universität
wien

DISSERTATION

Titel der Dissertation

Energy Efficient Resource Sharing for Networked Homes

Verfasser

Mag. Roman Weidlich

angestrebter akademischer Grad

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2010

Studienkennzahl lt. Studienblatt:	A 786 881
Dissertationsgebiet lt. Studienblatt:	Informatik
Betreuer:	Ao.Univ.Prof. Dr. Helmut Hlavacs

Acknowledgements

Thanks to my supervisor Prof. Hlavacs who thoroughly introduced me to research. He was always anxious to provide me with a work environment suitable for developing myself and always there to keep me on the right track.

Also thanks to the colleagues in my group for allowing me to feel at home at work due to the excellent organizational culture.

Further thanks to my parents who always gave me support that allowed me to pursue my self-fulfillment in academics.

And, over all, infinite thanks to my wife Nadia, who never had any doubts that I would succeed.

Contents

1 Introduction **1**
- 1.1 Wattage . . . 1
- 1.2 Power Saving . . . 5
- 1.3 Synopsis . . . 6

2 Related Work **7**
- 2.1 Resource sharing . . . 9
  - 2.1.1 Grid computing . . . 11
  - 2.1.2 Cloud computing . . . 13
- 2.2 Virtualization . . . 15
  - 2.2.1 Virtual Machine Monitor . . . 17
  - 2.2.2 Virtualization Technologies . . . 19
  - 2.2.3 PlanetLab . . . 22
  - 2.2.4 Live Migration . . . 25
- 2.3 Virtual Home Environment . . . 27
- 2.4 Own works . . . 31

3 Architecture **35**
- 3.1 Overlay . . . 35
- 3.2 Home States . . . 41
- 3.3 Front- and Backends . . . 43

4 Applications **47**
- 4.1 Download Sharing . . . 49
- 4.2 Video Encoding . . . 51
- 4.3 Home Management . . . 53
- 4.4 Replication . . . 54
- 4.5 Analytical Evaluation . . . 59
- 4.6 Measurement Study . . . 67
- 4.7 Model Extension . . . 70
- 4.8 Summary . . . 72

5 Simulation Model **75**
- 5.1 Basic Scenario . . . 82
- 5.2 Parameters . . . 86
- 5.3 Economic Model . . . 88

6 Evaluation **93**
6.1 Download Sharing (DS) . . . 94
6.1.1 Distribution . . . 97
6.1.2 Load . . . 99
6.1.3 Fairness . . . 100
6.2 Video Encoding (VE) . . . 102
6.2.1 Distribution . . . 105
6.2.2 Load . . . 107
6.2.3 Fairness . . . 107
6.3 Home Management (HM) . . . 110
6.3.1 Replication . . . 111
6.3.2 Failures . . . 113
6.3.3 Cluster recovery . . . 118
6.4 Combined Scenario . . . 124
6.5 Traffic Study . . . 127

7 Summary **133**
7.1 Conclusion . . . 133
7.2 Future work . . . 133

1 Introduction

Power saving, or energy efficiency, was and still is an important issue worldwide, even in the field of computers and computer networks. We all recognize the rising number of end users and home equipment that, of course, should ease our lives. Most people use computers at home, at work, or while traveling. Additionally, they often use these devices in a way that requires a stable connection to the Internet. When concentrating on the home or residential area we can observe a rising number of always-on and always-online equipment in a 24/7 manner like normal computers, media centers, and home automation devices, including sensors and actuators. Computers are running continuously for manifold reasons. For example, a continuously running and only downloading computer can be interpreted as power wastage, because this computer is normally underloaded and CPU time is only sparingly utilized.

This work explores power saving potentials for future networked home environments. The emphasis lies on the *future*, because, as we see later, more bandwidth and distribution enable better sharing and therefore more power saving. The achievable power saving is measured as the difference between the wattage caused by homes in the local case, where all homes execute their tasks locally without sharing, and the remote case, where homes concurrently execute their own and remote tasks.

The major problem with computers' power consumption is the relatively high consumption under low load or in idle state. If lightly loaded computers can be avoided and idle computers can be suspended, then power can be saved. To be more energy efficient it is crucial to reduce the power consumption for a given load. This can be done by resource and task sharing, albeit there are constraints like scalability, availability, reliability and fairness. Most content distribution systems are optimized versus performance, whereas my work does optimization versus energy efficiency.

1.1 Wattage

Always-on computers consume considerable amounts of energy worldwide and therefore energy efficiency has become a major topic in the last years. Households today contain a multitude of devices making our lifes more comfortable besides consuming energy to various extends. Especially even lightly loaded computers still consume energy, and of course exhibit at the same time unused resources.

In addition to increased CO_2 balance caused by high energy consumption, energy consumption is seen as major cost factor for servers. This is also becoming true for home networks. A doubling of energy consumption from 2000 to 2005 of volume, mid-

range, and high-end servers in the U.S. and worldwide[1] is reported [Koo07]. The total wattage reported in 2005 (including associated infrastructure) is equivalent to about five 1000 megawatt power plants for the U.S. and 14 such power plants for the world, tendency rising. Similar to server environments energy consumption is becoming a major problem in home networking, as energy costs tend to exceed that of hardware.

A similar tendency could be expected for always-on PCs. According to a 2006 survey commissioned by the EU [BA07], end devices in homes contribute significantly to the electricity consumption growth. According to measurements of ENERGY STAR[2] (a joint program of the U.S. Environmental Protection Agency and the U.S. Department of Energy) and Energy Star[3] from the European Union (Directorate-General for Energy and Transport) today's PCs consume around 100 watt when turned on (depending on the load) and only a few watt in suspended mode. Table 1.1 shows power consumptions in watt during 24/7 operation and busy or idle state of typical end user computers in 2008 [Lau08].

Computer configuration	**Idle W**	**Busy W**
AMD Opteron 144 (1.8 GHz), 1 GB RAM, Linux 2.6.17 (Ubuntu 6.10)	75	115
Intel Pentium 4 HT (2.8 GHz), 1 GB RAM, Nvidia GeForce FX5200LE, Windows Vista	70	125
Intel Pentium 4 HT (3.2 GHz), 2 GB RAM, Nvidia GeForce 6800, Windows XP SP2	90	155
Intel Core 2 Duo 6300 (1.87 GHz), 2 GB RAM, AMD Radeon X1300, Windows Vista	83	103
AMD Athlon64 X2 4400+ (2.3 GHz), 2 GB RAM, AMD Radeon X300, Windows Vista	70	125
Intel Core Duo (1.83 GHz), 17"-iMac, 1 GB RAM, Mac OS X 10.4.9	52	68
Intel Pentium D (2.8 GHz), 512 MB RAM, Intel 82945G, Linux 2.6.20 (Fedora Core 6)	100	184
Intel Pentium D (2.8 GHz), 512 MB RAM, Intel 82945G, Windows XP SP2	98	191
Playstation 3, Linux 2.6.21 (Fedora Core 7)	180	200

Table 1.1: Mean power consumption of typical end user computers in watt (W) in 2008.

On average a busy computer consumes 148 watt or idle still 100 watt. Another listing[4] is shown in Table 1.2, where power consumptions include a display for desktops and battery charging for laptops. The average power consumption of desktops is then 137.9 watt and for laptops 53.3 watt which is still a considerable amount of energy under 24/7 use.

Another measurement [Kat08] investigated wattages of CPUs, GPUs, and chipsets

[1] `http://hightech.lbl.gov/documents/DATA_CENTERS/svrpwrusecompletefinal.pdf`
[2] `http://www.energystar.gov`
[3] `http://www.eu-energystar.org`
[4] `http://www.upenn.edu/computing/provider/docs/hardware/powerusage.html`

Computer configuration	**Busy W**
Apple iMac/Intel 24-inch, Core 2 Duo, 4.0 GB RAM, 640 GB hard drive, Mac OS 10.6.2	150
Apple iMac/Intel 20-inch, Core 2 Duo, 2.0 GB RAM, 320 GB hard drive, Mac OS 10.6.2	117.5
Apple iMac/G5 20-inch, 2.0 GHz PowerPC G5, 1.0 GB RAM, 250 GB/7200 RPM hard drive, Mac OS 10.4.9	105.5
Dell OptiPlex 755 w/19-inch Dell LCD, Core 2 Duo, 2.0 GB RAM, 160 GB/7200 RPM hard drive, UltraSharp 1906FPV display, Windows Vista Business	128
Dell OptiPlex 745 w/19-inch Dell LCD, Core 2 Duo, 2.0 GB RAM, 100 GB/7200 RPM hard drive, UltraSharp 1907FPV display, Windows Vista Enterprise	122
Dell OptiPlex GX620 w/17-inch Dell LCD, 3.6 GHz Pentium 4 521, 1.0 GB RAM, 160 GB/7200 RPM hard drive, UltraSharp 1704FPV display, Windows XP Professional SP2	167
IBM ThinkCentre M52 w/19-inch IBM LCD, 2.8 GHz Pentium D 820, 1.0 GB RAM, 160 GB/7200 RPM hard drive, UltraSharp 1703FP display, Windows Vista Enterprise	175
Apple MacBook Pro 15-inch, 2.5 GHz Core 2 Duo, 4.0 GB RAM, 250 GB/5400 RPM hard drive, Mac OS 10.5.6	41
Apple MacBook Pro 13-inch, 2.53 GHz Core 2 Duo, 4.0 GB RAM, 250 GB solid state drive, Mac OS 10.5.8	58
Dell Latitude E4200 12-inch, 1.4 GHz Core 2 Duo, 3.0 GB RAM, 128 GB solid state drive, Windows Vista	68.5
Dell Latitude D420 12-inch, 1.06 GHz Core Solo, 1.0 GB RAM, 40 GB/4200 RPM hard drive, Windows Vista Ultimate	51
Lenovo ThinkPad X41 Tablet 12-inch, 1.5 GHz Pentium M, 1.5 GB RAM, 40 GB/4200 RPM hard drive, Windows XP Tablet SP2	51
Lenovo ThinkPad T400s 14-inch, 2.53 GHz Core 2 Duo, 3.0 GB RAM, 250 GB hard drive, Windows Vista Business SP2	50

Table 1.2: Mean power consumption of typical end user computers in watt (W) in 2009.

(motherboard). Deduced from that data, the average wattage of todays CPUs, GPUs and chipsets are about 89, 114.3 and 43.1 watt under load or 42.7, 63.3 and 43.1 watt if idle. Other end user equipment, like gateways (routers) or modems, consumes approximately 7 to 15 watt and devices like sensors and actuators consume only a few milliwatt or use a battery as power source.

Energy efficient computing is not a new topic. With the need of longer battery life in laptops for instance, several techniques have been developed as local power saving mechanisms. Power can be saved with various well known techniques. First, the processor can be powered down with mechanisms like *Enhanced Intel Speedstep* (EIST) and *Demand-Based Switching*[5] [Win07b], *Enhanced AMD PowerNow!* and *AMD Cool'n'Quiet Technologies*[6]. These technologies enable slowing down the clock speeds (clock gating) or powering off parts of the chips (power gating), if they are idle [BHK+07, Win07a].

In [ISG03, AIS04] algorithms for online *Dynamic Power Management*[7] (DPM) are presented. These algorithms are based on online learning and more than two states as idle and shutdown. The power state of a device is changed accordingly to current information available at runtime through sensing whether the device has been left idle or not. By sensing user-machine interaction different hardware parts can be incrementally turned off stepwise. This is usually applied to mobile devices but can also be used for desktop computers.

Other energy saving methods relate to communication energy cost. Battery lifetime of wireless devices can be improved by addressing several layers of the network protocol stack (Physical, Data Link (LLC, MAC), Network, and Transport layers), operating systems, middleware, and applications [JSAC01, YWL+06, GCN05].

All mentioned techniques can be categorized as local energy saving techniques.

There already exist trends for data centers to measure and reduce energy wastage, especially caused by underutilized hardware while taking cooling cost into account [Int06]. Thus, there is a requirement for the consolidation of servers. An energy saving method must be investigated within the context of data centers for turning off and on a part of machines as needed.

However, turning hardware off does not always imply energy efficiency. Energy efficiency can be measured [KBN+06] in performance per watt as with *SPECpower ssj2008*[8] or with a benchmark like *JouleSort* [RSRK07].

Virtualization [Int07] is a tool for consolidation of hardware. Virtualization can be seen as splitting an underlying hardware entity into smaller identical virtual entities which run isolated from each other. In data centers for instance, the rack-mounted servers were configured to run a single workload to guarantee reliability, availability, and scalability for the service. These servers are located in a controlled closed envi-

[5] `http://www.intel.com`
[6] `http://www.amd.com`
[7] `http://dynamicpower.sourceforge.net`
[8] `http://www.spec.org/power\_ssj2008`

ronment like a high bandwidth network grid, which can support migration of virtual entities between them without challenges existing for home networks like security, varying availability, and decentralized load sharing via the Internet. With virtualization a service is dedicated to a virtual entity, but can run transparently on any available server next to several other virtual entities. This effective consolidation of servers, i.e. running a machine at higher utilization, is usually administrated by central management mechanisms.

1.2 Power Saving

To achieve power saving through cooperation of home networks, power consumption should be globally minimized, whereas energy efficiency should be globally maximized. For a number H of different homes (h_i, $1 \leq i \leq H$) the basic energy consumption $E(T)$ in joule over time T can be expressed as

$$E(T) = \sum_{i=1}^{H} \int_{0}^{T} P_{h_i}(t)\ dt$$

where $P_{h_i}(t)$ is the power (joule per time unit or watt) consumed by home h_i. In absence of measurement possibilities, the power consumption of a home might as well be estimated by assigning an energy class level to the home. To calculate the energy efficiency, the workload introduced by homes is related to power consumption, thus the work carried out by all homes is defined as

$$L(T) = \sum_{i=1}^{H} \int_{0}^{T} L_{h_i}(t)\ dt$$

where $L_{h_i}(t)$ describes the load caused by a home at time t. Similar to [RSRK07] the global energy efficiency η of the system, which should be maximized, is

$$\eta(T) = \frac{L(T)}{E(T)} \tag{1.1}$$

with $E(T) \neq 0$. If the power consumption $E(T)$ of all homes can be reduced, the global energy efficiency η will increase with fixed load $L(T)$. Based on this, a distributed solution is proposed, where load in form of tasks created by homes is distributed among them to consolidate provided resources within a network of homes.

Networked systems offer the possibility to share resources, which has been extensively researched to achieve higher performance and availability in parallel and distributed systems. Using resource sharing to save energy is a radically new approach which needs thorough investigation.

The aim of my work is to investigate how to save energy by minimizing the number computers consuming power. Unused computers which are not necessary for the over-

all system operation will be suspended and only woken up by the system if needed. Additionally, computers should contribute and use resources fairly. The goal of my system is to find underutilized computers. To access computers inside a home, a special software must run. This software can be seen as threefold; it is responsible to hold the connection to the network of homes, to react adequately to events and changes and to do this under some security precautions. Each computer that should participate to the system must run such software. Furthermore this software must provide virtualization functionality. To sum up this software (acting as a P2P-client) denotes an entry point of the system into the home. The user can allow or prevent the system to control computers. A common logic, consisting of the behavior of homes manifested as distributed algorithms included in the software, ensures that homes act in the same way. The goal is to let homes self-organize for emerging a local power saving goal and to lower the global wattage caused by computation in the home network domain.

My work shows that it is possible to save power through resource sharing on the application layer in the domain of home networks without central management, based on current technologies for interconnection and mediation.

1.3 Synopsis

This chapter introduced the problem and goal of my work, whereas the next Chapter 2 gives an overview about related topics. Chapter 3 proposes an architecture for a network of homes suitable to manage home states and load. On top of this architecture the applications *Download Sharing*, *Video Encoding* and *Home Management* – completely different in their resource usage and applicable for power saving – are defined in Chapter 4. Especially Section 4.5 goes into detail with the analytical model and outlines the reachable gain of energy efficiency. Chapter 5 describes a simulation model that validates the analytical model and is used for extensive experiments. Chapter 6 presents simulation output and results are discussed. Finally, Chapter 7 provides a summary, conclusions and some ideas for future work.

2 Related Work

The home gateway as optimization point is suggested in [PCLP08, OHP08]. A home gateway connects a network to the access network. Normally the home gateway is designed to be always-on and connected for services inside the home as well as for requests from outside. An activity scheme is considered to determine periods without traffic flows. Power can be saved by using a newly created hardware component, the *Network Protocol Agent*, that acts as traffic monitor and triggers sleep and wakeup mechanisms for partly suspending or resuming the home gateway.

The works [PCLP08, OHP08] can be understood as supplement for my own concept which is based on applications assuming a stable connection to the access network in form of an always-on home gateway, whereas the home gateway is not the optimization point for power saving.

The work [LKP+09] is a follow-up of [PCLP08] where an *Energy-aware Framework* is introduced to manage and control the power status of consumer electronic devices in homes. The framework is based on a power management scheme based on the *Advanced Configuration and Power Interface*[1] (ACPI). Minimum controllable resource units are defined as *Energy-aware Control Elements* controlled by an *Energy-aware Plug and Play* protocol that provides an open API for managing them.

The work [LKP+09] can be seen as specific case for a locally or remotely running home management application as suggested in my own work, where sensors and actuators are controlled for supporting the residents of a home.

In [MSHK07] a home network consisting of control devices, sensors, information appliances, data devices and AV devices is modeled and analyzed to learn about the power consumption of devices inside under a typical use case. Energy efficient hardware and power management mechanisms, already used for battery-powered wireless devices, are applied on home devices. Power saving is reached by avoiding unnecessary communication between home devices.

The work [MSHK07] illustrates a home management application, as suggested in my own work, very well. However, I concentrate on inter-home cooperation and also on reliability and not on the intra-home management at all.

The assumption of instantaneous turning-on mechanisms for computers, my work considers, is investigated in [BH07a]. With improvements like replacing the *System off* by the *Suspended to RAM* method or the redesign of the circuit for less standby power consumption the wattage of a PC in standby mode can be lowered from 2.4 watt down to 0.7 watt while ensuring resuming within 3 seconds.

[1] An open specification developed by HP, Intel, Microsoft, Phoenix and Toshiba.

This work is a proof of concept for one aspect of my work where computers within homes can be turned off and on by the system as required for overtaking remote tasks and enabling the computer of the remote home to go into a low power mode.

In [HZCS09] a programmable bandwidth aggregation system for home networks is designed and implemented. For annihilating the shortcomings of current *Bandwidth Aggregation Systems* (BASs), deployed in public networks when they are directly used in context of home networks, they require to be easily and dynamically adaptable. The suggested *Programmable Bandwidth Aggregation System* (PBAS) can provide home networks improved performance through access bandwidth sharing. The PBAS can be understood as an open, scalable platform that exploits programs represented as Java byte code, and transparently aggregate bandwidth for them. The speed up is achieved by bandwidth sharing and multi-path communication.

The work [HZCS09] covers a further aspect which is not in scope of my own work. Of course, having two access connections at home would result in more available bandwidth enabled by a PBAS, but for my work it is sufficient to consider one access connection. As seen later, the bigger issue as total bandwidth is the type of access in terms of synchronous or asynchronous access.

An energy-efficient routing scheme for home automation is introduced in [OBC05]. Because of limited battery power of sensors, a point of application, as known from wireless sensor networks, is the routing protocol between the home base and sensors. The routing scheme here divides the home area into sectors and locates manager nodes for each sector. Manager nodes are also sensors and additionally collect data from sensors for forwarding it to the home base in bulk. This reduces the power consumption of normal sensors compared to conventional sensor routing schemes.

In the same area of interest is [CKC+09]. Data fusion is considered as a process for decreasing the transmission number of similar sensor values for home automation and thus saving power in sensors. Sensors' decisions for sending new data is based on previously sent data and on the last value sent by one-hop neighboring sensors. In this routing scheme temporal and spatial correlation is taken into account.

Also [KES+07] falls in the domain of home automation and sensors. In this work a *DLNA*[2]-compilant home network is connected to a *ZigBee*[3] sensor network for aiming at energy efficient sensor control through switching between unicast and broadcast for the sensor data gathering method in accordance with current network conditions. A home gateway architecture connecting the two networks is proposed.

A self-organizing clustered topology with a periodic and query-based data aggregation method is proposed in [KKSK09a] for saving sensor power. For some specific time one node is chosen as cluster head for collecting and routing the data and this role rotates within the cluster with increasing network lifetime. Then in [KKSK09b] the clustered topology is applied to a ZigBee network to make it self-organizing where the cluster head is chosen based on leftover power what outperforms normal ZigBee

[2]Digital Living Network Alliance, `http://www.dlna.org`
[3]ZigBee Alliance, `http://www.zigbee.org`

meshed networks in terms of energy consumption.

The work [PRV09] also aims at minimizing the communication overhead and therefore power consumption of sensors with a protocol for home automation emphasizing energy efficiency and service discovery. This is reached by device specific low power listening schemes for a minimal number of transactions.

These works address a further point of interest in power saving in the domain of home management. In my work optimization of sensor action is out of scope. In fact, the constraint of having an always-on home base is relaxed by executing home management remotely, where sensor data is sent to a remote home and control messages back.

Power saving for mobile devices is examined in three ways in [SEP05]. Firstly, at the network level a new routing protocol reduces power consumption on average by 15 %. Secondly, at the processer level the memory bandwidth directly influences performance and CPU frequency, which in turn affects the CPU speed setting and therefore the power consumption of the CPU. Thirdly, most interesting is local versus remote processing, investigated under relative performance and power consumption of local and remote systems, transmission bandwidth and network congestion. The communication and processing costs must be taken into account for deciding where to execute a task.

[SEP05] already rudimentary dealt with local versus remote execution of load in the area of mobile devices. My work is fully concentrated on this aspect for network of homes with less restrictions on computing power and network bandwidth.

2.1 Resource sharing

Resource sharing is strongly related to *Peer-to-Peer* (P2P) networks most popular for content distribution. P2P networks offer ad hoc collaboration for aggregating and sharing large amounts of resources in computer networks. Logical links are defined on top of a physical network. A single logical hop in the P2P network can be mapped to several physical hops in the underlying network.

As resources mainly disk space, network bandwidth and CPU time are considered. There is no distinction between resource consumer and resource provider; both roles are referred as *Peer*. In pure P2P networks [SMK+01] all peers are assumed to be equal. A peer can download data from another peer at the same time as uploading to other peers. In hybrid P2P networks some peers are distinguished from other peers, i.e. some peers have different capabilities than others [Tut04].

The most famous application of resource sharing is file sharing. Files are stored among many peers; the P2P network can be understood as distributed database. Many network protocols for file sharing have already emerged [ATS04].

Basically, P2P networks are characterized according to their topology [SW05]. In structured networks the range of unique identifiers is equally distributed over all peers. Each peer can be explicitly found; each peer has full knowledge about the network.

In case of churn[4] this implies repair mechanisms for ensuring a fully qualified address space of identifiers. Also in unstructured networks there are unique identifiers but peers do not have full knowledge about the network; the network grows and shrinks randomly. Such a network may be clustered with varying diameters[5]. Churn is less interruptive and this makes this type of networks more failure resistant. The drawback is that rare content may not be found by the system. In structured networks also content is addressed and each peer can find out where a certain data item is stored. These networks establish an addressing scheme that enables the addressing of peers as well as the addressing of content.

A further distinction are hierarchical networks compared to flat ones. In a flat network each peer is truly equal as initially considered. To overcome increased management effort in large networks, hierarchy in form of peers with special roles where introduced. Super peers act as information holder and entry point into the network. This concept is also used for bootstrapping where peers are given a list of previously known super peers. A peer connects to such a super peer for receiving references to other peers. Super peers themselves build their own logical network within the P2P network. This network inside the network provides peer and content discovery. Since super peers also act as normal peers, these peers must share more resources, mostly network bandwidth.

Whenever the logical addressing of a P2P network is established on a native network like the Internet, this can be understood as overlay. The overlay interconnects nodes for a special purpose. Nodes can participate in several overlays at the same time.

A special issue of resource sharing is fairness. A peer should exhibit a trade-off between consumed and offered resources and must be prevented from free-riding [FC05, RL02]. A peer that only consumes remote resources without contributing local resources are called a free-rider. The work [NWD03] presents an economic model to create incentives for fair resource sharing. Usage files must be exchanged and tell a peer about the sharing ratio of another peer. This randomly performed auditing mechanism between peers increases the probability that faked usage files are detected and malicious peers are penalized.

An incentive mechanism for fair cooperation, based on past interactions and a distributed algorithm encapsulated in a middleware, is presented in [AGR05]. A peer is basically not considered as cooperative or selfish, but rational and changes its behavior over time.

P2P networks solve three problems an architecture, as proposed in my work, must deal with: home network interconnection, resource mediation and resource allocation. What is not solved so far is fairness, because this is still an open research topic. My work does not solve fairness in P2P networks, but points out the cost of fairness.

[4]Churn denotes the stochastic process of peer turnover as occurring when peers join or leave the system.
[5]The longest distance in hops between to arbitrary peers.

2.1.1 Grid computing

In *Grid* computing high-end computers build a high performance network [Sto07]. Moreover, Grids are multi-institutional virtual organizations mostly consisting of networks by scientific and research communities. A commonly agreed definition of Grid computing by the *Open Grid Forum*[6] (OGF) is:

> "A system that is concerned with the integration, virtualization, and management of services and resources in a distributed, heterogeneous environment that supports collections of users and resources (virtual organizations) across traditional administrative and organizational domains (real organizations)."

The OGF introduced an architecture for Grid services interaction, called *Open Grid Services Architecture*[7] (OGSA) which is based on the *Web Service Resource Framework* (WSRF) specified by the *Organization for the Advancement of Structured Information Standards*[8] (OASIS).

The Grid computing model is a client-server model where servers offer specialized, reliable, highly advanced and sophisticated scientific applications. Grids require a predefined administrative infrastructure enforcing virtual organization policies. The roles, responsibilities and privileges of collaborating users are also predefined. In contrast, the P2P paradigm provides direct communication between peers without warranting any policy enforcement. Responsibilities and privileges of participating users are not defined a priori and are spontaneous. Every peer is responsible for maintaining the access to local resources.

The work [ALM04] is a comparison of Grid computing to P2P computing. Grid computing focuses on performance, control, security and *Quality of Service* (QoS), whereas P2P computing focuses on fault tolerance, resilience, decentralization, cooperation and best-effort. Grids are collaborative, because they comprise heterogeneous resources managed by several entities across multiple institutions. The institutions usually are closed environments enforcing strict policies describing their cooperation within the grid. Grids provide non-trivial QoS assurances by e.g. dedicated high-speed networks and can be distinguished into three models:

- *Traditional grids* are closed networks, tailored to the special requirements of driving grid members. Significant management overhead arises due to role-based usage by service provider, service developer, administrator and users despite the closed number of members.
- *Ad hoc grids* include also participants from non-scientific context, e.g. private home computers. Instead, the grid only consists of a predefined *Virtual Organization* (VO), the cooperation and resource power is mainly based on a transient,

[6] `http://www.ogf.org`
[7] `http://www.globus.org/ogsa`
[8] `http://www.oasis-open.org`

short-lived collaboration of huge numbers of computers. For this purpose, the VO must appoint administrative privileges and credentials, and provide a common middleware to every grid member. Unfortunately the administrative overhead for the VO surpasses its utility in most cases.

- *Federated grids* are an extension of ad hoc grids. A federated grid is a generic grid architecture where resource consumption is not only limited to members of the VO alone. Every participant can submit tasks to the grid while providing own resources. An economic model ensures sufficient incentive for fair resources usage. A contributor gains credits for donation of his resources, or pays credits in case of a negative sharing balance. Thus, resource providers become also resource consumers. Beyond this idea the next level of grids could be called *Public Grids* very similar to P2P networks. Users could dynamically establish VOs in an ad hoc fashion enabling collaborative resource sharing.

Since P2P networking and Grid computing are converging in terms of functionality, my work is related to both areas. There is no best concept or protocol for driving a network of homes with the goal of power saving through resource sharing. My focus is rather to show that it makes sense to interconnect home networks for home user tasks on top of an existing P2P or Grid system.

Basically, in Grid computing it is assumed that each participant is cooperative without free-rider problematic. After Grid computing steps toward the commercial world, also the issue of fairness arises similar to P2P networks. Since Grid computing is intended for solving computationally intensive tasks, the sharing of CPU time is most important.

In [IAA07] incentives are proposed for resource providers and for negotiating the execution of tasks. The donation of resources is rewarded by the grid while new members have minimum credits. An incentive-based algorithm fosters cooperative behavior of members for the timely execution of tasks within a given deadline.

The work [kE07] gives a survey about Grid computing traffic patterns. The majority of Grid traffic is not made up of enormous volumes of data and it is disproved that the TCP/IP stack alone prevents Grids from working on their full potential. Grid traffic can be classified in applications causing datasets under 10 MB, 100 MB and 1 GB. The surveyed applications run on top of middleware solutions. 93 % of the surveyed applications are deployed on dedicated clusters or mixed with desktop computers, where only 7 % of the surveyed applications are deployed solely on desktop computers.

Also for task sharing Grids are considered [LCP+05], because it is alluring to use worldwide idle desktop computers at homes. This is exploited by many projects running on top of the software platform *BOINC* [And04] (Berkeley Open Infrastructure for Network Computing). BOINC is an open source middleware for desktop Grid computing [Sch07]. BOINC assures that tasks are distributed above all participating computers and results are collected. A desktop grid differs from a dedicated grid in terms of performance, whereas in dedicated grids computers exclusively running for the

grid, in desktop grids any computer of the network edge can volunteer. A desktop grid is therefore a public server-centered resource sharing system and fits well with highly parallel computing. Projects hosted by BOINC span over various research fields as earth sciences, astronomy and physics, biology and medicine, mathematics and strategy games, etc. A user of BOINC can join several projects and divide idle cycles of his computer between them. The most famous project is *Seti@home*[9] (Search for Extraterrestrial Intelligence) to detect narrow-bandwidth radio signals from space. This project disseminates tasks with small data size and high CPU load. Therefore, it is easier to collect small sized data items, than transferring huge amount of data to clients. Additionally, tasks are not strictly ordered and can be assembled as they arrive.

BOINC has similarity to my approach but lags the missing decentralization. Where in BOINC tasks are sent out by a central server, in my work tasks can be sent out from every peer as usually in P2P networks. Further, I do not stick to small-sized tasks and also consider increasing tasks sizes; an original task object can consist of a description and source data, whereas a completed one may contain much more data. What my work and BOINC have in common is the software each client has to run. But instead of communicating only with the server, the software assumed in my work must behave like a P2P client.

2.1.2 Cloud computing

The latest step of on-demand and distributed computing is called *Cloud computing* which can be understood as advancement of Grid computing and aims at scalable services for end-users on the Internet [PRSBM+09]. Where a Grid is historically a closed environment with task scheduling, Service Level Agreements (SLAs) and policies, Cloud computing is not dedicated to virtual organizations or companies. A Service Level Agreement (SLA) is a contract between a service provider and service user. It regulates levels of *Quality of Service* (QoS) in terms of security, availability, performance etc. According to [BYV08] there is a requirement for market-orientation in Cloud architectures for regulating the supply and demand of resources. Service requests with specific QoS requirements must be met by providers for establishing SLAs. Dealing service requests as equivalent is usually done by centric resource management without incentives for the providers to share their resources. Thus, fairness is also an important issue in Cloud computing.

Cloud computing is still an open term and there exist a couple of definitions of it [Gee08], whereas in [BYV08] a Cloud is defined as follows:

> "A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers."

The term *Cloud* refers to the fact, that the Internet is often illustrated as a cloud

[9] `http://setiathome.berkeley.edu`

in network diagrams. Since the Cloud provides applications and storage this encourages companies to outsource their own IT infrastructure to Cloud providers. Cloud providers sell applications, storage or even computation time similar to public services like electricity, water, gas and telephone since the user only pays for utilization.

According to [Vou08], Cloud computing implies a *Service Oriented Architecture* (SOA) [PvdH07, ZHvdA06, SHM08, KLS08] where users access an integrated suite of functions through composition of services from possibly many different networks. SOA addresses the requirements of loosely coupled, standards-based and protocol independent distributed computing and allows the integration of applications as reusable services with platform independent specifications that abstract underlying complexity. These services are delivered through next-generation data centers that are built on compute and storage virtualization technologies. Consumers will be able to access applications and data from a Cloud anywhere on demand. The Cloud appears to be a single point of access.

The term *Software as a Service* (SaaS) stands for remote access on single applications via a web browser. The web browser is meanwhile a very suitable tool for accessing most important functionalities required by information workers. Since *Web 2.0*[TLT09] allows users not only to consume information on web pages, but also to add and customize information, the look and feel of web applications is not far away from native applications running on the desktop. Over this, *Cloud platforms* enable users to create own applications within the Cloud space which are then be used as SaaS.

As in the *Amazon Elastic Compute Cloud*[10] (Amazon EC2) *Virtualization* is applied. As execution environment Virtual Machines (VMs), e.g. based on Linux, encapsulate an operation system with all applications and data. All services inside Amazon EC2 run into VMs.

Cloud computing is not directly related to my work but must be considered for being in line with the trend and more important for fulfilling future end user requirements. Thus, the network of homes can be assumed as Cloud where homes conceptually have access to a single entity not caring about location of service and data. Further, virtualization is assumed to encapsulate tasks generated by homes and executed locally or remotely. As with Grid computing, also Cloud computing lags true decentralization. In my work no server is in place as information broker; thus a plain P2P system is enough for implementing my architecture.

Table 2.1 summarizes typical differences between the three discussed resource sharing concepts. Because of the focus of this work on home networks and private computers are involved, a P2P approach seems to be the most fitting resource sharing concept.

cdffdc

[10] `http://aws.amazon.com/ec2`

	P2P	Grid computing	Cloud computing
Organization	decentral	central	central
Access	open	closed	closed
Allocations	High number	Low number	High number
Request Type	tiny	big	small
Task Scheduling	no	yes	no
Reliability	no	yes	no
Service-Level Agreements (SLAs)	no	yes	yes
Dedicated Hosts	no	yes	yes

Table 2.1: Typical differences between resource sharing concepts.

2.2 Virtualization

Nowadays, system virtualization is successfully used to consolidate services in data centers. Several services can run separately on top of a single hardware, saving hardware costs, space and energy. In system virtualization a *Virtual Machine* (VM) is created, i.e. a full host is virtualized consisting of virtual CPUs, virtual memory, virtual hard disk, virtual network. A VM is a perfect recreation of a real machine in such a way that an operating system can be installed on it without being aware of the resource virtualization.

Virtualization in distributed systems refers to abstraction from physical characteristics and location of computing resources. Virtualization is used to aggregate a pool of hardware resources, to provide load sharing, to save hardware and energy, and to hide the complexity of a distributed system. But also to split hardware resources into separated parts as VMs which can run in parallel.

Large resource sharing systems leverage virtualization concepts for building clusters as well as for administering large networks [WCC+08]. VMs allow administrators to better control available resources through consolidation of hardware while also protecting the host from faulty or malicious software. This allows administrators to provide sandbox-like environments with little performance reduction.

According to [QNC06] the main motivations for virtualizing computer resources are:

- A VM provides a confined environment where non-trusted applications can run.
- A VM can limit hardware resource access and usage through isolation techniques.
- A VM allows adaption of the runtime environment to the application instead of porting the application to the runtime environment.
- A VM allows using dedicated or optimized operation system mechanisms for each application.
- Applications and processes running within a VM can be managed as a whole.

The IT industry needed major efforts to make data centers more energy efficient. Energy efficiency in data centers relies heavily on virtualization. In data centers virtual server entities are created, copied, moved and deleted depending on management decisions. Energy efficiency is then achieved by consolidating hardware and reducing redundancy. Some computers run at higher load while idle computers are hibernated or even stay in a low power mode. The management process itself is borrowed from Grid technology, where virtualized hardware resources are allocated in a centralized way. In order to reduce the complexity of the management process and simplify the trust relationship, shared server hardware is usually located close to each other (e.g. in racks) and interconnected with high-bandwidth links.

Users may access aggregated hardware as a single virtual environment (e.g. a single Linux shell). This kind of virtualization is shown in Figure 2.1 a). A number of real machines is aggregated to a single virtual environment. In contrast to the compositional Grid virtualization, server virtualization uses virtualization methods in a segmenting manner. Server virtualization aims to split hardware resources into several smaller virtual environments, enabling more than one virtual environment on a single hardware. Servers are virtualized to achieve load-balancing, to increase resilience, and to save energy by consolidation. In Figure 2.1 b) this kind of resource virtualization is shown. A single hardware is split into several virtual environments.

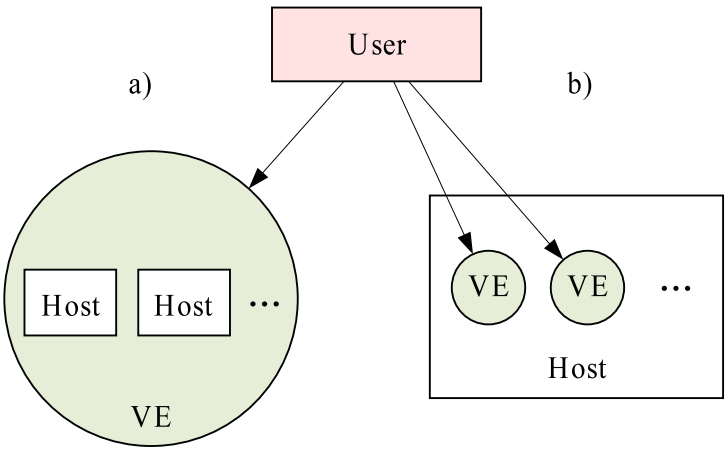

Figure 2.1: a) Grid virtualization and b) server virtualization.

