more intelligent ones, called *control points* (CPs). The basic protocol mechanism is that a CP continuously probes a device that in turn replies to the CP, if it is still present. The CP adapts the probing frequency automatically in case a device tends to get over- or underloaded. The CPs are dynamically organized in an overlay network by letting the device, on each probe, return the ids of the last two (distinct) processes that probed it. On detecting the absence of a device, the CP uses this overlay network to inform all CPs about the leave of the device rapidly. The latter information dissemination phase of the protocol is inspired by Chord [18]. It is further considered in Section 6.

Device behaviour. A device maintains a probe-counter *pc* that keeps track of the number of times the device has been probed so far. On the receipt of a probe, this counter is incremented by the natural Δ, and a reply is sent to the probing CP with as parameters the (just updated) value of *pc*, and the ids of the last two distinct CPs that probed the device. The latter information is needed to maintain the overlay network of CPs[1], whereas the returned value of *pc* is used by CPs to estimate the load of the device. Δ is device-dependent, and typically only known to the device. Its value may change during execution. A CP cannot therefore distill the actual probing frequency of a device, but only its own perceived probing frequency, the *experienced probe load*, denoted as L_{exp} .

The factor Δ is used by a device to control its load, e.g., for a larger Δ, CPs consider the device to be more (or even over-) loaded sooner, and will adjust (i.e., lower) their probing frequency accordingly resulting in a lower probe-load at the device. This works as follows. We assume a so-called *ideal probe load*, denoted by L_{ideal}, known to all CPs and devices. L_{ideal} is the ideal probe load that a device should be able to cope with, but it is a reference constant only and must have an unrealistic high value. CPs try to keep their L_{exp} values close to L_{ideal}.

In addition, a device is assumed to have a private *nominal probe load* L_{nom}, representing the actual load the device can or wants to maintain during normal operation. Defining now

$$\Delta = \left\lceil \frac{L_{ideal}}{L_{nom}} \right\rceil ,$$

enables the device to slow down the CPs (assuming $L_{ideal} \gg L_{nom}$) as it informs the CPs that it is a factor Δ more "busy" than is really the case. Note that the optimal probe frequency for a CP with k CPs in total is L_{nom}/k, but as neither k nor L_{nom} are known to a CP, the adaptive mechanism described below must take care of keeping the probing frequencies of the CPs reasonably close to this optimum.

[1]By returning two distinct CP ids, the overlay network forms a tree with depth $\log_2 N$ where N is the number of CPs, with a high probability.

CP behaviour. The behaviour of a CP is more intricate. The basic mechanism for communicating with a device is a bounded retransmission protocol (à la [8]): a CP sends a probe ("are you still there?"), and waits for a reply. In absence of a reply, it retransmits the probe (cf. Fig. 1). Otherwise, the CP considers the reply as a notification of the presence of the device, and continues its normal operation. Probes are retransmitted maximally three times. If on none of the four probes a reply is received, the CP considers the device to have left the network, and starts to disseminate this information to other CPs using the overlay network. The protocol allows to distinguish between the timeout value TOF after the first probe and the timeout value TOS after the other (maximally three) probes. Typically, TOS $<$ TOF. In all simulation studies in this paper TOF equals 0.022 (i.e., two times the round-trip delay of the considered network + the maximal computation time of the device), and TOS equals 0.021 (1 times round-trip delay + maximal computation time of the device).

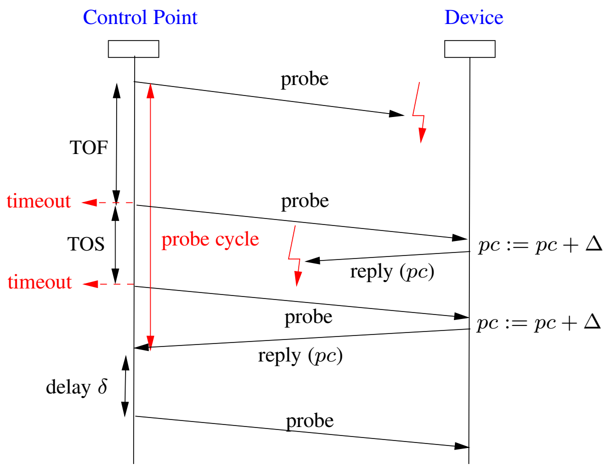

Figure 1. Elementary protocol mechanism

Adapting the probing frequency. Let us now consider the mechanism for a CP to determine the probing frequency of a device. A *probe cycle* starts with a probe and ends with either a reply (a successful probe) or with a timeout after three retransmissions of the probe (an unsuccessful probe), cf. Fig. 1. Let δ be the delay between two consecutive probe cycles. There is a minimal and maximal inter-probe-cycle delay, i.e., a CP has to obey:

$$\delta_{min} \leqslant \delta \leqslant \delta_{max}$$

for constants δ_{min} and δ_{max} with $\delta_{max} \gg \delta_{min}$. The value of δ is adapted to keep the probe load L_{exp} as perceived by a CP "close to" the ideal probe load:

$$\frac{1}{\beta} \cdot L_{ideal} \leqslant L_{exp} \leqslant \beta \cdot L_{ideal} \text{ for } \beta > 1.$$

