Adaptive Failure Detection and Correction in Dynamic Patient-Networks

Martin Ringwelski[1], Andreas Timm-Giel[1], and Volker Turau[2]

1 **Institute of Communication Networks**
2 **Institute of Telematics**
Hamburg University of Technology
Hamburg, Germany
{martin.ringwelski,timm-giel,turau}@tuhh.de

Abstract

Wireless sensors have been studied over recent years for different promising applications with high value for individuals and society. A good example are wireless sensor networks for patients allowing for better and more efficient monitoring of patients in hospitals or even early discharge form hospital and monitoring at home. These visions have hardly led research as reliability is and issue with wireless networks to be known error-prone. In life critical applications like health care this is not an aspect to be handled carelessly. Fail-safety is an important property for patient monitoring systems.

The Ambient Assistance for Recovery (AA4R) project of the Hamburg University of Technology researches on a fail-safe patient monitoring system. Our vision is a dynamically distributed system using suitable devices in the area of a patient. The data in the network is stored with redundancy on several nodes. Patient data is analyzed in the network and uploaded to a medical server.

As devices appear, disappear and fail, so do the services being executed on those devices. This article focuses on a Reincarnation Service (RS) to track the functionality of the processes. The RS takes suitable actions when a failure is detected to correct or isolate the failure. Checking of the nodes is done adaptively to achieve a good response time to failures and reduce the power consumption.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Wireless Sensor Networks, Fail-Safety, Health Monitoring, Failure Masking, Distributed Systems

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.38

1 Introduction

Until today, patients need to stay in the hospital after their treatment for monitoring their recovery or even for diagnosing. With wireless sensor networks (WSNs) these doctors would be able to monitor the health signals of the patients remotely, while they can be at their familiar environment. This would not only help reduce costs by reducing the occupied beds in the hospitals, but might also help people recover. People tend to recover better in their families and many infections happen in hospitals[1].

[1] See http://www.bmg.bund.de/praevention/krankenhausinfektionen/fragen-und-antworten.html (2014-02-10)

Medical Cyber Physical Systems – Medical Device Interoperability, Safety, and Security Assurance (MCPS'14).
Editors: Volker Turau, Marta Kwiatkowska, Rahul Mangharam, and Christoph Weyer; pp. 38–48

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Using a WSN for patient monitoring rises several problems regarding the fail-safety of the system. Wireless links are volatile and nodes can fail. While those problems also affect other kinds of WSNs, in patient monitoring they can be life critical.

This work is part of the Ambient Assistance for Recovery (AA4R)[2] project of eight institutes at the Hamburg University of Technology. The project aims at building a patient monitoring system to support recovery of patients and relieve of hospital personal. This will be achieved by an uninterrupted use of technology from the ward of a hospital over a rehab center to the patients home. By this, we are closing the gaps between ambient assisted living in home care, telemonitoring and telediagnosis. Fail-safety is addressed at different subsystems such as the communication infrastructure or the sensors themselves. Further, the fail-safety is addressed at system-level. In this paper we primarily focus on fail-safety in the communication infrastructure.

The system is distributed using resources available in the area. We point out and model different sources of failure and discuss strategies to reduce them and the risk of a total failure of the system. To achieve that, we build a monitoring service for the system, called Reincarnation Service (RS). The RS discovers faults and takes actions to solve them.

In the next chapter, we review related work and show the differences to our approach. The vision of our scenario is introduced in the third chapter, followed by possible failures and failure rates. We discuss our approach and the expected improvements in the fifth chapter. In chapter six, we give a conclusion and discuss the future work.

2 Related Work

Chipara et al. [4] built and tested an IEEE 802.15.4 based wireless monitoring system for patient data. In contrast to our idea of monitoring the patient at home for longer periods, they only measured for up to three days during the patients stay at the hospital. The sensor data was limited to the pulse rate and the oxygen level of the blood. Data rates were only in the range of several bytes per minute. As the patients only moved inside the hospital and several relay points where installed to forward the data, the network reliability was not problematic in this study. The system was centralized and the nodes preconfigured. Problems of reliability in a distributed system were not investigated. The work of Ko et al. [7] invested a similar system.