My work assumes both types of virtualization. First Grid virtualization is applied for the view of the single home on the rest of the network of homes. A unified interface must provide access to any resources available at the moment. This can be done by a software, i.e. a P2P client. Server virtualization is used as local view within homes. Since local and remote tasks are executed concurrently, each of these tasks can be represented by a VM. All VMs together are subdividing disposable resources. With the help of these two virtualization concepts and in conjunction with P2P a system as proposed by my work can be implemented.

2.2.1 Virtual Machine Monitor

As surveyed in [RG05] at the end of the 1960s, the *Virtual Machine Monitor* (VMM) was introduced as software layer which partitions a hardware platform into VMs. Each of these VMs has been sufficiently similar to the underlying physical machine to run existing software unmodified. At this time, general-purpose computing was the domain of large and expensive mainframe hardware. Users found that VMMs provide a compelling way to multiplex resources among multiple applications. This technology flourished both in industry and in academic research. The 1980s and 1990s brought modern multitasking operation systems and a simultaneous drop in hardware costs, which eroded the value of VMMs. To reduce the effects of system crashes and lag of performance, system administrators again resorted a computing model with one application running per machine. This increased hardware requirements, imposing significant cost and management overhead. As mainframes gave way to minicomputers and then PCs, VMMs disappeared to the extent that computer architectures no longer provided the necessary hardware to implement them efficiently. By the late 1980s, neither academics nor industry practitioners saw VMMs as much more than a historical curiosity.

In the 1990s, Stanford University researchers began to look at the potential of VMs to overcome difficulties that hardware and operation system limitations imposed. This time the problems stemmed from *Massively Parallel Processing* (MPP) machines that were difficult to program and could not run existing operation systems. Besides, moving applications that once ran on many physical machines into VMs and consolidating those VMs onto just a few physical platforms increased use efficiency and reduced space and management costs. Thus, the VMM's ability to multiplex hardware, this time for server consolidation and utility computing, again led it to prominence.

With VMs, researchers found they could make various architectures look sufficiently similar to existing platforms to leverage the current operation systems. From this project came the people and ideas that underpinned *VMware*[11], the original supplier of VMMs for commodity computing hardware. Today, in research labs and universities researchers are developing approaches based on VMs to solve mobility, security and manageability problems.

As shown in Figure 2.2 a VMM decouples software from the host by forming a layer of indirection between applications running in the VM, and host hardware. This layer of indirection can be seen as a virtualization layer and lets the VMM control how guest operation systems inside a VM access host hardware. The VMM can be a normal application or an integrated functionality of a *Hypervisor*. Either a Hypervisor or a normal operation system executes on the host. A normal operation system enables to execute native applications beside the VMM, whereas a true Hypervisor environment dedicates the virtualized host. On the VMM level a unitary virtual host is emulated. The VM encapsulates a guest operation system which accesses hardware of that virtual

[11] `http://www.vmware.com`

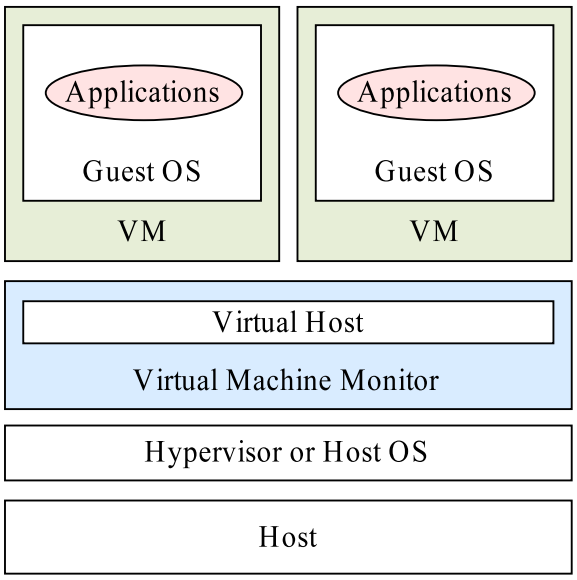

Figure 2.2: The basic principle of a Virtual Machine Monitor.

host. Inside the VM arbitrary operation systems are possible. For all applications, there is no difference in running inside a VM or a native operation system.

Disadvantages of virtualization are the additional overhead and considerable complexity. But advantages of virtualization are isolation, encapsulation, more flexibility, higher availability and good scalability:

- Isolation inhibits mutual annoyance of processes. Operations of one server hosted within an VM, can not affect the execution of another server running in another VM on the same computer. E.g. the Mail and Web servers run on the same machine, but a malfunction of one server can not affect other servers.

- Encapsulation means that the software within a VM is self-contained. A self-contained environment does not require external functionality like software libraries or application level drivers for execution. This is a strong base for whole operation system and application configurations without need for manual configuration.

- Flexibility comes from the ease of creation, migration, resumption and duplication of VMs. Basically a VM is a normal file. In case the underlying Hybervisor or VMM is capable of executing manifold VMs, there is no limitation on type and number of VMs.

- Availability is reached through replication of VMs. A crashed VM can be resumed on another machine or a passive replicate, running alongside as an on-the-fly copy of the active master VM, takes over immediately. This can be done by monitoring and then delegating the master role to a certain VM within a pool of slaves. In the best case the user does not notice the crash of a virtual service.

- Scalability is provided by live migration. Fully loaded computers can be discharged by migrating several VMs to another machine.
- Overhead and complexity are caused by the additional virtualization layer. A crash of a physical machine can cause the breakdown of many VMs, thus many service breakdowns. A system administrator must take this into account.

2.2.2 Virtualization Technologies

Resources can be virtualized on different layers implementing different forms of virtualization [WCC+08].

Full virtualization is also sometimes called hardware emulation. An unmodified operation system uses a VMM or Hypervisor as layer between guest and host operation system to trap and safely execute [WCC+08] privileged instructions at runtime. Trapping privileged instructions can lead to less performance. To overcome this, one strategy is to aggregate multiple instructions and translate them together, or another strategy is binary translation [AA06].

The basic binary translation technique is to run privileged mode code (kernel code) under control of the binary translator. The translator converts the privileged code into a similar block by replacing the problematic instructions, which lets the translated block run directly on the CPU. The binary translation system caches the translated block in a trace cache so that translation does not occur on subsequent executions. When full virtualization is supported, the virtualization software simulates full featured hardware and runs on top of the local operation system. An example of full virtualization is the VMware workstation.

Para-virtualization uses a Hypervisor and VMs to refer to virtualized operation systems. The work [WSG02] revitalizes para-virtualization by selectively modifying existing virtual architecture principles to enhance scalability, performance and simplicity for non legacy operation systems. Purely virtual instructions that have no counterpart in the physical architecture were introduced. These instructions are conceptually similar to operation system calls, except that they are non-blocking and operate at the architectural level instead of at the level of operation system abstractions. Existing instruction semantics are modified and certain rarely used instructions are classified as deprecated. Virtual registers as a lightweight mechanism for passing data between the VMM and VMs were added to the virtualization architecture. These registers are mapped to a well-known region of a VM's address space. Thus, bidirectional communication between VMM and VM is possible; e.g. the VMM can be noticed about changed resource requests or releases of the VM. Virtual I/O devices export a simplified architectural interface, designed to minimize VM/VMM boundary crossings. Unlike full virtualization, para-virtualization requires changes to the guest operation system. Guest operation systems are modified in order to perform so called hyper calls instead of system calls, which leads to higher performance as e.g. used by Xen[12] [BDF+03]. In Xen 3.0 [FHL+01] guests can be virtualized without modifying them,

[12] `http://www.xen.org`

using the virtualization support of X86 CPUs. This avoids some performance issues as experienced in full virtualization and the use of privileged instructions is reduced through VM-to-Hypervisor coordination. The advantage of para-virtualization is the relative better performance compared to full virtualization, whereas the disadvantage is the requirement to modify the para-virtualized operation system to be ready for the Hypervisor. Mostly this is only possible for open source operation systems.

Operation system virtualization has been proposed by the Linux-VServer[13] [Lig05], a kernel based virtualization. All guest operation systems are sharing the same kernel, while isolated from each other. Operation system virtualization does not rely on a Hypervisor, because the operation system itself is modified to run multiple instances of a guest operation system isolated. Guest operation systems are referred for example as *Virtual Private Servers* (VPSs). Because of no instruction trapping, the advantage of operation system virtualization lies mainly in the near-native performance. A disadvantage is that all VPSs share one kernel which could compromise all VPSs if the kernel crashes.

Native virtualization is virtualization support within a processor itself which allows to run unmodified operation systems concurrently and directly on the processor. Native virtualization does not emulate a processor like full virtualization. Both, Intel and AMD support virtualization for their x86 64 processor architectures through Intel-VT[14] or AMD-V[15] virtualization extensions respectively.

Table 2.2 summarizes typical differences between the four discussed virtualization concepts. For my work currently most feasible is full virtualization, since private

	Full	**Para**	**OS**	**Native**
Performance (lower is faster)	4	3	2	1
Application Modification	no	no	no	no
OS Modification	no	yes	yes	no
Sandbox	yes	yes	no	yes

Table 2.2: Typical differences between virtualization technologies.

computers are involved. In a second step native virtualization would be beneficial because of the performance gain. I do not investigate virtualization at all, but my architecture assumes a virtualization technique used by the P2P client for exchanging VMs between homes.

Especially *VMware*[16] is a virtualization software [WCC+08, QNC06] for machines based on x86 architecture. E.g. VMware Workstation and VMware Server require a host operation system, but are highly portable and do not require any modification of the host operation system. WMware ESX is itself an operation system. It provides

[13] `http://linux-vserver.org`
[14] `http://www.intel.com/technology/virtualization`
[15] `http://www.amd.com/virtualization`
[16] `http://www.vmware.com`

better performance at the cost of reduced portability. In VMware virtualization works at the processor level. VM privileged instructions are trapped and virtualized by the VMware process. Other instructions are directly executed by the host processor. VMware can be seen as the market leader in virtualization technology which includes support for both, full and native virtualization. Unlike VMware ESX, VMware Server runs on Linux or Windows, which allows freedom in hardware use, but causes additional overhead and less performance. VMware Server supports bridged, NAT, and host-only networking. Bridged networking allows VMs to act as distinct hosts with own IP addresses, where NAT networking allows VMs to use the same IP address, and host-only networking allows the VM to directly communicate with the host without true network interface.

Parallels[17] is a virtualization software like VMware. With the Parallels Desktop 4.0 application it is possible to run Windows, Linux or any other 32 or 64 bit operation system on a computer with the Macintosh operation system natively (OS X 10.4.11 or later). VMware also entered the market of virtualization on Macintosh with VMWare Fusion 2.0. Parallels introduced the support for multi core systems. Also seamless drag and drop of files between host and guest operation system is now possible.

VirtualBox[18] is similar to VMware or Parallels a virtualization software for x86 computer architectures. Formerly an open source project, Sun adopted the project still offering VirtualBox for free. VirtualBox runs on Windows, Linux, Macintosh and OpenSolaris and support guest operation systems of such types.

OpenVZ[19] is the open source base of Parallels Virtuozzo Containers[20]. It uses operation system virtualization to achieve near native performance for guest operation systems under Linux. The main advantage of OpenVZ lies in the resource control which is mainly missing in full virtualization or para-virtualization. Especially guest communication buffer, kernel memory, memory pages and disk space accessible by guest operation systems can be limited. For networking a virtual network device or a virtual Ethernet device can be chosen. While the virtual network device can not be modified by the guest operation system, the virtual Ethernet device is configurable as a standard Ethernet device.

KVM (Kernel-based Virtual Machine) is open source and included in Linux since version 2.6.20 and consists of a kernel module (kvm.ko) and a specific processor module. The processor virtualization instruction sets *Intel VT*[21] and *AMD-V*[22] are supported by using the corresponding module (kvm-intel.ko or kvm-amd.ko). KVM requires *QEMU*[23], an open source processor emulator or virtualizer. QEMU can run in emu-

[17] http://www.parallels.com
[18] http://www.virtualbox.org
[19] http://wiki.openvz.org
[20] http://www.parallels.com/virtuozzo
[21] http://www.intel.com/technology/virtualization
[22] http://www.amd.com/virtualization
[23] http://www.qemu.org

lator or virtualizer mode. In emulator mode QEMU is a machine emulator emulating different computer architectures. In virtualizer mode for x86 architectures QEMU executes guest operation systems with near native performance by directly using the host CPU. In that case a host driver called *KQEMU*[24] is needed.

Xen[25] is the most popular para-virtualization implementation. Guest operation systems exhibit, thanks to a small performance overhead, near-native performance. Xen manages mainly memory and CPU allocation, whereas storage is organized as either a single file on the host file system, or as partitions or logical volumes. Networking is realized as a series of virtual Ethernet devices created on the host system. These virtual Ethernet devices, each with own MAC address, function as endpoints of network interfaces in guest operation systems. In Xen, one primary domain has direct access to the host's hardware. Through this primary domain the Hypervisor monitors normal domains of guest operation systems. Hardware access of guests is redirected and granted only by the primary domain. The primary domain allows the Hypervisor also launching and shutting down normal domains.

A full virtualization solution like VMWare or the free VirtualBox is suitable to provide the functionality for my system. All necessary functions must be seamlessly included in a P2P client executed within homes. Thus, the P2P client must create, start, stop, pause, resume, send and receive VMs concurrently. This of course requires computer resources and could be improved by a native virtualization solution included and standardized in future operation systems.

2.2.3 PlanetLab

A special platform related to my work, is *PlanetLab*[26][PR06, AR06, PBFM06] (used e.g. by *Emulab*[27]). PlanetLab envisions an open distributed platform for deploying, executing and evaluating planetary-scale network services. PlanetLab is shared, built and maintained by a community of researchers at about 500 sites with more than 1000 nodes (checked in Jan. 2010). In exchange for hosting one or a small number of nodes, participants obtain access to resources across the entire platform.

PlanetLab faces a complex, distributed scenario of virtualization. Hardware resources are spread all over the planet, interconnected via the Internet without use of special high-performance links. Within PlanetLab every single machine is split into *Virtual Machines* (VMs) similar to server virtualization. These VMs are organized in slices, which are network-wide containers that isolate services from each other. Services belong to slices running concurrently and sharing the same global resources. Slices enforce two kinds of isolation: resource and security isolation. The former minimizes performance interference and the latter eliminates namespace interference.

[24] `http://kqemu.sourceforge.net`
[25] `http://www.xen.org`
[26] `https://www.planet-lab.org`
[27] `http://www.emulab.net`

More precisely, a slice is represented by one single VM per available PlanetLab machine. Thus, a user who has booked a slice can create one Linux-shell per PlanetLab machine. This scenario is illustrated in fig. 2.3. It is important to see that VMs in a slice are not aggregated like resources in a Grid. No further abstraction than one shell per machine is provided, leaving users with the problem of dealing with dozens or even hundreds of shells simultaneously.

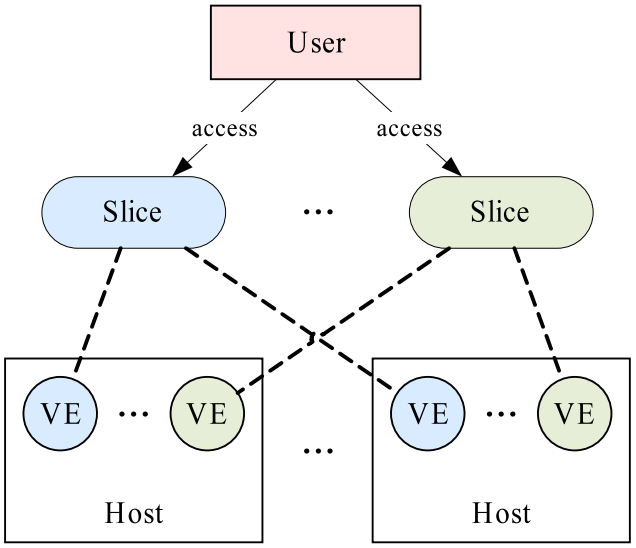

Figure 2.3: Virtualization in PlanetLab.

Services and applications run in a slice that can also be seen as a set of nodes on which the service receives a fraction of each node's resources in form of a VM. What is done in PlanetLab is distributed virtualization: the acquisition of a distributed set of VMs treated as a single, compound entity by the system. PlanetLab aims to isolate services and applications from each other for maintaining the illusion that each service runs on a distributed set of private machines.

The management system in PlanetLab is quite simplistic, since it allocates resources to users without caring of their load or bandwidth requirements. A central management authority provides slices to users; each slice is a collection of VMs (one on each PlanetLab node). The user must centrally upload code (via SSH for instance) by addressing each node individually. This is different to the controlled approach in Grid computing where a central management allocates exact resources to certain users and guarantees a given service level. In contrast, distributed virtualization such as that in PlanetLab connects virtual resources over the Internet while relying on less strict availability and reliability of the end systems.

With the *PlanetLab Central* (PLC) nodes can be remotely managed. Each node itself runs a *Node Manager* (NM) for controlling own VMs on top of a *Virtual Machine Monitor* (VMM). Slices are created by users through operations available on the PLC. The PLC uses the NM at nodes for creating VMs. A set of such VMs, possibly distributed on different nodes, defines a slice; they are a global abstraction.

PlanetLab uses simple central management, based on reservation of this slices on machines for running Linux shells. Access to global resources is given by this slices. PlanetLab users who wish to deploy applications acquire a slice. The slice abstraction itself, as a distributed collection of VMs, is implemented by slice creation services.

In some sense, PlanetLab is a distributed operation system. While PlanetLab provides weak isolation between slices at the node level, it provides no isolation at the level of aggregated resources across the entire system. Similarly, PlanetLab has only a single type of user; all experiments are equally powerful, even those written by novices.

PlanetLab applies unbundled management where services, used to manage PlanetLab itself, should be deployed as normal user services. This allows the system to more easily evolve, to permit third-party developers to build own services, and to permit decentralized control over PlanetLab resources.

To take advantage of virtualization, VM-management has to be done. In Grids available resources have to be adequately allocated. In data centers VMs have to be moved, copied, created and deleted e.g. for load balancing or consolidation. Similar to the resources of Grids, server hardware is usually located close to each other as in racks or data centers, and interconnected with high-bandwidth links. Therefore, the management of virtualization in Grids and data centers is mainly implemented in a centralized way, where a central management element allocates resources. VMware for instance, provides such a centralized management element to manage VMs in data centers. Although virtualization itself is highly distributed in PlanetLab, the management of hardware and slices is rather centralized. Slices are created, allocated and managed via a central server. Also the user of a slice is a central point of management, having to cope with hundreds of VMs.

A distributed management and dynamic behavior, as proposed in my work, is essentially missing in a platform like PlanetLab. There, virtual environments are created centrally, one virtual environment on each participating machine. Moreover, in PlanetLab shifting of load is not trivial and consolidation of machines to run at a higher load is not yet possible. Also, there is no automation in allocating virtual resources to a given user or to a special application. In an architecture for homes in the residential domain, the automatic allocation of resources and consolidation of resources must be possible.

My work adopts the concept of slices and VMs from PlanetLab. VMs containing tasks are send to other homes for remote execution. The single home knows locations of own tasks; this is the slice. But slice management should be considered decentralized simply in terms of scalability due to the vast number of possible participators in form of homes. Since in P2P systems it may be negligible to have knowledge about the entire system, a fixed management infrastructure as used in PlanetLab could be avoided and this assumption underpins my work.

2.2.4 Live Migration

Another important topic related to my work is *Live Migration* [HH09]. When resources are virtualized within VMs, an additional management of VMs is required to create, terminate, clone or move VMs from host to host. Migration of VMs can be done offline (the VM is suspended) or online (while running the VM). The management solution of VMware is supporting live migration.

Migrating VMs across distinct physical hosts is a useful tool for administrators of data centers and clusters. It allows a clean separation between hardware and software, and facilitates failure management, load balancing and system maintenance.

Migrating an entire VM allows to avoid many difficulties faced by process-level migration. In particular the narrow interface between a VM and the VMM avoids the problem of *residual dependencies* in which the original host must remain available on behalf of certain system calls or memory accesses on behalf of migrated processes. With VM migration the original host may be released once migration has completed.

Migrating at the level of an entire VM means, that the working memory state can be transferred in a consistent fashion. Theoretically, a game server or streaming media server can be migrated at runtime without requiring clients to reconnect. Live migration of VMs allows a separation of concerns between users and operator. Users need not provide any remote access to the administrator (e.g. a root login to suspend client-side processes prior to migration). Similarly, the operator does not need to be concerned with details of what is going on within VMs. Instead they can simply migrate and then resume an entire running system as a single unit. If a physical machine needs to be removed from service an administrator may migrate all VMs to alternative machines.

As stated by [CFH+05] there are three strategies to copy a VM from one host to another host:

1. *Pure stop-and-copy* involves halting the original VM, copying all pages to the destination, and then starting the new VM. This has advantages in terms of simplicity, but means that both downtime and total migration time are proportional to the amount of physical memory allocated to the VM. This can lead to an unacceptable outage if the VM is running a live service.

2. *Pure demand-migration* involves a short stop-and-copy phase that transfers essential kernel data structures to the destination. The destination VM is then started, and other pages are transferred across the network on first use. This results in a much shorter downtime, but produces a much longer total migration time; and in practice, performance after migration is likely to be unacceptably degraded until a considerable set of pages have been faulted across. Until this time the VM will fault on a high proportion of its memory accesses, each of which initiates a synchronous transfer across the network.

3. *Pre-copy* combines a bounded iterative push phase with a typically very short stop-and-copy phase. Iterative means that pre-copying occurs in rounds, in which

the pages to be transferred during round t are those that are modified during round $t-1$ (all pages are transferred in the first round). This strategy is based on the assumption that every VM will have some set of pages that it updates very frequently and which are therefore poor candidates for pre-copy migration which implicates that the majority of pages rarely changes.

The migration procedure itself can be divided in two types:

- *Managed migration* is performed by migration daemons running on source and destination hosts. Those are responsible for creating a new VM on the destination machine, and for coordinating the transfer over the network. When transferring the memory image of a running VM, daemons perform rounds of copying in conjunction with complete scans of the VM's memory pages. Although in the first round all pages are transferred to the destination machine, in subsequent rounds this copying is restricted to pages that were updated during the previous round, as indicated by e.g. a dirty bitmap that is copied at the start of each round.

- *Self migration* places the majority of migration functionality within the system being migrated. In this design a migration stub must run on the destination machine to listen for incoming migration requests, to create an initial VM, and to receive memory images by the source machine. This time the operation system maintains a dirty bitmap itself with the difficulty to transfer a consistent operation system checkpoint. Further, the operation system must continue to run in order to transfer the final memory state. This results in additional rounds of copying but avoids the full pause during managed migration.

Yet live migration was only considered in local area networks and for relative small-sized VMs[HH09]. Live migration of large enterprise applications is a major problem, because significant parts of memory pages are dirtied at least as fast as the transfer over network is possible. To overcome this, *Dynamic Rate-Limiting* is applied by increasing the bandwidth for migration at cost of network congestion, or *Rouge Process Stunning* by freezing processes with to much activity. The drawbacks are possible bottlenecks in terms of network bandwidth or response time, which is dangerous for highly available services hosted within VMs.

In my work, migration of VMs must not be necessarily *live*. Since each task is an atomic operation, migration takes place only if a new task is created or finished (*stop-and-copy*). I investigate a system under the worst case; interrupted tasks must be restarted and there is only one place of execution for each task. If the system performs under this constraints, it will also do so with live migration which in turn would improve the outcome by allowing more exact determination of migration time points.

Virtualization nowadays is a well established technology for breaking up the dependence between host, operation system, and applications. Virtualization is assumed as basic part of this work and enables to abstract load through tasks that can be exchanged between homes to fulfill power saving goals. If VMs are mentioned later, then

it is meant that VMs only contain minimal operation system and application bundles. These (tiny) VMs contain only the necessary parts of e.g. Linux to boot and start the application executing a task. For this purpose a solution like a VMware or VirtualBox is considered. Deamons as part of the P2P client of each home could handle the migration of VMs and beyond that could assure reliability. This work does not focus on virtualization itself, but is based upon it.

2.3 Virtual Home Environment

This section gives an introduction about connectivity inside and between homes. Many technologies came up in recent years for interconnecting devices of a home network. Also concepts for inter-home cooperation were suggested.

According to [BH06] a home network interconnects home appliances, media systems, PCs, and various kinds of sensors via bridges. A residential gateway runs remotely managed applications. The gateway operator maintains the gateway itself without paying attention to applications. The service aggregator is responsible for deployment and configuration of applications. Ideally, the customer may not be aware of updates for applications running on the gateway. The network operator provides access to the Internet and monitors a variety of performance parameters.

A definition by the 3rd Generation Partnership Project[28] (3GPP)[PM02] describes a *Virtual Home Environment* (VHE) as a concept for personal service environment portability across network boundaries and between terminals. 3GPP is a collaboration agreement since December 1998 that brings together a number of telecommunications standards bodies which are known as *Organizational Partners*. Users are consistently presented with the same personalized features, interface customizations and services in whatever network, wherever the user may be located. Further the *Open Service Access* (OSA) framework for separating network and service layers was proposed by 3GPP.

This OSA framework in conjunction with the *Parlay Application Programming Interfaces*, specified under the so called *Parlay/OSA Specifications*, was the base technology applied in the project *VESPER* [RSO01, RJX+02] (Virtual Home Environment for Service Personalization and Roaming Users). The project aimed to define, demonstrate and promote a service architecture for provision of VHE across a multi-provider, heterogeneous network and system infrastructure. The key objective of VESPER was to define a VHE architecture validated by some sample implementations of services. The *Parlay Group*[29] is a consortium formed to develop open, technology-independent APIs that enable the development of applications operating across converged networks.

Also the *European Institute for Research and Strategic Studies in Telecommunications*[30] (Eurescom) described a VHE [Geu01] as an environment enabling users to

[28] `http://www.3gpp.org`
[29] `http://www.parlay.org`
[30] `http://www.eurescom.de`

receive customized and personalized services, regardless of location, access network or terminal type in a way that users will not see a difference in using services at home or while roaming in other networks. A VHE promises to provide users with a common look and feel of services. Eurescom is a private organization for collaborative research and development in European telecommunications and performs multinational research projects on networks, services, applications and further aspects of telecommunications.

Similarly in [LYBP02] the VHE concept pursues the idea of service universality, which allows users to transparently access services anytime, anywhere with any type of terminal. This concept allows users to be consistently presented with the same personalized features and preferences, regardless of the context. In practice this implies, that users can access VHE services in the home or anywhere. Hence their *Home Environment* becomes a *Virtual Home Environment*. Figure 2.4 shows the generic VHE roaming model deduced from [RCXA03] and [DM02]. Internet access, home control and service management are distributed among three providers. The user is connected to the Internet via the network provider at his current location. A user at home is connected to his *Local Network Provider* and outside to a currently available *Remote Network Provider*. The *VHE Provider* is responsible to manage the home's equipment through a residential application layer gateway [DM02]. Such a gateway may be equipped with *OSGi*[31] to manage services that control home devices. OSGi is a dynamic module system for *Java*[32]. The *OSGi Alliance* is a worldwide consortium of technology innovators that advances a proven and mature process to assure interoperability of applications and services based on its component integration platform. This alliance provides specifications, reference implementations, test suites and certification. Java is an object-oriented programming language developed by *Sun*[33]. Locally, the user establishes a connection to his home along the edges denoted with L and remotely along the edges denoted with R. The remote connection can be seen as roaming via *Remote Network Provider* and *Remote VHE Provider*. The model also considers a *3rd Party Service Provider* that offers services to the home, accessible over the corresponding VHE Provider (edges denoted with E).

Here is the own home virtually extended to another home from where services of the own home can be remotely accessed. The relevance for my own work is the fact, that results of tasks, sent out previously at home, could be received at another home location for e.g. home management applications. Thus, one can create, send and receive tasks at different locations, but the system gives the feeling of being always in the own home network.

The work [RCXA03] proposed a *Roamer Agent*, already defined in the VHE architecture of the project VESPER and responsible for dynamic selection of an appropriate Remote VHE Provider within the range of a single roaming user. The Roamer Agent negotiates user preferences and ensures Quality of Service aspects during the roaming

[31] `http://www.osgi.org`
[32] `http://java.sun.com`
[33] `http://www.sun.com`

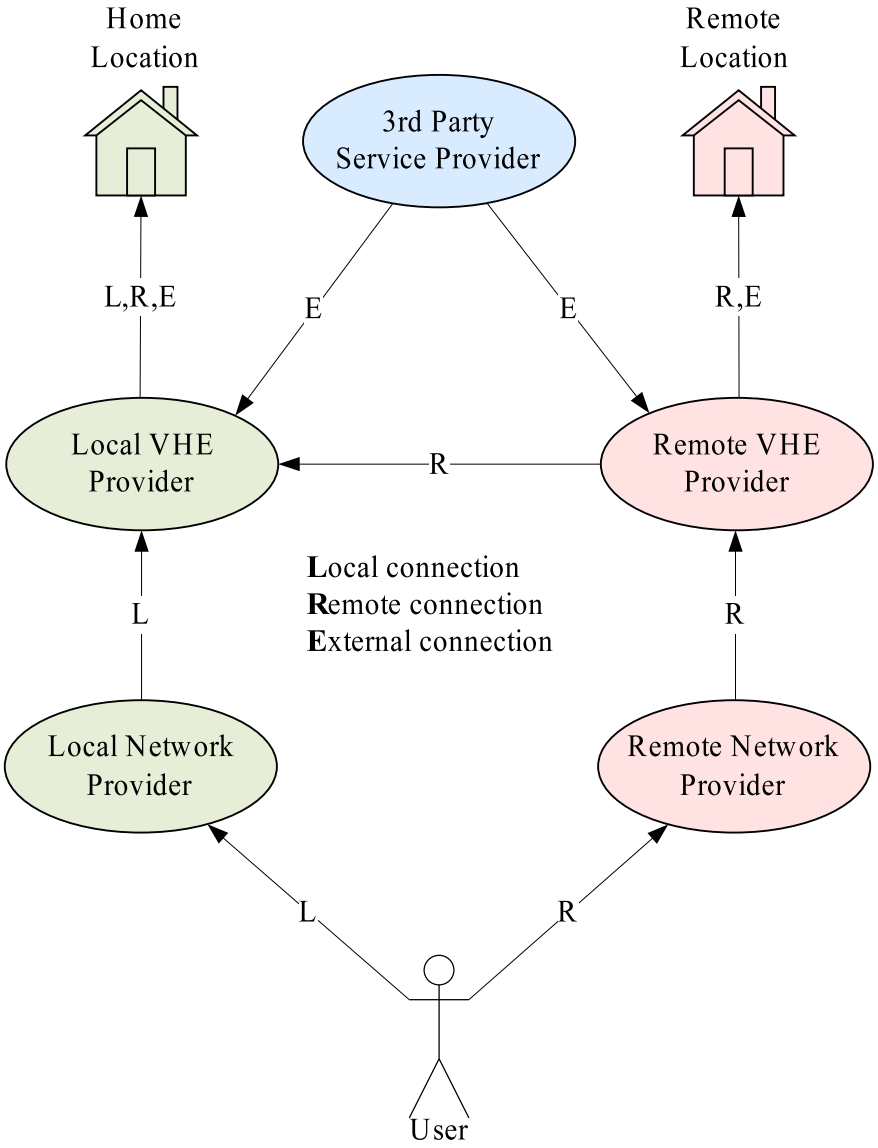

Figure 2.4: The generic VHE roaming model.

session of the user.

A major problem of the VHE model is the requirement to dynamically adapt network capabilities to enable optimal service provision [DM02]. As in [LYBP02] outlined, the key problem of managing services provided by a VHE, are effective adaption mechanisms. The service-context leads to a multi-dimensional adaption problem. In addition to terminal, network and user preferences, also location and time adaption must be considered. The work [LYBP02] focuses on the service management aspects of a VHE-middleware for context-aware, adaptable services.

In [TCdM02, TdMV+02] a multimedia delivery service in the context of the VESPER project is described to demonstrate the features of the VESPER prototype. This

service provides a mechanism to distribute and adapt multimedia streams according do the user's terminal, connectivity and preferences. Adaption is done in terms of user interaction (terminal capabilities, user preferences, networks) and in terms of the media content. Code mobility in form of mobile agents are conceived to partition the adaption between server (encoding) and terminal (decoding).

In [NUT+02] a virtual overlay network for integrating networked home appliances is proposed while also considering media streaming and disk sharing. The aim is that virtual overlay networks should build an Internet-scale ubiquitous computing environment. Application layer gateways are connected through the Internet and translate between different protocols like *Jini*[34], *UPnP*[35], *HAVi*[36], *SOAP*[37] or *Bluetooth*[38], supported by home appliances. Jini is a service oriented architecture based on Java and defines a programming model for the construction of secure, distributed systems. The Universal Plug and Play (UPnP) architecture offers pervasive P2P network connectivity for computers, intelligent appliances, and wireless devices. Home Audio Video interoperability (HAVi) is a specification for interconnecting home entertainment products. SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. Bluetooth is a protocol for wireless short-range communication.

The work [FXY+04] analyzed in details the requirements of service content adaption in a VHE system and created a policy based multi-dimension framework to implement adaption due to user profiles, terminal capabilities and network resources. The framework consists of a service layer, transmit layer and terminal layer. In the service layer occurs the context-based adaption. The transmit layer assures Quality of Service whereas the terminal layer adjusts the content to user requirements.

The work [LMBE03] designed an architecture for integrating homes connected to the Internet with OSGi-capable gateways. *JXTA*[39] is combined with OSGi to build a P2P-overlay without central server for communication, media sharing and distributed device control. The JXTA technology is a set of open protocols that enable any networked device to communicate and collaborate in a P2P manner.

Another work [YHJ+08] deals with the interoperability of various home network middlewares like UPnP, HAVi and Jini. A so called *Multi Middleware Bridge* (MMB) facilitates interoperability of different bridges for this middlewares. Normally the bridges of two different middlewares communicate in one-to-one manner, but a bridge repository is used for coordinating the bridge systems and to provide load balancing and dynamically linking of additional home networks.

[34] http://www.jini.org
[35] http://www.upnp.org
[36] http://www.havi.org
[37] http://www.w3.org/TR/soap
[38] http://www.bluetooth.com
[39] http://www.jxta.org

The VHE-concept explained above is more restricted to a home centered scenario where the user leaves the home network and come into range of another network (e.g. in the car or at work) whereas data and services of the home network remain accessible. Then the user remains *virtually* always within the own network.

The extended case would be that the system tracks user position and delivers information and content without user intervention to the right place. A user starts and gives over some tasks to the system at home. The user changes his position, moves into another network and will be notified on task completion at his current place. So, tasks sent out at one place, were executed within the system, and sent back to another place. E.g. one wants to encode a video, sends the video source to the system at home. Then goes to work and some time after the system sends the outcome of the task to the current position of the user, the computer at the workplace. This implies that all computers (home, work) must be part of the system and logically networked to an VHE.

2.4 Own works

This work precedes some already published works I was involved in. In the next few paragraphs I introduce works already published since 2007.

The baseline paper [HHW+07] outlines the energy efficiency optimization goal and the importance of decentralization and virtualization. Then an architecture for task sharing is introduced. The analytical model for the application *Download Sharing* shows the possible benefit in terms of power saving:

In this paper, a new architecture for sharing resources amongst home environments is proposed. Our approach goes far beyond traditional systems for distributed virtualization like PlanetLab or Grid computing, since it relies on complete decentralization in a peer-to-peer like manner, and above all, aims at energy efficiency. Energy metrics are defined, which have to be optimized by the system. The system itself uses virtualization to transparently move tasks from one home to another in order to optimally utilize the existing computing power. An overview of our proposed architecture is presented as well as an analytical evaluation of the possible energy savings in a distributed example scenario where computers share downloads.

My thesis is mainly related to this paper for the idea of energy efficiency and the architecture.

In [GBH+08] an economic model for fair resource sharing in the context of the architecture already explained in [HHW+07] is developed:

Home networks recently gained importance due to their development from pure internal networks in form of an Ethernet LAN to converged networks integrating home, Internet, and access provider infrastructure. In emerging future home networking scenarios, service provisioning and network management is proposed by distributed architectures forming Virtual Home Environments (VHEs). This paper provides a service description and corresponding traffic and cost model for fair resource sharing in VHEs.

The objective of the proposed cost model is to allow an evaluation of the contribution and consumption for each user participating in the VHE to find an economic balance in the distributed behavior. Hereby, the contribution counts positively and the consumption negatively. The economic balance controls the load in the VHE and further limits the consumption of resources by users which over-pass a corresponding threshold. A negative balance leads to an exclusion from the VHE, if the negative balance is not equilibrated over a mean- or long-time horizon.

My thesis is mainly related to this paper for the economic model.

In [HWH+08] results of the analytical model of [HHW+07] were extended and first simulation output added:

We present an overview of our proposed architecture, consisting of a middleware interconnecting computers and routers in possibly millions of homes using P2P technologies. For demonstrating the potential energy saving of distributed applications, we present an analytical model for sharing downloads, which is verified by discrete event simulation. The model represents an optimistic case without P2P overhead and fairness. The model allows to assess the upper limit of the saving potential. An enhanced version of the simulation model also shows the effect of fairness. The fairer the system gets, the less efficient it is.