Note that β is just a constant. Assume the CP receives a reply on a probe with probe-count *pc* at time t. (In case of a failed probe, the time at which the retransmitted probe has been sent is taken.) The next reply is received at time $t' > t$, and let *pc'* be its returned probe-count. $t' - t$, thus, is the time delay between two successive successful probes. The experienced probe load is now given by:

$$L_{exp} = \frac{pc' - pc}{t' - t}.$$

The inter-probe-cycle delay δ is adapted according to the following scheme, where δ' and δ refer to the new and current value of δ, respectively:

$$\delta' = \begin{cases} \min\left(\alpha_{inc} \cdot \delta, \delta_{max}\right) & \text{if } L_{exp} > \beta \cdot L_{ideal} \\ \max\left(\frac{1}{\alpha_{dec}} \cdot \delta, \delta_{min}\right) & \text{if } L_{exp} < \frac{1}{\beta} \cdot L_{ideal} \\ \delta & \text{otherwise} \end{cases} \quad (1)$$

where $\alpha_{inc} > 1$ and $\alpha_{dec} > 1$.

This scheme is justified as follows[2]. In case the just perceived probe load L_{exp} exceeds the maximal load, the delay δ is extended (by a factor $\alpha_{inc} > 1$) with the aim to reduce the load. As δ should not exceed the maximal delay δ_{max}, we obtain the first clause of the above formula. This rule thus readjusts the probing frequency of a CP in case the number of CPs (probing the device) suddenly increases. If L_{exp} is too low, the delay is shortened in a similar way while obeying $\delta_{min} \leqslant \delta$. The second rule thus readjusts the probing frequency of a CP in case the number of CPs (probing the device) suddenly decreases. In all other cases, the load is between the maximal and minimal load, and there is no need to adjust the delay. Note that the maximal frequency at which a CP may probe a device—given that the protocol is in a stabilized situation—is given by $\min(\frac{1}{\delta_{min}}, \beta \cdot L_{nom})$.

3. Modeling and Analysis

The probe protocol has been modeled using MODEST, a modeling formalism for stochastic and timed systems [5, 7]. MODEST is a formalism that is aimed to support (i) the modular description of reactive system's behaviour while covering both (ii) functional and (iii) non-functional system aspects such as timing and quality-of-service constraints in a single specification. The language contains features such as simple and structured data

[2]To avoid clustering of CPs, in fact, a CP adds a small random value to δ'. For the sake of simplicity, this is not described any further here.

types, structuring mechanisms like parallel composition and abstraction, means to control the granularity of assignments, exception handling, and non-deterministic and random branching and timing. Elaborate descriptions of the MODEST language can be found in [5, 7]. Here, we only explain the language constructs needed for the main parts of the protocol model.

The entire model consists of the parallel composition of a number of CPs, a number of devices and a network process. By making a precise model of the probe protocol, some small unclarities in the original protocol description [3] were revealed, such as, e.g., the way in which the ids of the last two (distinct) probing CPs were managed. The network is modeled as a simple first-in, first-out buffer with a sufficient capacity such that the probability of a full buffer is negligible. Three probabilistic cases are distinguished: a fast, slow or standard transmission.

Device model. Each device has an input and an output buffer (denoted `in` and `out`, respectively). Access to these buffers is determined by a shared variable `lck`. The model of the device is as follows, where for the sake of simplicity, the returned CP ids in a reply (and the accompanied administration) are omitted.

```
process Device (int id) {
  const int Delta = (int)(#eval(L_IDEAL / L_NOM));
  clock x;                  // timer for reply time
  int cp, pc = 0;           // probe counter
  float r;                  // reply delay

  do {
  :: when (in.lck)          // probe in input-buffer
        handle_probe {=
          pc += Delta,
          r = min + (max - min)*Uniform(0,1),
          cp = in.src,    // source of probe
          x = 0 =};       // set clock
     when (x == r && !out.lck)
        send_reply {=
          out.lck = 1,    // reserve out-buf
          out.src = id,
          out.dst = cp,
          out.pc = pc,    // send new probe cnt
          in.lck = 0 =}
} }
```