Chen et al. [3] investigated a routing protocol for patient monitoring and fall detection at home. In case of emergency the system transmits the ECG and in-door position of the patient to first responders. Their protocol ensures a fast and reliable delivery of the data. The system is closed and no ambient sensors or other devices are used.

Preventing or avoiding faults is not possible in complex distributed systems. We plan to detect and correct or isolate faults in the system. To achieve high reliability, Herder et al. [5] described a Reincarnation Server in the Minix 3 operating system. This server periodically checks the state of the other services in the system and restarts them when they are broken. The described method is used on one machine and the checking time is constant. Nevertheless, we can use that idea of failure detection and correction and thereby increase the overall reliability of the system.

This approach can also be read out of the white paper of IBM [6] about self-healing, self-configuring and self-optimizing. They identified a four state healing loop consisting of

[2] http://www.aa4r.org/

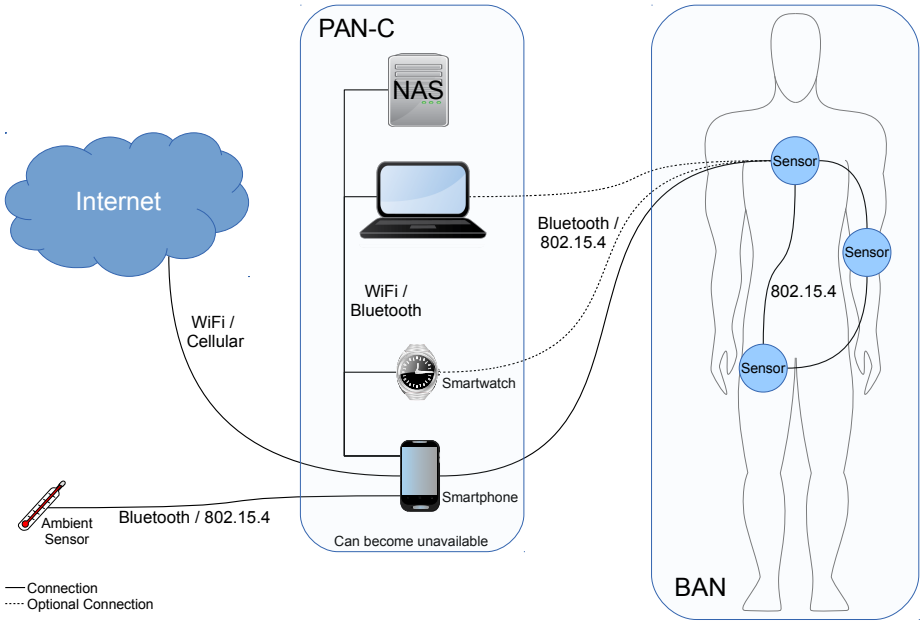


Figure 1 Network Architecture.

monitoring the state, detecting errors, analyzing the failure to plan the repair and executing the repair.

Our implementation will use the CometOS framework [12]. CometOS allows us to write code that can be simulated in Omnet++ and compiled for testbed hardware. This reduces the risk of failures during the migration of the software.

3 Scenario

The AA4R project aims at people recovering from different kinds of incidents. From a broken leg over to heart attack. These people need to stay in hospitals to monitor their recovery and ensure they are not suffering a fall-back. Another scenario are people coming to the hospital without knowing what is wrong. They also need to stay in the hospital to monitor their health values and be able to make a diagnosis. We want those people to be able to go home and continue with their lives. Our monitoring system will assist patients in their recovery and the physicians in their diagnosis. It will help prevent dangerous situations and call help in emergency situations.