My thesis is mainly related to this paper for the validation of the analytical model by simulation.

In [HWT08, HWT10] the concept of task virtualization is pursued and a prototypical implementation presented and verified by simulation and then extended in terms of power saving and verified with simulation:

Already, hundreds of millions of PCs are found in homes, offering high computing capacity without being adequately utilized. This paper reveals the potential for energy saving in future home environments, which can be achieved by sharing resources, and concentrating 24/7 computation on a small number of PCs. We present three evaluation methods for assessing the expected performance. A newly created prototype is able to interconnect an arbitrary number of homes by using the free P2P library FreePastry. The prototype is able to carry out task virtualization by sending virtual machines (VMs) from one home to another, most VMs being of size around 4 MB. We present measurement results from the prototype. We then describe a general model for download sharing, and compare performance results from an analytical model to results obtained from a discrete event simulator. The simulation results demonstrate that it is possible to reach almost optimal energy efficiency for this scenario.

My thesis is mainly related to this paper for the idea of task virtualization as basic assumption for the architecture.

In [HHWdM10] besides the application *Download Sharing*, two new applications are analytically investigated. The application *Video Encoding* models more extensive resource requirements compared to *Download Sharing*, whereas the application *Home Management* represents a highly available service with moderate resource requirements:

In this paper, we present a distributed approach for saving energy by sharing computing load in home networks. In our approach, possibly thousands of home computers may cooperate and send each other tasks which include services and applications running on a 24/7 basis. By concentrating as many tasks as possible on a small number of computers, idle computers may go asleep and thus consume almost no energy. We present the general architecture of our approach and analyze several 24/7 applications by modeling the potential energy consumption with and without application sharing, as well as the aspect of availability.

My thesis is mainly related to this paper for the applications.

In [BWS+09] a traffic study about the architecture is given:

Home environments have a great potential of resource sharing and energy saving. More and more home computers are running on an always-on basis (e.g. media-centers or file-sharing clients). Such home environments have not been sufficiently analyzed regarding their energy-efficient operation, yet. This paper discusses network virtualization methods that are required in future home environments to enable the energy-efficient cooperation of home networks. End-users share their available hardware resources (e.g. CPU, disk, or network resources) with other users in an energy-efficient and balanced way. To achieve such an envisioned future home environment, an architecture is suggested that combines different virtualization methods. In this paper, virtualization related requirements of the suggested architecture are discussed in detail. Network virtualization methods and concepts are compared to each other with respect to their usability in the architecture. In addition, initial virtualization approaches are simulated and evaluated with regard to benefits and complexity in the suggested architecture.

My thesis is mainly related to this paper for the traffic study of the P2P network of homes.

Works listed above cover these topics:

- Power wasting in networked home environments.
- Architecture for resource and task sharing for power saving.
- Energy efficiency enabling characteristics of applications.
- Fairness for resource sharing.
- Feasibility of task virtualization and corresponding performance.
- Traffic overhead caused by P2P signaling.

My work is a roundup and comprehensive extension in terms of energy efficiency problem, related work, architecture, applications, analytical model and evaluation through simulation.

3 Architecture

This chapter introduces an architecture as basis for a network of homes exchanging load in form of tasks with the aim of power saving. Tasks are aggregated on a subset of participating homes, whereas the majority of homes eventually switch into a low power mode. The envisioned architecture applies virtualization for computer resources. Resource usage must be distributed among homes with respect to energy efficiency. The aim is to save power in each home and also to reduce the wattage of the whole network of homes. Since the system works without central management, various functionality must be performed by all homes like in P2P networks.

Requirements of an architecture enabling energy efficient networking highly depend on the application domain (closed or open, centralized or decentralized). Hence, I put my attention to the demanding application domain of home networks and huge numbers of interconnected homes worldwide. Because of bad scalability, it is not feasible to assume a central managing component. An open and decentralized system of homes as non-dedicated heterogeneous resource providers is envisioned, without given structure in terms of hardware and system software.

It is not clear to which extent energy efficiency can be achieved by means of resource sharing under given system properties and constraints. The general objective of the proposed architecture can be stated how to provide a distributed energy efficient system which fulfills stated requirements. The idea is to optimize this system for energy efficiency, thus saving power while providing at least similar degrees of availability, *Quality of Service* (QoS), security and privacy as with a local solution where homes do not cooperate.

3.1 Overlay

Figure 3.1 summarizes common elements that must be addressed by an architecture for energy efficient resource sharing in terms of context (home networking domain), outcome (energy efficient resource sharing), key technologies (virtualization, resource management, decentralization) and constraints (availability, QoS, security, fairness, privacy).

From virtualization the concept of *Virtual Machines* (VMs) is borrowed as encapsulation of tasks. Resource management and decentralization is covered by *P2P* and *Virtual Home Environments*. Security and privacy rely on the chosen P2P technology, whereas fairness is based on an economical model and must be included into

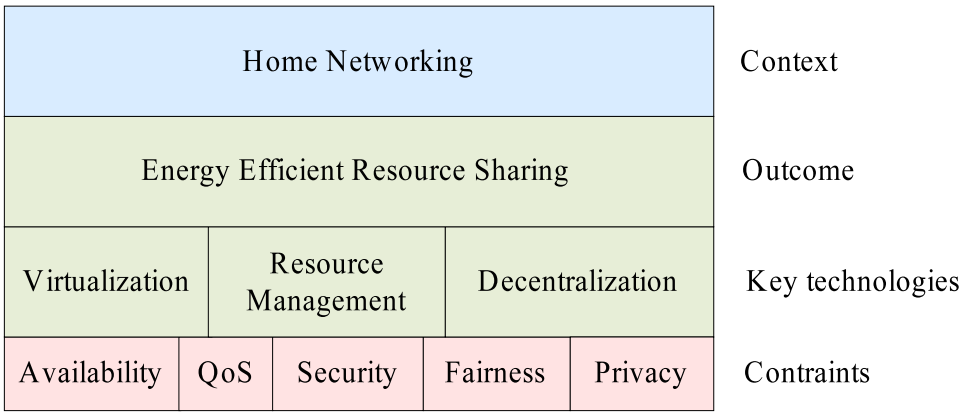

Figure 3.1: Elemens of an architecture for energy efficient resource sharing.

the communication protocol of the P2P network. Availability and QoS are strongly oppositional to the goal of power saving and must be seen as trade-off.

For the architecture proposed in this work, homes must be interconnected and addressable for other homes. The addressing of homes has to be solved in a distributed way since a server is neglected. Mediation of available hardware resources has to be established and idle resources have to be discovered and addressed (resource mediation). Another requirement of this architecture is the distributed management of resources (resource allocation). No central element is available that manages the balanced cooperation of homes and the access to available resources. Energy-efficient resource sharing has a number of constraints. Examples are fair distribution of power consumption or the provision of sufficient QoS to users. The distributed management has to be aware of the different home states and must also know resources available at a certain point of time.

Networking requirements of a network of homes can be partly met by network virtualization methods [CB08, FGR07]. Two kinds of virtualized networks are widely used today: *Virtual Local Area Networks* (VLAN) and *Virtual Private Networks* (VPN). VLANs like *IEEE 802.1Q* operate mainly on the link layer, subdividing a switched *Local Area Network* into several distinct groups either by assigning different ports of a switch to different VLANs or by tagging link layer frames with VLAN identifiers and then routing accordingly. VPNs like *IPSec* on the other hand, establish a network layer tunnel to either connect two networks (site-to-site), one network and a host (site-to-end) or two hosts (end-to-end) with an encrypted and/or authenticated channel over the Internet. However, these kinds of virtualization methods target mainly the sharing of links among users.

Besides the virtualization of links, also the virtualization of routers has been investigated in several approaches. In [BFdM09] system virtualization (e.g. based on *Xen* [BDF+03] or *VMWare ESX Server*) is applied to routers to create virtualized networks with special features. In [EGH+07, MCZ06] performance challenges are identified that have to be tackled when virtual routers are based on Xen. Other forms of router vir-

tualization are already available in commercial products. Such solutions mainly allow the concurrent usage of network infrastructure.

For this work the unstructured and hybrid *Overnet*[1] P2P overlay can be suggested as a solution for network virtualization. It has the capability to solve all of the mentioned requirements and is resistant to high churn rates. Overnet is a very popular file sharing P2P network with a high amount of users (and traffic) [Tut04] that has practically proven to be very scalable. Another reason for this choice is the similarity of the hybrid Overnet structure to the structure of the network of homes suggested in this work (computers and gateways).

Two types of nodes are participating in the Overnet network: peers and super nodes (index servers). Peers are providing and consuming resources, similar to the computers in homes. Super nodes form a separate network to share information as gateways could do. A peer can report its available resources to the super node and request the location of hardware resources from the super node. The location of a gateway within this architecture is very similar to the location of a super node – the gateway is physically the first node of each home. This location makes the gateway the natural place for gathering statistics about the home that are required to enable a fair and energy-efficient resource sharing among homes. In addition, the gateway is supposed to be always-on, which enables it to manage and distribute information among other gateways.

Sure, selection of super nodes among nodes and also the optimal number of super nodes is an own problem and not necessarily in focus of my work. Super node selection is rudimentarily addressed in Chapter 5.

Commonly, the selection of super nodes can be formulated as *P-Median* problem [FH09] among location allocation problems, i.e. which nodes have best positions to all other nodes to minimize distances. The best P nodes are selected, which should provide services required by all nodes. In a computer network, the selection of the P nodes may depend on available bandwidth and delays between nodes. The P-Median problem is not trivial as we see in equation

$$\binom{N}{P} = \frac{N!}{P!(N-P)!)}$$

where N is the number of nodes in the network and P the given number of super nodes to be selected among them.

Another point of view is super node selection versus demand and fixed cost. The *Warehouse Location Problem* (WLP) [SS07] describes the problem of finding one or more locations for a warehouse to supply the demand of costumers. The sum of transport and fixed costs

$$\sum_{n=1}^{N}\sum_{m=1}^{M} c_{nm}x_{nm} + \sum_{m=1}^{M} f_m y_m$$

[1] `http://www.overnet.org`

in the sense of a computer network, must be minimized where f is the fixed cost of one node acting additionally as super node, c the transport costs between N nodes and M super nodes with binary variables x and y.

The architecture is depicted in Figure 3.2. The proposed architecture consists of a P2P-based overlay of interconnected homes. The overlay could also be a structured one, based on a *Distributed Hash Table* (DHT) [HC07]. Each home, represented by a cycle, features a home network (stylized as a bus system) consisting of an always-on gateway connecting the home to the Internet, one or several computers (as abstraction for any device that can share resources), and sensors and actuators as usually existent in smart homes [RMD+07, CEEC08]. The home network itself may include multi networks, for example wireless and wired networks for connecting computers, sensors, actuators, and broadband access to the Internet. The gateway is a lightweight device which does not share resources - only essential management functionality is embedded. Virtualization techniques are applied in two ways. First, the network of homes appears as an abstract *Virtual Organization* (in compliance with Grid computing) to the single home and is executed transparently on participating homes. Second, load distribution and shifting is implemented by a virtualization layer.

Figure 3.2 points out that the intelligence of the distributed management layer is situated in each contributing node. This may be both, a full-blown PC with large computational resources (but also large power consumption), or a gateway which is assumed to be a simple Linux-based diskless computer with small energy consumption. However, this gateway is not able to contribute its own resources to be used by other homes, but its computational power should be sufficient to maintain a permanent entry in the overlay for representing its particular home. Since gateways are assumed to run permanently, churn is thus almost zero.

Depicted modules for *Distributed Algorithms*, *P2P-Based Virtualization* and *Security* implement the management layer. The architecture inside computers is divided in a control and an execution part. The control part integrates three modules:

1. *Distributed Algorithms* are the behavior of homes and pose the business logic of the overlay. All application-directed communication is defined here. Since the system operates in true P2P-manner without servers, distributed algorithms are the result of home-interaction according to exact equal implemented algorithms in each home.

2. *P2P-based Virtualization* is the base functionality provided by the chosen P2P-technology for discovery, resource allocation and aggregation, persistence, search, and messaging – in short all signaling communication necessary to maintain the overlay.

3. *Security* is mandatory for authentication, secure channels between homes, encryption of data stored inside homes, and sandboxed task execution. Since the module for *Distributed Algorithms* uses functionality from the module for *P2P-based Vir-*

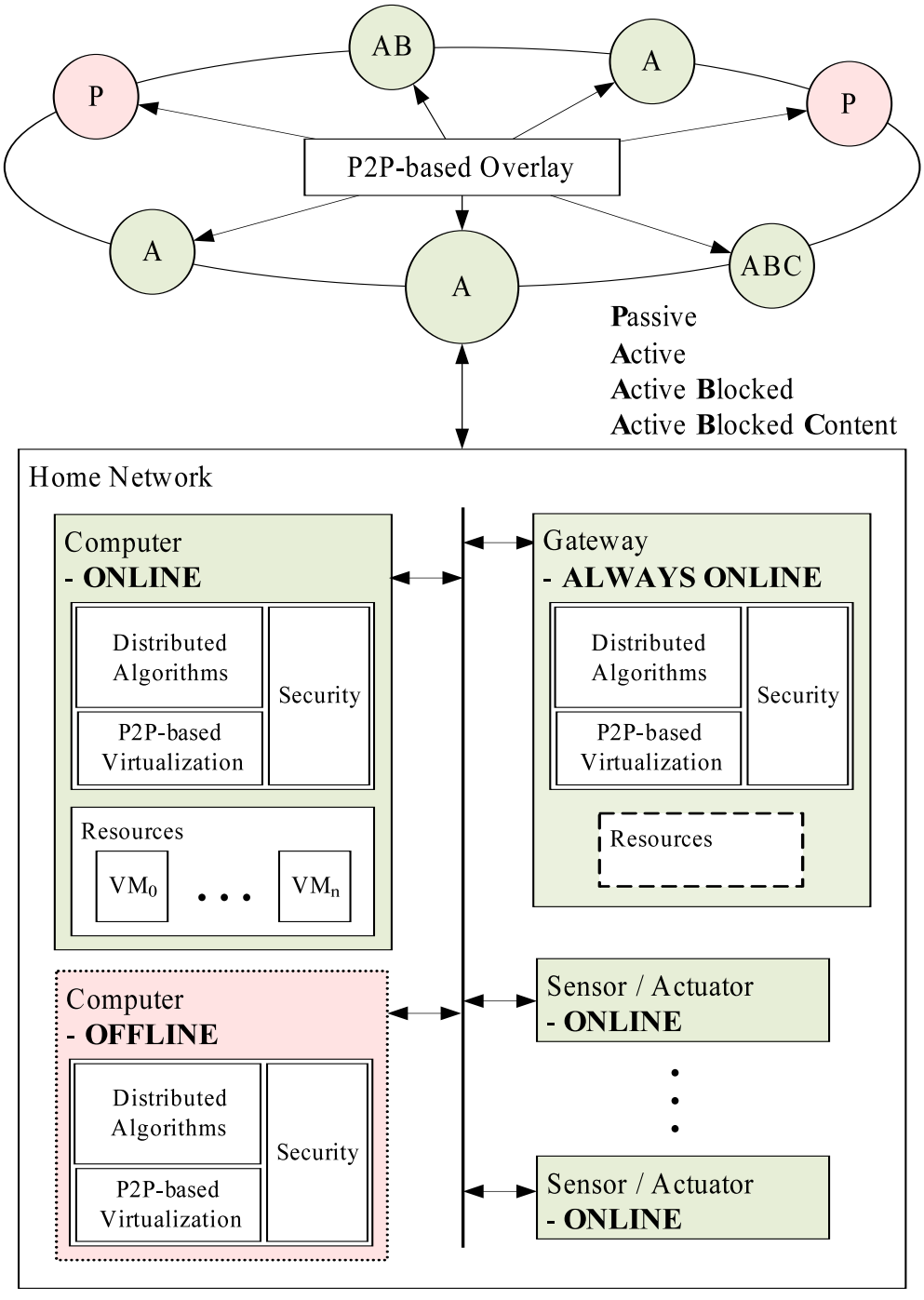

Figure 3.2: An architecture for distributed energy efficient resource sharing.

tualization, the security concept must span over both modules. Partly security will be already provided by the chosen P2P-technology, but the other part is up to the business logic (e.g. statistics, trustbase, voting). This module assures that attacks to the distributed system will be detected and consequences of intrusion and sabotage are avoided. At the P2P layer it provides services for encryption, authentication and secure key exchange. At the algorithm layer it mainly governs the distributed voting process for minimizing the malicious host problem. Voting is necessary because malicious homes may try to create damage to others. For *Home Management*, shutting down the heating is unpleasant in winter. Such applications might rely on majority voting with quorums [Hou08, OFGG06, GM05], where for instance gateways act as a policers, and only those commands may pass, which have been signed by several other gateways.

All three modules of the control part are integrated into a P2P-client which itself is bridged to a *Virtual Machine Monitor* (VMM) (or *Hypervisor*) which is the execution part. Inside this VMM, tasks can be executed to the extend as resources are shared on this computer. Tasks are, in conjunction with a minimal execution environment, encapsulated in a *Virtual Machine* (VM) compatible to the VMM. Up to the shared resources of the computer, a certain number of VMs can concurrently run. The gateway itself does not provide resources but must be capable of running the P2P-client whereas a minimal functionality of these modules run continuously on gateways to keep the P2P network alive and avoid high churn [GSS06, SR06, BQ06]. Sensors and actuators do not run any P2P-client, because they only send or receive values to the gateway.

The module for *Distributed Algorithms* is the true intelligence of the system. It can be divided into five submodules:

1. The *Energy Efficiency* submodule is responsible for finding a global optimum in terms of energy efficiency given by global and local constraints. The main constraint is energy saving for individual homes, but also for the whole system. Once a home requests remote resources, this submodule tries to identify a set of remote homes which should be selected, because selecting them would maximize global energy efficiency.

2. The *Fairness* submodule controls the contribution of local resources to the system in order to avoid free-riding [KKU08, KSTT04], e.g. as seen in P2P systems like *Gnutella* [HCW05]. For achieving a fairness goal, cheaters must be revealed based on monitoring by other participants. This submodule contains an economic model and uses statistics about recent and past sharing behaviors of homes. The system requires distributed storage for persisting and accessing statistics, which must be guarded from being manipulated by cheaters. Given a resource request and a set of homes selected by the the *Energy Efficiency* submodule, this submodule identifies homes that should improve their sharing balance towards more fairness.

3. The *Availability* submodule assures service availability based on redundancy, diversity and system statistics. For instance, storing data in a distributed manner or

doing remote *Home Management* forces to consider reliability by using replication and fault-hypothesis.

4. The *Privacy* submodule maximizes the degree of privacy a home is experiencing. Proxy chaining and onion routing [Owe07, FJS08, For06] cloaks the identity of a remotely managed home. It guarantees that users of currently *active* homes can not figure out the owners of their hosted remote VMs. Mechanisms, based on encryption and tracking of privacy threats for deriving a convenient notification about the privacy assurance level of the system, must be taken into account.

5. The *Quality of Service* (QoS) submodule decides whether a particular home is able to host the requested tasks dependent on a given resource request. It describes performance characteristics (e.g. response time, bandwidth, packet loss, jitter) and utilization. For instance, if the user wants to remotely encode video files, the remote home should exhibit sufficient down- and uplink access bandwidth and enough free CPU time. These resources would be used only once. A slower home on the other hand might be sufficient to handle messages from *Home Management* devices. This particular service then would run for a very long time, thus achieving fairness in contrast to the encoding home. A third example for QoS decisions is given by the tradeoff between QoS and privacy. Consider again remote *Home Management*. When using long proxy chains, the degree of privacy is high, whereas the important QoS parameters latency and bandwidth will be much worse. There is a tradeoff between QoS and privacy.

3.2 Home States

Basically a home is in a certain state. Depending on the state of computers inside, the gateway of the home reports a certain joint state to the overlay.

The home is for a certain time in one of these states:

- *Active* (A)
- *Active-Blocked* (AB)
- *Active-Blocked-Content* (ABC)
- *Passive* (P)

Figure 3.3 shows the homes' state cycle and introduces possibilities for transitions where P[event] denotes the possibility that a home transitions to another state due to an event.

A home is in state A when at least one computer contributes to the system. The state A is the only state where the home contributes to the system. In all other states the home does not contribute to the system. Only in this state the home can help

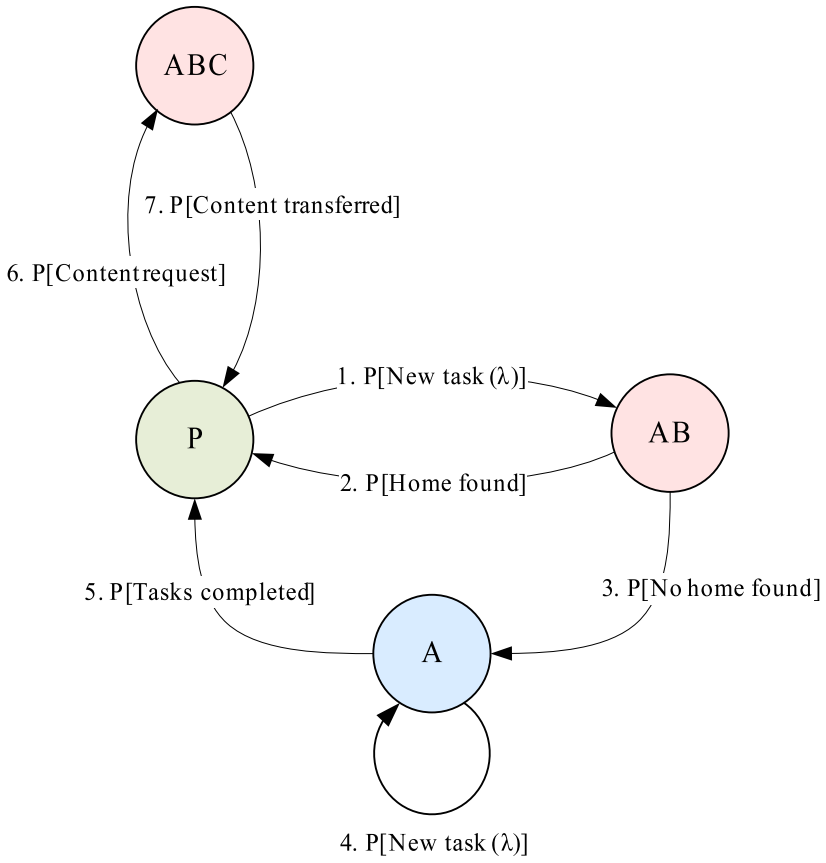

Figure 3.3: State cycle of homes.

other homes to save power, because in this state the home either executes only local or also remote tasks.

The state *AB* supports users who want to stay in control of own computers. For example, in the state *AB* the home is not ready for contributing, because if the user wants to join an online computer game, bandwidth and CPU time should not be contributed for energy efficient resource sharing, otherwise the gaming experience might be negatively influenced. In this state homes do not provide any services to the system.

The state *ABC* defines the phase during which a home receives results from the system, e.g. finished tasks. In this state the home does not contribute to the system, but consumes power as long the transfer of the results lasts. Further, the home currently being in state *ABC* was woken up by the system automatically and will be hibernated again by the system. In this state the home does not overtake remote tasks. The user must always declare the state *A* for being accessible for other homes.

A home is in state *P* if all computers of the home are in a low power mode (hibernating). This home does not contribute to the system but saves power. This is the desired state. The aim is to maximize the number of *passive* homes and minimize the number of *active* homes. The smaller part of *active* homes should serve the majority of *passive* homes.

Note, that also in case of a *passive* or *active-blocked* home, the gateway is still up

and the home might consume services from other homes. Thus, minimal power consumption by gateway, sensors and actuators are not avoidable. Optimization addresses the wattage of computers.

To better understand possible states of computers and homes Table 3.1 lists all possible state combinations for a home with two computers. A computer is *ONLINE* if it is ready to execute remote tasks, or only *online* if it is connected to the overlay but not ready to execute remote tasks, or even *offline* for saving power as defined in the architecture Figure 3.2. For example, a home with one *online* computer is reported as

Computer 1	**Computer 2**	**Home State**
ONLINE	ONLINE	A
ONLINE	online	A
online	ONLINE	A
ONLINE	offline	A
offline	ONLINE	A
online	offline	AB(C)
offline	online	AB(C)
offline	offline	P

Table 3.1: Possible home states with two computers inside.

AB by the gateway. A home with two computers, one *ONLINE* and the other *online*, is still *A*. If all computers are *offline*, the home is *P*. To sum up, if there is at least one computer *ONLINE*, then this home is stated as *A*. If all computers are *offline*, the home is reported as *P*.

With these states the system can distinguish contributing from other homes. A home can obtain a list of currently *active* homes to select one which is then contacted for executing a task. Naturally, the user should only decide if a computer is ready do participate and do work for others (*A*) or is exclusively used (*AB*), because the system will set any idle computer automatically to *P* (or temporarily to *ABC*). The state cycle of homes is explained in detail in Chapter 5.

3.3 Front- and Backends

Thanks to the expected increase of access bandwidth in the residential domain in future, it is imaginable that whole user desktops (Windows, Linux), encapsulated in VMs, could be persisted, accessed, and composed in a distributed manner at runtime. Due to virtualization, applications logically separated into *frontends* and *backends* could transparently allocate resources like CPU time, disk storage, disk or network bandwidth without knowledge about location or configuration of remote computers.

The frontend implements a user interface for unified access to backends, while backends implement the heavy-loaded business logic of applications. Many backends could be assigned to one frontend while the distributed execution is hidden. The user only starts the frontend instead of each application directly. The frontend is like a remote

desktop tool and knows current remote locations and states of own backends. Encapsulating frontends and backends in VMs enables dynamical migration of load between homes.

The idea is as follows: the user starts up the frontend which is a VM with any operation system. This frontend is the work environment or home desktop and only provides access to currently running backends in the network. Then the user starts some tasks where for each of them a VM containing minimal execution environment, task description, and source data is created; these are the backends. The backend is an atomic unit and is sent to the network by the frontend. The frontend memorizes the location of the backend. If the location of a backend changes during a frontend is offline, redirection is used. At this point the frontend may go offline while backends are distributed and running among other homes. Once tasks executed within backends are completed, results are transferred from backends to the frontend as soon as the frontend is online. The frontend stores results on any location. During the work phase of backends, the computer running the frontend can be powered off or even hibernated. Backends are aggregated on remote computers for consolidation.

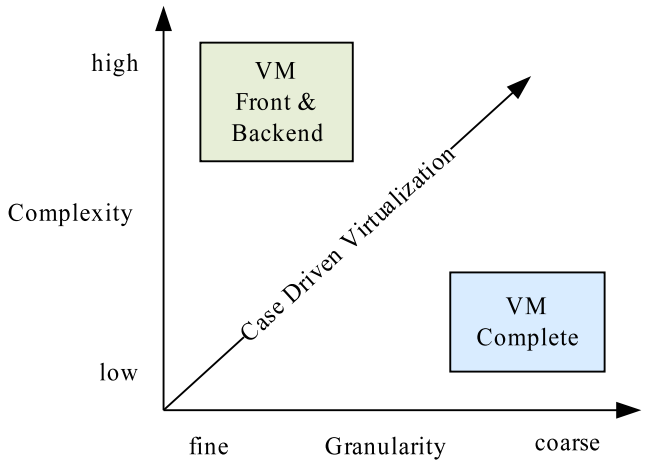

Figure 3.4: Case driven virtualization.

The goal is to go away from computers with fixed operation system and application installation. Everyone has a work environment (frontend), data and applications (backends) are stored distributed. One can request so much resources as required, as done in Cloud computing.

An interesting question is the path between granularity versus complexity of virtualization as depicted in Figure 3.4. Coarse granularity means that services and functionality hosted inside a VM can not be divided into smaller units (e.g. frontend). High complexity is featured by front- and backend combinations of VMs. The problem is to determine the best solution for a specific application.

Either whole VMs are migrated between homes or VMs are divided into heavy-

weighted backends and light-weighted frontends as mentioned earlier. Depending on the type of service, an indicated form of virtualization should be applied. Remote access to home's appliances can be provided by a frontend-backend combination, whereas the location change of a user leads to a full migration of the work environment (frontend). With the front- and backend concept it would not be necessary to always migrate whole VMs between homes. This decreases resource requirements and improves energy efficiency. In my work the process of backend migration is explored.

4 Applications

This chapter presents applications suitable to act as backend in context of the architecture introduced in Chapter 3. For energy efficient resource sharing it is necessary to find use cases for homes where the *active* time of a home is minimized. And this under the assumption that homes generate a certain level of load in form of tasks. The idea is to model load as application-specific tasks. These tasks are aggregated on a part of *active* homes. For these tasks latencies, caused by the transfer of source and result data, must be taken into account or be negligible compared with local execution.

These applications are introduced:

- *Download Sharing* (DS)
- *Video Encoding* (VE)
- *Home Management* (HM)

It is assumed that homes generate load over time based on these applications. It is further assumed, that this load is composed of many atomic tasks that can be executed locally or remotely (backends). A specific task, stemming from one of these applications, is referred as DS-task, VE-task or HM-task. A task of these applications is encapsulated in a *Virtual Machine* (VM) that also holds all necessary components for running autonomously on any computer. Task descriptions, executed at initialization time of the VM, define actions during startup, execution and shutdown of the VM. After task completion, depending on the task type, the modified usually bigger VM is scheduled to be sent back to the creator of the task.

A DS-task is the download of a specific file from a source outside the considered network of homes. A user in *Home A* (H_A) wants to download some content and instructs the P2P-client to search a remote place for task execution. The system tries to find another home (H_B), that is ready to take the task over. A prepared and secured VM with included task description is sent to H_B. H_B executes the DS-task and sends the modified VM, enriched with the download content, back to H_A.

A VE-task is the encoding of a source video to a compressed format, e.g. with a video codec such as *Xvid*[1]. The user in H_A creates a VE-task and commits it to the system. As for DS-tasks, a VM with included task description will be sent to another H_B. H_B downloads the source video from H_A and then encodes the video. After encoding, H_B sends the modified VM back to H_A.

[1] `http://www.xvid.org`

A HM-task is intended to continuously run for controlling all the technical equipment of a (future) smart home. Therefore, the HM-task is modeled as lightweight task in terms of resource usage, but with high availability requirement. Each home must run one such task, but may outsource the VM containing the HM-task to another home. The procedure is quite similar as with the first to tasks, except that the VM is sent back whenever the user of H_A wants to regain control of the own *Home Management*.

H_A creates its HM-task that controls all technical equipment inside. Then H_A uses the system to find another H_B that can take over this task. H_A transfers the HM-task to H_B and is then ready to go *passive*. Since the gateway of each home is assumed to be always-on, the control of H_A's technical equipment is done remotely by H_B which receives e.g. sensor values and sends back commands to actuators. Especially for reliability this type of application requires some extra precautions as analyzed later in Chapter 6.

Since tasks run on computers within homes, they have certain resource requirements as shown in Table 4.1. The intention of these tree task types is as follows: A DS-task

Task Type	**Frequency**	**Access Bandwidth**	**Disk Space**	**CPU Utilization**	**Availability**
DS-task	high	middle	middle	low	low
VE-task	middle	high	high	high	low
HM-task	low	low	low	low	high

Table 4.1: Characteristics of tasks.

is frequent, where resource usage is moderate. The VE-task is a power task engrossing many resources on typical computers. Both, DS-task and VE-task, feature low availability and no real time communication. On the contrary, a HM-task is infrequently, but must be very responsive to react accordingly to data recorded by e.g. sensors. With these three task types my work proofs how energy efficient resource sharing can be reached and what are the limitations.

Since *Download Sharing* and *Video Encoding* feature low availability, best effort is considered for avoiding overhead of repeated or resumed tasks. Failed downloads or encodings are lost but this assumption in uncritical in reality because task creation and execution not necessarily underlie time constraints. Also the length of the period from starting the task until the user receives the result is uncritical. The situation is completely different for *Home Management*. Only an uninterrupted controlling of home equipment makes sense. That is the reason why reliability is very important for *Home Management* but neglectable for *Download Sharing* and *Video Encoding*.

A *local case* is distinguished from a *distributed case*. In the local case each home executes all tasks itself, thus we say local execution. In the distributed case each home tries to send tasks away for remote execution in other homes. Remote execution must consume less resources as local execution to achieve a power saving.

Let W_L be the wattage caused by local execution of a task and W_D the wattage

caused by distributed execution of a task. More precisely, let W_A and W_B be the wattages caused by the execution of one task in H_A and H_B respectively. Further, let T_A be the time elapsed, if a task is locally executed in H_A, or let T_D be the time elapsed if the task migrates from H_A to H_B, is executed within H_B, and then is sent back from H_B to H_A. The overall wattage W_L for one task in the local case is

$$W_L(T_A, W_A) \tag{4.1}$$

which says that in the local case only the wattage and execution time of H_A is relevant, whereby the wattage W_D in the distributed case is

$$W_D(T_D, W_A, W_B) \tag{4.2}$$

which depends on the wattage of H_A as well as on the wattage of H_B, and on the overall execution time T_D including the task migration time from H_A to H_B, the execution time of the task in H_B and the the back migration of the result from H_B to H_A. The distributed wattage must be significantly below the local wattage

$$W_D < W_L \tag{4.3}$$

otherwise remote execution would not result in global power saving.

4.1 Download Sharing

In case of a DS-task the work time T_A consumed in H_A in the local case is now

$$T_A = \frac{F}{D_A} \tag{4.4}$$

and for H_B

$$T_B = \frac{F}{D_B} \tag{4.5}$$

whereas F is the size of the download and D_A or D_B is the downlink bandwidth of H_A or H_B. The time T_D is taken by remote execution of a DS-task in the distributed case

$$T_D = \frac{V_A}{\min\{U_A|D_B\}} + T_B + \frac{V_B}{\min\{U_B|D_A\}} \tag{4.6}$$

and depends on the filesizes of the original VM encapsulating the task V_A by H_A and the modified VM encapsulating the competed task V_B by H_B, and the execution time in H_B as denoted in (4.5). T_D depends also on the available uplink or downlink bandwidth of H_A (U_A or D_A) and H_B (U_B or D_B). Before transferring V_A from H_A to H_B, or after task execution transferring V_B from H_B to H_A, both homes must negotiate a minimum bandwidth.

Revisiting (4.1) and (4.2) for introducing wattage yields

$$W_L = T_A W_A = \frac{W_A F}{D_A} \tag{4.7}$$

and

$$W_D = \frac{(W_A + W_B)V_A}{\min\{U_A|D_B\}} + \frac{W_B F}{D_B} + \frac{(W_A + W_B)V_B}{\min\{U_B|D_A\}} \tag{4.8}$$

Notice, that W_L only depends on parameters of H_A, whereas W_D depends on wattages (W_A, W_B) of cooperating homes as well as their downlink and uplink bandwidths (U_A, D_A, U_B, D_B). Moreover, W_D depends on the filesizes (V_A, V_B) of the VMs and naturally on the size F of the download itself.

To save power, there has to be at least disparity in wattage or performance between H_A and H_B. Thus, if H_A wastes less power than H_B, but H_B has a faster access bandwidth, then it could be still cheaper to migrate the DS-task to H_B, to hibernate H_A, and then send the result back to H_A later, instead of doing the task within H_A for a quite longer time due to a slow connection. The lower wattage of H_A would be relativized by the much faster access bandwidth available in H_B. To express this case, α denotes the coherence between task work times (4.4) and (4.5) as follows

$$T_B = \alpha \times T_A \qquad 0 < \alpha \leq 1 \tag{4.9}$$

Equally, for the wattage the coherence is denoted by β

$$W_B = \beta \times W_A \qquad 0 < \beta \leq 1 \tag{4.10}$$

Theoretically it is possible that H_B with poor performance ($\alpha > 1$), but highly effective power supply (β near 0), outperforms H_A, but we neglect this extreme case by restricting to values less than 1. Values of 1 for α or β mean that wattage or performance of H_B is equal to that of H_A which does not bring any benefit if both becomes true.

Using (4.9) and (4.10) in (4.8) results in

$$W_D = \frac{(W_A + \beta W_A)V_A}{\min\{U_A|D_B\}} + \frac{\alpha\beta F W_A}{D_A} + \frac{(W_A + \beta W_A)V_B}{\min\{U_B|D_A\}} \tag{4.11}$$

which expresses the overall wattage in the distributed case with introduced ratios for wattage and performance between H_A and H_B. Again, taking the relation between distributed and local wattage (4.3) into account and substituting the negotiated bandwidths between H_A and H_B with B leads to

$$\frac{(W_A + \beta W_A)V_A}{B} + \frac{\alpha\beta F W_A}{D_A} + \frac{(W_A + \beta W_A)V_B}{B} < \frac{W_A F}{D_A} \tag{4.12}$$

and says that the wattage in the distributed case must be below the wattage in the local case.

Simplifying to B finally gives

$$\frac{(1+\beta)(V_A+V_B)D_A}{(1-\alpha\beta)F} < B \tag{4.13}$$

whereas the left side yields the bandwidth necessary for VM-migration in the distributed case. This minimum bandwidth is the negotiated bandwidth B between H_A and H_B based on the corresponding minimum of U_A and D_B for migration or U_B and D_A for back-migration. Also the left side reveals that α and β directly affect B.

Table 4.2 shows some results for meaningful ranges of α and β and fixed $V_A = 10$ MB, $V_B = 710$ MB, $D_A = 2500$ kbit/s, $F = 700$ MB.