The process behaviour starts with some declarations. In addition to the typical data types, clock variables can be used. These are real-valued variables that increase implicitly and all run at the same pace. Clocks can be reset to zero and can be compared to expressions. Thus, the bahviour of a process can be made depending on time explicitly. The behaviour of a device is non-terminating as indicated by the outermost `do`-loop. In that repetitive behaviour, a device waits until it receives a probe, i.e., until its input buffer is non-empty. This is indicated by the initial `when`-clause. Once the condition of the `when`-clause holds, an action is executed (that can be used for synchronization purposes in a parallel composition), and the assignments in the {= .... =}-clause are executed atomically. In particular, `r` is determined by sampling a uniform distribution, and—simultaneously—clock `x` is reset. The subsequent `when`-clause is executed as soon as `r` time units have elapsed (and the output buffer is empty). Then the repetitive behaviour restarts by waiting for the next probe.

CP model. The model for the CP is a bit more intricate and its essential behaviour is as follows. For sake of brevity, the model has been condensed and follows not the precise MODEST syntax. The complete model is available from the net [10].

```
process CP (int id ) {
clock x, y;
  do {
  :: probe {= i = 1, x = 0, to = TOF =};
     do {      // wait for reply or timeout
     :: when ( (x >= to) && (i < 4) )
          probe {= i += 1, x = 0, to = TOS =};
     :: when ( (x < to) && in_buf )
          handle_reply {=
            prd = y, L_EXP = (CP[id].pc - pc)/prd,
            y = 0, pc = CP[id].pc, i = 0,
            if L_EXP > beta * L_IDEAL? :
                       d = min(d*a_inc, d_max),
            if L_EXP < (1/beta) * L_IDEAL? :
                       d = max(d*1/a_dec, d_min) =};
        when (x >= d) break  // wait and re-probe
     :: when ( (x >= to) && (i == 4) )
          dev_abs {= i = 0, pc = 0 =};
        when (x >= d) break  // wait and re-probe
] }  }
```

Like for the device, the behaviour of a CP is non-terminating. Its behaviour starts by probing the device (for simplicity we assume in this description a single device; in case of more devices there is one thread per device). In the innermost `do`-construct it is waited until one of three alternatives occurs: (i) if a timeout appears and more retransmissions are allowed, a probe is resend, (ii) if a reply is received in time, the experienced probe load is determined (using clock `y`), and the inter-probe-delay `d` is adjusted according to equation (1), and (iii) if a timeout occurs and the maximal number of retransmissions has been reached, the device is considered absent. In the latter two cases, a new probe cycle is initiated after respecting the inter-probe-delay `d`.

Steady-state analysis. In order to obtain more insight into the protocol's behaviour we first carried out a steady-state simulation using the batch-mean technique and confidence interval 0.1 with a confidence level of 0.95. The MOTOR-MÖBIUS integrated modeling tool [4] has been used for that purpose. The Mobius project files are available from [10]. The values for the parameters that determine the extent to which to enlarge or shorten the delay δ between probe cycles are given by [3]: $\alpha_{inc} = 2$ and $\alpha_{dec} = \frac{3}{2}$. Other important parameter values that were used for the steady-state simulation were: $\beta = \frac{3}{2}$, $L_{ideal} = 10^6$ and $L_{nom} = 10$ (yielding $\Delta = 10^5$), $\delta_{min} = 0.02$ and $\delta_{max} = 10$. Furthermore, it has been assumed that there is a single device, and $k = 20$, i.e., 20 CPs are continuously present. In a stabilized situation, one would expect the CPs probing at a frequency of $L_{nom}/k = 0.5$. To avoid buffer overruns, the network buffer size has been fixed to 20,000 elements. The network delay has been modeled as a uniform probabilistic choice between three modes of operation: a slow, a medium and a fast mode. We have experimented with several other types of networks, and obtained similar phenomena for all of them.

The analysis focused on the probe frequency $\frac{1}{\delta}$ of the CPs. The simulation revealed that indeed network buffer overflow is a seldom phenomenon as the average buffer length is very small (≈ 0.004). Surprisingly, however, the mean delay of almost all CPs was about 10.0, whereas two CPs had a delay of only 0.4. Both values are far away from the optimal delay, $k/L_{nom} = 2$. Whereas the self-adaptive mechanism should—surely in a static system situation—lead to an equal spreading of the probe frequencies, this is thus apparently not the case. Secondly, some CPs have a high variance in their computed delays, whereas others have only minimal variation. The most extreme case is a CP with a mean delay of 8 and a variance of about 13.5. Despite this abnormal behaviour of the CPs, the device load is quite good (i.e., it is near to $L_{nom} = 10$, and has a low variance). These unexpected steady-state simulation results motivated a set of more detailed simulations in order to obtain a better insight into these phenomena.