The system we envision consists of a Body-Area-Network (BAN) of sensors, collecting vital information, ambient sensors and other External Devices (EDs), building a Personal-Area-Network (PAN). EDs, like smartphones, smartwatches or laptops, are used for the PAN-Controller (PAN-C). Figure 1 shows the network architecture of the Patient-Network (PN) including the BAN, EDs building the PAN-C and an ambient sensor. The sensors in the BAN might have direct connections to several EDs, but only only one connection will be preferred, depending on the stability of the connection and the resources of the device. On the other end of the Internet will be a medical server, which stores the information of the patients under pseudonyms for diagnosis by physicians.

The PAN-C is a distributed system consisting of different services in the network. A

service will be delegated to the most suitable device, determined by their available resources and connections. Those services are:

- **Forwarding**: We want to send the data to a medical server. The server can automatically analyze the data and prepare relevant graphs for physicians or present raw data for doctors to monitor the values or figures remotely. This service needs to be done by a device having a reliable Internet connection.
- **Storing**: Storing the data that is collected in the BAN and other ambient sensors needs to be done redundantly by several devices. The data is needed for analyzing the health status in the network and to maintain the state and integrity of the system. Not only the current, but also the past data of the medical sensors is important to understand the situation of the patient. This service becomes critical when lacking an Internet connection.
- **Plausibility**: This service looks for anomalies in the sensor data to detect drifts or other failures of sensors. To achieve that, we need to have past data available.
- **Inclusion**: This service can be described as the coordination service. New devices for services of the PAN-C need to be found and included in the network. Also, ambient sensors that might be helpful to analyze the patient data, e.g. room temperature and humidity, need to be included. This service can be compared to the Membership Service in the work of Rodrigues et al. [11].
- **Control**: The control service allows medical applications to use possible actuators in the BAN. A patient with diabetes can get his insulin automatically or a patient with strong pain can get his analgesics. This service also needs to eliminate the risk of overdosing the patient.

Services on the EDs run as virtual machines to be separated from the rest of the devices system. This also enables us to simply copy the service to a different device and switching the responsibility to the new device.

Having several devices responsible for the services in the network reduces the risk, that all devices fail at once, but it increases the risk that devices running important services fail. Also consistency of data and decision are critical points in the network. Failures of services need to be detected to take measures to isolate and try to repair the failures. For that, a Reincarnation Service (RS) will run on every device to monitor the state of the connected devices.

The overlying application should not be concerned with failures of devices in the network. Only if a service can not be fulfilled by any other device or compensated by another service, the application should be informed to take actions.

4 Failures

As this system is responsible for a human being, a failure of one single component could lead to an undiscovered critical situation of the patient and thereby to death. That is why we need to analyze the possible faults in the system. We identified seven fault categories:

- **No Internet connection**: The data needs to be transmitted to a medical server for remote monitoring by a doctor. If the connection fails, the data needs to be temporarily stored locally and transmitted when a connection comes up again. For diagnosis, this will be a sufficient solution. In emergency situations, this problem can be life critical. Depending on the previous condition of the patient, a timeout might be used to trigger an emergency call on the server.

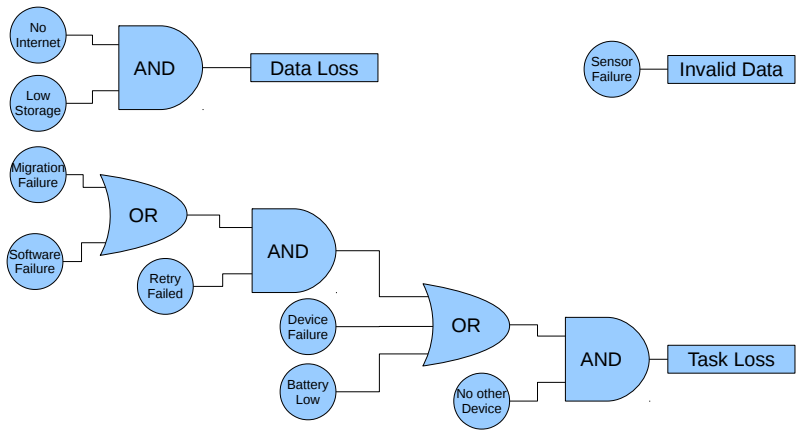


Figure 2 Fault Tree.