α	β	**kbit/s**
1.0	0.8	23143
0.9	0.8	16531
0.8	0.8	12857
0.7	0.8	10519
0.6	0.8	8901
0.5	0.8	7714

Table 4.2: *Download Sharing*: Minimum transfer bandwidth required.

The scenario is as follows: a user wants to download a file with size F. The basic VM containing the DS-task has size V_A, because only a rudimentary small featured VM including a task description is sent to another home. After the task is completed, a VM with size V_B is sent back. The DS-task is carried out with an average speed of D_A, taking into account that many tasks run inside H_B. So, the factors α and β are important variables that specify the ratio between performance (here downlink bandwidth) and power efficiency (here wattage) of homes. We see, the smaller the performance factor a, thus the faster H_B is, the less bandwidth between H_A and H_B is necessary to gain advantage of the distributed case.

4.2 Video Encoding

For a VE-task T_A and T_B are encoding times of a video in H_A and H_B respectively. The focus lies now on the duration of encoding a source video (e.g. MPEG-2[2]) to a compressed format (e.g. DivX[3] or Xvid[4]). Nevertheless, also the migration of the original VM from H_A to H_B and the back-migration of the result VM from H_B to H_A are critical. We take wattages for the local (4.7) and distributed (4.8) case and simplify to

$$W_L = T_A W_A \tag{4.14}$$

[2] `http://www.mpeg.org`
[3] `http://www.divx.com`
[4] `http://www.xvid.org`

and

$$W_D = \frac{(W_A + W_B)V_A}{\min\{U_A|D_B\}} + T_B W_B + \frac{(W_A + W_B)V_B}{\min\{U_B|D_A\}} \tag{4.15}$$

with T_A and T_B as corresponding encoding times of H_A and H_B. Again using the relations between H_A and H_B of performance (4.9) and wattage (4.10) results in

$$W_D = \frac{(W_A + \beta W_A)V_A}{\min\{U_A|D_B\}} + \alpha\beta T_A W_A + \frac{(W_A + \beta W_A)V_B}{\min\{U_B|D_A\}} \tag{4.16}$$

Again taking into account that the distributed wattage must be less than the local wattage (4.3) and substituting up- and downlink bandwidths of both homes with the negotiated bandwidth B, leads similar to *Download Sharing* (4.12) to

$$\frac{(W_A + bW_A)V_A}{B} + \alpha\beta T_A W_A + \frac{(W_A + bW_A)V_B}{B} < W_A T_A \tag{4.17}$$

Simplifying to B finally gives

$$\frac{(1+\beta)(V_A + V_B)}{(1-\alpha\beta)T_A} < B \tag{4.18}$$

which is similar to the final expression for *Download Sharing* (4.13) and expresses the bandwidth necessary for gaining a benefit to send the source video out and get the encoded video back instead of local encoding.

Table 4.3 shows the minimum required bandwidth between two homes for a VE-task for encoding a 60 min MPEG2-DVD video with Xvid. The VM containing the source video has 2451,6 MB and the VM with the encoded video has 507,6 MB. For calculating T_A, real measurements of Xvid encoding times on up to date computers were taken from the well-kept benchmark desktop CPU chart on Tom's hardware[5]. CPU time available for encoding in H_B is set to a 2.4 Ghz (single core processor). We can see,

α	β	**kbit/s**
1.0	0.8	22763
0.9	0.8	16259
0.8	0.8	12646
0.7	0.8	10347
0.6	0.8	8755
0.5	0.8	7588

Table 4.3: *Video Encoding*: Minimum transfer bandwidth required.

for higher performance (shorter encoding time) of H_B, the necessary bandwidth B between H_A and H_B may be lower. If the difference of the encoding time $T_A - T_B$, depending on the performance factor α is small, then more bandwidth B is necessary to take advantage of the distributed case. It make only sense to transfer the video to another home that can quicker encode it.

[5] www.tomshardware.com

4.3 Home Management

For a HM-task the model must be rethinked. The intention of the HM-task is a software that controls a home's technical equipment like heating or home appliances. Such a HM-task could be set up with some parameters (e.g. the preferred room temperature), sent to another home, and remotely control the own home for allowing local resources to be released. Thus, H_A is controlled by its HM-task running in H_B until the user wants to regain control over its own HM-task for perhaps modifying the setup.

Since a HM-task is meant to be a lightweight process in terms of resource usage, aggregation of many HM-tasks on a single home should be possible. But the situation becomes more difficult if additionally reliability is considered. To prevent homes without control in case of a breakdown of HM-tasks, copies or replications of each task must be taken into account.

Concerning reliability, the model for DS- and VE-tasks must be extended by replication. A replication factor R is introduced which says that a HM-task will be sent to R different remote homes. If $R = 3$, then three other homes will receive a HM-task of H_A, where only one remote home acts as master controlling H_A and the other two homes act as slaves and help out if the master fails. Sure, there must be some logic to handle the temporary cluster of replicate-holders, emerged by cooperation between H_A and the three remote homes, but this is not focus of this section and now we will concentrate again on the wattage caused by HM-task replication.

Now T_A and T_B are merged to T and interpreted as the time the HM-task will run either in H_A in the local case or in remote homes, each denoted as H_B, in the distributed case. Modifying the wattages for local (4.14) and distributed (4.15) execution of a VE-task, and introducing the replication factor R leads to

$$W_L = TW_A \tag{4.19}$$

and

$$W_D = R\frac{(W_A + W_B)V_A}{\min\{U_A|D_B\}} + TW_B + R\frac{(W_A + W_B)V_B}{\min\{U_B|D_A\}} \tag{4.20}$$

with T as pure execution time of a HM-task similar to the encoding time of a video. Transfer costs are multiplied with R, because R HM-tasks must be sent. Note, since only one replicate of a HM-task is taken by a specific home over, R is omitted for the execution ($R = 1$).

The distributed wattage W_D again must be less than the local wattage W_L which leads to

$$R\frac{(W_A + W_B)V_A}{\min\{U_A|D_B\}} + TW_B + R\frac{(W_A + W_B)V_B}{\min\{U_B|D_A\}} < TW_A \tag{4.21}$$

Substituting W_B with the wattage relation for H_B (4.10) and the minimum bandwidths

with B gives

$$R\frac{(W_A + \beta W_A)V_A}{B} + \beta T W_A + R\frac{(W_A + \beta W_A)V_B}{B} < T W_A \tag{4.22}$$

Simplifying to B finally yields

$$\frac{(1+\beta)(V_A + V_B)R}{(1-\beta)T} < B \tag{4.23}$$

where the left side again is the bandwidth necessary to send the HM-task from H_A to R other H_B. Table 4.4 shows the necessary bandwidth for different wattage factors β under assumed VM file sizes for V_A and V_B of 100 MB, $T = 8$ hours and $R = 3$. Even the remote H_B is not much more economical (β up to 0.9) as H_A, only a relatively low bandwidth is necessary to transfer HM-tasks to several homes and back. These low

β	**kbit/s**
0.9	1425
0.8	675
0.7	425
0,6	300
0,5	225
0,4	175

Table 4.4: *Home Management*: Minimum transfer bandwidth required.

bandwidths are very compatible with nowadays residential access bandwidths and the main problem lies in the fact that replication can counteract the advantage of doing home management remotely. This issue is analyzed more in detail in Chapter 6 through simulation.

4.4 Replication

This section refers to [HHWdM10] and will focus on the sensing aspect[6] [OBC05] of *Home Management* which relies on sensors, actuators and services controlling future homes with utmost autonomy as suggested.

Sensing a variety of different home conditions is a major characteristic of smart homes. From a networking perspective, each sensor typically causes periodic traffic of constant and low intensity [CSP04]. The actual message payload of sensor information is only a few bytes[7]. The sampling and sending frequencies depend on the type of sensors and the services aggregating sensor values. Table 4.5 shows example sensors and the estimated load caused by these sensors when assuming a UDP message of size 1280 bytes for delivering the sensor readings to a computer running the HM-task. For indoor movement we assume a movement speed of 1 m/s and a position accuracy of

[6] As designed by my colleague Karin Hummel.
[7] `http://www.tinyos.net`

one meter. We further assume that a refrigerator sends once a day an update of e.g. expired food.

Sensor type	**Samples/min**	**Bytes/min**
Temperature, wind, humidity	4	5120
Indoor position sensing	60	76800
RFID-based refrigerator	1/1440	0.89
Overall home		81921

Table 4.5: Estimated example sensor sampling/sending rates for *Home Management* services.

Hence, the example home would approximately cause average sensor traffic of about 81921 bytes/min ($\approx$ 10922 bit/s). When assuming a realistic access bandwidth of 4 Mbit/s, each home can deal with sensor data sent by about $M = 366$ remote homes. Depending on the services' requirements and in terms of real-time responses and processing times, this number will have to be decreased because the shared CPU time is likely to be the bottleneck of homes. In this scenario it is possible to save energy for a number up to 365 homes by distributed *Home Management*.

For a general model we assume N homes and that each home is able to manage up to M different remote homes without considering failures or related redundancy at this point. Then the number of necessary *active* homes A_0 is calculated as

$$A_0 = \left\lceil \frac{N}{M} \right\rceil$$

Since *Home Management* targets for embedded computing resulting in physical effects for humans, availability is a major concern and will now be included into the model. In the home, sensors, actuators and computers (including their hardware and peripheral connected components) may fail and impair the availability of services. To investigate energy consumption for distributed *Home Management*, we concentrate on computer failures only and do not include failures of sensors.

In detail, the fault-hypothesis for *Home Management* includes only crashed homes, caused either by intentionally leaves or peer failures. These failures, usually referred to as peer churns, are modeled by defining two random variables X_{on} and X_{off} for a peer's online and offline time in accordance to [BL07] and their expectation values T_{on} and T_{off}. To assure trust in the system, the *Home Management* system is defined to be available if all homes can be served (all home services are available). The availability of a home is defined as

$$A_v = \frac{T_{on}}{T_{on} + T_{off}}$$

The corresponding error rate for a home is $\lambda = 1/T_{on}$ and the repair rate $\mu = 1/T_{off}$. Repairing means that a home joins the P2P system. The availability can be increased by redundant execution of services on k different homes to tolerate up to $k-1$ failures for each home, where each service receives the sensor values by means of push information

dissemination. To simplify the model, it is assumed that services supported by one home are replicated as a group on a disjunct set of other homes. Thus, these homes must be considered as being of the same type (types 1 to A_0 as shown in Figure 4.1). In the faultless state the system consists of A_0k homes. Figure 4.1 depicts the birth-death process of the fault-tolerant system, where the states correspond to the number of faults in the system. These faults are distributed among the home types and their replicas.

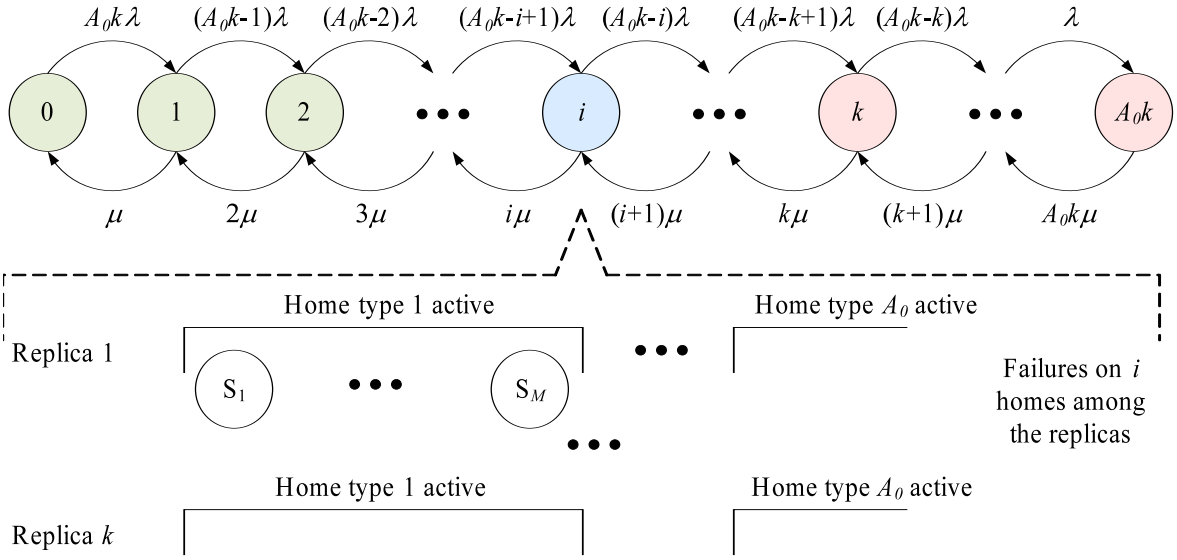

Figure 4.1: Failures modeled as birth-death process for *Home Management*.

- Each home failure leads to a system state transition between state 0 and up to A_0k failures.
- Until $k-1$ failures the system is in any case available for all homes.
- Within the range of k to $A_0(k-1)$ failures the system availability depends on which homes fail in parallel.
- In failure state $A_0(k-1)+1$ the system is no longer available (i.e. at least one home type or M services $S_1, \ldots, S_m$ are not available any more).

Figure 4.2 shows the state diagram for $A_0 = 5$ and $k = 3$:

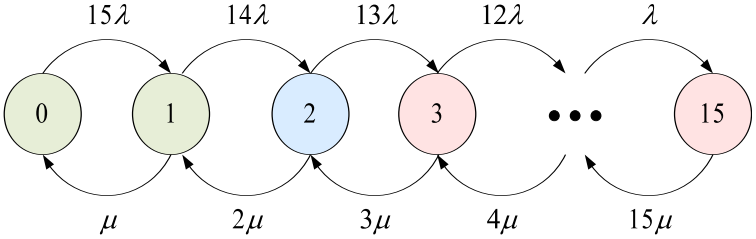

Figure 4.2: Failures modeled as birth-death process for *Home Management* with 15 homes.

- Each home failure leads to a system state transition between state 0 and up to 15 failures.
- Until 2 failures the system is in any case available for all homes.
- Within the range of 3 to 10 failures the system availability depends on which homes fail in parallel.
- In failure state 11 the system is no longer available (i.e. at least one home type or M services $S_1, \ldots, S_m$ are not available any more).

For the states i where $k \leq i \leq A_0(k-1)$, the probability that homes of similar type fail can be calculated using the hypergeometric distribution to find the number of occurrences H_i of a set of interesting jk failures among the i overall failures (i is here the number of the sample set and jk is both the number of occurrences in the basic set A_0k and the number of occurrences in the sample set). Semantically, the jk are the number of failures of homes with same type, where j is the number of different such k-sets which fail in parallel.

$$\begin{aligned} H_i(X = jk) &= \frac{\binom{jk}{jk}\binom{A_0k - jk}{i - jk}}{\binom{A_0k}{i}} \\ &= \frac{\binom{A_0k - jk}{i - jk}}{\binom{A_0k}{i}} \qquad 1 \leq j \leq \lfloor i/k \rfloor \end{aligned}$$

To cover all sets, this probability has further to be multiplied by the binomial coefficient of j and A_0 set, i.e. the number of possible j sets in the A_0 original set. Additionally, in case more than one home type fails, these cases are included multiple times, hence the inclusion-exclusion principle has to be applied. Considering these two issues, the availability in state i can be calculated by building the sum of the malicious cases. For each i the following calculations are used

$$Av_i = \begin{cases} 1 & 0 \leq i \leq k-1 \\ 1 - \sum_{j=1}^{j=\lfloor i/k \rfloor} (-1)^{j+1} \frac{\binom{A_0k - jk}{i - jk}}{\binom{A_0k}{i}} \binom{A_0}{j} & k \leq i \leq A_0(k-1) \\ 0 & i > A_0(k-1) \end{cases}$$

The probabilities for the system being in one of the states i is given by p_i. The

probabilities of the system's steady state are given as follows [BGdMT06], where

$$\sum_{i=0}^{A_0 k} p_i = 1$$

and

$$p_0 = \left(\sum_{i=0}^{A_0 k} \left(\begin{array}{c} A_0 k \\ i \end{array} \right) \left(\frac{\lambda}{\mu} \right)^i \right)^{-1}$$

and

$$p_i = \left(\begin{array}{c} A_0 k \\ i \end{array} \right) \left(\frac{\lambda}{\mu} \right)^i p_0 \qquad 0 < i \leq A_0 k$$

Thus, the availability of the system can be calculated as

$$Availability = \sum_{i=0}^{A_0 k} p_i A v_i$$

The overall number A of active homes necessary to maintain home services with a configurable redundancy value k is calculated by considering the probabilities of the system being in one of these faulty states, where i of the $A_0 k$ homes fail

$$A = \left\lceil \sum_{i=0}^{A_0 k} p_i \left(A_0 k - i \right) \right\rceil$$

We now investigate the number of *active* homes and the availability under varying conditions. We assume that *active* homes support up to 1000 sensor-enriched remote homes and provide triple redundant service execution ($k = 3$). To investigate the influence of the maximum load possible at homes, the load parameter ($M = 50, 100, 200$) is varied. Different failure- and repair rates are selected in accordance to the assumptions and findings in [BL07] ($\lambda = 1/15, 1/10$, $\mu = 1/10, 1/15$) to demonstrate the influence of these rates.

Figure 4.3 shows the *active* homes necessary to support up to 1000 HM-tasks. Without sharing each home would run the own HM-task which provides less availability ($k = 1$) by consuming more energy. The curve shows steps due to the load M per home. The trends show that the number of *active* homes is lower for homes capable of serving more load. The failure- and repair rates also influence the number of *active* homes. In case of higher failure rates and lower repair rates, less homes will be *active* and less energy will be consumed. Both energy saving and achievable availability have to be considered to derive a trade-off depending on the services' requirements.

By investigating availability, it is clear that higher failure rates and lower repair rates decrease availability, as shown in Figure 4.4 by the cases $M = 50$. Additionally, the

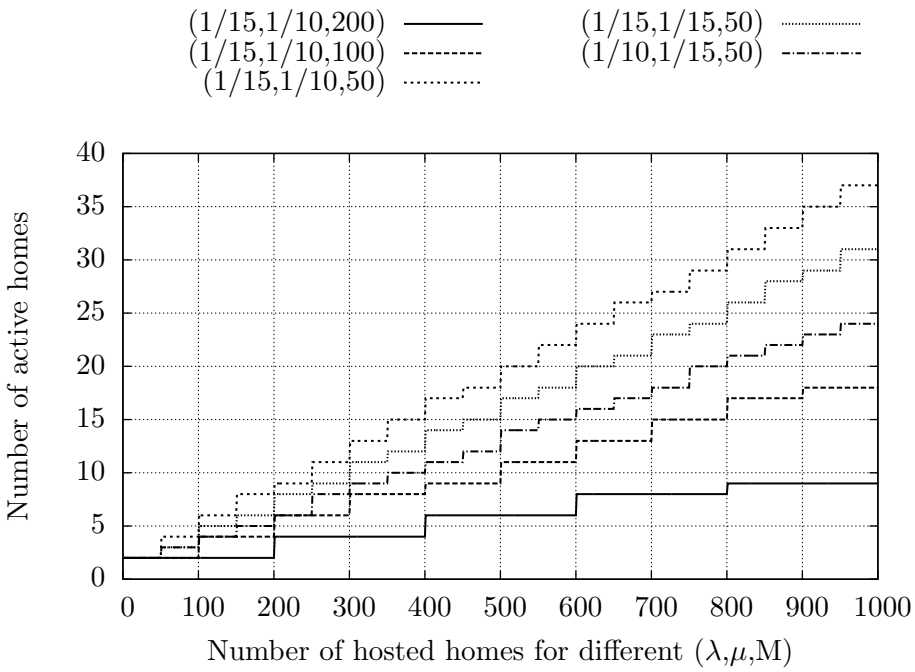

Figure 4.3: *Home Management*: Number of *active* homes.

availability also decreases with lower load capabilities of homes due to the definition that the HM-task is operable if at least one of the k replicas for each home type is running. In case of lower load capabilities, more homes are necessary to execute the tasks and more home types have to stay operational in parallel. Or in other words, it is harder to remain available (as expressed by lower calculated availability). This is of interest, because energy efficiency and availability are not contradictory to *Home Management*.

4.5 Analytical Evaluation

In order to show the potential of power saving through task sharing, as suggested in the architecture proposed in my work, this section outlines an analytical model for *Download Sharing*. Since we are only interested in power saving, security and privacy concerns are not included into this model.

Home A (H_A) sends a VM containing the DS-task to *Home B* (H_B). Once H_B finished the DS-task, H_B sends the result (the VM containing the downloaded file) back to H_A, here waking up H_A, which will then again consume energy as long as the transfer is going on. As a simplification we assume that homes being *active* for downloading for others, always download their own files. Furthermore, it is assumed that downloads do not use the whole downlink bandwidth B_d as given by the Internet connection. Instead, as it is experienced with real life file-sharing tools, the download bandwidth for one single file is limited to some upper limit, and on average uses B_l kbit/s with $B_l < B_d$. B_l usually depends on the number of seeders and on properties

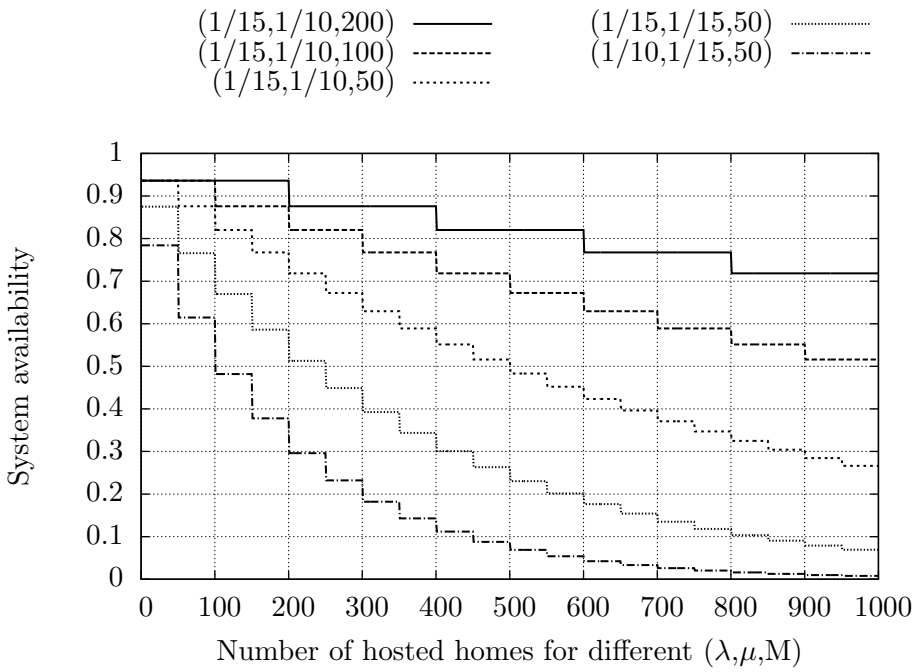

Figure 4.4: *Home Management*: Achieved availability.

of the used file-sharing tool.

The scenario is described by the following parameters. Parameter N denotes the number of homes in the scenario, while $M = \lfloor B_d/B_l \rfloor$ denotes the number of DS-tasks that may be carried out in parallel by each single home. For instance, if we assume that a home's raw downlink bandwidth is $B_d = 4$ Mbit/s, and each DS-task on average consumes $B_l = 200$ kbit/s, then $M = 20$ downloads can be carried out concurrently. Parameter λ denotes the arrival rate of DS-tasks at each single home, F denotes the average filesize, $t_l = F/B_l$ denotes the average time it takes for downloading a file, and thus $\mu = 1/t_l$ denotes the rate at which each DS-task is finished. For instance, if the size of a file on average is $F = 100$ MB, and $B_l = 200$ kbit/s, then $\mu = 1/4000$ DS-tasks are finished per second. In order to make the model analytically tractable, it is assumed that DS-tasks arrive accordingly to a Poisson process, and download times (and thus file sizes) are distributed exponentially.

The Poisson process is a memoryless (the future evolution of the process is statistically independent of its past) point process of an indexed collection of random variables with the probability that two or more arrivals happen at once is negligible. A point process is a sequence of events called arrivals occurring at random in points of time t_i with $i = 1, 2, ..., n$ and $t_i + 1 > t_i$. The index $_i$ is used to model time as point of arrivals, i.e. t_i is the time of the ith arrival that joins a queue. A point process can be defined by its counting process $\{N(t), t \geq 0\}$ where $N(t)$ is the number of arrivals occurred within $[0, t)$. Therefore the Poisson process is a continuous-time counting process with rate $\lambda > 0$ if it satisfies the following three conditions:

1. $N(0) = 0$
2. The number of arrivals in two non-overlapping intervals are independent.
3. The number of occurrences in an interval of length t has Poisson distribution with man λt.

This implies also time-homogeneity in the sense that occurrences are equally distributed over all time. The inter-arrival times of occurrences are exponentially distributed with parameter λ. The properties independence and time-homogeneity of the Poisson process provides a sample with observations that are not biased.

We investigate three cases:

1. The local case where no sharing occurs (*local*).
2. The ideal case with resource sharing but no back transfer of the result (*ideal*).
3. The corrected case with resource sharing and back transfer of the result (*corr*).

The two latter cases differ in the way they deal with the actual transfer to the requesting home: while in the ideal case this transfer is neglected, in the corrected case this transfer is included (resulting in additional wake-up time for the requesting home).

First, we assume that DS-tasks are carried out on the home that created the request while no sharing is going on. We start by modeling one single home. The number of DS-tasks carried out by this home can be modeled by a birth-death process. The process is in state k if the home is currently carrying out k DS-tasks. Since M is the upper bound of DS-tasks, the process has exactly $M + 1$ states. It is further assumed that if the process is in state M, newly generated DS-tasks are lost. Process states and transition rates are shown in Figure 4.5.

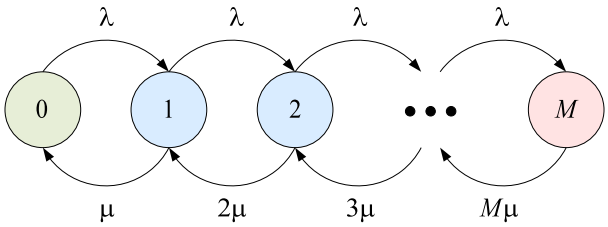

Figure 4.5: Birth-death process for local DS-tasks of one single home.

Simple analysis shows, that the probability π_k for being in state k is given by [BGdMT06, Zuk09]

$$\pi_k = \pi_0 \frac{1}{k!} \left(\frac{\lambda}{\mu}\right)^k \qquad 1 \leq k \leq M$$

with

$$\pi_0 = \left[1 + \sum_{k=1}^{M} \frac{1}{k!} \left(\frac{\lambda}{\mu}\right)^k\right]^{-1}$$

Since π_0 denotes the probability that no DS-task is going on, $1 - \pi_0$ denotes the probability that at least one DS-task is going on and the home is *active*. If there are N homes, then the mean number of *active* homes N_l for local DS-tasks is only given by

$$N_l = N\,(1 - \pi_0)$$

and hence by

$$N_l = N\left(1 - \left[1 + \sum_{k=1}^{M} \frac{1}{k!}\left(\frac{\lambda}{\mu}\right)^k\right]^{-1}\right) \tag{4.24}$$

In the next scenario we assume that homes share DS-tasks; if a home creates a DS-task with rate λ, it first searches for an *active* home to pass the DS-task to. If there is none, it will start the DS-task itself. Again the scenario is modeled by a birth-death process, this time by modeling the state of all homes. Since there are N homes, and each is able to carry out M DS-tasks in parallel, in total $M \times N$ DS-tasks can simultaneously be carried out. The process has $M \times N + 1$ states as shown in Figure 4.6.

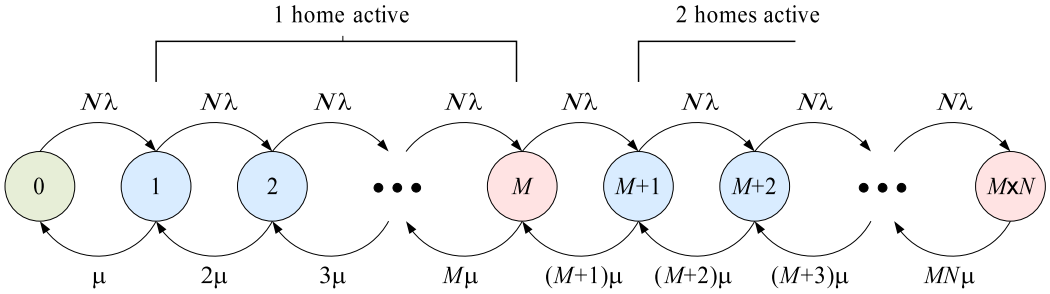

Figure 4.6: Birth-death process for simultaneous DS-tasks of N homes.

The solution of this process is similar to the one above, yielding

$$\hat{\pi}_k = \hat{\pi}_0 \frac{1}{k!}\left(\frac{N\lambda}{\mu}\right)^k \qquad 1 \le k \le MN$$

with

$$\hat{\pi}_0 = \left[1 + \sum_{k=1}^{MN} \frac{1}{k!}\left(\frac{N\lambda}{\mu}\right)^k\right]^{-1}$$

When assuming zero communication overhead, and not taking into account sending results back (*ideal*), then the number of *active* homes necessary to carry out k DS-tasks is $a = \lceil k/M \rceil$. In other words, no home must be *active* in state zero, $a = 1$ home must

be *active* in the states 1 to M, $a = 2$ for the states $M + 1$ to $2M$, and so on. The probability for needing exactly one *active* home is thus given by the sum of the π_k for $1 \leq k \leq M$, and in general the probability for needing exactly a (*active*) homes is therefore the sum of the π_k for $(a-1)M+1 \leq k \leq aM$. For computing the expectation N_i of a for the ideal case we derive

$$N_i = \sum_{a=1}^{N} a \ \mathrm{P}(a \text{ homes necessary}) = \sum_{a=1}^{N} a \sum_{k=(a-1)M+1}^{aM} \hat{\pi}_k \tag{4.25}$$

In order to catch the effect of the transfer of the result to H_A after the download has finished on H_B, the system is observed for a long time T. Then the total time that homes are *active* within T is given by $N_i\,T$, and the time that the system was in state k is given by $\hat{\pi}_k\,T$. It follows, that the number of finished DS-tasks while being in state k, is given by $\hat{\pi}_k\,T\,k\,\mu$. Since all N homes contribute equally to the system load and create DS-tasks with the same λ, the origins of DS-tasks are distributed evenly amongst all homes, but only $\lceil k/M \rceil$ of them are *active*. It follows that on average the number of DS-tasks finished in state k, carried out for a currently *passive* home, is given by

$$\hat{\pi}_k\,T\,k\,\mu\,\frac{N - \lceil k/M \rceil}{N}$$

The time for sending back the result to the initiating home is given by $t_u = F/B_u$, here taking the full raw uplink bandwidth B_u given by the Internet connection (e.g. $B_u = 1$ Mbit/s), which is considered to be much faster than the average download bandwidth B_l limited by the file-sharing tool. Thus, when sending back the result to a home that was *passive* previously, the *passive* home must wake up and must be *active* for at least t_u seconds. It follows, when observing the system for T seconds, the additional *active* time T_a for the back transfer to homes which have been *passive* previously is given by

$$T_a = t_u \sum_{k=1}^{MN} T\,\hat{\pi}_k\,k\,\mu\,\frac{N - \lceil k/M \rceil}{N}$$

When considering additionally that $t_u = F/B_u$ and $\mu = B_l/F$, the number of homes N_a additionally running for receiving transfers is then given by

$$N_a = \frac{T_a}{T}$$

and finally

$$N_a = \frac{B_l}{B_u} \sum_{k=1}^{MN} k\,\hat{\pi}_k\,\frac{N - \lceil k/M \rceil}{N} \tag{4.26}$$

Equation (4.26) is in accordance with the simple intuition that *active* time in the sharing scenario is determined by the relation between the download bandwidth B_l and the raw uplink bandwidth B_u.

The corrected average number N_c of *active* homes observed, including the time for downloading and for transferring back, is then defined as

$$N_c = N_i + N_a \tag{4.27}$$

and further

$$N_c = \sum_{a=1}^{N} a \sum_{k=(a-1)M+1}^{aM} \hat{\pi}_k + \frac{B_l}{B_u} \sum_{k=1}^{MN} k\,\hat{\pi}_k \frac{N - \lceil k/M \rceil}{N} \tag{4.28}$$

Equation (4.28) is in accordance with the simple intuition that *active* time is likely to be saved only if the download bandwidth B_l is smaller than the raw uplink bandwidth B_u.

Figure 4.7 shows results for $N = 1000$, $F = 100$ MB, $B_d = 4$ Mbit/s, $B_l = 200$ kbit/s, and $B_u = 1$ Mbit/s. Each single home generates a certain number of DS-tasks per week,

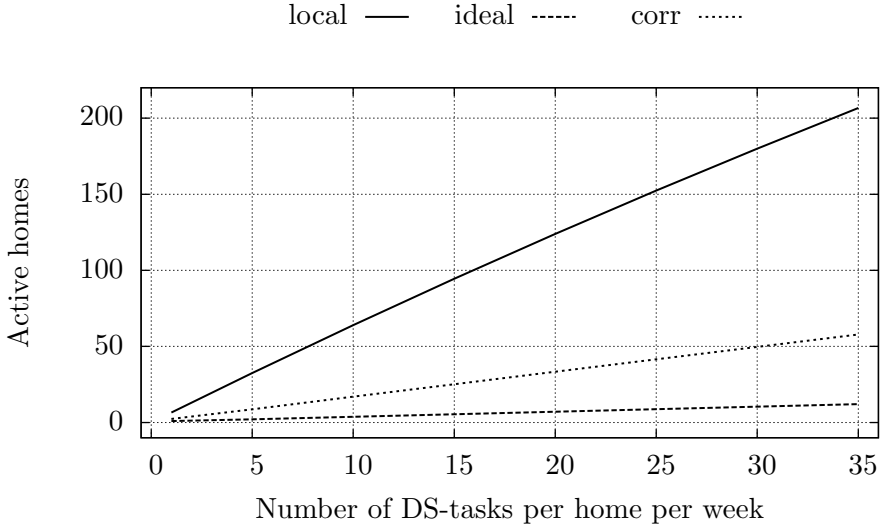

Figure 4.7: Number of *active* homes.

shown at the x-axis. The possible saving of home power is reflected by the difference between the number of *active* homes in the local case (4.24) and the corrected case (4.28). It can be seen that even when taking into account the distribution overhead, the corrected case (*corr*) can save a substantial amount of power. For instance, when assuming that each home consumes 100 W and creates 35 DS-tasks every week, without cooperation 1000 non-cooperative homes would constantly consume more than 20 kW on average just for downloading files, while cooperating homes would only consume about 5.7 kW for the same task.

However, the distribution overhead of sending files back to the requesting home clearly dominates the shared scenario, which can be seen by the difference between the ideal and the corrected case, and which is mainly determined by the relation between B_l and B_u. Note that changing B_l alone does not have a large effect in (4.28), since B_l also determines M, and a smaller B_l will result in a larger M, enabling a larger degree

of sharing. On the other hand, increasing B_u has a dramatic effect and yields to much better energy efficiency.

The energy efficiency η given by (1.1), here in DS-tasks per kWh, is shown in Figure 4.8. The energy efficiency of the sharing scenario (*corr*) is clearly much better than the one for the scenario without cooperation (*local*). It can be seen that if the load is too small, then downloads are usually carried out sequentially, and even the ideal case cannot save much energy by clustering DS-tasks. For increasing load the energy efficiency approaches a system-specific upper limit.

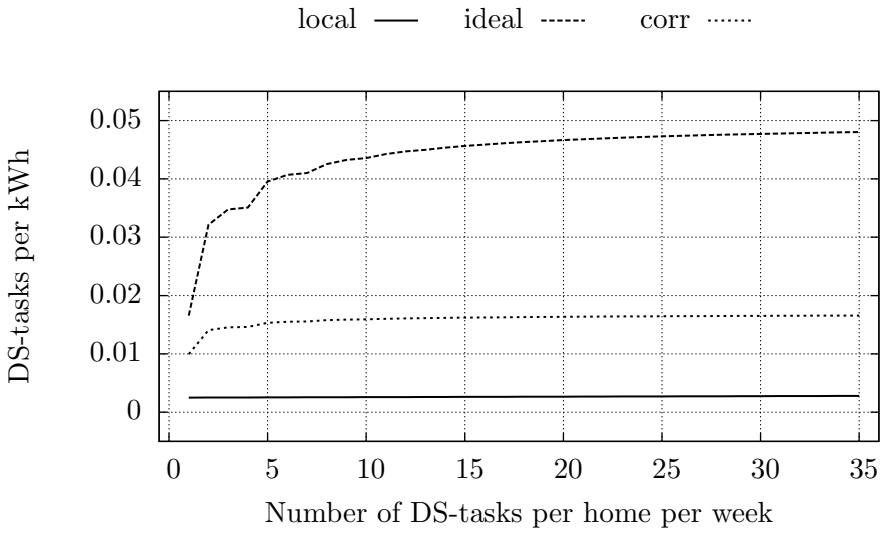

Figure 4.8: Energy efficiency η.