Transient simulation. Various transient simulations have been carried out that simulated the initial 20,000 seconds of the probe protocol. All studied configurations consist of a single device and a number of CPs. Whereas for one or two CPs, the probe frequencies were balanced and exhibit almost no variance as expected, for three CPs this is not the case: Fig. 2 shows the probe frequencies of the 3 CPs (y-axis) versus the elapsed time (x-axis). Note that after a short initial phase, one CP is probing less and less frequent, and is not recovering from this (undesired) situation. Another observation is that although the remaining two CPs tend to "stabilize" their probing frequencies, there remains to be a rather high variance in their frequencies. Simulations of other scenarios with different numbers of CP show similar behaviour: some CPs probe rather fast, whereas others are probing (unexpectedly) slower and

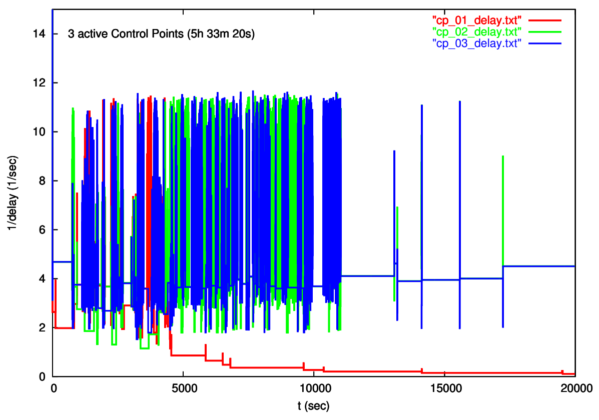

Figure 2. Probe frequencies for 3 CPs

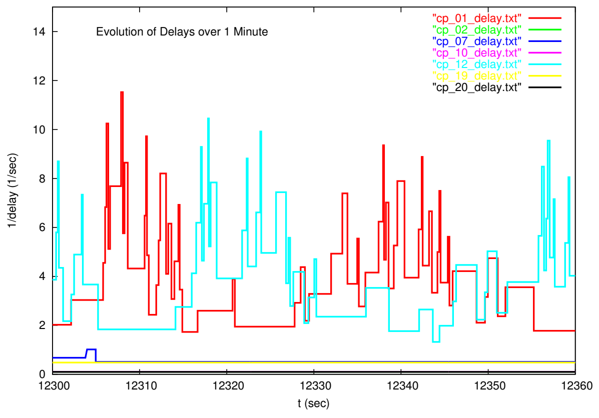

Figure 3. Probe frequencies for 7 (out of 20) CPs for 1 minute

slower, and do not recover from this situation. Similarly, high variances in the individual probe frequencies of a single CP occur. Fig. 3 shows for 7 arbitrary CPs out of a collection of 20 CPs how their probing frequencies evolve over a short interval of 1 minute. We like to stress that in all these scenarios the number of CPs is completely static, i.e., no CPs join or leave the system.

Fig. 4 shows the effect for two CPs that start in a configuration of 20 present CPs of which 18 suddenly leave simultaneously. Whereas in a static scenario with just two CPs, their frequencies are equal, we see that in this dynamic scenario, there is neither a load balance between the CPs nor a low variance.

Some considerations. The above analysis revealed (at least) two undesired phenomena: the variance in prob-

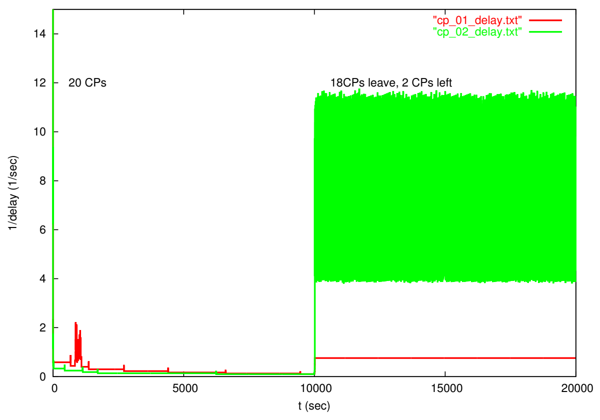

Figure 4. Suddenly 18 out of 20 CPs leave

ing frequencies of (individual and between) CPs may be extremely large, and CPs can even be completely starving without the possibility to recover from this situation. Would it not be possible to adapt the protocol to "synchronize" CPs in some sense thus obtaining a stable and equal probe frequency for all CPs? An important source of these undesired phenomena is that the experienced probe load L_{exp} of a CP may be misleading. In particular, a CP cannot distinguish between many CPs that probe the same device at medium rate, or a few CPs probing the device at a high frequency. In both cases, the device tends to be overloaded, and the difference $pc' - pc$ is large, resulting in a lowering of the CP's probe frequency. Moreover, if the device load drops below the threshold L_{nom}, the CPs with a higher pinging frequency will detect this faster. Due to the greedy nature of the protocol they will therefore increase their respective probe frequencies, thus increasing the probe load of the device. CPs with a low frequency are therefore with high probability too late to get their share of the available bandwidth. The adaptive probe protocol as described here has therefore an inherent fairness problem.