- **Low Storage**: The last data should always be held on nodes in the network for a better analysis of the current patient state. In case of a lost Internet connection the data needs to be kept even longer, so that the storage may get sparse.
- **Migration failure**: The state of the network needs to be known by every device. During transfer of a service from one device to another, inconsistencies in the distributed information might arise.
- **Sensor failure**: Sensors might drift, have an offset or even stop working. Those errors are not further considered in this work, as they are part of the analysis that other members of the AA4R project are working on.
- **Device failure**: Any device can randomly fail because of bad production or wear-out.
- **Software failure**: We can not guarantee an error-free program. An error may also be caused by a memory shortage. In this case it might work again after a restart.
- **Battery low**: As we are using wireless devices running on battery, any device can run low on battery. This failure can be easily foreseen.

The fault tree in Figure 2 shows the impact of the given failures. Only if one of the three subsystems fails and we encounter data loss, service loss or a sensor failure, the application must be informed. Otherwise the faults can be masked for the application, be it through a state-machine or a primary-backup approach [8]. Invalid data will be handled by the analysis service, which might just discover a drift or switch to a redundant sensor, if available.

Masking failures allows the application to concentrate on its service rather than having to deal with error handling. It is still possible for faults to occur, thus fault handling can not completely left out.

4.1 Expected Failure Rates

Failure rates, hazard rates or hazard functions are names for the expected failures at a given time. In contrast to the probability density function f(t), which depicts the overall probability of failures per time, the hazard function h(t) depicts this probability under the assumption, that no error happened before. It is the fraction of the probability density function by the survival rate, whereas the latter is $S(t) = 1 - F(t)$.

For electronic devices Nowlan and Heap [10] proposed a hazard function with an infant mortality and a constant failure rate. The infant mortality is caused by incapable devices, program or migration errors. Those errors are more likely to happen at the beginning of the lifetime of a device. Devices that survive the infant mortality phase are not likely to suffer from those errors. The constant failure probability is due to battery drain, interfering failures or devices moving out of reach. Those errors can happen at any time. A wear-out mortality is not included, as the system is not expected to run tens of years. We can also expect that the patient is taking care of his or her smart phone battery.

The infant mortality can be described with the Weibull distribution (equation 1), with a $0 < k < 1$ [13]. k is the shaping factor, where $k < 1$ results in a decreasing failure rate over time. λ is the scaling parameter. $f_{im}(t)$ describes the error probability at a certain time, (2) is the cumulative error probability and (3) shows the hazard function:

$$f_{im}(t) = k\lambda_{im}^{k} t^{k-1} e^{-(\lambda_{im} t)^{k}} \tag{1}$$

$$F_{im}(t) = 1 - e^{-(\lambda_{im} t)^{k}} \tag{2}$$

$$h_{im}(t) = k\lambda_{im}^{k} t^{k-1} \tag{3}$$

Failures by interfering signals or devices moving out of range can occur at any time. Those failures are not more likely to happen at the beginning or after some time. The constant failure rate can be described by an exponential distribution. Equations (4), (5) and (6) show the probability density function, the cumulative distribution and the hazard function of the exponential distribution:

$$f_{c}(t) = \lambda_{c} e^{-\lambda_{c} t} \tag{4}$$

$$F_{c}(t) = 1 - e^{-\lambda_{c} t} \tag{5}$$

$$h_{c}(t) = \lambda_{c} \tag{6}$$

The hazard functions can be easily combined by addition to get the system hazard function in equation (7). For any given interval $[t_a; t_b]$ we can calculate the probability of a failure, under the assumption that no failure occurred before time t_a, with the equation (8):

$$h(t) = k\lambda_{im}^{k} t^{k-1} + \lambda_{c} \tag{7}$$

$$P_{fail}(t_a, t_b) = \frac{F(t_b) - F(t_a)}{1 - F(t_a)} = 1 - \frac{e^{-\lambda_{c} t_b - (\lambda_{im} t_b)^{k}}}{e^{-\lambda_{c} t_a - (\lambda_{im} t_a)^{k}}} \tag{8}$$

Figure 3 depicts sample hazard functions for equations (3), (6) and (7). The used example values are later discussed in section 5.1.1.