It must be noted that the corrected case does not take into account several details such as representative file size distribution and protocol overhead. Additionally, it assumes that after the last DS-task has finished, the downloading home immediately may go *passive*, while in reality it must still send the file to the requesting home if possible. On the other hand, a requesting home, which is just receiving a finished DS-task, is a good candidate for starting DS-tasks itself. The latter two issues may cancel each other out to some extend and thus have not been included into the model. In order to include all the above mentioned issues, a discrete event simulator is developed and introduced in Chapter 5 to evaluate the energy consumption for various applications and sharing patterns.

We can also consider more future homes with synchronous access bandwidth to the Internet. Figure 4.9 shows results for $N = 100$ homes, $F = 100$ MB, $B_d = B_u = 8$ Mbit/s, and $B_l = 1$ Mbit/s, while Figure 4.10 shows the same for $N = 1000$ homes.

With arrival rate λ, each single home generates a certain number of DS-tasks per week, shown at the x-axis. The possible power saving is reflected by the difference between the number of *active* homes in the local case (4.24) and the corrected case (4.28). It can be seen that even when taking into account the distribution overhead, i.e. sending back the files to the requesting homes, the shared scenario (corr) can

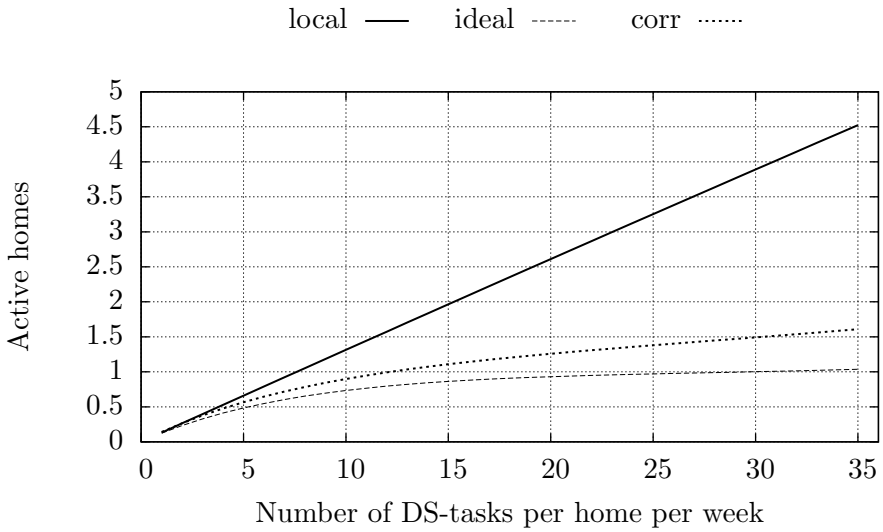

Figure 4.9: Number of *active* homes for $N = 100$ homes.

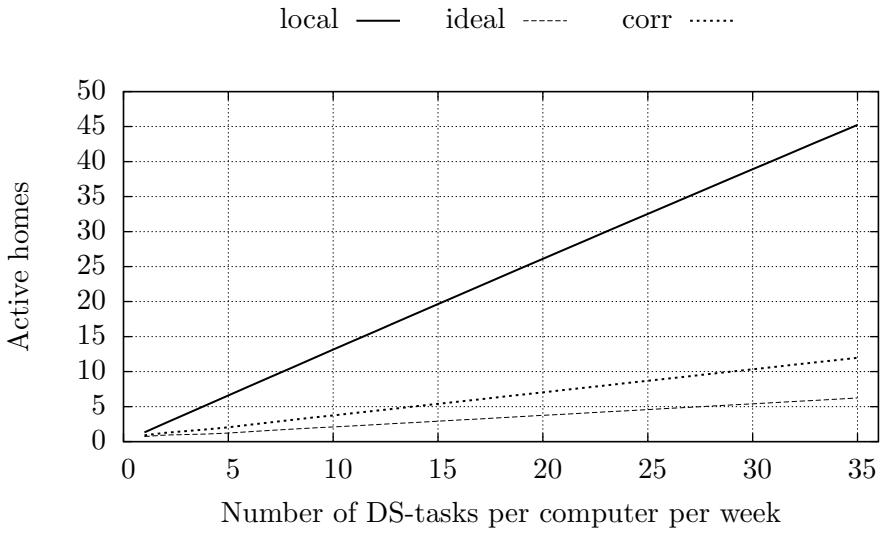

Figure 4.10: Number of *active* homes for $N = 1000$ homes.

save a substantial amount of energy. For instance, in the scenario $N = 1000$ when assuming that each home consumes 100 W and creates 35 DS-tasks per week, without cooperation 1000 homes would constantly consume more than 4.5 kW on average just for downloading files, while cooperating homes would only consume about 1.2 kW for the same task.

What we also can see is the power of large numbers. Larger networks enable better resource sharing through better distribution. This can be seen by comparison of the networks with $N = 100$ and $N = 1000$ homes in the corrected case. The larger network with 1000 homes saves 9 % more homes as the smaller network with 100 homes compared each to their local case. Thus, if the number of participating homes increases, distribution of load becomes better, homes' resources can be consolidated better, and therefore more power can be saved.

4.6 Measurement Study

This section refers to [HWT08, HWT10] where the idea of task virtualization for applications, introduced in Chapter 3, is to migrate a VM containing only a specific task. Currently, task virtualization is only possible within expert environments, since due to the complexity of decentralized systems, users are normally not able to create tasks for remote execution. Solving this usability problem would enable most home users to participate in effective resource sharing and is an important part of future work.

A prototype[8] called *vPastry* allows easy to use task virtualization for home environments. The prototype connects to other instances by using the open source library *FreePastry*[9].

A minimalistic resource/performance model is used comprising information about:

- CPU: Number and performance of CPU cores.
- RAM: Size and speed of main memory.
- Disk: Disk space and speed.
- Energy efficiency of the system as a whole (e.g. power supply).
- Connection: Access bandwidth to the overlay.

CPU, RAM and disk metrics can be understood as how much a user is willing or able to contribute. This is important as it enables other participants of the overlay to find a host with desired abilities for a specific task.

To find capable hosts for a specific task, a minimalistic search algorithm is used. When a node receives an invitation for a task it requested, the user is prompted to select a VM to send while a file transfer channel is established in the background.

[8] Developed by my colleague Thomas Treutner.
[9] `http://www.freepastry.org`

VMs following the *Just enough Operating System* (JeOS) approach for trying to reduce the size of the VM to a minimum. Currently available task VMs include VMs for performance tests, converting an MP3 file to *OGG Vorbis*[10], a *Personal Stream Recorder* (PSR) capturing N seconds of a predefined radio web stream and saving it as MP3, stress tests for stability analysis, downloading a specified file using *BitTorrent*[11], and downloading a specified file using the command *wget*[12].

Task VMs consist of a minimal Linux kernel image, a minimal root file system built by *OpenEmbedded*[13] and a task file system layer, which is put on top of the read-only root file system by *UnionFS*[14]. The VM's task file system shelters a shell script which is executed after the VM has started and disk space for installing required packages (e.g. *lame*[15]) and for writing resulting data (e.g. an MP3 file) was allocated. When the executor receives a VM, it is decompressed into a temporary place and started by a runtime execution using the *Kernel-based Virtual Machine*[16] (KVM) driver and *QEMU*[17].

Once a task is finished (which basically means that the respective VM has exited), the VM task file system is compressed and can be sent back to the task owner. Only the task layer is sent back, as the root file system is mounted read-only and has encountered no changes (and never will).

Along the described process, owner and executor of a task track the status of a given task. When vPastry is quitted, this information is serialized to disk and vice versa. When vPastry is started, it looks for a yet existing session to continue. Hence, the owner of the task can turn off his computer after the task is migrated to the executor. When the owner starts vPastry again, it automatically asks the overlay if the previously outsourced tasks are finished yet.

When the task file system layer is migrated back, it is decompressed and moved to a specific file and can be loopback-mounted on Linux or opened with *explore2fs*[18] on Windows.

Although the described work is just a tiny fracture of what would be necessary to realize the envisaged architecture, it is the very first working prototype for distributed energy saving for homes in a global manner and can give an insight what wattage is produced by this system.

Now it is investigated how energy can be saved by sharing home resources, here using *Download Sharing* (DS) as example. In this scenario N homes are interconnected with gateways. For simplicity we assume a homogeneous environment. For future homes we assume a synchronous access with either $B_d = B_u = 50$ Mbit/s, or $B_d = 8$ Mbit/s

[10] `http://www.vorbis.com`
[11] `http://www.bittorrent.com`
[12] `http://www.gnu.org/software/wget`
[13] `http://wiki.openembedded.net`
[14] `http://www.filesystems.org/project-unionfs.html`
[15] `http://lame.sourceforge.net`
[16] `http://www.linux-kvm.org`
[17] `http://www.qemu.org`
[18] `http://uranus.chrysocome.net/linux/explore2fs-old.htm`

downlink and $B_u = 4$ Mbit/s uplink bandwidth. The local bandwidth within the home is 100 Mbit/s. A DS-task uses on average $B_l = 2400$ kbit/s of the available access bandwidth and has on average a size of $F = 700$ MB. It follows that a home can carry out $M = B_d/B_l$ DS-tasks in parallel. Following a Poisson arrival process, home users create DS-tasks with arrival rate λ; the service rate is given by $\mu = B_l/F$. As shared CPU time 3 GHz are considered, further 100 GB of shareable disk storage.

In order to validate the resource consumption of executing VMs, following experiments were carried out:

- A PC (AMD Phenom 9550 Quad-Core Processor (2.20 GHz) with 8 GB main memory) was setup to act as a host.
- A client instance of vPastry then sent $1 \leq k \leq 5$ DS-tasks (VMs) to the host, which decompressed and ran the VMs.
- The DS-tasks then downloaded a file via wget at 300 kB/s from an FTP server connected via GigabitEthernet.

The total uplink bandwidth of the FTP server was limited to 8 Mbit/s to simulate a typical ADSL access network with 8 Mbit/s downlink bandwidth. The uplink bandwidth of a single FTP connection was limited to 300 kB/s to simulate a download that does not fully utilize the available total downlink bandwidth. The intention was to assess the resource demands of such a scenario.

Figure 4.11 (fat lines) shows results, consisting of the long term resource utilization of the CPU, the main memory and the network card of the host PC. As result on the

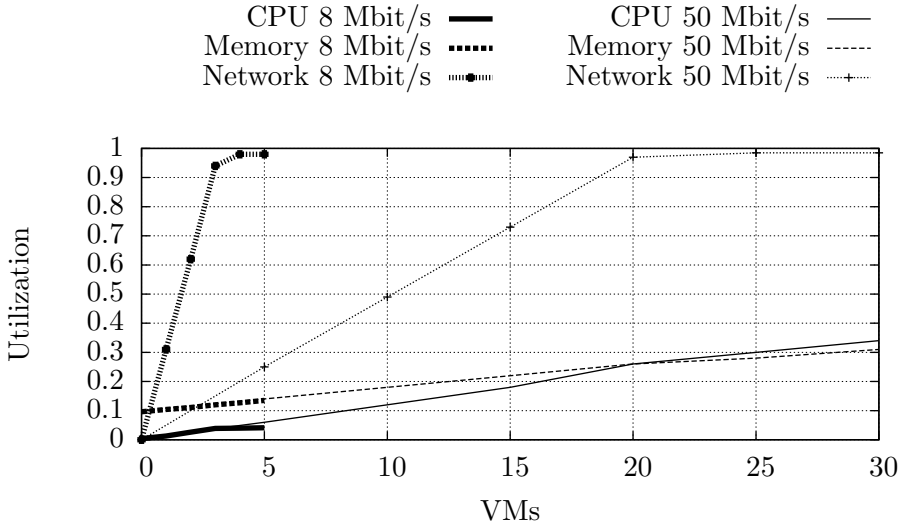

Figure 4.11: Resource utilization (CPU, memory, network) of a download scenario. Fat lines denote results for a 8 Mbit/s access network (i.e. saturation occurs for $\lceil 8/2.4 \rceil = 4$ parallel downloads), thin lines for a 50 Mbit/s access network.

quad core under investigation, the resource utilization clearly depends linearly on the

number of VMs being hosted. Memory and CPU utilization grow slowly due to the little resource demands of our minimalistic Linux operating system. This indicates, when hosting DS-tasks on a modern PC, the system bottleneck indeed is the network, but CPU and memory can be used to host other types of tasks. When the network is beginning to be the bottleneck of the system, an interesting behavior can be observed: With three DS-tasks downloading a file at 300 kB/s (2.4 Mbit/s) each, the resulting downstream utilization is clearly within the available downlink bandwidth of 8 Mbit/s and the network is not yet a bottleneck. Beginning with four VMs the network becomes a bottleneck as the desirable downlink utilization would be 9.6 Mbit/s, and for five VMs 12 Mbit/s. As the downlink bandwidth is limited to 8 MBit/s, the incoming amount of data and therefore the amount of data that has to be written to disk is limited too. In our experiments we observed that the CPU utilization rises more slowly when the system bottleneck is fully utilized. At this point there is no sense in adding more VMs as the resource utilization rises with no benefit of work being done.

In a second experiment the FTP server's total uplink bandwidth was limited to 50 Mbit/s to simulate a future Internet access like FTTH or VDSL (see thin lines in Figure 4.11). A client instance of vPastry sent $1 \leq k \leq 30$ VMs containing a download task (wget, 300 kB/s). The intention was to verify the linear resource demand of a *Download Sharing* scenario also for future Internet access bandwidths. On the quad core under investigation again it can be seen that the CPU utilization U clearly depends linearly on the number of VMs k being hosted. As shown in Figure 4.11, for the situation without network contention ($1 \leq k \leq 20$) this relation is

$$U(k) \approx \frac{0.26}{20} k \tag{4.29}$$

As a result our approach scales well to a large number of VMs being hosted. Even then when the system bottleneck is the network with enough CPU and memory resources being available for hosting other types of tasks. Furthermore, a large number of (small) VMs, possibly > 100, can be hosted with modern PCs, which is important for other scenarios like *Home Management*, where only little CPU and network resources might be required. When the network begins to be the bottleneck of the system, which happens in this case between 20 and 25 VMs, the above mentioned behavior is to be observed here too.

4.7 Model Extension

Yet it was assumed that energy consumption or saving is only derived from the mean number of *active* homes required to cope with a given load compared to the mean number of *passive* homes. Now, based on above measurements presented in Section 4.6, we extend the model slightly by considering the fact that a home hosting several VMs is more utilized as a home hosting less VMs. A more utilized home consumes more energy compared to a lightly loaded one.

Utilization, depending on the number of VMs k, is assumed to be given by (4.29).

As a general model for the consumed energy $E(U)$, depending on the CPU utilization U, we use [FWB07b] which is approximately

$$E(U) \approx 0.5 + \frac{1}{2}U \qquad 0 \leq U \leq 1 \tag{4.30}$$

where a home that does nothing approximately consumes half the energy of a fully loaded home. Combining (4.29) and (4.30) yields

$$E(k) \approx 0.5 + \alpha k \qquad 0 \leq k \leq M \tag{4.31}$$

with $\alpha \leq 0.5$, in our case $\alpha = 0.13/20$. This can then be inserted into (4.24) for the mean number of *active* homes for the local case without sharing and (4.28) for the corrected case with sharing. In the local case each home downloads only its own files, and the energy required for a single home is (setting $\rho = \lambda/\mu < M$):

$$\begin{aligned}
E_{l,1} &= \sum_{k=1}^{M} E(k)\pi_k \\
&= \sum_{k=1}^{M} (0.5 + \alpha k)\pi_k \\
&= 0.5 \sum_{k=1}^{M} \pi_k + \alpha \sum_{k=1}^{M} k\pi_k \\
&= 0.5(1 - \pi_0) + \alpha\rho \sum_{k=1}^{M} \pi_0 \frac{1}{(k-1)!}\rho^{k-1} \\
&= 0.5(1 - \pi_0) + \alpha\rho \sum_{k=0}^{M-1} \pi_0 \frac{1}{k!}\rho^{k} \\
&= 0.5(1 - \pi_0) + \alpha\rho(1 - \pi_M)
\end{aligned} \tag{4.32}$$

In the local case the energy E_l consumed by N homes is then given by

$$\begin{aligned}
E_l &= N\left(0.5(1 - \pi_0) + \alpha\rho(1 - \pi_M)\right) \\
&= 0.5N_l + N\alpha\rho(1 - \pi_M)
\end{aligned} \tag{4.33}$$

It must be noted, that $N\alpha\rho(1-\pi_M)$ will be quite small (relatively to N_l) for reasonable values of ρ and the effect of this term is thus limited. In the corrected case, when exactly k downloads are going on, then $a = \lceil k/M \rceil$ homes are running and the energy $E_{c,k}$ consumed by them is

$$\begin{aligned}
E_{c,k} &= (a-1)E(M) + E(k - (a-1)M) \\
&= (a-1)(0.5 + \alpha M) + (0.5 + \alpha(k - (a-1)M)) \\
&= 0.5a + \alpha k
\end{aligned} \tag{4.34}$$

where $E(k)$ is given by (4.31). Summing up over all k and setting $\hat{\rho} = N\lambda/\mu < N\,M$,

and considering the number of busy homes in the ideal case, N_i (4.25) yields the energy consumption E_i due to the downloads

$$\begin{aligned} E_i &= \sum_{a=1}^{N} \sum_{k=(a-1)M+1}^{a\,M} \hat{\pi}_k E_{c,k} \\ &= 0.5 \sum_{a=1}^{N} a \sum_{k=(a-1)M+1}^{a\,M} \hat{\pi}_k + \sum_{k=1}^{N\,M} \alpha k \hat{\pi}_k \\ &= 0.5 N_i + \alpha \hat{\rho}(1 - \hat{\pi}_{NM}) \\ &= 0.5 N_i + N \alpha \rho (1 - \hat{\pi}_{NM}) \end{aligned} \tag{4.35}$$

Additionally there are also transmissions for sending the content back to the content owner, represented by N_a given in (4.26). The additional effort is due to starting the owner home which only downloads one file, i.e. the own download content. Its energy consumption is actually $E(1) = 0.5 + \alpha$, and the energy consumed for doing this is

$$E_a = E(1) N_a = (0.5 + \alpha) N_a \tag{4.36}$$

Combining the energy consumption E_i in the ideal case (4.35) with the additional energy consumption E_a for the back transfer (4.36), and taking into account (4.27) as the mean number of active homes N_c in the corrected case, the overall energy E_c consumed in the corrected case is then

$$\begin{aligned} E_c &= E_i + E_a \\ &= 0.5 N_c + \alpha N_a + N \alpha \rho (1 - \hat{\pi}_{NM}) \end{aligned} \tag{4.37}$$

Again we see that the energy consumption is equal to the consumption of idle homes (which is essentially the number of running homes divided by 2) plus αN_a which is for modern systems negligible due to the small α, further on a term that depends on the load $\rho = \lambda/\mu$ and N. Again we can state that for reasonable values of ρ the influence of this term will be limited, but of course its influence grows as ρ grows. By inspecting the solutions, the influence of the utilization on the overall energy consumption depends very much on the load. In general, the system is energy efficient if the mean number of homes N_c in the corrected case is smaller than the mean number of homes N_l in the local case to compensate the additional mean number of homes N_a that are *active* for receiving results. N_a can be seen as the cost of distribution.

4.8 Summary

The quintessence of this chapter was to show that it makes sense for homes to share resources for power saving. Three possible and typical applications were introduced; each covering a specific type of resource requirements. The moderate application *Download Sharing* counts up for many tasks relying mainly on the downlink bandwidth of homes. The more comprehensive application *Video Encoding* requires much CPU time and so

the difference of homes in terms of performance is important. The *Home Management* application needs high availability and in conjunction with replication also reliability. By examining each of this applications in more detail as done in Chapter 6, we can see better how to achieve any advantage in form of power saving.

It is also important how homes differ in performance α and wattage β. The factors α and β can be used as weights for initial simulation parameters for CPU speed and power consumption of homes. Especially CPU-intensive tasks strongly depend on the performance factor α as well as on the wattage factor β.

5 Simulation Model

This chapter explains the simulation model and related concepts for proofing the feasibility of the idea of task sharing in terms of power saving. The simulation model consists of nodes representing homes connected to the Internet and providing network bandwidth, CPU time and disk space to other nodes. Nodes generate and share load, i.e. tasks as introduced in Chapter 4. It is assumed that *Virtual Machines* (VMs) as introduced in Section 2.2 and considered more in detail in Section 4, encapsulate all parts necessary to exchange tasks. Simulation experiments for *Download Sharing* (DS), *Video Encoding* (VE) and *Home Management* (HM) show the feasibility of this load sharing approach for future networked homes. Further, the simulation implements a *Birth-and-Death* (BD) process to model a population of load that fills up the network of homes with tasks. The population grows due to new tasks and shrinks while homes complete tasks. The theory beyond a BD process is well explained in [KM08, BGdMT06]. The simulation verifies the analytical model presented in Chapter 4.

The simulation itself is a self-driven stochastic (also called synthetic or distribution-driven) discrete-event simulation [LK99] with asynchronous timing [KM08, BGdMT06]. In combination with the stochastic feature, the simulation is designed for steady-state analysis; after the beginning transient phase the variables average out to steady-state values. Besides, the simulation is sequential in the sense, that events are executed on one single machine. Latter implies a strictly event-oriented simulation that evolves over time by executing events in increasing time order. The simulation implements a continuous-time but discrete-state model [Jai91] in the sense that homes produce load continuously but there is a countable number of tasks around. The simulation is probabilistic and results differ on repetitions despite of the same initialization parameters.

The simulator architecture outlined in Figure 5.1 is called event-scheduling or event-driven approach, because the simulation control scans the event list, identifies the next event (usually the event with the smallest time) and executes the actions defined by this event. Since the simulation is a discrete event simulation, an event is scheduled for a given time point and events may schedule further events for the future. So, the simulation flow steps from event to event until the termination conditions are reached. The termination condition normally is an event with a timestamp that exceeds the desired simulation time.

The simulation [LK99] is self-driven because *Random Number Generators* (RNGs) are used for generating several rates. The arrival rate is calculated based on a RNG and the customers constitute a homogeneous set. Arrival times of customers vary in

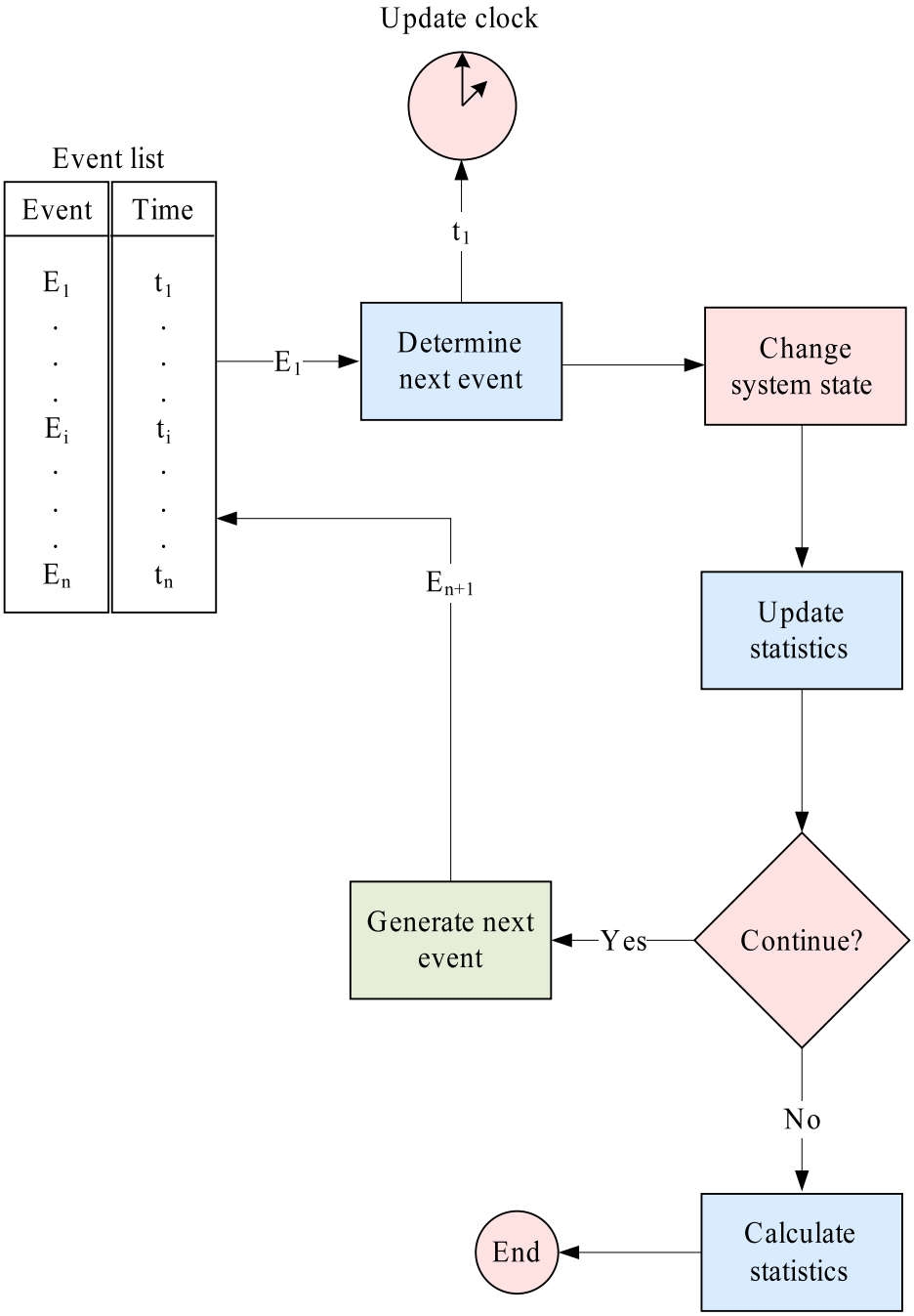

Figure 5.1: The rudimentary simulator architecture.

an unpredictable fashion and are only characterized in terms of i.i.d. (independent, identically distributed) first-order statistics. Asynchronous timing (or event advance) differs from synchronous timing in the way the simulator clock is advanced. With asynchronous timing the simulator clock is increased by a variable amount of time rather than a fixed time step, as with synchronous timing defined. Discrete-event, because the simulation is kept running by discrete events occurring at given times. The simulation clock is set to the event times. Since an event can cause future events, the simulation goes further on the time axis.

Events are stored in a data structure called event list. The event list is an ordered FIFO-list (First In First Out) organized as heap, a special case of a binary tree [Jai91].

A self-driven simulation requires a mechanism for generating sequences of events which in turn govern or imitate the dynamic behavior of the system under investigation. The random nature of events is characterized by underlying probability distributions. The simulator must produce sequences of variates (sample values of a random variable) from continuous probability distributions. For generating variates from any specified distribution it is necessary to generate variates drawn from the uniform distribution. Based on this uniformly distributed variates an e.g. exponential distributed variate can be calculated. The simulation uses instead of Java's built-in RNG, which is a *Linear Congruential Generator* (LCG) for uniform distributed variates (java.util.Random) according to Knuth [Knu97], a Java implementation[1] of the approximately 1/3 faster *Mersenne Twister*[2] [MN98] algorithm. Furthermore Mersenne Twister offers a very high equidistribution up to 32 bits accuracy. Although Mersenne Twister also only generates pseudorandom sequences, no important statistical tests reveal a significant discrepancy from a truly random sequence.

Since no special network protocol must be simulated and to have a generic framework with maximum adaptability, the discrete event simulation, for simulating network communication on application level, is implemented in the general-purpose programming language *Java*[3] from scratch. For simulating the application communication of *Download Sharing*, *Video Encoding* and *Home Management*, proper event handling and network routing mechanisms have been implemented. To be independent from given simulation frameworks like *OPNET*[4] or *OMNeT++*[5] and because of not simulating a specific network protocol, as can be done with the open network simulation *ns-2*[6], an own simulation framework was created from our group to quickly generate first results and then implementing more comprehensive scenarios. Over this, the simulation verifies an analytical model.

A middleware, where investigated applications run, divides the simulation in a network and application layer. Figure 5.2 illustrates the simulated network. There are

[1] `http://goui.net/doc/net/goui/util/MTRandom.html`
[2] `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`
[3] Java SE Development Kit (JDK) 6 Update 17
[4] `http://www.opnet.com`
[5] `http://www.omnetpp.org`
[6] `http://www.isi.edu/nsnam/ns`

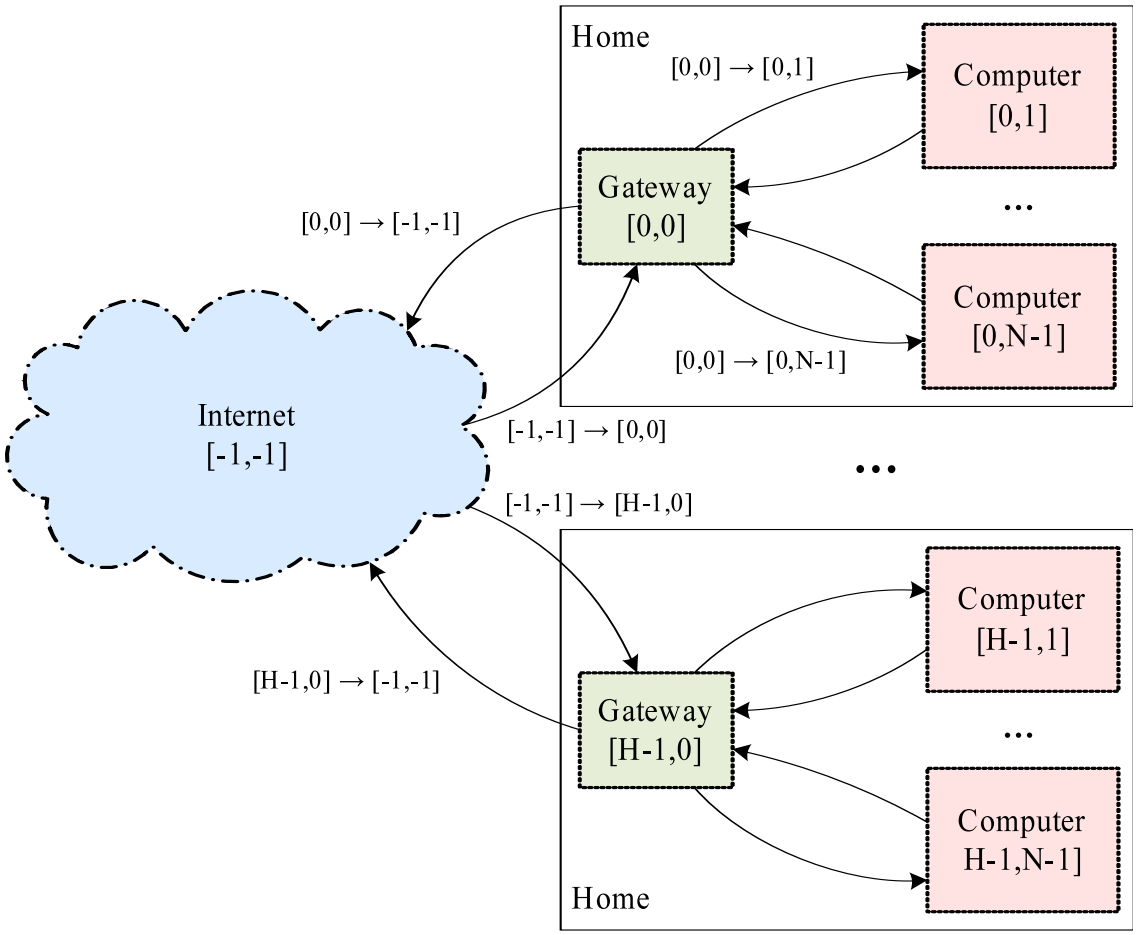

Figure 5.2: Simulated network.

three types of nodes:

- A computer as resource provider inside a home.
- A gateway without resources, but with no churn that represents the home in the overlay.
- The Internet as special node for interconnection and delay between gateways.

The simulated network consists of homes, where in each home are one gateway and at least one computer. All homes together build an overlay. On gateways and computers run the middleware which implements the application logic. The middleware of a gateway only handles supportive overlay related operations, where the middleware of a computer implements distributed algorithms for application-specific task sharing.

An addressing scheme enables routing. The address of a certain node within the simulation is of form $[H, N]$. The address space defines $H - 1$ homes and $N - 1$ nodes inside a home. The Internet as special node has the address $[-1, -1]$. In Figure 5.2 e.g. if the computer with address $[0, 1]$ wants to send a message to computer with address $[H - 1, 1]$ we have the trace $[0, 1] \rightarrow [0, 0] \rightarrow [-1, -1] \rightarrow [H - 1, 0] \rightarrow [H - 1, 1]$ since the Internet and gateways must be passed as in reality. The simulation

abstracts the Internet as single node because routing inside any overlay, as used in common P2P-overlays, is not in scope of this work. The interest lies in application scenarios, underlaying distributed algorithms suitable for task sharing, and the amount of resources required for a given load.

Homes are modeled as nodes cycling through several states. The transition between states are triggered by events during time. There are user events, like the creation of a new task or system events created by the underlaying distributed logic of inter-home communication. The four possible home states, as already introduced in Chapter 3, are:

- An *active* (A) home executes tasks for remote homes. This state is the most cooperative state, because the home does work for other homes. Tasks can be also added locally by the own user, but it is assumed that all tasks can be carried out without user invention or failures in the basic scenario.

- An *active-blocked* (AB) home generates new tasks but is not available for remote tasks. Additionally, the home tries to find another home for created tasks. In this state the home is not under system control, i.e. the user has exclusively access on his resources.

- An *active-blocked-content* (ABC) home is sending back the result of a completed task. Once a remote task is finished, the executer (home) notifies the owner (home). If the owner is ready to receive the result as DS-task, the executer uploads the result to the owner. In the case the owner was in state P, the state ABC is only for awaking the owner from hibernation as long as the transfer lasts. Indeed, a home in state ABC switches back to state P immediately after all transfers are done. Note, that transfers can also happen if the owner is in state A. In this case the executer tries to upload to the owner if the owner has currently enough resources, otherwise the executer queues the completed task and retries later.

- A *passive* home currently does not contribute resources to the system, but saves power and helps to reach the global optimum of energy efficiency. The home can be waken up by a user or a system event.

Figure 5.3 shows the homes' state cycle and introduces possibilities for transitions where P[event] denotes the possibility that a home transitions to another state due to an event.

The simulation uses an arrival process with interarrival times i.i.d. exponentially distributed with rate λ. The arrival rate λ is Poisson distributed [LK99, Agr02] based on uniform variates obtained by Mersenne Twister. The interarrival time T of customers is based on a time period P and the load L (number of tasks within the period). The arrival rate is then

$$\lambda = \frac{L}{P} \tag{5.1}$$

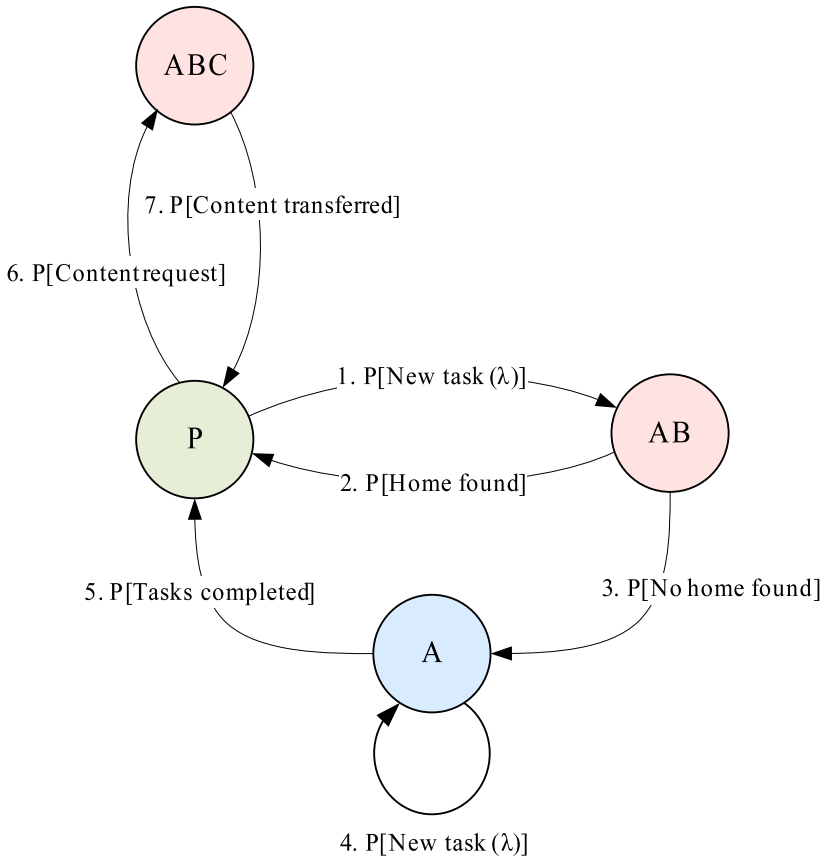

Figure 5.3: State cycle of nodes.

and the mean interarrival time is

$$T = \frac{1}{\lambda} \tag{5.2}$$

Poisson variates can be calculated from the cumulative distribution function of the exponential distribution with mean μ

$$F(x) = 1 - e^{-\mu x} \tag{5.3}$$

by setting (5.3) equal to a decimal number u from $U(0, 1)$

$$u = 1 - e^{-\mu x} \tag{5.4}$$

and inversing it to

$$F^{-1}(u) = x = -\frac{\ln(1-u)}{\mu} \tag{5.5}$$

Since $1 - u$ is itself a random number, (5.5) can be simplified to

$$x = -\frac{\ln(u)}{\mu} \tag{5.6}$$

and this finally yields with (5.1)

$$x = -\frac{\ln(u)}{\lambda} \tag{5.7}$$

as interarrival times for transitions 1. and 4. of Figure 5.3.