This raises the question, why devices not simply communicate other information that CPs could use to determine their probing frequency more accurately? These considerations form the basis for the next protocol.

4. An Improved Lightweight Probe Protocol

The protocol described in this section is intended to overcome the fairness problem encountered just above. Like before, there are two types of nodes (devices and CPs) and the basic protocol mechanism is that CPs continuously probe a device to check its presence. Instead of keeping track of the number of times the device has been probed so far, a device simply schedules when a probing CP is allowed to probe it again. The device replies to a probe with a delay value indicating to the respective CP how long it has to wait before it may probe again. This mechanism can be viewed as a bakery protocol: on probing, a CP receives an indication about the next time instant at which it can probe the device. To facilitate this scheme, the device keeps track of the first free time slot at which it wants to be probed (*i.e.*, first ticket count that has not been assigned yet). In addition to the ticket count, a device sends the ids of the last two (distinct) CPs that probed it as part of the reply. This information is used (as before) for a quick dissemination of absence information concerning a device using the overlay network. As we will see, CPs will now probe a device in a fixed order, and therefore, the overlay network will be a ring.[3]

Device behaviour. A device keeps track of nt, the first time instant for which the last pinging CP has been scheduled to probe again. On receipt of a new probe from a CP at time t, this variable is incremented by Δ ($nt'{=}nt{+}\Delta$)[4] and a reply is sent to the probing CP with as parameters the delay $nt' - t$ until the time instant where the pinging CP is scheduled for its next probe and the ids of the last two distinct CPs that probed the device. In the following we describe how we choose an appropriate value for Δ.

As in the adaptive protocol, we assume that the device has a certain maximal probe load, L_{nom} which it is able or willing to cope with. We define $\delta_{min} = 1/L_{nom}$. Additionally, we must assume that also a CP does only want to ping with a maximal frequency, f_{max}, and we define $d_{min} = 1/f_{max}$. In order to obtain a reasonable value for Δ, the following two constraints should be fulfilled. Assume again that a probe has just arrived at time t. Then: (i) $\Delta \geqslant \delta_{min}$, which asserts that two consecutive probes are at least δ_{min} time units apart, and (ii) $nt{-}t{+}\Delta \geqslant d_{min}$, which states that the CP should not probe earlier than $t{+}d_{min}$. Summarizing, we have:

$$\Delta \geqslant \max\{\delta_{min}, d_{min}{-}(nt{-}t)\}.$$

In order to be as fast as possible, we choose in the sequel:

$$\Delta = \max\{\delta_{min}, d_{min}{-}(nt{-}t)\}. \tag{2}$$

CP behaviour. The CP behaviour is, compared to the adaptive protocol, much simpler. The CP shows the same behaviour with respect to the probing and re-probing of a device, however, the delay between two probe cycles is

[3]This may be improved by letting the device keeping track of more CPs that have probed, and sending back a randomly selected subset in a reply. Such mechanisms are, however, not further considered here.

[4]Do not confuse this Δ with the one used for the adaptive protocol in Section 2.

now directly determined by the device, rather than computed from a probe counter. A CP has actually no notion anymore of the experienced or real probe load the device has to cope with. Each reply to a probe is accompanied with a delay d. On receipt of such reply, the CP sets a timer and waits until d time-units have passed before it initiates the next probe cycle.

5. Modeling and Analysis

Device model. We have analyzed the protocol again with the MODEST-MÖBIUS tool tandem. Since the CP behaviour is simpler than before, we omit its MODEST specification. The interesting part is the device, which is shown in a condensed form in Fig. 5. The constant `MINDELTA` corresponds to δ_{min}, and `MINDELAY` to d_{min}. When a

```
process Device (int id) {
  clock st = 0;      // time since system start
  float nt = st;     // last scheduled ping

  do {
  :: when (in.lck)        // probe in input-buffer
        handle_probe {=
          d = max(MINDELTA,
                    MINDELAY-(nt-st)),
          nt += d,
          r = min + (max - min)*Uniform(0,1),
          cp = in.src,    // source of probe
          x = 0 =};       // set clock
     when (x == r && !out.lck)
        send_reply {=
          out.lck = 1,    // reserve out-buf
          out.src = id,
          out.dst = cp,
          out.delay = nt-st, // delay for CP
          in.lck = 0 =}
} }
```

Figure 5. Device behaviour improved protocol

probe is detected in the input buffer, the device computes the maximum of `MINDELTA` and `MINDELAY-(nt-st)`, according to (2). It then adds this maximum to `nt`. This is the next free time (slot) that the device has not yet assigned to a CP. The delay that is returned to the probing CP is computed as `nt-st`.