5 Counter-measures

To achieve fail-safety, the system needs to be in a stable state, although inevitable faults may occur. For that, we need to detect the failures as fast as possible and take measures to

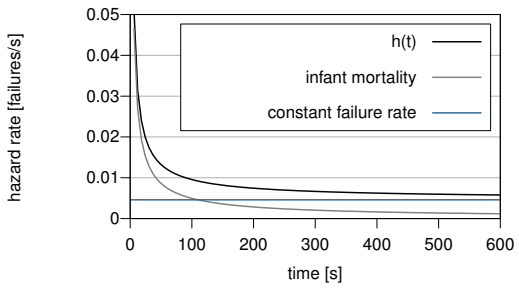


Figure 3 Sample of hazard functions for equations (3), (6) and (7) with k = 0.2, $\lambda_{\text{im}} = 0.9335\frac{1}{\text{s}}$ and $\lambda_{\text{c}} = 0.0046\frac{1}{\text{s}}$.

solve them. The system must not rely on one single node. Also, even if a service or data is lost, the system needs to be aware of the situation and keep on working with its limited capabilities.

There are no metrics for fail-safety, but we can use the Mean-Time-To-Failure (MTTF), Mean-Time-To-Recovery (MTTR) and the fraction of components allowed to fail, before the system becomes unstable. Those figures are used to describe the reliability.

For the different kind of failures, we have to take specially assembled counter-measures:

- **Link Quality Prediction**: By predicting the quality of a link, we can take measures to switch a service from a node to another one, before it is disappearing. This would extent a Link-Quality-Estimator (LQE) to a Link Quality Predictor.
- **Redundant storage**: As nodes with important information can disappear without warning, we need to have redundant data in the network. Acedański et al. [1], Nguyen et al. [9] and Rodrigues et al. [11] have analyzed distributed network storage with random linear network coding. Nevertheless they can not assure that data is available after some time. Depending on the sensors data-rate and the number of devices and their capacities, data needs to be deleted eventually. We want to use an approach that assure data availability and prioritizes the data by importance, so only dispensable data gets lost.
- **Reincarnation Service**: For detecting failures in software and loss of connection, a Reincarnation Service (RS) checks the state of the devices and processes. In case of failure, the process can be restarted or a substitute can be found.

The RS itself also needs to be distributed. Every device running a service has a RS responsible for the services on that device. This ensures reliability of services on the devices. Figure 4 shows the layered software architecture on two connected devices. Services that are not present on a device, due to limited resources, can be used from another device. The medical application itself does not know, where the services are hosted. The RS is responsible for the services and the application to be running and available.

As devices can also fail or disappear, the device that has include another device, that means it has delegated a service to that device, is responsible for that. The Reincarnations Service has to check the Reincarnation Service on the other device.

5.1 Detecting Failures

To detect failures, processes and devices need to be checked periodically. The question remains how often a check should be performed. Too frequent checks will drain the battery and cause congestion, too few lead to long unrecognized failures which can cause other failures.