Homes build an overlay without central unit like a server that has all knowledge about homes. The basic simulated network structure is shown in Figure 5.2. Due to the consecutively numbering of homes' addresses (as equivalent to IP-addresses in the Internet), the overlay network can be imagined as ring as illustrated in Figure 5.4. To replace server functionality, a hierarchical and structured P2P approach is used.

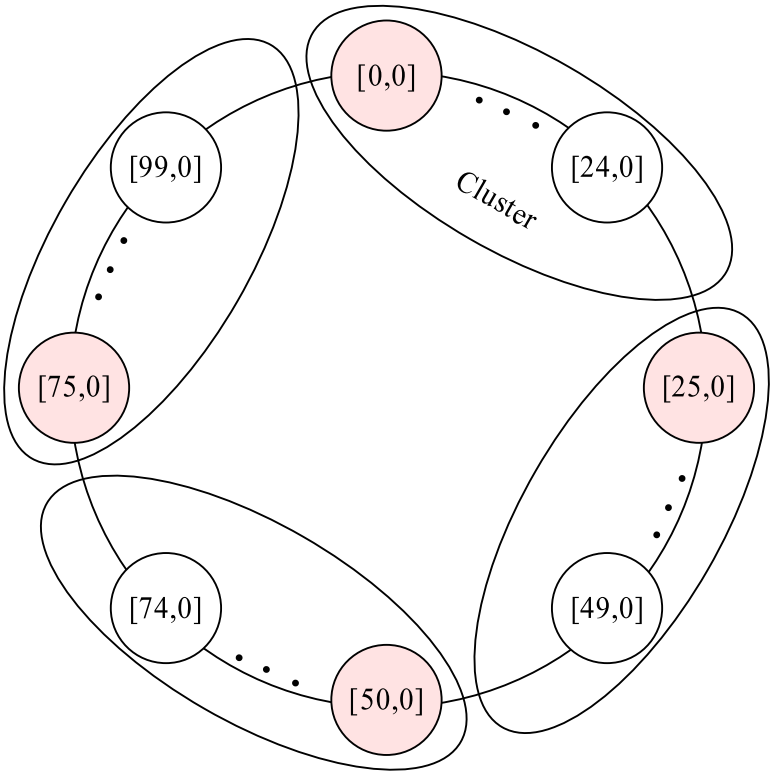

Figure 5.4: An overlay for 100 homes and 4 super homes.

Super Homes (SHs) segment the overlay into clusters. The gateway of a SH has the additional role of a directory service where state information about homes is stored. For a given network size, a certain number of SHs is defined like in Figure 5.4 with 100 homes, where 4 homes ([0,0],[25,0],[50,0],[75,0]) act as super homes. Each normal home sends state information to its SH. The home can calculate the network address of his SH S by

$$S = \left\lfloor \frac{H}{C} \right\rfloor \times C \tag{5.8}$$

where H is the network number and C is the cluster size, i.e. how many homes are

assigned to a SH according to Figure 5.4. C is simply calculated with (5.9).

$$C = \left\lfloor \frac{numHomes}{numSuperHomes} \right\rfloor \tag{5.9}$$

A home can determine, that it is a SH if

$$0 = H \bmod C \tag{5.10}$$

is true. Further, a super home can calculate the network address of all other SHs by incrementing or decrementing the own network address by C within the range of the number of homes ($numHomes$).

SHs replicate their knowledge about normal homes. A normal home sends a state object (including information about the own state, the load, etc.) to its gateway. The gateway forwards the information to the gateway of its SH. SHs forward state objects to other SHs. Therefore, in case of a resource request by a specific home, almost full knowledge about currently *active* homes is available. Gateways of homes and SHs build a proprietary overlay providing base P2P functionality and produce communication costs. The message flow of state information replication is shown in Figure 5.5 with N considered SHs.

As state information is forwarded by each passed gateway, it is propagated through the network, i.e. becomes ready for home requests. Homes also ask their gateways for other *active* homes. The resource request for a list of *active* homes is forwarded to the corresponding SH. The SH directly sends a network-wide actives list to the requesting home.

This basic overlay adds communication costs to the simulation. Further it allows to study the impact of missing state information in case of failures and high churn in the network. This will be especially investigated in Section 6.3 where replication for *Home Management* tasks is used to simulate compensation of failed or malfunctioning homes in the network. Because state information is propagated asynchronously among SHs, this overlay adds some uncertainty which itself causes costs if homes reported as *active* fail and must be substituted.

5.1 Basic Scenario

The basic simulation scenario, according to Figure 5.3, is as follows: A user decides to start a new task (DS-task, VE-task, HM-task, or a combination of them) with rate λ. The creation of a new task implies a home's transition from state P to AB (transition 1), thus the home is awaken and causes wattage through running (a) computer(s). In state AB the user has exclusively access to his resources and passes the newly created task to the system (e.g. via a special P2P client). The system now searches an executer which is also connected to the overlay where the task can be executed. Figure 5.6 shows all steps for the default communication pattern of task sharing with an initial *passive* home 1 and an *active* home 2 with the corresponding SH.

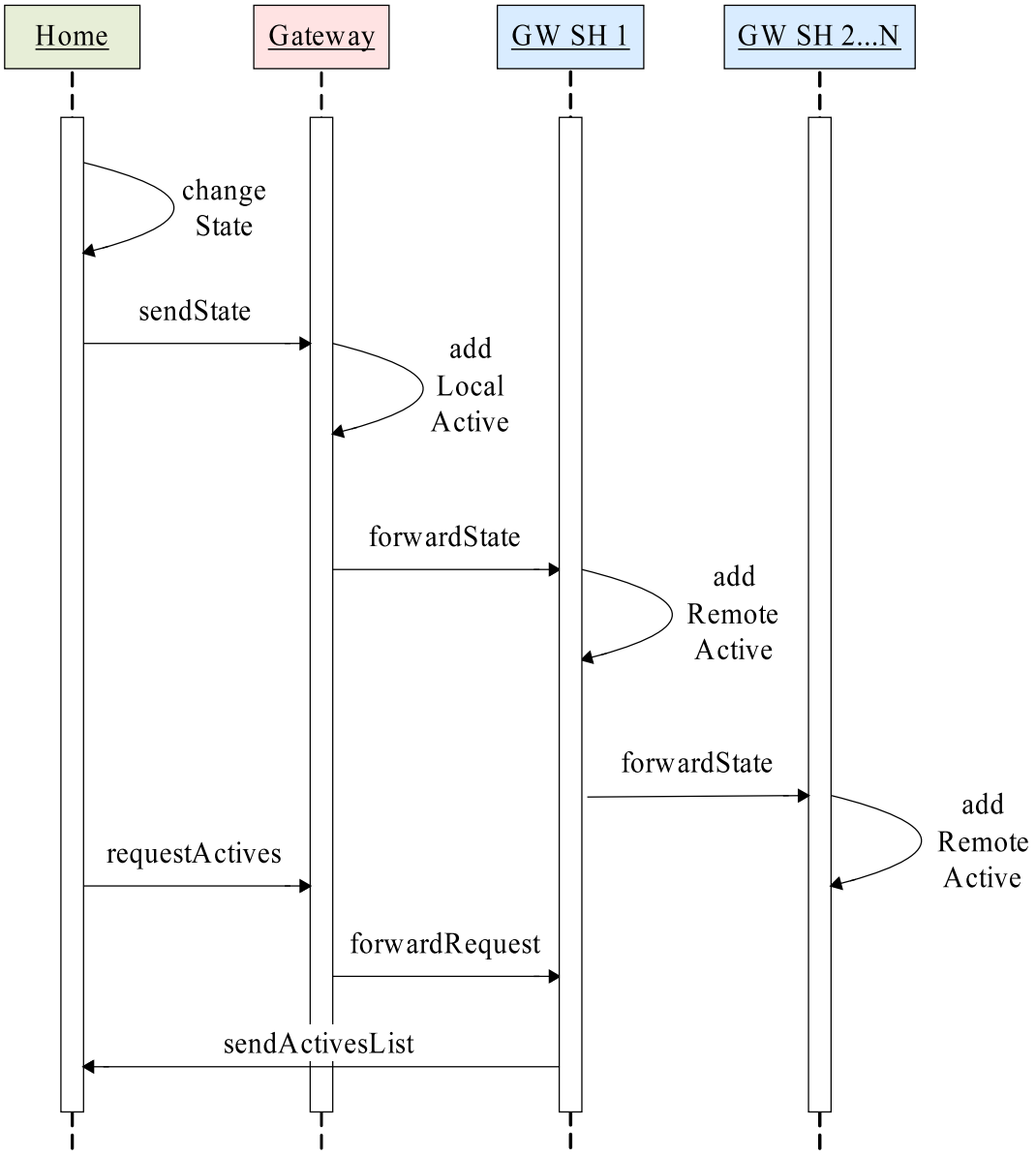

Figure 5.5: State information replication.

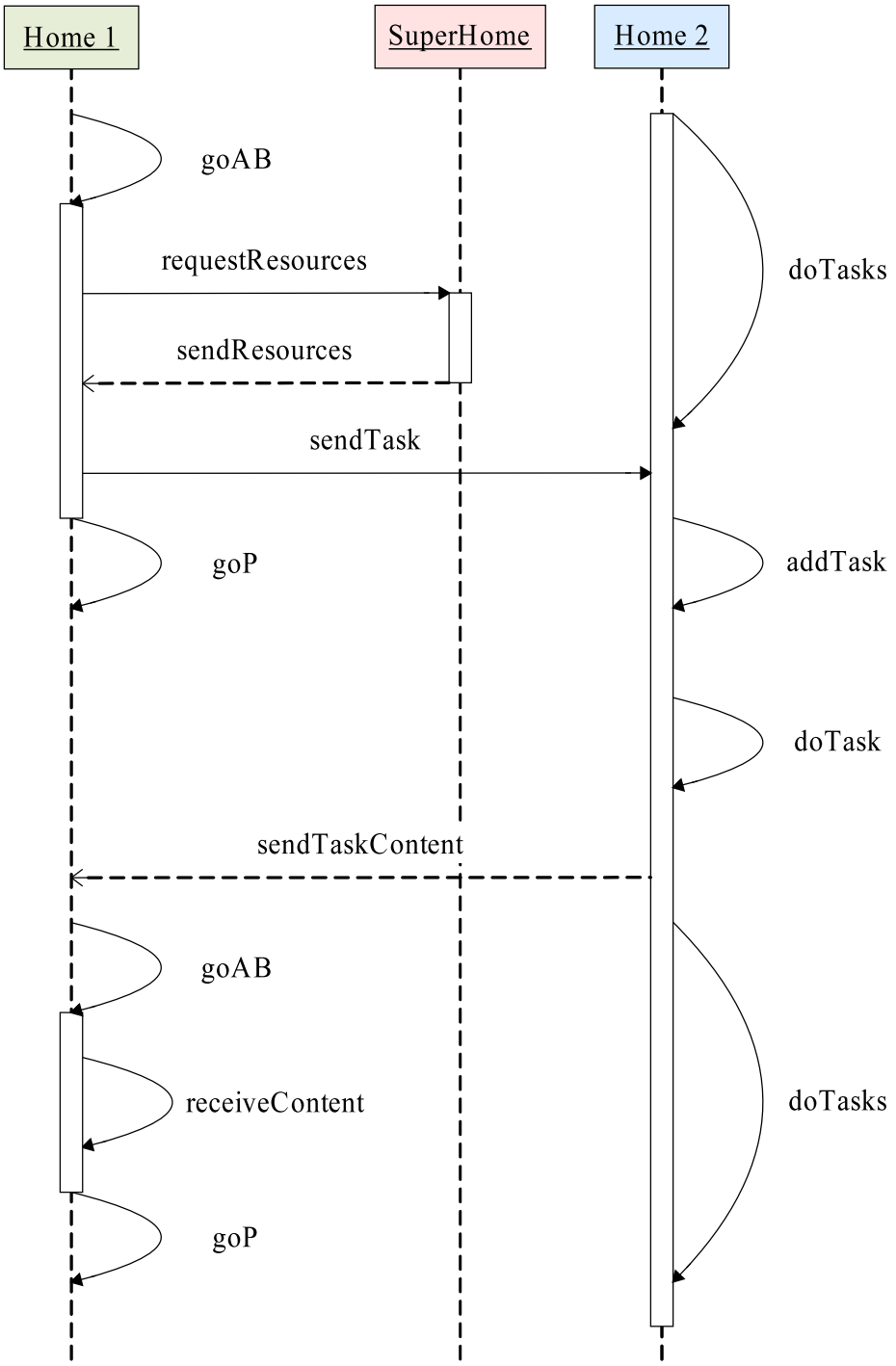

Figure 5.6: Default communication pattern of task sharing.

If an executer could be found, the VM containing the task is sent to this home and the owner becomes *passive* (transition 2) to save power. Otherwise the owner itself becomes *active* to execute the task locally (transition 3) as shown in Figure 5.7 with an initial *passive* home 1 and an *active* home 2. While in state A, a home can receive tasks allocated by the system or added by the local user (transition 4). In state A the home may concurrently execute own and remote tasks and accepts new arriving tasks until a threshold. After that threshold the home only finishes running tasks to prepare for state P.

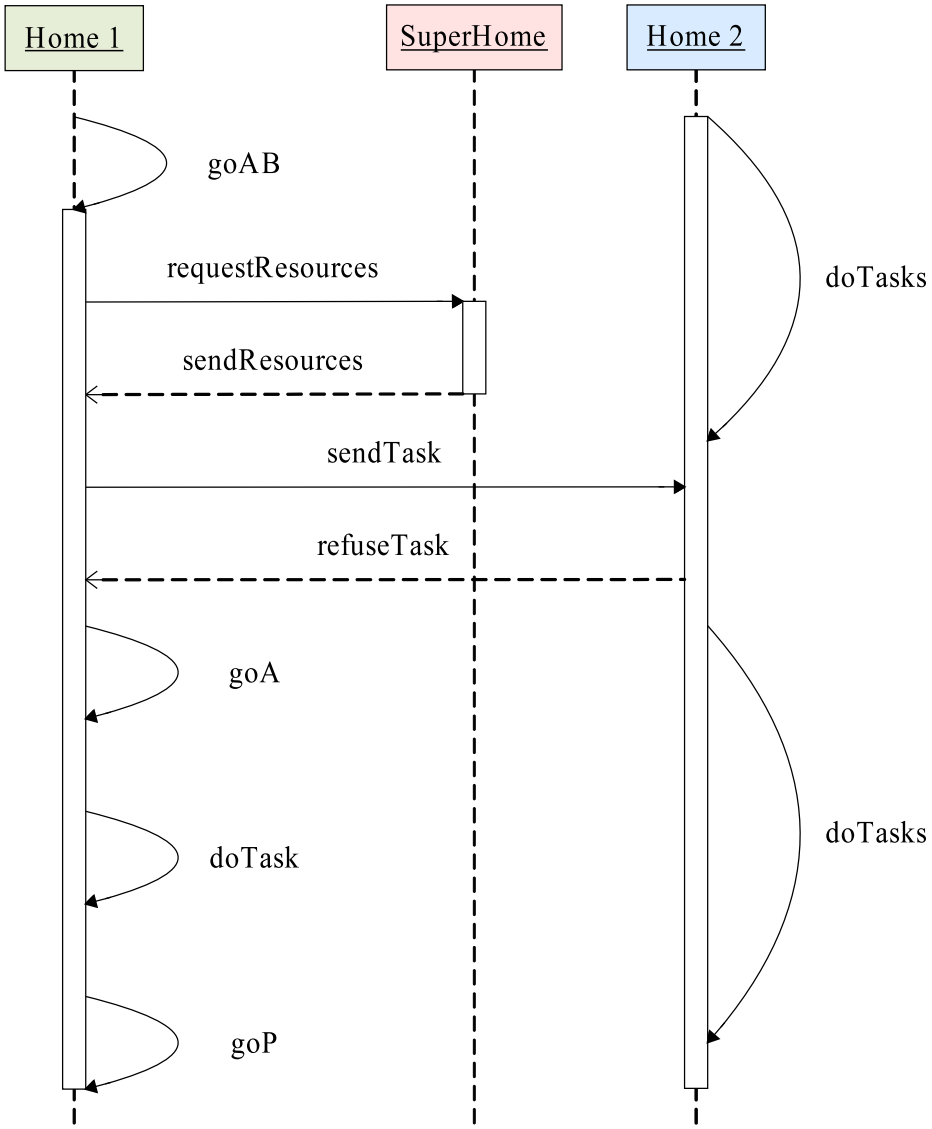

Figure 5.7: Communication pattern of task sharing if the task can not be outsourced.

If all tasks are done, the home switches to state P (transition 5) to save again power. If a home is *passive* and a request for the content of a previously finished task arrives, the home will go into state ABC (transition 6) as shown is Figure 5.6. In state ABC the home downloads the content (result) of its task from the executer. After that, the home becomes again *passive* (transition 7). Therefore, the time a home resides in state

ABC shortens the interarrival time of transition 1.

5.2 Parameters

The simulation is parameterized with variables derived from reality and experience. Table 5.1 gives an overview about relevant parameters used for all simulation runs evaluated in the rest of this work.

The simulation network consists of a number of homes (*numHomes*). For concentrating on inter-home instead of intra-home behavior, the home is assumed as computing unit; therefore a home is understood as abstraction for all computing equipment inside. Among homes, a certain number of homes (*numSuperHomes*) is assigned with a special role. These super homes act like a directory service and store state information about homes. Super homes replicate their knowledge among each other. Factors for performance (α) and wattage (β), as introduced in Chapter 4, weight home resources according to two classes of homes.

The home's shared resources are expressed by the downlink (*bwDn*) and uplink (*bwUp*) access bandwidth, a synchronous local bandwidth (*locBw*) for inter-home communication, the CPU time (*cpu*), and the disk space (*mem*). A wattage parameter (*watt*) defines how much power is consumed on average if the home is under load.

For building the simulation network, homes are distributed in 2D-space to simulate partitioning of homes in several aggregated areas like cities in reality. The vicinity probability (*vicinityProp*) controls how near homes are situated and the position adjustment (*posAdjust*) avoids too dense positioning. The average base delay (*delayBase*) for messages passing the Internet is fixed like an additional distance delay (*delayDist*) that depends on the distance between homes (point to point delay). Parameters for the latency within the Internet (*latInternet*) or within homes (*latLocal*) add communication costs. For the sake of completeness, the message size (*msgSize*) defines the default size of control messages. Delay, latency and message size may sum up to considerable communication costs in the simulation.

As in the evaluation sections for *Download Sharing* 6.1, *Video Encoding* 6.2 and *Home Management* 6.3 showed, results are based on a given load (*load*), generated within a given simulation period (*simPeriod*). The service time (*serviceTime*) specifies the maximum time period an *active* home will accept incoming tasks. In standard simulations a home will accept incoming tasks from the point of time it changed to state *A* until the service time is expired. The simulation time (*simTime*) says how much real time is simulated.

The average downlink bandwidth per DS-task (*dsBwDn*) can be fixed as threshold like in common download clients for P2P networks [MRPM08]. The average uplink bandwidth per DS-task (*dsBwUp*) is also assumed as fixed. Similar the average CPU usage by a DS-task (*dsCpu*) and the average disk space allocated by a DS-task (*dsMem*) are fixed values.

For a VE-task the length of the source video (*veSrcLength*) is the playtime of a video. Additionally, with bit rates for the video (*veSrcVidRate*) and audio (*reSrcAudRate*)

Parameter	Unit	Description
numHomes	Number	Number of homes.
numSuperHomes	Number	Number of super homes.
α	Factor	Adjusts the performance relation between homes.
β	Factor	Adjusts the wattage relation between homes.
bwDn	kbit/s	Home's max. shareable downlink bandwidth.
bwUp	kbit/s	Home's max. shareable uplink bandwidth.
locBw	kbit/s	Home's synchronous local bandwidth.
cpu	Mhz	Home's max. shareable CPU time.
mem	MB	Home's shared disk space.
watt	watt	Home's mean power consumption under load.
vicinityProp	Probability	Clustering of spatial near homes.
posAdjust	Factor	Adjusts positions of spatial near homes.
delayBase	ms	Base delay within the Internet.
delayDist	ms	Point to point delay between homes.
latRemote	ms	Latency of links within the Internet.
latLocal	ms	Latency of links within homes.
msgSize	bytes	Default size of control messages.
load	Number	Number of task arrivals per week.
simPeriod	s	Mean interarrival time of tasks.
serviceTime	s	Period for that an *active* home accepts new tasks.
simTime	s	Simulation time.
dsBwDn	kbit/s	Average downlink bandwidth per DS-task.
dsBwUp	kbit/s	Average uplink bandwidth per DS-task.
dsCpu	Mhz	Average CPU time per DS-task.
dsMem	MB	Average disk space allocated per DS-task.
veSrcLength	min	Length of the source video.
veSrcVidRate	kbit/s	Mean source video bitrate (DVD).
veSrcAudRate	kbit/s	Mean source audio bitrate (DVD).
veTarVidRate	kbit/s	Mean target video bitrate (Xvid).
veTarAudRate	kbit/s	Mean target audio bitrate (Xvid).
veMinCpu	Mhz	Minimum granted CPU usage for encoding.
hmBwDn	kbit/s	Average download bandwidth per HM-task.
hmBwUp	kbit/s	Average upload bandwidth per HM-task.
hmCpu	Mhz	Average CPU usage per HM-task.
hmMem	MB	Average disk space per HM-task.
hmRep	Number	Number of HM-task replications.
hmMeanFailures	Number	Mean number of failures.

Table 5.1: Simulation parameters.

part of the source material (DVD-Format with MPEG2 compression), the size of the source file can be calculated. Also the file size of the compressed target video is specified by video (*veTarVidRate*) and audio (*veTarAudRate*) bit rates according to e.g. Xvid[7] encoding. A parameter for CPU usage (*veMinCpu*) defines the minimum CPU time that must be granted for *Video Encoding*.

The average downlink bandwidth per HM-task (*hmBwDn*) is fixed and based on the assumption that only small scaled AV-streams and sensor/actuator data is considered. Just as well as the average uplink bandwidth per HM-task (*hmBwUp*), the average CPU usage per HM-task (*hmCpu*), and the average disk space per HM-task (*hmMem*) is fixed. For availability analysis, HM-tasks are replicated by a certain number (*hmRep*) of copies within the network. Then, the mean occurrence of HM-task failures (*hmMeanFailures*) per given simulation period (*simPeriod*) yields the possibility whether and when a home fails during the *active* period. All running tasks of a failed home are lost. If there is replication with $hmRep > 0$ other homes will recover the failed HM-tasks.

Depending on the intended application type (DS, VE, HM), each simulation run is initialized with most of these introduced parameters which will be repeated in more detail beneath corresponding results in Chapter 6.

5.3 Economic Model

An economic model is necessary to avoid free-riding as experienced in P2P-networks. The problem of free-riding are selfish peers trying to get content from the P2P-network without contributing own content or resources. There should be a mechanism for detecting such bad sharing behavior, for measuring the sharing balance of peers, and also for enforcing policies to avoid or penalize selfish behavior of peers.

The service time (*serviceTime*) addresses the fairness of the system. The service time is the amount of time a home must reside *active* to be accessible by the system, i.e. for receiving remote tasks for execution.

Longer service times cause longer *active* periods for homes. Longer service times also make it easier for homes to find another home for their tasks. On the other side, a low service time makes a home's *active* period shorter and therefore homes may change to state P earlier.

The supposed fairness of the system with a short service time is higher than with a long one. With a short service time only a fraction of the load must be handled by homes compared to a long service time. A long service time causes less concurrently *active* homes, but leads to a worser load distribution; thus fewer homes must handle the whole load and reside longer *active*. A short service time may cause more *active* homes, but each of them must be *active* for less time. Also with a long service time the load moves slower between homes. On the contrary, a short service time implies better load distribution because it avoids the repeated selection of *active* homes due

[7] `http://www.xvid.org`

to their long stay in state A.

An economic model modifies the service time based on past behavior of the home. The model relates sent tasks S to accepted tasks A and calculates the service time for the next *active* period for each home. Sent tasks are those tasks, which the home tries to sent out to another *active* home. Sent tasks are the cost for a home. However, accepted tasks are remote incoming tasks. An *active* home accepts incoming tasks by other *active-blocked* homes. The accepted tasks are the revenue of the home. If the home is in state A, every accepted task neutralizes one sent task. Therefore equal numbers of sent and accepted tasks implies that the home has so much contributed to the system as consumed and no modification to the service time for the next *active* period will happen.

Equation (5.11) is the short cost C_t after the *active* period of a home at time point t.

$$C_t = A - S = \begin{cases} < 0 & S > A \\ 0 & S = A \\ > 0 & S < A \end{cases} \tag{5.11}$$

The set C of short costs with

$$C_t \in C \qquad [tdt, (t+1)dt) \qquad 0 \leq t \leq T \tag{5.12}$$

is growing with time where T is the time point of the latest determined value. The minimal short cost C_{min}

$$C_{min} = \min C \tag{5.13}$$

and maximal short cost C_{max}

$$C_{max} = \max C \tag{5.14}$$

are used for calculating the normalized short cost N_t for an ended *active* period at t with

$$N_t = \begin{cases} \frac{1}{2} & C_{min} = C_{max} \\ \dfrac{C_t - C_{min}}{C_{max} - C_{min}} & C_{min} \neq C_{max} \end{cases} \tag{5.15}$$

where

$$N_t \in N \qquad [tdt, (t+1)dt) \qquad 0 \leq t \leq T \tag{5.16}$$

Additionally the limited short cost L_t for an ended *active* period at t with

$$L_t \in L \qquad [tdt, (t+1)dt) \qquad 0 \leq t \leq T \tag{5.17}$$

is used as

$$L_t = \begin{cases} \frac{1}{2} & C_{min} = C_{max} \\ \dfrac{-C_{min}}{C_{max} - C_{min}} & C_{min} \neq C_{max} \end{cases} \tag{5.18}$$

L_t is the marginal cost where the economic model decides that the home was neither a donor (a home that has contributed to the system) nor a leecher (a home that has more consumed as contributed) in the past *active* period. The average normalized cost C_{avg} is calculated according to

$$C_{avg} = \frac{\sum_{t=1}^{T} N_t}{T} \tag{5.19}$$

for all determined normalized short costs N_t. The service time S_{t+1} for the next active period is then

$$S_{t+1} = \begin{cases} S_t \times (1 - C_{avg}) & N_t > L_t + \epsilon \\ \\ S_t \times (1 + C_{avg}) & N_t \leq L_t - \epsilon \end{cases} \tag{5.20}$$

for

$$0 \leq \epsilon \leq 1 \tag{5.21}$$

where S_t is the current service time and ϵ is the boundary for deciding if the home was a donor, a leecher, or had a balanced behavior in the last *active* period.

Figure 5.8 depicts the coherence between introduced values and service time modification. We can see the limited cost L_t is the value for deciding if the home is currently

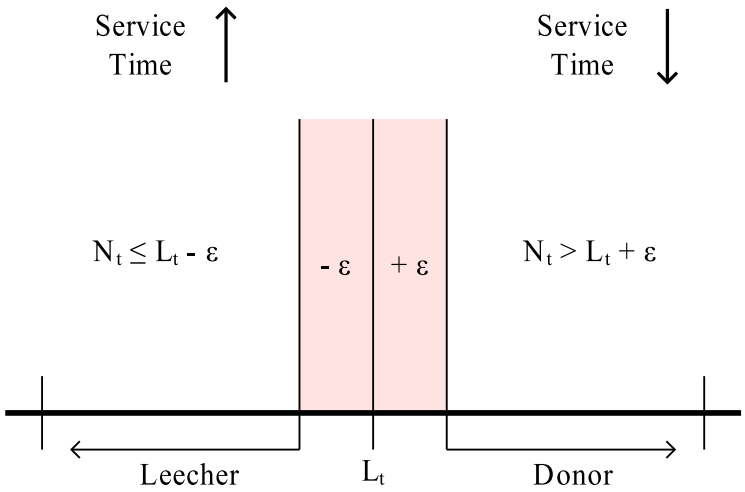

Figure 5.8: Behavior classification for service time modification.

a leecher or donor. The parameter ϵ is the bias and extends the range for L_t. If ϵ is big, then the economic model will hardly set a new service time and the home will be handled as neutral, i.e. $S = A$. The model will react slowly to changes of the sharing behavior of homes. On the other side, if ϵ is small each change of the sharing behavior of a home will be considered immediately. The aim of the economic model is to shorten the necessary service time homes must reside *active* with only small additional effort and therefore a tradeoff between fairness and wattage.

For example, for following calculations we assume $C_{min} = 2$, $C_{max} = 20$, $C_{avg} = 0.5$, $\epsilon = 0.5$, and $S_t = 28800$ seconds (8 hours).

Now a home accepted $A = 15$ tasks by remote homes and sent $S = 5$ tasks to other homes in the last period. According to (5.11) $C_t = 10$; the home was a donor in the last period and it depends on its behavior of previous periods, more precise on C_{min} and C_{max}, how the service time will be modified for the next period.

According to (5.15) $N_t \approx 0.44$ and to (5.18) $L_t \approx -0.11$. Since according to (5.20) $N_t > L_t + \epsilon$, the service time for the next period $S_{t+1} = 14400$ seconds; thus reduced by 50 % down to 4 hours.

Otherwise, if $A = 5$ and $S = 15$, then $C_t = -10$ and the home was a leecher. According to (5.15) $N_t \approx -0.67$ and again $L_t \approx -0.11$. Now, according to (5.20) $N_t < L_t + \epsilon$ and the service time for the next period $S_{t+1} = 43200$ seconds; thus increased by 50 % up to 12 hours.

But there is also an indifferent case where the service time will not be altered with e.g. $A = 10$ and $S = 10$. In this case the home was neither a donor nor a leecher in the last period with $C_t = 0$. $N_t = L_t \approx -0.11$ and $L_t - \epsilon < N_t < L_t + \epsilon$ which yields according to (5.20) $S_{t+1} = S_t = 28800$ seconds; thus again 8 hours.

The economic model presented in this section is implemented in the simulation and some experiments are discussed in Chapter 6.

6 Evaluation

This chapter presents results of simulation runs for the applications introduced in Chapter 4, produced with the simulation framework introduced in the simulation Chapter 5. As variance reduction techniques *Independent Replications* and *Steady-state* simulation are applied [Whi91]. Simulation results are tested if they are below a relative statistical error of 0.05. To avoid correlated observations, random numbers are generated based on the parallel version of the *Linear-Feedback Shift-Registers generator* (Tausworthe), which is called *Twisted Generalized Feedback Shift-Register* or simply *Mersenne Twister*[1]. This generator provides a sufficient large cycle of random numbers and is commonly used to produce uncorrelated random numbers.

For the network part of the simulation model, following values are default as shown in Table 6.1. A third of homes are near together (*vicinityProp*), thus have low communication cost, where no home is situated at the same place (*posAdjust*). The communication cost consists of an experienced base delay (*delayBase*) of 10 ms plus a distance-depended delay (*delayDist*) between two homes. The latency outside the home (*latRemote*) is set to 20 ms and inside the home (*latLocal*) to 6 ms. A fixed message size (*msgSize*) of 1000 bytes is assumed for all control messages. These values are based on experienced data [YGM02, HHS08].

Parameter	**Unit**	**Value**
vicinityProp	Probability	0.3
posAdjust	Factor	1.0E-3
delayBase	ms	10
delayDist	ms	60
latRemote	ms	20
latLocal	ms	6
msgSize	bytes	60

Table 6.1: Simulation parameters for the network topology.

Homes share a certain amount of their access bandwidth, disk space and CPU time. Also homes have a maximal local bandwidth and a peak power consumption. Default values are shown in Table 6.2. As basic setup, all homes provide equal resources and share a synchronous access bandwidth of 50/50 MBit/s (*dnBw, upBw*), a local bandwidth of 100 MBit/s (*locBw*), at maximum 3 Ghz of CPU time (*cpu*), and 100 GB of disk space (*mem*). Each home has a peak power consumption of 100 (*watt*) if busy.

[1] `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`

Parameter	Unit	Value
dnBw	kbit/s	50000
upBw	kbit/s	50000
locBw	kbit/s	100000
mem	MB	100000
cpu	Mhz	3000
watt	watt	100

Table 6.2: Simulation parameters for homes.

The simulation experiments are focused on the feasibility of load distribution in the context of *future home environments*. Hence, compared with todays usual bandwidths (e.g. 8/4 MBit/s) of residential access, a very high synchronous down- and uplink access bandwidth is supposed (50/50 MBit/s). Disk space (*mem*) is not a bottleneck nowadays. The most influencing parameter, as we will see in next sections, is the shared commuting capacity (*cpu*) of homes.

6.1 Download Sharing (DS)

Since this application causes not much CPU usage, performance and wattage factors are fixed; thus homogeneous homes are considered. The basic scenario is explained in Section 5.1. The default parameter space, derived from Table 5.1, is shown in Table 6.3. For this default setup, a simulation run is for a network of 100 or 1000 homes

Parameter	Unit	Value
numHomes	Number	100 or 1000
numSuperHomes	Number	4 or 40
α	Factor	1
β	Factor	1
load (A)	Number	$1 \leq A \leq 35$
simPeriod	s	604800
serviceTime	s	28800
dsBwDn	kbit/s	2500
dsBwUp	kbit/s	200
dsCpu	Mhz	10
dsMem	MB	700

Table 6.3: Simulation parameters for *Download Sharing*.

(*numHomes*) segmented in 4 or 40 clusters, each of them served by one home acting as super home (*numSuperHomes*).

Homes create 1 to 35 DS-tasks (*load*) per week. Per default this value is fixed to $A = 5$. The parameter *simPeriod* is fixed to one week as mean interarrival time for new DS-tasks with different load levels. The service time (*serviceTime*) is fixed to 8 hours because it is assumed that *active* homes will accept new tasks for a period of 8 hours.

This is a fairness parameter. A higher service time causes more available resources but longer *active* times and in turn higher wattage. The economic model introduced in Section 5.3 alters this parameter based on past behavior of homes.

The next four parameters define a DS-task. The parameter *dsBwDn* is the maximum downlink bandwidth a DS-task may use. Since a home can carry out a couple of DS-tasks, it is reasonable to restrict the bandwidth for all tasks equally with e.g. 2500 kbit/s. Further, the downlink bandwidth reachable in common P2P file-sharing systems is the combined bandwidth from (many) different sources. Based on the real world of P2P file-sharing, the same is considered for the uplink bandwidth (*dsBwUp*) which is much lower with 200 kbit/s. Only very low CPU usage (*dsCpu*) is considered for each DS-task, since the main usage lies on the memory that must be allocated. A typically compressed video file for one movie has approximately 700 MB (*dsMem*).

To further refine the resource profile (including VM overhead) for a DS-task this parameter is Pareto distributed. Since file sizes on Internet servers can be supposed as Pareto distributed [BGdMT06], the size of the VM containing a completed DS-task is a random variate of the Pareto distribution with

$$F(x) = \begin{cases} 1 - \left(\frac{k}{x}\right)^{\alpha} & x \geq k,\ \alpha > 0,\ k > 0 \\ \\ 0 & x < k \end{cases} \tag{6.1}$$

where k is the mode or scale parameter, α the shape parameter of the function, and x the target value. Setting (6.1) equal to a decimal number u from $U(0, 1)$

$$u = 1 - \left(\frac{k}{x}\right)^{\alpha} \tag{6.2}$$

and inversing it to

$$F^{-1}(u) = x = \frac{k}{(1-u)^{\frac{1}{\alpha}}} \tag{6.3}$$

gives the Pareto distributed download size. The larger α, the closer are results for x to the desired minimal download size specified by *dsMem*. Default simulation runs are made with $\alpha = 3$.

The plot in Figure 6.1 compares, for default DS-scenario parameters of Table 6.3, the local versus the distributed case. The y-axis shows the mean number of busy homes (homes in either the state A, AB or ABC) and the x-axis the simulation time in days up to one year. What Figure 6.1 says is, if all homes execute their DS-tasks locally then on average approximately 2.15 homes are busy constantly throughout the year. On the other hand, if homes cooperate and tasks are executed in a distributed way, only approximately 1.1 homes must be busy. Thus, the half of resources is required; only 1.1 homes must be busy instead of 2.15 homes. Note, that not the same 1.1 or 2.15 homes do the work throughout the whole year. The load is distributed about 100 homes being in arbitrary states, but there are always sufficient homes in state A which

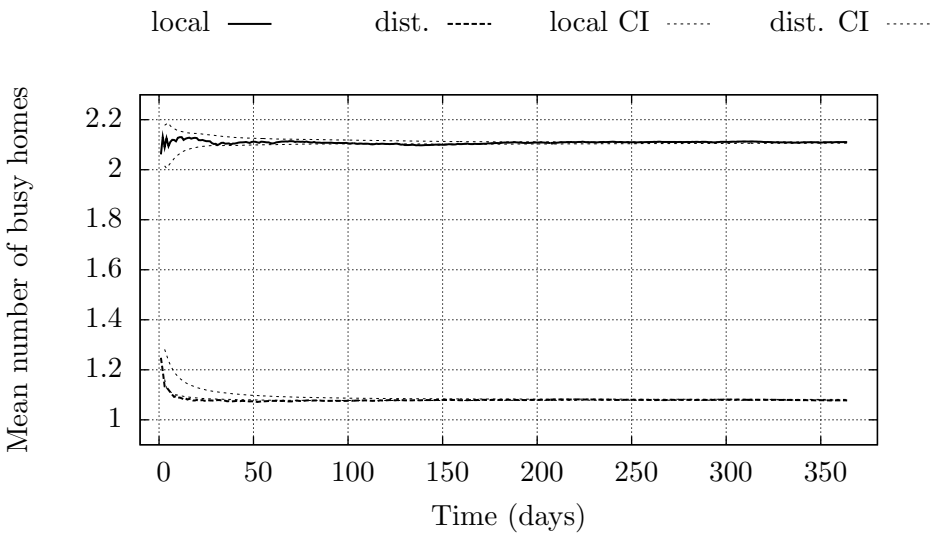

Figure 6.1: *Download Sharing*: Mean number of busy homes with 95 % confidence intervals (CI).

cope with the complete load as in the local case.