Dynamic scenarios. Due to its deterministic nature, the protocol ensures that once a situation is reached where the number of probing CPs does not change, the device has a probe load of L_{nom}, and the probe frequency is nearly the same for all CPs. Therefore, the protocol has a big advantage over the adaptive protocol, and is even computationally simpler.

In a dynamic scenario, where CPs join or leave the network in a nondeterministic fashion, and perhaps even in bursts, it is less obvious to determine whether the protocol can meet its expectations. CPs that join the network and start probing the device are unaware of the current schedule laid out by the device. Their entrance will, therefore, disturb the neat pattern of the probe schedule, yielding a (temporarily) increase of the device's load.

Simulations have been carried out to find out how the device load is affected in a dynamic environment. These studies concentrated on the average load and its variance. Consider a worst case scenario, where the number of active CPs is uniformly chosen from the set $\{1, \ldots, 60\}$. This choice is repeated every X time-units, where X is exponentially distributed with rate 0.05. That is to say, with a mean of 20 seconds, the number of active CPs will change. Packet losses are not considered, i.e., every transmitted probe will eventually be answered. The value of δ_{min} has been set to 0.1, and d_{min} equals 0.5.

In this scenario, the mean load of a device in steady-state is $9.7\ probes/s$, and the variance 20.0, yielding a standard deviation of $\approx \pm 4.5$. Statistically, the probability

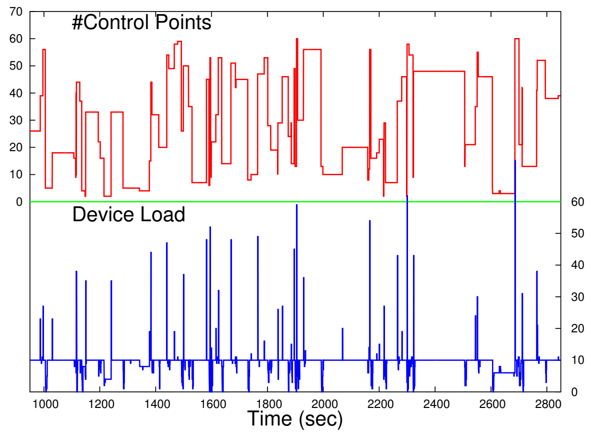

Figure 6. Load and #CPs over 30 min

of exceeding the nominal probe load is low. Fig. 6 depicts the device load and the number of CPs over an interval of 30 minutes (1800 s). Despite the low variance, the device load has some peaks, especially when many CPs join the network simultaneously. However, the load falls off very quickly again towards $L_{nom} = 10 = 1/\delta_{min}$ as the device rapidly incorporates the newcomers into the current schedule. It is important to realize that is a worst-case scenario, and it makes the unrealistic assumption that all probes in a burst will eventually be replied to (no packet loss). In case of packet losses, however, which will occur in bursts due to the limited capacity of devices, the load caused by new CPs will be spread better over time, since some CPs will only receive a reply after some re-probing. We can therefore expect that in practice the peaks in the device load as they appear as spikes in Fig. 6 will not be a bit wider.

6 The Proxy-Bye Protocol

The protocols discussed so far aim to quickly detect that a device has left the network. When a device does not respond to four consecutivce probes sent by a CP, the CP considers the device to be absent. To inform other CPs about the absence of the device, the so-called *proxy-bye* protocol [3] is initiated. This dissemniation protocol will be described and analyzed in the following. As the protocol does not affect the device's behaviour, we focus on the CP.

CP behaviour. Assume there are n CPs $c_1, \ldots, c_n$. As mentioned in the beginning of Section 2, a device sends the ids of the last two CPs c_j, c_k (with $j \neq k$) that have probed it as part of the reply to a probe of CP c_i ($i \neq j, k$). In this way, c_j and c_k become the logical neighbours of CP c_i in an overlay network. On detecting the absence of a device, this overlay network is used to rapidly inform all CPs about the leave of a device. Once a CP, c_i, say, pretends the absence of a device (as it did not receive a reply to any of its four probes), it initiates the proxy-bye protocol by sending a message to its logical neighbours c_j and c_k informing them about the absence of the device. On receipt of a proxy-bye message, a CP probes the device *once*, out-of-schedule, in order to check whether the device has really gone. If no reply is received (after the timeout period of length TOF), the CP considers the device to be absent, and initiates the proxy-bye protocol by sending a proxy-bye message to its logical neighbors. This results in a flooding of proxy-bye message through the overlay network.