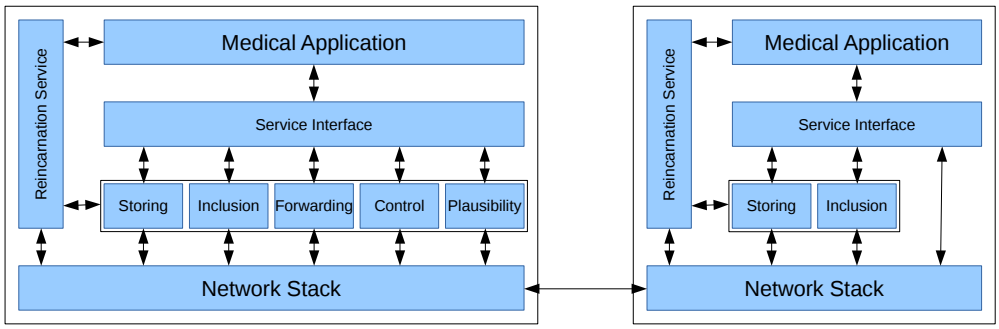
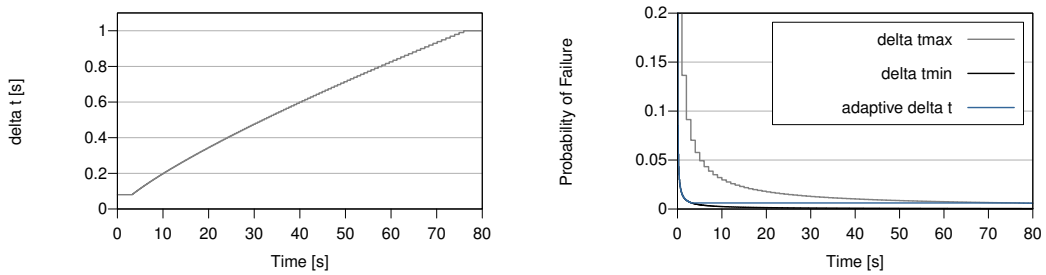


Figure 4 Software Architecture, showing two devices offering different services



(a) Adaption of Δt

(b) Expected failures per Interval

Figure 5 Influences of Δt.

Setting the probability of failures in an interval to a fixed value $\text{fail}_{\max}$, we can calculate the next time we need to check if the service is still alive. By that, we achieve an adaptive testing of the process, depending on the expected failure rate. But solving that equation is not trivial. We use the Newton method to approximate the next time to check (equation 9):

$$0 = \lambda_c \Delta t + {\lambda_{im}}^k \left((t + \Delta t)^k - t^k \right) + ln(1 - P_{\text{fail}_{\max}})$$

$$\Delta t_{next} = \Delta t_{min} + \frac{\lambda_c \Delta t_{min} + {\lambda_{im}}^k \left((t + \Delta t_{min})^k - t^k \right) + ln(1 - P_{\text{fail}_{\max}})}{-\lambda_c - k{\lambda_{im}}^k (t + \Delta t_{min})^{k-1}} \tag{9}$$

If that equation results in a Δt_{next} less than Δt_{min}, Δt_{next} is set to Δt_{min}. To make sure that a failure does not stick undetected for a long time, it is best to also set $\Delta t_{next} < \Delta t_{max}$. Figure 5a shows a sample curve for an adaptive Δt, whereas Figure 5b depicts the failure probability during the different intervals for fixed Δts and the adaptive one from Figure 5a.

After detecting a failure, other processes must be tested again, regardless of their next checking time. This way, we can check if the failure affected other processes or was introduced by the failure of another process.

5.1.1 Calculating the parameters

To create a fitting curve, we need to know the parameters k, λ_c and λ_{im}. k and λ_{im} will be set to fixed values. They can be estimated in advance by empirical values of failures.

For every device that was already used for the PAN-C, we can adaptively compute λ_c with an exponentially weighted moving average (EWMA) filter. As for a constant failure rate, the MTTF is the inverse value of λ_c. We can use the last time to failure to update our estimation. Devices that are known to stay in the network for longer times will be checked less frequently than devices that disappear after short times.