Energy is measured in *joule*. Power is energy divided by time and measured in *watt*. A watt is a joule per second. A computer with a power consumption of 100 watt uses 100 joule of energy every second. Multiplying a power unit (e.g. kilowatt) by a time unit (e.g. hours) gives an energy unit (kilowatt hours or shortly kWh).

The plot in Figure 6.2 shows, similar to the previous Figure 6.1, the difference between local and distributed case, but now in terms of energy (wattage) caused by homes for executing the same load. In the local case each home has a power consumption of

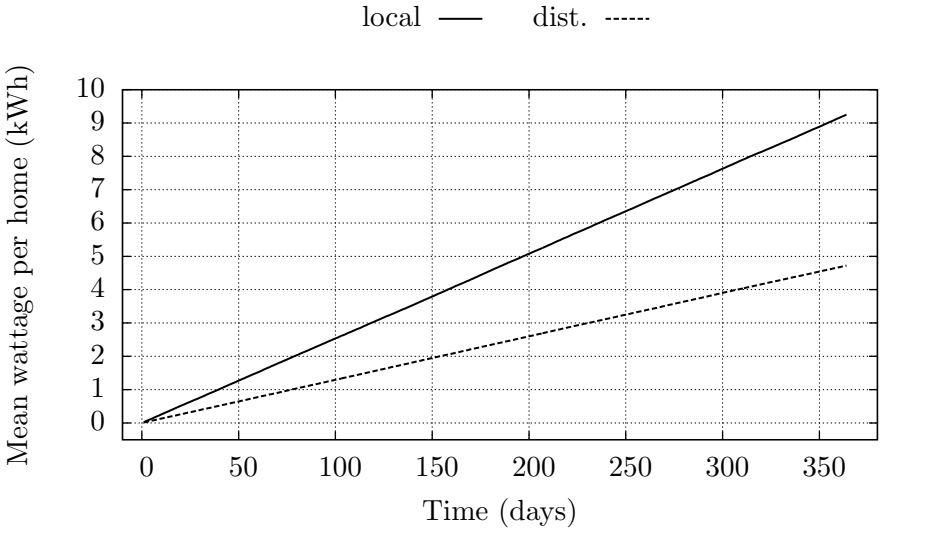

Figure 6.2: *Download Sharing*: Mean wattage per home.

approximately 9.3 kWh after one year of executing DS-tasks. Comparatively, the power

consumption only counts up to approximately 4.9 kWh in the distributed case. This is a local power saving of 4.4 kWh per home for this CPU non-intensive application. In sum, the global power saving is 440 kWh within one year for 100 homes.

As the power consumption can be correlated at most to CPU usage [FWB07a, BH07b], the wattage is measured as the sum of the idle and the busy wattage. The busy wattage is the difference between idle and busy peak wattage and is weighted by the CPU utilization, which depends on the number of running tasks. It is assumed that *active* but idle homes without tasks cause the half of the assumed peak wattage; thus a fixed amount of 50 watt. The remaining 50 watt are weighted with the CPU utilization which is variable for each home. We see that there is a correlation between the number of busy homes and the wattage.

The joint plot, shown in Figure 6.3, combines the plots of Figure 6.1 and Figure 6.2; the mean number of busy homes and the mean wattage per home, again for 100 homes. We see that the remote case clearly outperforms the local case in both, the

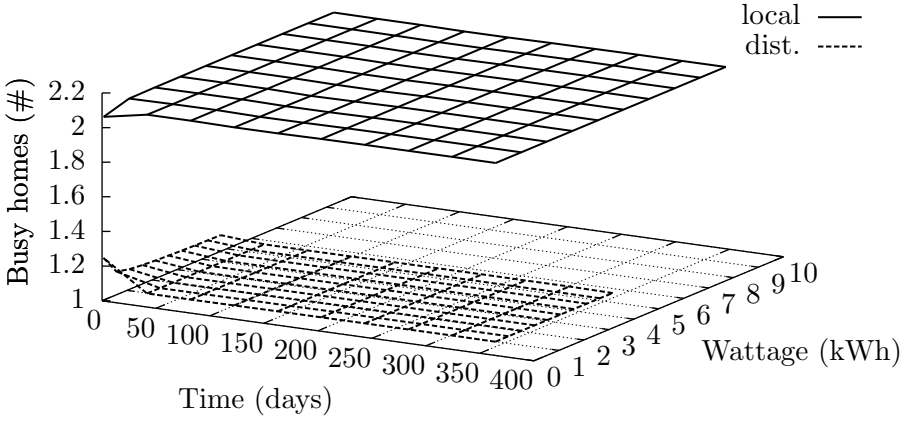

Figure 6.3: *Download Sharing*: Joint plot of mean busy homes and mean wattage per home.

mean number of busy homes and the mean wattage per home throughout the whole simulation period.

6.1.1 Distribution

A very interesting point is, that optimization toward energy efficiency can be achieved easier with a larger network of homes. The plot in Figure 6.4 compares a network of 1000 homes with the hitherto network size of 100 homes. Now we pay attention to the relative difference of the mean number of busy homes between local and distributed case regarding to the network size. For the local case it is clear that the mean number of busy homes must be much higher for a network of 1000 homes, than for a network of 100 homes; always about 2.1 % of homes are busy. But the relative difference of

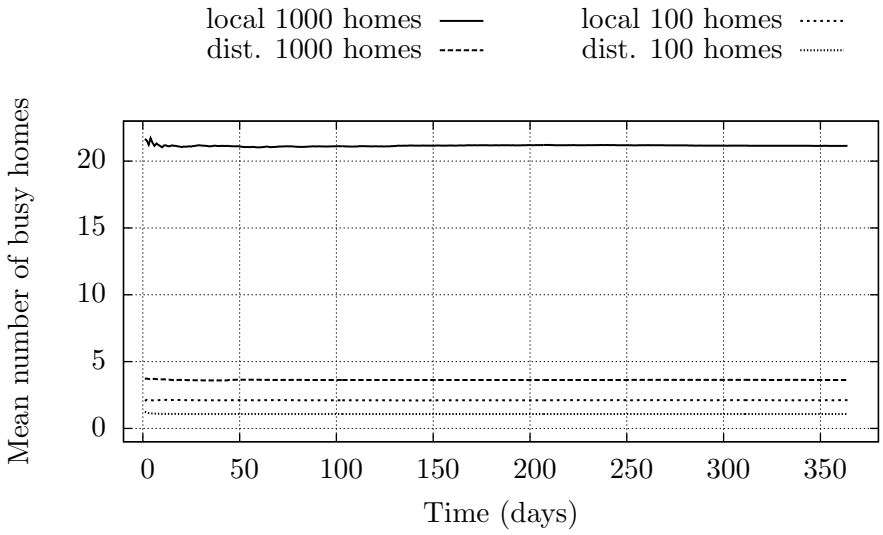

Figure 6.4: *Download Sharing*: Mean number of busy homes for 1000 and 100 homes.

the mean number of busy homes in the distributed case between a network of 1000 and 100 homes can be clearly seen. Along 100 homes around 1.1 % are busy, whereby along 1000 homes only 0.37 % have to be busy to cope with the same load of 5 DS-tasks per week. This absolutely postulates, if we rise the network size, also the degree of distribution is rised and this unleashes more power saving potential due to better distribution.

But this is only the effect on the mean number of busy homes. The network size has more influence on the mean power consumption (wattage) per home as shown in Figure 6.5. Firstly, we should agree that the mean wattage per home is the same, independent

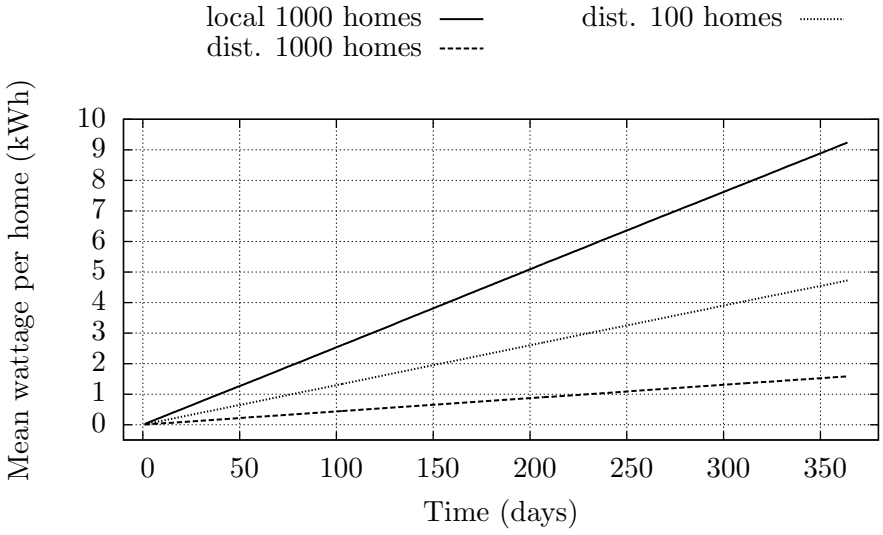

Figure 6.5: *Download Sharing*: Mean wattage per home for 1000 and 100 homes.

from the number of homes, which is verified by the top curve with a wattage of around

9.3 kWh after one year of local downloading for 1000 and 100 homes respectively. The effect of a bigger network size directly influences the mean wattage per home to a larger extend as the mean number of busy homes. We see, the distributed case among 100 homes amounts approximately to 4.8 kWh per year per home and approximately only to 1.6 kWh per year per home for a network of 1000 homes. The point is, that we have an improvement of local power saving by 3.2 kWh per home and an improvement of global power saving of 1120 kWh if the number of participating home is increased by factor 10. A 10 times bigger network of homes can save up to 34 % more power. Further, the big difference of the distributed case in comparison to the local case among 1000 homes, justifies the feasibility of this task sharing approach especially for bigger networks.

6.1.2 Load

All preceding plots were for a load of 5 DS-tasks per week. In the next plot 6.6 the load is altered from 1 to 35 DS-tasks per week. Figure 6.6 shows the mean number of busy

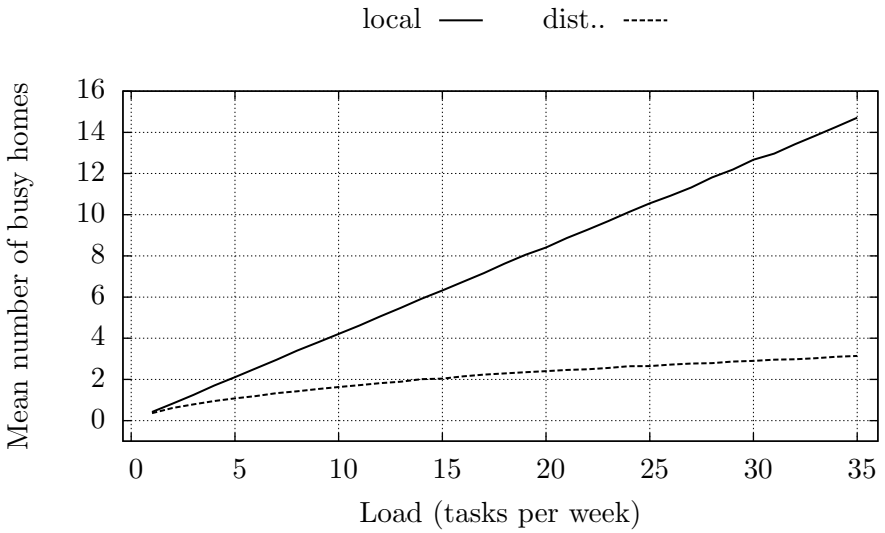

Figure 6.6: *Download Sharing*: Load plot for 1 to 35 DS-tasks per week.

homes for the setup of Table 6.3 and 100 homes. In the local case, where each home execute its task self, nearly 14.7 homes are busy to cope with the load of 35 DS-tasks per week; in the distributed case only approximately 3.14 homes are required. Thus, similar to the effect of a lager network size, with rising load the difference between local and distributed case grows considerably.

The relation between local and distributed case can also be seen in terms of wattage as shown in Figure 6.7. In the local case with increasing load, the wattage per home increases linearly. In the distributed case the increase in wattage is logarithmic. For the load of 35 DS-tasks per week, each home can achieve a local power saving of 50.9 kWh per year. The global power saving of the whole network of homes would be approximately 5090 kWh per year. This are nearly 78.6 % less wattage for the load

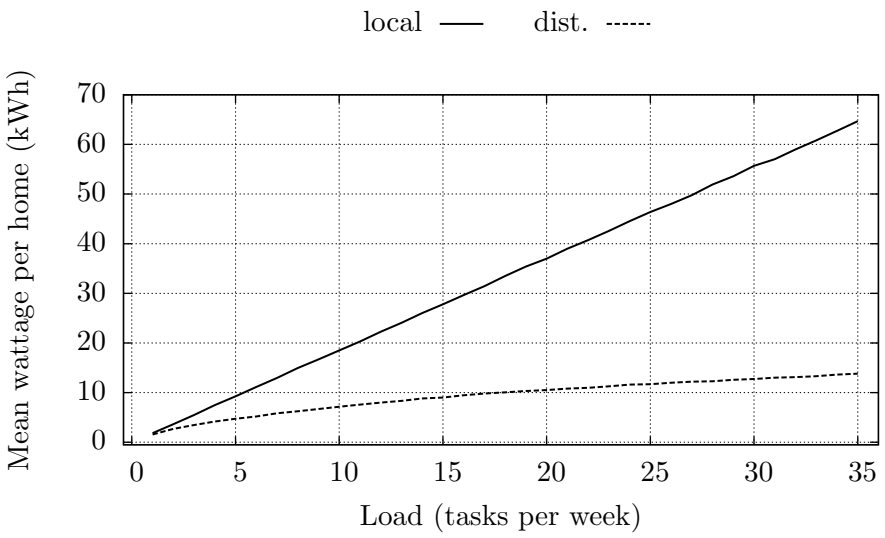

Figure 6.7: *Download Sharing*: Wattage plot for 1 to 35 DS-tasks per week.

caused by on average 35 DS-tasks per home per week throughout one year.

6.1.3 Fairness

Another aspect is fairness. Yet homes stay *active* for a fixed period of 8 hours defined by the service time (*ServiceTime*) mentioned in Table 6.3. The economic model, introduced in Section 5.3, affects the service time based on past behavior of homes.

Figure 6.8 shows the mean number of busy homes under fairness, i.e. with enabled economic model for the distributed case. To see the effect of the economic model,

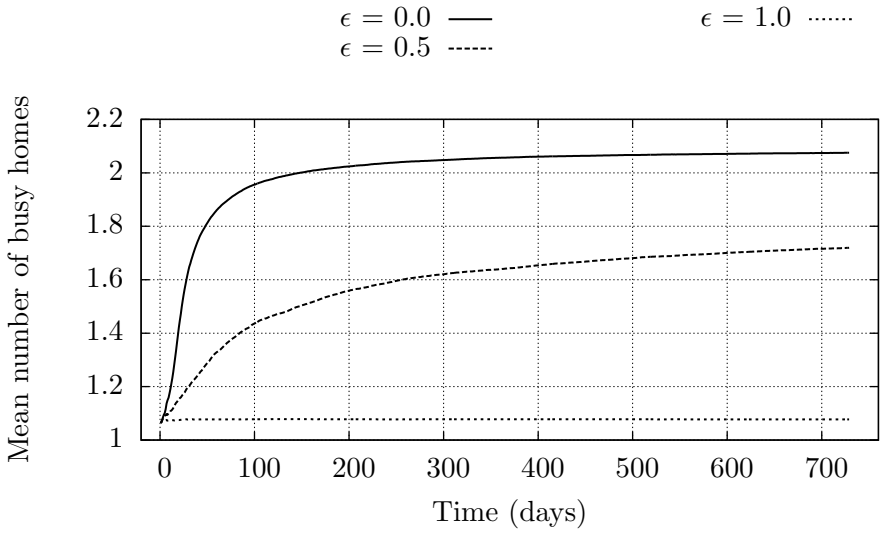

Figure 6.8: *Download Sharing*: Mean number of busy homes with fairness.

the fairness parameter $0 \leq \epsilon \leq 1$ is varied. With $\epsilon = 0$ each change in the sharing behavior of the home immediately affects the service time for the next *active* period.

This means that homes that have been donors for long time and once become a leecher, will immediately be penalized with a higher service time for the next *active* period.

With $\epsilon = 1$ the economic model has no effect and the outcome is equal to no fairness control; thus whatever behavior the home had, it will never be penalized or rewarded. Values for ϵ in between are a gradation of the sensibility of the economic model in terms of sharing behavior.

We see, if we immediately enforce fairness ($\epsilon = 0.0$), the system requirements more *active* homes, because under fair resource sharing (positive sent and accepted tasks balance), homes are rewarded by shorter service times. For coping with the same amount of load, this in turn requires more *active* homes.

A good tradeoff between fairness and the mean number of busy homes is $\epsilon = 0.5$ because with this value the service time modification accompanying with 19 % less busy homes as with a full sensible economic model ($\epsilon = 0.0$).

In Figure 6.9 we see the mean service time per home under fairness, i.e. with enabled economic model for the distributed case. With $\epsilon = 1.0$ we have the case without fairness

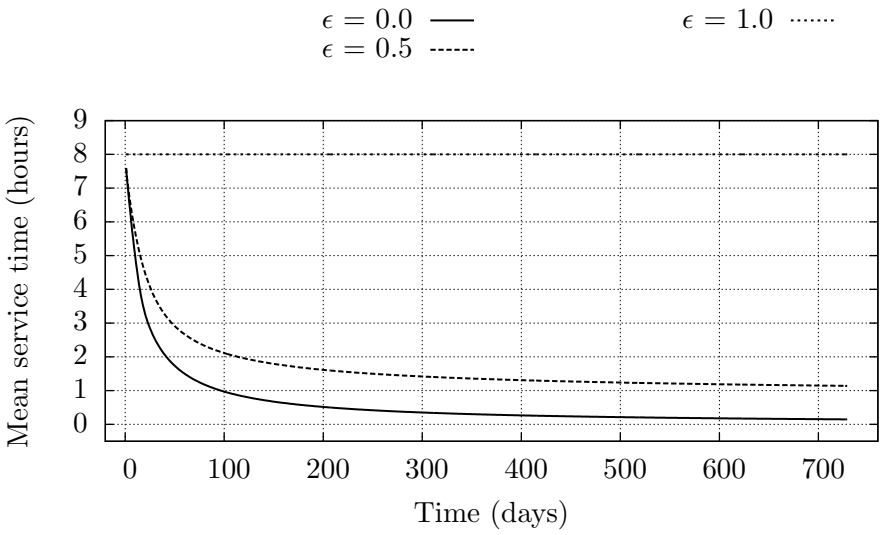

Figure 6.9: *Download Sharing*: Mean service time per home with fairness.

as simulated before; since the service time is initially 8 hours, it is the same forever. With $\epsilon = 0.5$ the service time can be shorten from 8 hours by 86.3 % down to 1.1 hours which results only in a neglectable increase in the mean number of busy homes as we have seen in Figure 6.8 compared to the saving here. With $\epsilon = 0.0$ it is only 8.9 minutes which represends an extremly dynamic sharing behavoir. Note, that despite of a very short service time of 8.9 minutes, this does not mean that homes reside *active* for only 8.9 minutes. During this 8.9 minutes homes accept new tasks and then block new ones, whatever time it takes to work out that accepted tasks.

We can see the small impact of shorter service times in terms of wattage in Figure 6.10. Without fairness ($\epsilon = 1.0$) each home has with 9.5 kWh the lowest wattage after

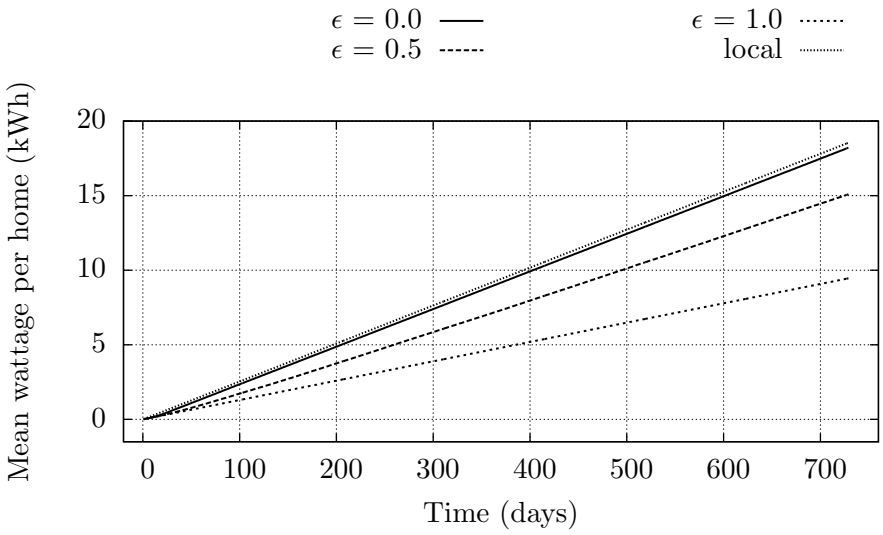

Figure 6.10: *Download Sharing*: Mean wattage per home with fairness.

two years. If we consider the previously as best working value of $\epsilon = 0.5$ then the wattage per home steps up by 58.9 % to 15.1 kWh. As also shown, the wattage of the local case is just above the wattage of the distributed case with best fairness ($\epsilon = 0.0$) and this means that fairness is naturally limited in terms of the past behavior of participating homes. If we allow long service times, then we can achieve a considerably power saving, but if we want maximal fairness, the power consumption will be almost the same as in the local case. Only in time the gap between wattages of the distributed case with extreme fairness and the local case slightly increases.

To summarize, if the service time is shortened by 86.3 % this costs the home 58.9 % more power which is a tradeoff between fairness and wattage. But as anticipated, fairness has a cost, namely power consumption.

6.2 Video Encoding (VE)

First insights in this application were given in Chapter 4. The goal is to encode a video. A VE-task including the source video, created by the owner (home), is sent to another executer (home), which in turn encodes and compresses the video. The new video is then sent back to the owner. This application strongly depends on the performance factor α and the wattage factor β, already introduced in Chapter 4. As for the evaluation of *Download Sharing*, the basic scenario for *Video Encoding* is explained in Section 5.1.

Simulation parameters for the default VE-Scenario are listed in Table 6.4. As explained in Section 6.1, the default setup for a VE simulation run consists of a network of 100 homes (*numHomes*) segmented into 4 clusters by super homes (*numSuperHomes*). Performance (α) and wattage (β) factors are set to 0.6 indicating that the half of homes share 40 percent less CPU time and therefore consume 40 percent less power for the

Parameter	Unit	Value
numHomes	Number	100 or 1000
numSuperHomes	Number	4 or 40
α	Factor	0.6
β	Factor	0.6
load (A)	Number	$1 \leq A \leq 35$
simPeriod	s	604800
serviceTime	s	28800
veSrcLength	min	100
veSrcVidRate	kbit/s	5000
veSrcAudRate	kbit/s	448
veTarVidRate	kbit/s	1000
veTarAudRate	kbit/s	128

Table 6.4: Simulation parameters for the default *Video Encoding* Scenario.

same utilization. Thus, we have fast and slow homes.

Also the load (*load*) varies in the range from 1 up to 35 tasks per week per home with a mean interarrival time (*simPeriod*) of 1 week. The default service time (*serviceTime*) is again 8 hours.

The next five parameters describe a VE-task. The idea is to convert a video of e.g. 100 minutes (*veSrcLength*) in DVD format to one in AVI format. You can easily do this with the DivX[2] or Xvid[3] codec manually. The source video has a video bitrate (*veSrcVidRate*) of approximately 5000 kbit/s and an audio bitrate (*veSrcAudRate*) of 488 kbit/s. The encoded target video has a video bitrate (*veTarVidRate*) of 1000 kbit/s and an audio bitrate (*veTarAudRate*) of 128 kbit/s. Hence, the filesize of the target video is considerably smaller than the filesize of the source video. For example, 100 minutes of a video in DVD format requires 4086 MB disk space; the encoded video in AVI format requires with almost 846 MB only 20.7 % of the size of the source video.

Because the source video is sent to the executer and the encoded target video back to the owner, at least 4932 MB of data has to transferred between these homes. In between, the encoding work must be done by the executer. This application produces load in the sense of power tasks, because serious amounts of network bandwidth, CPU time and disk space are required. Nevertheless, following simulation runs will show the benefit by doing such VE-tasks remotely.

The plot 6.11 outlines the difference of the mean number of busy homes within one year to cope with the given load of 5 VE-tasks per week per home. It can be clearly seen that in the local case approximately 13 homes must be busy to work out the load, whereas in the distributed case only 8.1 homes are necessary. This is a reduction of 4.9 homes and causes therefore a saving of 37.7 % of busy homes in the distributed case.

[2] `http://www.divx.com`
[3] `http://www.xvid.org`

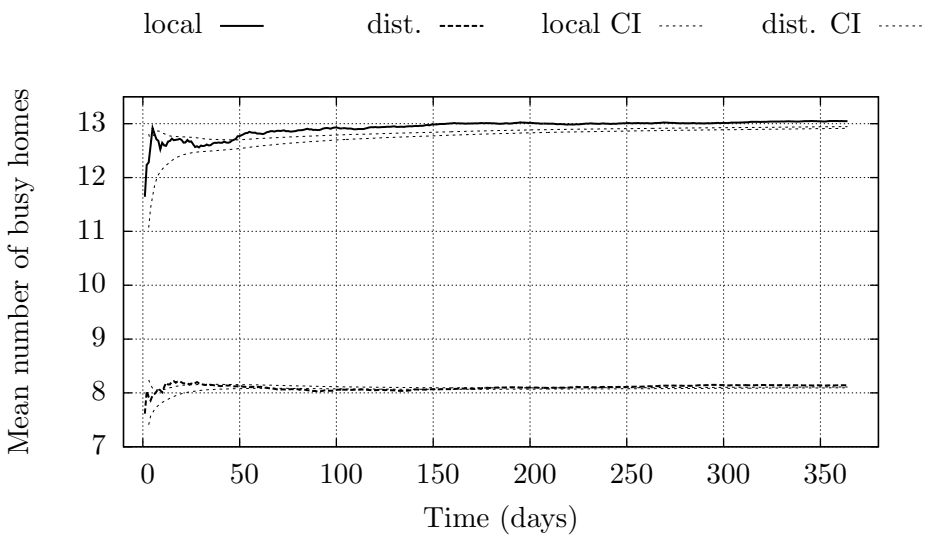

Figure 6.11: *Video Encoding*: Mean number of busy homes with 95 % confidence intervals (CI).

The improvement can also be seen in terms of wattage in Figure 6.12, because the mean wattage per home within one year is lower in the distributed case as in the local case. The local case causes for each home a wattage of about 54.7 kWh after one year.

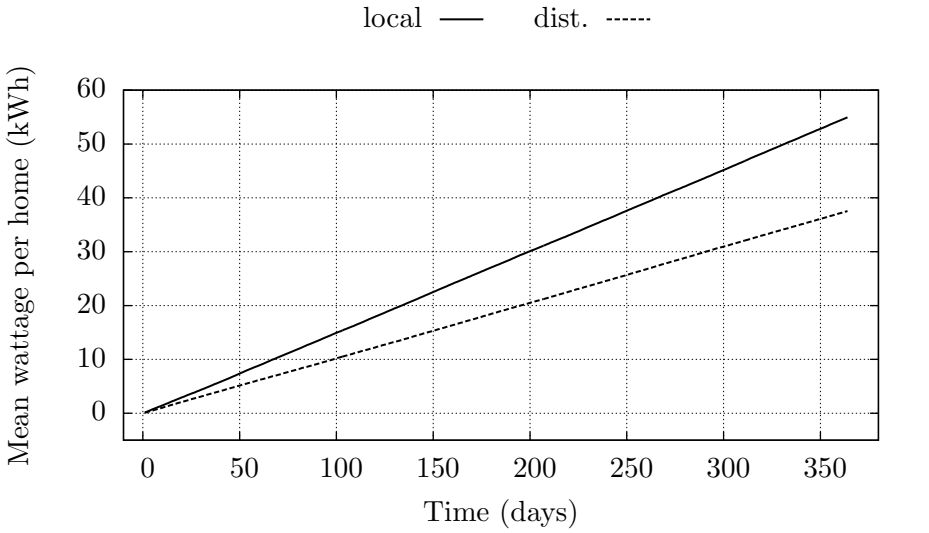

Figure 6.12: *Video Encoding*: Mean wattage per home.

In the distributed case each home causes only about 37.5 kWh on average and this is equivalent to a reduction of 31.4 % or a power saving of 17.2 kWh per home. The relative better reduction of mean busy homes, compared to the reduction of the mean wattage per home, comes from the fact, that *Video Encoding* is strongly related to CPU usage and therefore has direct impact on the power consumption of homes that actually do the work. Because we have a performance factor $\alpha = 0.6$, most executers share 40 % more CPU time as corresponding owners and this homes cause more wattage.

Thus, the smaller part of busy executers tends to cause more wattage relative to the same amount of owners. This is based on the assumption, that *Video Encoding* can only work in heterogeneous networks where certain members provide the calculation power that the other part does not have. In such cases the load will aggregate on homes providing the requested resources and this causes the main wattage.

This of course leads to a skew toward resource provider and should be compensated by fairness models similar to the one introduced in Section 6.1.3. Besides, bringing up the past behavior of members, also a mix of light and strong tasks in terms of resource requirements is thinkable. Homes that offer much CPU time do the calculation work, whereas homes with less CPU power must do downloading or storing data. This issue is addressed later in Section 6.4.

We can see, the coherence of the mean number of busy homes and the mean wattage per home at a glance in Figure 6.13. In the same period, local *Video Encoding* causes

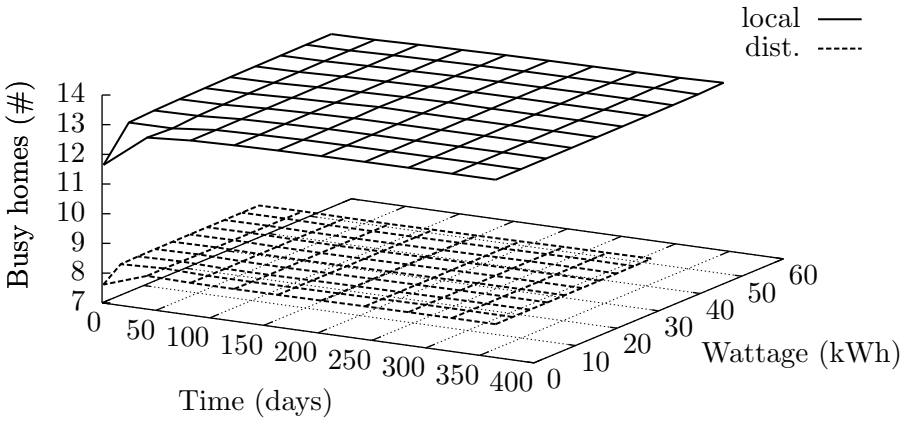

Figure 6.13: *Video Encoding*: Joint plot of busy homes and wattage.

more than twofold. First, more homes must be busy to cope with the given load of VE-tasks and second a bit more than a third more wattage is caused by each busy home. This can be clearly seen by the two planes in Figure 6.13. Generally, *Video Encoding* exhibits more power saving potential as *Download Sharing* but requires heterogeneous homes in terms of shared CPU time.

6.2.1 Distribution

As *Download Sharing* also *Video Encoding* performs better with a higher number of participating homes as seen in Figure 6.14. In the local case approximately 13 % of 100 or 1000 homes are busy. On the contrary, in the distributed case only appoximately 8.1 % of 100 homes or 7 % of 1000 homes are busy to cope with the same load of 5 VE-tasks per week per home. For a network of 100 homes the mean number of busy

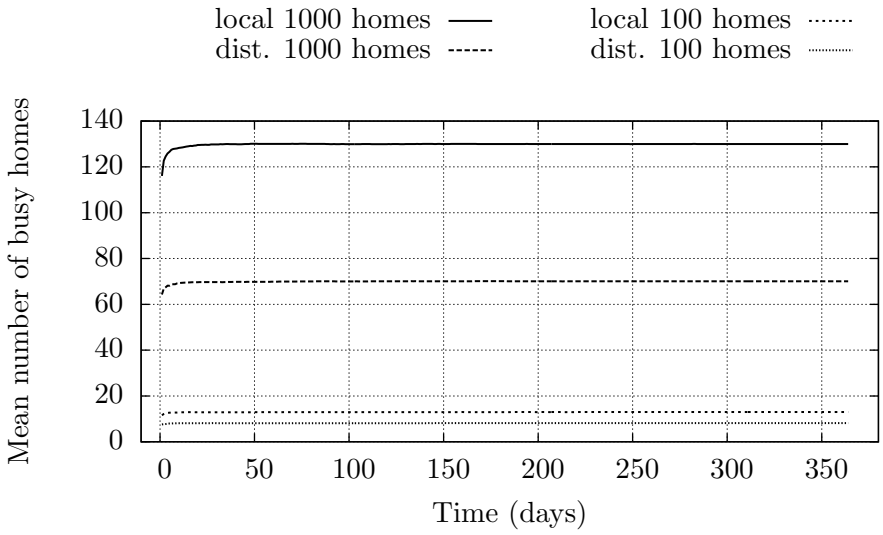

Figure 6.14: *Video Encoding*: Mean number of busy homes for 1000 and 100 homes.

homes in the distributed case is 37.7 % smaller than for the local case. On the other side, for a network of 1000 homes is the mean number of busy homes 46.2 % smaller than for the local case. Thus, if the network is bigger, then task sharing gains additonal advantage; The possibility to find a home willing to take over a task is bigger if more homes participate.

Also the application *Video Encoding* unveils power saving potential and the difference in wattage can be seen in Figure 6.15. Generally, the mean wattage per home with a

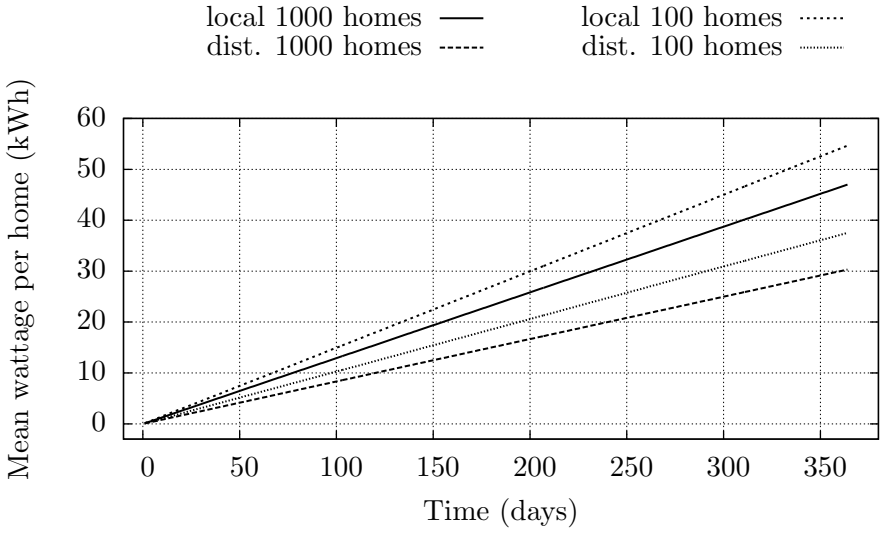

Figure 6.15: *Video Encoding*: Mean wattage per home for 1000 and 100 homes.

network of 100 homes lies above the corresponding case with a network of 1000 homes. The rule is also here: better distribution brings lower wattage. In the distributed case one home of 1000 with 30.3 kWh causes by 35.5 % less wattage as one home of 1000 in

the local case with 47 kWh. On the contrary, one home of 100 causes in the distributed case with 37.5 kWh only 31.4 % less wattage as one home of 100 in the local case with 54.7 kWh. The big network outperformes the small network with 4.1 % less wattage which shows the considerable additional power saving potential due to distribtion.