7 Analyzing the Effect of Proxy-Bye

As CPs can (and will) detect the absence of a device on an individual basis using the probe protocol, the proxy-bye protocol may seem to be superfluous. Its introduction, in fact, raises the immediate question to what extent the absence detection of devices is affected by the proxy-bye protocol. Will all present CPs now know faster that a device is absent? This is studied by considering two scenarios: device absence detection without and with the proxy-bye mechanism.

The analysis of the device absence detection has again been done by means of the MODEST/MÖBIUS tandem. In order to deal with proxy-bye messages, part of the model had to be modified, especially the model of a CP. Here, we will only describe the relevant functionality. Fig. 7 shows the basic structure of the CP model. Inside the iteration of the try-block, it is waited until one of five alternatives occur: (i) it is time to start a new probe cycle (i.e., perform action `send_probe`), (ii) a probe has to be resent (action `send_probe_again`), (iii) no reply has been received after four probes (send then proxy_bye message), (iv) a reply has been received (action `handle_reply`), or (v) a proxy-bye message has been received (action `handle_proxybye`).

```
...
try {
  do {
    :: when ( /* time for next ping cycle */ )
         send_probe {= i = 1, timeout = TOF ... =}
    :: when ( /* no reply received and i < 4 */ )
         send_probe_again {=
           i += 1,
           timeout = TOS,
           ... =}
    :: when ( /*no reply received, i == 4 */ )
        // send  proxy-bye packet to known CPs
         init_proxybye {= ... =};
         throw device_gone  // terminate CP
    :: when ( /* reply received */ )
         handle_reply {=
           i = 0,
           /* set time for next ping-cycle*/,
           ... =}
    :: when ( /* proxy-bye packet received */)
           handle_proxybye {=
             i = 4,
             timeout = TOF,
             // send one probe
             ... =}
  }
} catch (device_gone) {
      // CP terminates
      bye-bye
}
```

Figure 7. Sketch of CP model with proxybye

The measure-of-interest is the mean time (and variance) between the device going off-line, and the last CP noticing its absence. To obtain this measure, we have conducted a series of terminating simulations. We assume a stable scenario, i.e., 60 CPs over the initial period of the simulation. The device goes off-line at time 50. Without proxy-bye, the time until the last CP notices the absence of the device is very near to 6 seconds. This is not surprising: since the CPs still follow the schedule that has been dictated by the device, and considering that the nominal ping load L_{nom} of the device was set to 10, the last CP that has successfully probed the device will be scheduled to probe again after a 6 second delay. Only then this CP can detect the absence of the device.

With proxy-bye, the result is much different. In fact, the time until the last device considers the device to be absent is only 0.7 seconds after the device actually leaves the network. This delay, moreover, has a very low variance of $1.8 \cdot 10^{-6}$. Fig. 8 shows how the number of CPs that have realized that the device is gone (y-axis) grows when time elapses (x-axis). Note that the device leaves at time t=50.

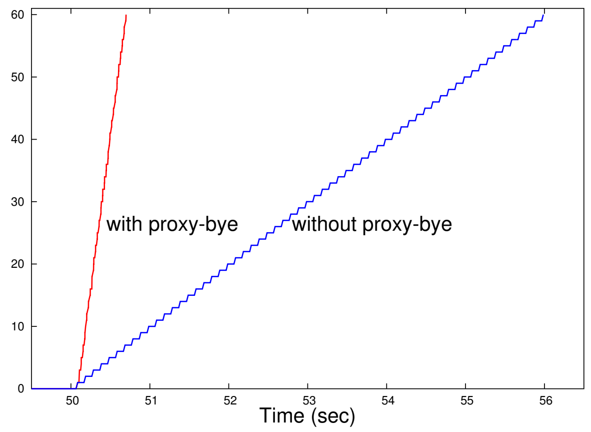

Figure 8. # informed CPs vs time when device leaves at t=50

8. Concluding Remarks

This paper described two approaches to the absence detection of devices in self-organizing networks, and reported on their modeling and analysis by means of simulation.

Our analysis has shown that the adaptive probe protocol suffers from a fairness problem: the protocol fails to ensure that CPs probe a device with roughly the same frequency. Instead, some CPs can have lower and lower probing frequencies, whereas other CPs probe very fast, and even have oscillating frequencies. The slow CPs will therefore take a long time before detecting the absence of a device. Even though the proxy-bye protocol can cushion this effect, the unfairness is still an undesirable effect—the faster CPs send more packets than really necessary and have a lot of computation to do in order to adjust their frequencies. This leasd to a waste of computing recources and an sharp increase of power consumption, which for e.g., handheld devices is most undesirable.