5.1.1.1 Example

Assume we have a smartphone with a MTTF of two years, the battery holds for 24 hours and we expect the device to stay in the network for 8 hours. If we experience a migration failure, it is expected to occur in the first second. If the software has a failure we expect it to occur in the first minute. The MTTF of the whole system can be calculated by [2]:

$$\frac{1}{\text{MTTF}} = \sum \frac{1}{\text{MTTF}_{\text{subsystem}}} .$$

We do that separately for the infant mortality failure and for the failures with constant failure rate. The migration failure and the software failure are part of the infant mortality, with an MTTF of about 0.9836 s. The MTTF for this is the expected value of the Weibull distribution, which is calculated by:

$$\text{E}(X) = \frac{1}{\lambda_{\text{im}}}\Gamma\left(1+\text{k}\right) \quad \Longrightarrow \quad \lambda_{\text{im}} = \frac{\Gamma\left(1+\text{k}\right)}{\text{MTTF}} .$$

Assuming $\text{k} = 0.2$, we calculate $\lambda_{\text{im}} = 0.9334\ \frac{1}{s}$. The lower we set k, the more we decrease the influence of infant mortality failures to the long term failure rate.

λ_c is simply the inverse of the MTTF for the device, the battery and the device availability. By that we get $\lambda_c = 4.63 \cdot 10^{-5}\ \frac{1}{s}$. The Round-Trip-Time (RTT) between two neighbor nodes in IEEE 802.15.4 can be estimated with 20 ms (including MAC retries and Back-offs), so Δt_{min} should not have a lower value. To also allow other packets in the network to freely flow, we set $\Delta t_{\text{min}} = 80$ ms. Δt_{max} is set to one second, so we are still able to catch failures after at least that time.

Those calculated values were used in the example Figures 5a and 5b. The maximum failure probability per interval was set to $\text{fail}_{\text{max}} = 0,62\%$.

When the smartphone disappears from the network after two hours, λ_c will be recalculated for the next time. The experienced time to failure would result in a $\lambda_{\text{c,exp}} = 1.39 \cdot 10^{-4}\ \frac{1}{s}$. A new λ_c will then be calculated by:

$$\lambda_{\text{c,new}} = \alpha\lambda_{\text{c,old}} + (1-\alpha)\lambda_{\text{c,exp}} .$$

5.2 Defeating Failures

When failures occur, it is important to understand their cause. When the device failed, the service needs to be switched to another device, but if it was a random failure it might be sufficient to restart the process. On the other hand, if the service program for that device is error-prone, the service needs to be updated.

Of cause, switching the service to another device is not an option if no other is available. If the device is still available and no other is capable of fulfilling it, the service needs to be restarted. Otherwise, if the device is not available anymore, the service must be set on hold.

The checking of devices and services by the RS must not only rely on a simple echo ping, but must ask for the devices and services state. Services might need to do a simple job, to

see, if the service is still working properly. Answers by devices need to inherit the battery status, link quality, RAM usage and a version of the information used on that device. With this information, we can derive a probable cause for the failure. We can see if the information is out of date, the processes ran out of ram, the link went weak or if the device went out of energy.

The gathered information about the device can also influence the next checking time. A device responsible for the Internet connection with a bad link quality to the WiFi should be tested more often, than with a good link quality.

Another way to defeat failures is to have redundancy. We already mentioned to store data redundantly on the devices of the network, but we can also have several devices fulfilling the same service, as it is done on aircrafts. But in contrast to aircrafts, we have a dynamic system with changing devices and can only do that, when spare resources are available. The up-lying applications have to make their requests through the RS, unknowing where the services are fulfilled. That way failures can be masked from the applications and only get an error, when a service is not able to run anymore.

5.3 Expected Improvements

By implementing this adaptive checking behavior, we expect to keep the probability of undetected failures below a set maximum failure probability $P_{fail_{max}}$, while also not keeping the network occupied with checking packets and having a low energy consumption. Network capacities force us not to stick to that optimum, when the failure rate would expect a checking time below the round-trip-time. The adaption by an EWMA filter will make sure, that the assumed hazard rate will reflect the experienced disappearing of a device.

With the RS we also mask failures of devices from applications. Only when service can not be fulfilled anymore, the applications will be informed and can take actions on their own.