In order to overcome the unfairness problem, we have proposed a variant of the probe protocol, where the CPs do not adjust their probing frequencies, but where the device is responsible to schedule the CPs for the next time they are allowed to probe the device. Our analyses have shown that this protocol variant does not suffer from the fairness problem. Moreover, the protocol is capable of keeping the load of a device within desirable limits, and—due to its intrinsic simplicity—is amenable to implementation in small computing devices.

Finally, we have considered the proxy-bye protocol which is used to propagate the absence of a device to other CPs. Our analysis has shown that proxy-bye indeed speeds up the absence detection enormously in the analysed cases.

The analysis results have been obtained using the MODEST/MÖBIUS tool suite. MODEST is a modeling language with a formal semantics [7] expressed in terms of (extended) labeled transition systems. The formality of the language allows not only for the integration with other formal analysis tools (such as model checkers), but, more importantly, is essential to carry out semantically sound simulation runs with MÖBIUS. This results in a trustworthy analysis chain. Standard simulation environments are risky to use instead, because they have been found to exhibit contradictory results (both quantitatively and qualitatively, i.e., difference in behaviour) even in simple case studies [6].

Acknowledgement. The authors like to thank Lex Heerink and Maarten Bodlaender, both of Philips Research, for their comments on a draft of this paper.

References

[1] C. Baier, F. Ciezinski, and M. Größer. PROBMELA: a modeling language for communicating probabilistic processes. In: *ACM-IEEE Int. Conf. on Formal Methods and Models for Codesign*, ACM Press, 2004.

[2] G. Behrmann, A. David, and K.G. Larsen. A tutorial on UPPAAL. In: *Formal Methods for the Design of Real-Time Systems*, LNCS 3185: 200–237. Springer-Verlag, 2004. (see also `www.uppaal.com`).

[3] M. Bodlaender, J. Guidi and L. Heerink. Enhancing discovery with liveness. In: *IEEE Consumer Comm. and Networking Conf.*, IEEE CS Press, 2004.

[4] H. Bohnenkamp, T. Courtney, D. Daly, S. Derisavi, H. Hermanns, J.-P. Katoen, V. Lam and W.H. Sanders. On integrating the Möbius and MODEST modeling tools. *Dependable Systems and Networks*, pp. 671–672, 2003, IEEE CS Press.

[5] H. Bohnenkamp, P.R. D'Argenio, H. Hermanns, and J.-P. Katoen. MODEST: A compositional modeling formalism for real-time and stochastic systems. CTIT Tech. Rep. 04-46, 2004.

[6] D. Cavin, Y. Sasson, and A. Schiper. On the accuracy of MANET simulators. In *ACM Workshop On Principles Of Mobile Computing*, pp. 38–43, 2002.

[7] P.R. D'Argenio, H. Hermanns, J.-P. Katoen and J. Klaren. MODEST: A modelling language for stochastic timed systems. In: *Proc. Alg. and Prob. Methods*, LNCS 2165: 87–104, 2001.

[8] P.R. D'Argenio, J.-P. Katoen, T.C. Ruys and G. Tretmans. The bounded retransmission protocol must be on time! In *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1217: 416–431, 1997.

[9] D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derasavi, J. Doyle, W.H. Sanders and P. Webster. The MÖBIUS framework and its implementation. *IEEE Tr. on Softw. Eng.*, **28**(10):956–970, 2002.

[10] `http://wwwhome.cs.utwente.nl/~bohnenka/liveness-model.tar.gz`

[11] W.K. Edwards. *Core Jini*. Prentice-Hall, 1999.

[12] J. Gorter. Modeling and analysis of the liveness UPnP extension. Master's thesis, Univ. of Twente, 2004.

[13] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. IETF, RFC 2608, 1999. (available at `www.rfc-editor.org/`).

[14] M. Jelasity, W. Kowalczyk and M. van Steen. Newscast computing. Tech. Rep. IR-CS-006, Vrije Univ. Amsterdam, 2003.

[15] M. Raynal and F. Tronel. Group membership failure detection: a simple protocol and its probabilistic analysis. *Distrib. Syst. Engng*, **6**: 95–102, 1999.

[16] R. van Renesse, K.P. Birman and W. Vogels. Astrolab: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Tr. on Comp. Sys.*, **21**(2): 164–206, 2003.

[17] R. van Renesse, Y. Minsky and M. Hayden. A gossip-style failure detection service. In: *IFIP Conf. on Distributed Systems, Platforms, and Open Distributed Processing*, pp. 55–70, 1998.

[18] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. In: *ACM SIGCOMM*, 149–160, ACM Press, 2001.

[19] UPnP Forum. *UPnP Device Architecture, Version 1.0.* (available from `www.upnp.org`).