6 Conclusion and Future Work

In this work we presented our vision of a fail-safe dynamic Patient-Network for health monitoring. To achieve fail-safety, among other counter-measures we propose an adaptive checking of the functionality of the components by a Reincarnation Service (RS). Instead of a fixed interval to check the components, we use an interval corresponding to the expected failure-rate. By that, we hope to keep the probability of undiscovered failures below a set threshold, while not flooding the network with checking packets and also keeping the energy consumption low.

Those measures for a fail-safe distributed health monitoring system, have not been implemented and tested yet. We want to build an Omnet++ simulation to validate our assumptions. We also need to further investigate the possible steps to be taken, when a failure is discovered.

In this paper we only focused on an the adaptive checking of processes and devices. We have not discussed how to check the functionality of a process. It is also our aim to investigate in a fail-safe distributed storage and the prediction of disappearing nodes with LQE.

Other works of the AA4R project will focus on the security of the system, analysis of the data and validation of the model.

References

1 Szymon Acedański, Supratim Deb, Muriel Médard, and Ralf Koetter. How good is random linear coding based distributed networked storage. In *Proceedings of the WINMEE, RAWNET and NETCOD 2005 Workshops*, Riva del Garda, Italy, April 2005.

2 A. Birolini. *Quality and reliability of technical systems: theory, practice, management.* Springer, 1997.

3 Shyr-Kuen Chen, Tsair Kao, Chia-Tai Chan, Chih-Ning Huang, Chih-Yen Chiang, Chin-Yu Lai, Tse-Hua Tung, and Pi-Chung Wang. A reliable transmission protocol for zigbee-based wireless patient monitoring. *Information Technology in Biomedicine, IEEE Transactions on*, 16(1):6–16, Jan 2012.

4 Octav Chipara, Chenyang Lu, Thomas C. Bailey, and Gruia-Catalin Roma. Reliable clinical monitoring using wireless sensor networks: Experiences in a step-down hospital unit. In *The 8th ACM Conference on Embedded Networked Sensor Systems (SenSys 2010)*, pages 155–168, Zurich, Switzerland, November 2010. ACM.

5 Jorrit N. Herder, Herbert Bos, Ben Gras, Philip Homburg, and Andrew S. Tanenbaum. Minix 3: A highly reliable, self-repairing operating system. *SIGOPS Oper. Syst. Rev.*, 40(3):80–89, July 2006.

6 IBM Corp. *An architectural blueprint for autonomic computing.* IBM Corp., USA, October 2004.

7 JeongGil Ko, Tia Gao, R. Rothman, and A. Terzis. Wireless sensing systems in clinical environments: Improving the efficiency of the patient monitoring process. *Engineering in Medicine and Biology Magazine, IEEE*, 29(2):103–109, March 2010.

8 Sape Mullender, editor. *Distributed Systems.* ACM, New York, NY, USA, 1989.

9 Kien Nguyen, Thinh Nguyen, Y. Kovchegov, and Viet Le. Distributed data replenishment. *Parallel and Distributed Systems, IEEE Transactions on*, 24(2):275–287, 2013.

10 F. Stanley Nowlan and Howard F. Heap. Reliability-centered maintenance. Technical Report AD-A066-579, United Airlines and Office of Assistant Secretary of Defense, December 1978.

11 Rodrigo Rodrigues, Barbara Liskov, Kathryn Chen, Moses Liskov, and David Schultz. Automatic reconfiguration for large-scale reliable storage systems. *IEEE Transactions on Dependable and Secure Computing*, 9(2):146–158, March 2012.

12 Stefan Unterschütz, Andreas Weigel, and Volker Turau. Cross-platform protocol development based on omnet++. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 278–282, ICST, Brussels, Belgium, 2012. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

13 M. Xie and C.D. Lai. Reliability analysis using an additive weibull model with bathtub-shaped failure rate function. *Reliability Engineering & System Safety*, 52(1):87 – 93, 1996.