6.2.2 Load

Figure 6.16 depicts the mean number of busy homes of 100 from 1 up to 35 VE-tasks per week per home for one year. Under increasing load the distributed case works

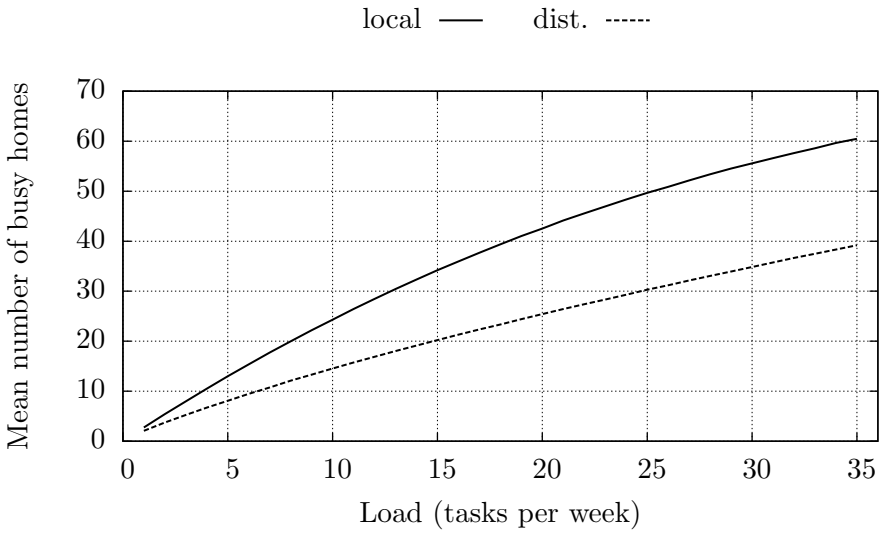

Figure 6.16: *Video Encoding*: Load plot for 1 to 35 VE-tasks per week.

better than the local one. At 35 VE-tasks per week per home saves the distributed case with 39.2 busy homes 35.2 % of resources compared to the local case where on average 60.5 busy homes are needed to cope with the same load.

The relation between the local and distributed case can also be seen in terms of wattage shown in Figure 6.17. For a load of 35 DS-tasks per week each home can achieve a local power saving of 30 % or nearly 98.3 kWh per year in the distributed case compared to the local case.

6.2.3 Fairness

As in *Download Sharing*, also here exists a tradeoff between fairness and wattage. The number of homes is 100 and Figure 6.18 shows results for the economic model and varied fairness parameter ϵ as introduced in Section 5.3 and with 5 VE-tasks per week per home.

The case without fairness is represented by $\epsilon = 1.0$ where 8.1 busy homes are necessary to cope with the load with a constant service time of 8 hours. If we choose $\epsilon = 0.5$ then the mean number of busy homes steps up to 11.5 busy homes but the service time can be shorten from 8 to 4.7 hours as shown in Figure 6.19. This means that more fairness causes after two years 42 % more busy homes but also 41.3 % less

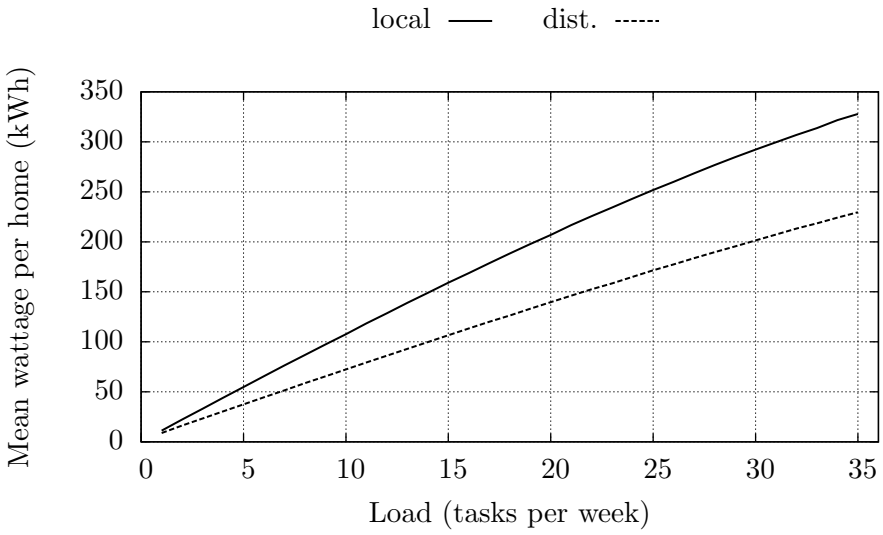

Figure 6.17: *Video Encoding*: Wattage plot for 1 to 35 DS-tasks per week.

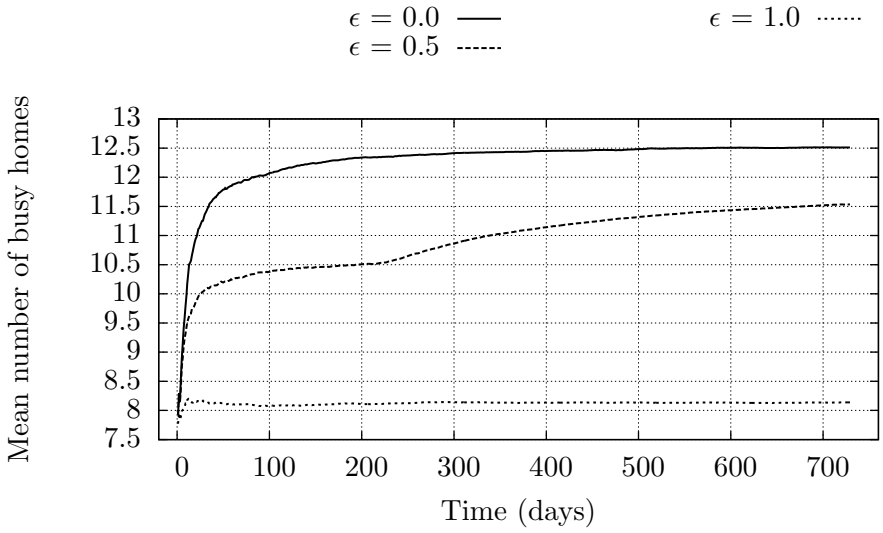

Figure 6.18: *Video Encoding*: Mean number of busy homes with fairness.

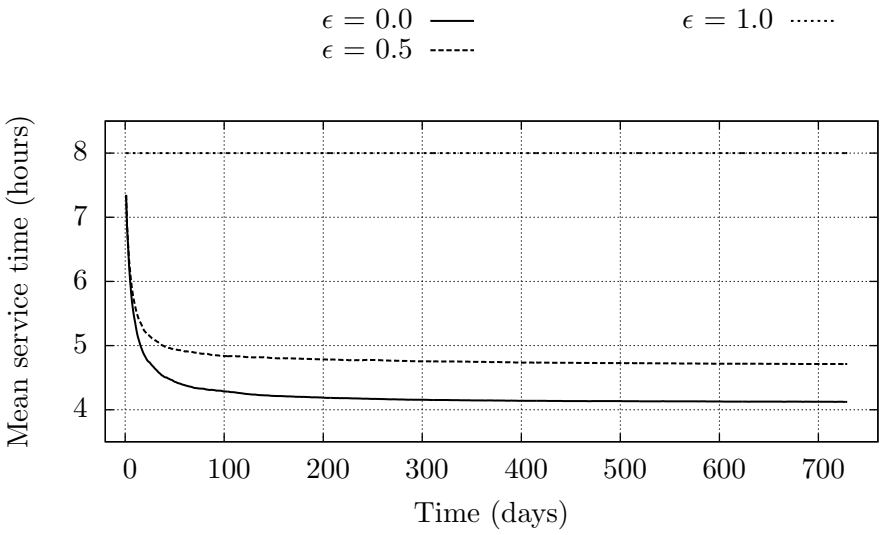

Figure 6.19: *Video Encoding*: Mean service time per home with fairness.

service time which is a good tradeoff in terms of fairness. Note, if we set $\epsilon = 0.0$, thus making the economic model highly sensible and therefore react to the past behavior of homes immediately, then the service time can be further lowered to 4.1 hours but this results in 54.3 % more busy homes compared to the case with $\epsilon = 1.0$.

This is also expressed in terms of wattage shown in Figure 6.19. The power consump-

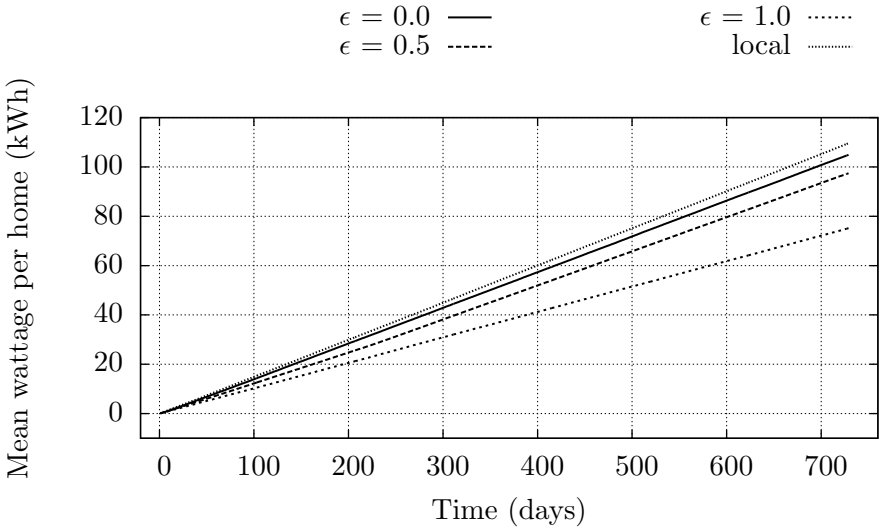

Figure 6.20: *Video Encoding*: Mean wattage per home with fairness.

tion for the distributed case with fairness ($\epsilon = 0.5$) is around 29.5 % higher as for the distributed case without fairness ($\epsilon = 1.0$). Although with $\epsilon = 0.0$ the wattage is 39.5 % higher as without fairness, the mean wattage per home is with 104.9 kWh after two years below the wattage of the local case with 109.6 kWh. Thus, with the application *Video Encoding* there is a natural limit to which extend fairness can be applied. With

the current type of economic model, based on the past behavior of homes, is it possible to gain shorter service times, i.e. more fairness, but always staying below the wattage of the local case.

6.3 Home Management (HM)

The *Home Management* application was introduced in Chapter 4 and simulation parameters in Chapter 5. Relevant simulation parameters, derived from Table 5.1 for the simulation of *Home Management*, are listed in Table 6.3.

Parameter	**Unit**	**Value**
numHomes	Number	100
numSuperHomes	Number	4 or 40
α	Factor	1
β	Factor	1
load (A)	Number	$1 \leq A \leq 7$
simPeriod	s	604800
serviceTime	s	28800
hmBwDn	kbit/s	500
hmBwUp	kbit/s	500
hmCpu	Mhz	300
hmMem	MB	500
hmRep (R)	Number	$0 \leq R \leq 6$
hmMeanFailures (F)	Number	$1 \leq F \leq 7$
hmHbeat	s	300
hmCheck	s	60
hmTimeout	s	600

Table 6.5: Simulation parameters for the default *Home Management* scenario.

The default setup consists of a network of 100 homes (*numHomes*) segmented into 4 clusters by a super home (*numSuperHomes*). Performance (α) and wattage (β) factors are set to 1 indicating homogeneous homes.

As load (*load*) there are 1 up to 35 HM-task arrivals per week; assumed with a mean interarrival time (*simPeriod*) of one week. The default service time (*serviceTime*) and interarrival time is set to 8 hours and 1 week.

The next four parameters define a HM-task. The mean downlink bandwidth (*hmBwDn*) and mean uplink bandwidth (*hmBwUp*) specify the bandwidth requirements of a HM-task. Assuming audio and video surveillance, a continuous AV-stream is considered on the downlink. Similar, for dissemination of results of analyzed data to many other homes, the same bandwidth requirement is considered uplink. The size (*hmMem*) of a HM-task is fixed to 500 MB, based on a survey on resource needs of home automation software. The number of replicates (*hmRep*) is the number of additional copies of one HM-task that must be hosted within the network of homes to improve

reliability. The number of mean failures (*hmMeanFailurs*) defines how many homes can crash in a given time period. Various settings for this parameters will be examined in Section 6.3.2 to observe the system under different fault hypothesis. Basic parameters for failure detection are the time between two heartbeats (*hmHbeat*) a home sends messages as proof to be operative, the time between to heartbeat checks (*hmCheck*) by other homes, and the timeout for a heartbeat (*hmTimeout*), i.e. after which time period a heartbeat is expired and the corresponding home considered as failed.

6.3.1 Replication

The fraction of busy homes B, with parameters defined in Table 6.5, can be calculated according to

$$B = (1 + R) \times \max\left\{\frac{hmCpu}{cpu}, \frac{hmMem}{mem}, \frac{hmBwDn}{bwDn}, \frac{hmBwUp}{bwUp}\right\} \tag{6.4}$$

and because of

$$\frac{hmCpu}{cpu} > \frac{hmBwDn}{bwDn} \geq \frac{hmBwUp}{bwUp} > \frac{hmMem}{mem} \tag{6.5}$$

the offered CPU time per home is the bottleneck which yields

$$B = (1 + R) \times \frac{hmCpu}{cpu} \tag{6.6}$$

Based on chosen values from Table 6.5 theoretical values for B are listed in Table 6.6.

Replication Degree R	**Fraction of busy homes B**
0	0.10
1	0.20
2	0.30
3	0.40
4	0.50

Table 6.6: Calculated fraction of busy homes.

The plot in Figure 6.21 shows simulation results for the number of busy homes (homes that are in either the states A, AB or ABC) for 100 homes. In the local case the expected number of busy homes converges quickly to the network size, because each home hosts its own HM-task without sharing. In the distributed case, but without replication ($R = 0$), all HM-tasks are aggregated on a much lower number of about 10 homes as calculated with (6.6). A higher replication degree denotes higher B. Finally, simulation results clearly show how the number of busy homes increases for

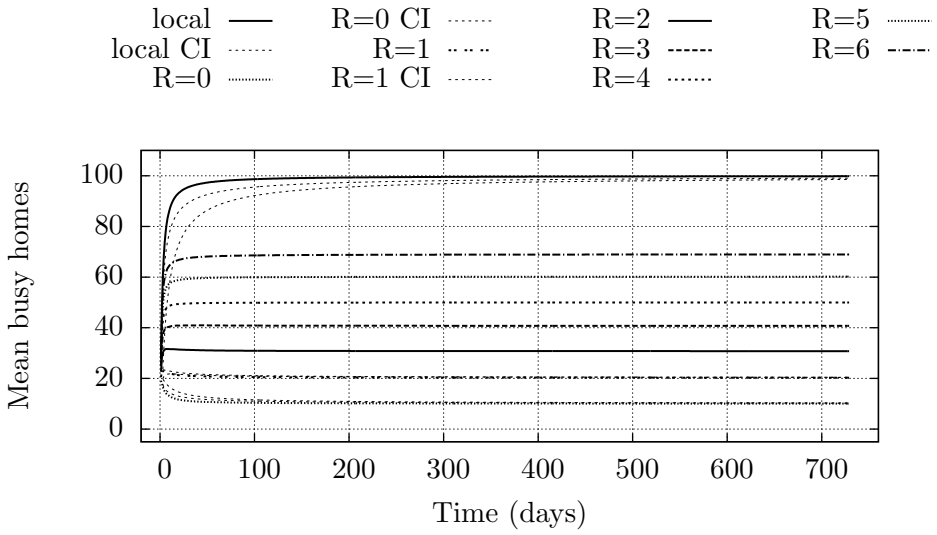

Figure 6.21: *Home Management*: Mean busy homes under various replication degrees with 95 % confidence intervals (CI).

each additional replicate ($0 \leq R \leq 6$) per 10 percent.

Figure 6.22 shows the relation between replication degree and wattage. A home

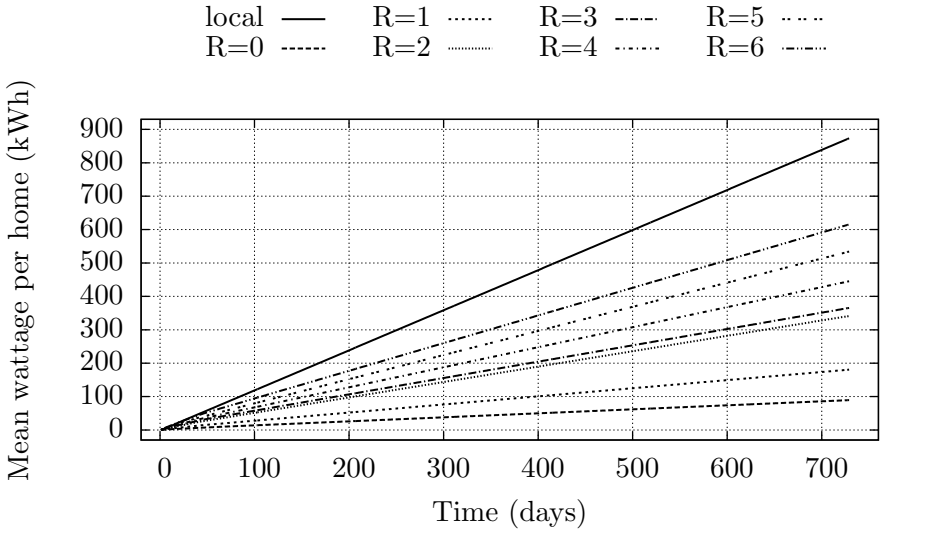

Figure 6.22: *Home Management*: Mean wattage per home under various replication degrees.

executing its HM-task is always running but not fully loaded, and this leads to a wattage of approximately 873 kWh for two years in the local case. For comparison, a fully loaded home with peak power consumption of 100 watt, as defined in Table 6.2, causes a wattage of approximately 1747.2 kWh for two years. Thus, the execution of one HM-task causes near the half wattage as the home would be under full load. For *Home Management*, the local case is a vast power wastage and it is better to run one home at a higher load and to avoid local task execution in other homes. The wattage

can be significantly lowered by distribution and even with a high replication degree ($R = 6$) it is clearly below that of the local case.

From Figure 6.22 results that after two years each can save without replication ($R = 0$) on average 89 % of the wattage compared to the local case. Even with high replication ($R = 6$) homes still save about 29 % of their wattage in comparison with the local case.

We see, more reliability and therefore higher replication degrees compensate the advantage of distribution to a certain extend. This leads directly to the next question how many replications are necessary to keep the system running, i.e. to assure that each home's HM-task is running continuously despite of failures under availability assumptions and makes this sense in terms of power saving? On the base of this observations availability under several failure hypothesis is examined in the next Section 6.3.2 implicating home failures.

6.3.2 Failures

For testing the behavior of the simulated network of homes, this section explains the extension of the *Home Management* scenario with home failures.

In the normal distributed case a home creates one HM-task. In the distributed case with replication the home creates some additional HM-tasks; the replications. One HM-task is executed by the creator immediately and the replications migrate to other homes. Therefore, some homes build a temporary cluster as depicted in Figure 6.23. As we see *Home 1* is member of two home clusters. In *Home Cluster 1* this home hosts its own master task *HM1* and as member in *Home Cluster 2* it hosts a replication of task *HM2*. In the case of a crashed *Home 1*, one instance of *HM1* and *HM2* is lost. Thus, both clusters are in error state and should compensate the crashed instance by finding another home willing to overtake.

The simulated procedure is as follows:

- A home fails according to a failure rate and all HM-tasks are lost.
- A failed home is set to state P and will come into state AB according to the arrival rate.
- A failed home will be detected by at least one other home of the cluster.
- Only the very first detection of a lost HM-task triggers the recovery process.
- The home that detected the failed HM-task creates a copy of its own HM-task and tries to migrate it to any other home in the network.
- The own HM-task of the home, that recovered the failed HM-task, is now the master.
- In the best case, next time the previously failed home becomes AB, it has nothing to do because the cluster repaired itself in the meantime.

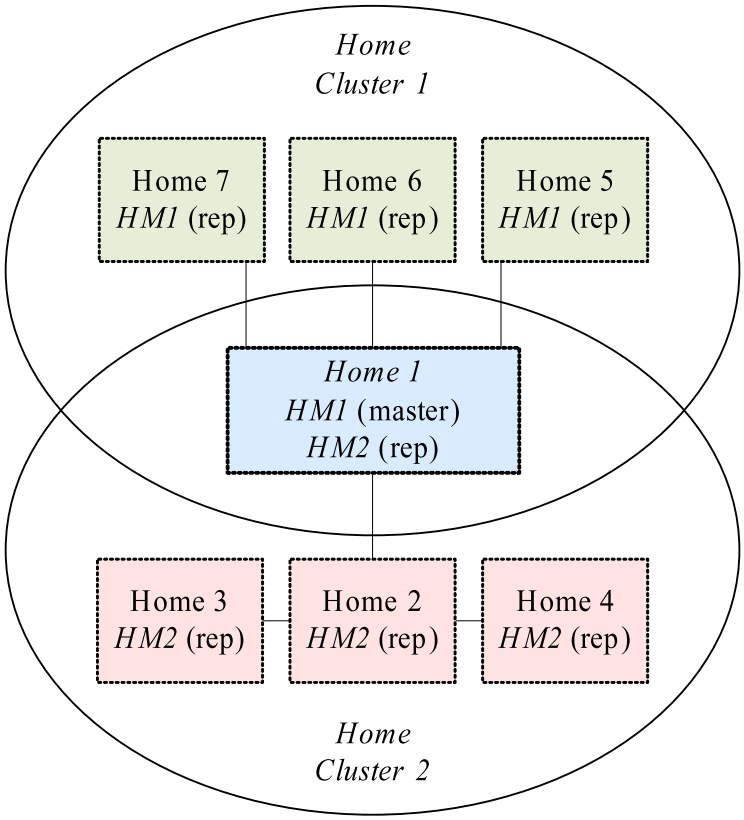

Figure 6.23: *Home Management*: Home cluster.

- In the worst case, the home cluster is still down and the home must recreate own HM-tasks up to the desired replication degree.

According to two reasons the occurrence of failures in all subsequent simulations are Weibull distributed. First, as investigated in [SR06] on three different P2P systems (Gnutella, BitTorrent, Kad) the Weibull distribution most adequately models churn in P2P systems, because churn consists not only of a sequence of completely independent events which normally leads to exponential distribution. Users' presence differs in terms of daytime and resource offer. Second, the possibility that a home fails depends more on the user as on the equipment. Then, failures in this context can be seen as arrivals and modeled similar.

To calculate a Weibull variate one must take the Weibull cumulative distribution function [Jai91]

$$F(x) = 1 - e^{\left(\frac{x}{\alpha}\right)^{\beta}} \quad x \geq 0 \tag{6.7}$$

where α is the scale and β the shape parameter of the distribution. Setting (6.7) equal to a decimal number u from $U(0,1)$

$$u = 1 - e^{\left(\frac{x}{\alpha}\right)^{\beta}} \tag{6.8}$$

and inversing it to

$$F^{-1}(u) = x = -\alpha \times ln(u)^{\frac{1}{\beta}} \tag{6.9}$$

gives the Weibull variate x for the failure interarrival time with α as mean. The Weibull distribution models lifetimes of components and for $\beta > 1$ it gives an increasing failure rate, i.e. failure interarrival times are short. Note that for $\beta = 1$ the failure rate is constant, but we use $\beta < 1$ for modeling a failure rate decreasing with time; the failure rate decreases monotonically and is convex. This models early failing homes, but more reliable homes on the longer run as it can happen for technical components which do not endure a certain load at the beginning and fail promptly, or continuously work error less under near constant load.

For now note the death process in the sense, that altough crashed HM-tasks will be restarted by other cluster members, crashed home clusters will be not recovered. The death process implies whenever a home fails, it will not recreate own HM-tasks if it comes online again and the cluster is down in the meantime; the system goes down over time as home failures occur.

Figure 6.24 shows cumulative numbers of home cluster crashes under different replication degrees with a mean failure rate of 1 failure per week ($F = 1$) and with a mean arrival rate of 5 arrivals per week ($A = 5$). That is, homes come online 5 times per week to once start the HM-task and immediately return to state P if so done before. The chosen replication degree should have a direct impact on the number of HM-tasks and indirectly also on the number of cluster crashes.

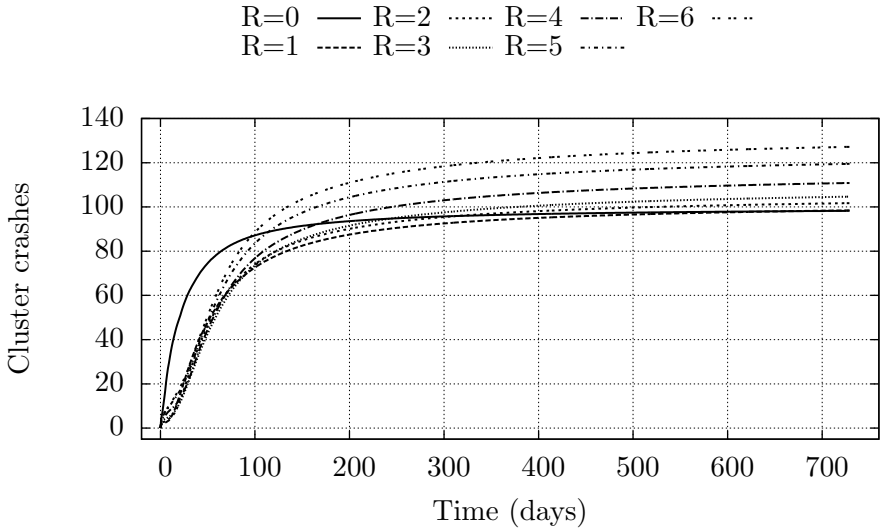

Figure 6.24: *Home Management*: Number of cluster crashes for 100 homes under various replication degrees.

Without replication ($R = 0$) each HM-task exists only once and therefore with a network of 100 homes also 100 pseudo clusters, each consisting of one home, are up. After 2 years nearly all clusters are crashed at least once in this scenario, which means that no home will be controlled by its HM-task.

For a specific replication degree in sum

$$M = (1 + R) \times numHomes \tag{6.10}$$

HM-tasks are running among homes; organized in so many clusters as homes in the network. With replication ($1 \leq R \leq 6$) clusters crash later as without replication ($R = 0$). In the simulated scenario homes monitor clusters' health. If one HM-task, because of a home failure, terminates, then other cluster members try to recover the lost HM-task.

Note, dependent on the replication degree, replication can revive crashed clusters before they must be reestablished by the corresponding homes. A recovered HM-task is initially marked as sleeping, because the recovering home already executes a replication of this HM-task. The recovering home will try to migrate the sleeping HM-task to a new home, yet not member of this cluster. Also homes with previously recovered and now sleeping HM-tasks can fail. This leads finally to a total system breakdown if all home clusters are crashed.

However, there is a possibility that HM-tasks of the same cluster crash in series which means that a cluster can be down for some period, but can be reestablished by some recovered still sleeping HM-tasks. Such sleeping HM-task becomes then operative. This is why home cluster can crash several times with replication and is shown in Figure 6.24 as a number of cluster crashes higher than the network size ($2 \leq R \leq 6$).

The slower rising number of cluster crashes is one outcome of replication in the *Home Management* scenario modeled in this work. The next viewpoint is the achievable availability expressed as the number of running clusters as shown in Figure 6.25.

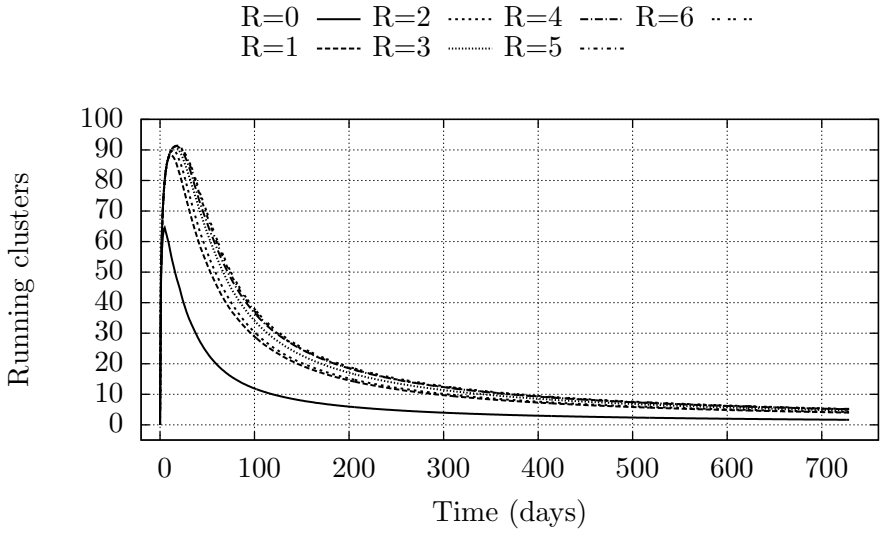

Figure 6.25: *Home Management*: Number of running clusters for 100 homes under various replication degrees.

At the beginning, homes create their HM-tasks and the availability (number of working clusters) increases, but because of chosen replication degree and failure rate never

reaches the maximum availability. The simulated network size is 100 homes and at least one HM-task for each home must run in order to fulfill the defined availability constraint. Without replication ($R = 0$) the system never reaches a fully operative state; although the peak of running clusters is 60.9 % after 8 days, the system goes down quickly and is minimzed to 1.6 % of homes after 2 years. This becomes better with replication, but only about 5.2 % of clusters can be kept operative with $R = 6$ because the slope asymptotically converges to a level clearly below the desired number of 100 running clusters.

Nevertheless, replication of course causes some extra power consumption that can be seen in the next Figure 6.26.

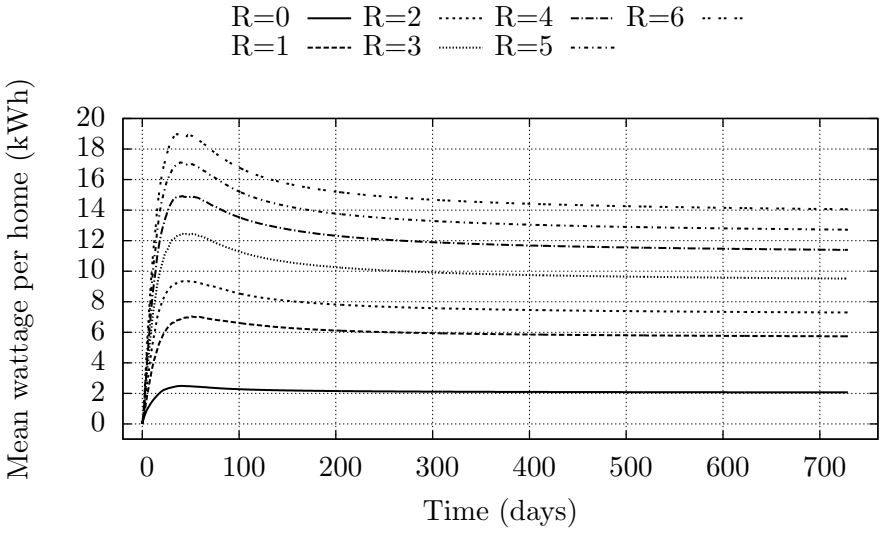

Figure 6.26: *Home Management*: Mean wattage per home for 100 homes with varying replication degrees.

Similar to the case without home failures, higher replication degrees cause higher wattage. With the most effective replication degree $R = 1$, the wattage is 2.8 times higher as with $R = 0$, but on the other hand, the availability is also always higher. The stepwise higher wattages of replication degrees $2 \leq R \leq 6$ are not justified compared to the reachable availability shown in Figure 6.25 above, because in the long term replication degrees $R > 1$ can not guarantee significantly better availability but cause linearly higher wattages.

Generally, because of the death process, wattages are much lower as in the case without failures and give only an idea about the wattages relatively to replication degrees. To become a deeper insight how much power can be saved under given replication degrees, the next Section 6.3.3 introduces the recovery of crashed clusters by respective cluster owners.

6.3.3 Cluster recovery

This chapter introduces a modification of the *Home Management* scenario. Now the cluster owner (the home that is controlled by the cluster) tries to reestablish the own cluster after a crash. A home being in state *AB* firstly checks if the own cluster is operative. It creates the required number of replications if the own cluster is not working, hosts one replication self, and tries to migrate the other replications. After some time it naturally returns to state P, because the own cluster is *active* among other homes; i.e. all HM-tasks could be migrated to other homes. Off course, recovered clusters can fail again which represents the continuous case where homes fail, clusters go down in series, and go up repaired by respective cluster owners.

Figure 6.27 shows cumulative numbers of home cluster crashes under different replication degrees and a mean failure rate of $F = 1$ failure per week and with a home's mean arrival rate of 5 arrivals per week ($A = 5$).

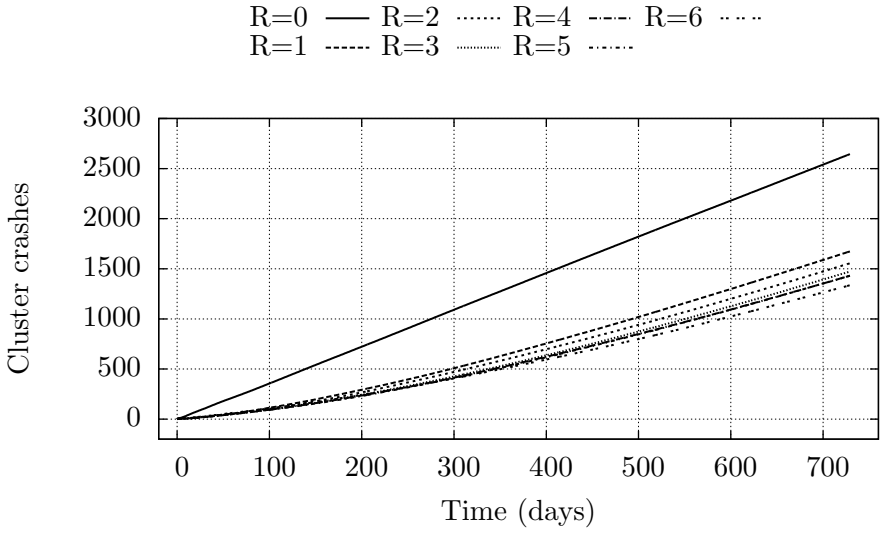

Figure 6.27: *Home Management*: Number of cluster crashes for 100 homes under various replication degrees with cluster recovering.

Note that the arrival rate can now be interpreted as repair rate. Whenever a home becomes busy, it checks if the own cluster is up and recovers (repairs) it on demand. This is the difference to the scenario explained in the previous section. Naturally, the cumulated number of cluster crashes is much higher, because clusters may crash several times during a period. Already a low replication degree $R = 1$ clearly outperforms no replication $R = 0$ with 1.58 times less cluster crashes after two years. High replication $R = 6$ avoids around 50 % of the cluster crashes compared to $R = 0$ after two years. Over this, higher repliation degrees ($2 \leq R \leq 6$) lowers the number of cluster crashes stepwise. We see the big difference between no replication and even the lowest possible replication degree $R = 1$ which postulates, that too high replication degrees do not gain a considerable advantage in terms of availability.

Figure 6.28 shows the number of running clusters under recovering and for two years. Most notable, in the long term replication has no impact on the availability here and

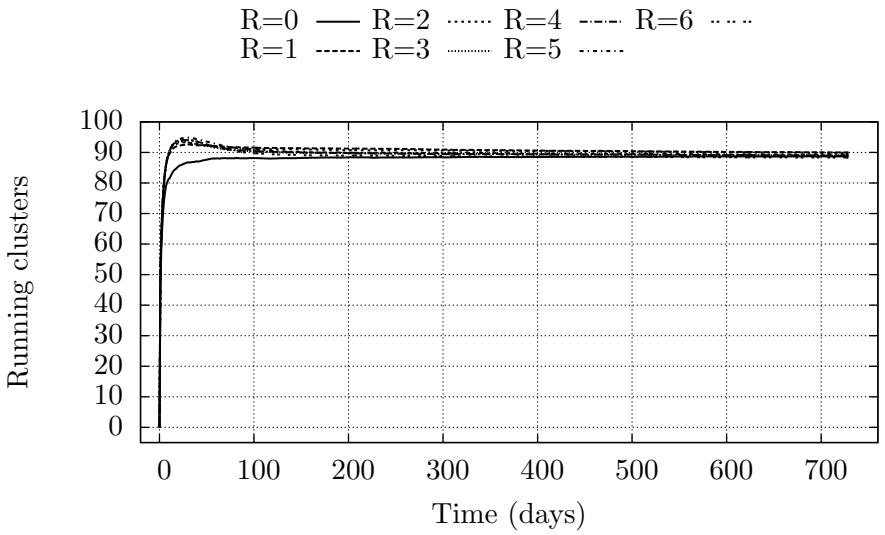

Figure 6.28: *Home Management*: Availability for 100 homes under various replication degrees with cluster recovering.

essentially depends on the relation between failure and repair rate. This lies on the fact that the repair rate of 5 arrivals per week per home ($A = 5$) is high enough to compensate the failure rate of one failure per week. However, because of the supposed failure rate the system can never reach the full operative state were all homes would be managed at the same time. If homes recover their clusters self, replication is only required in the transition phase to quickly rise the level of availability. In the steady-state phase homes recover clusters quicker, as the cluster can heal itself. We see the availability can be improved in the beginning by replication, but the question is how much this harm the overall optimization goal wattage.

The wattage can be seen in Figure 6.29 and clearly shows that more reliability through a higher replication degree implies more power consumption, but still outperforms the local case without replication with 79.3 % less wattage for $R = 6$ or anyhow 83.4 % less wattage for $R = 1$.

Note, that we cannot rise the number of replication boundlessly. The *Home Management* application modeled in this work is a task with considerable resource requirements. Only a limited number of concurrent HM-tasks can be executed by homes with typical resource offer. Big tasks can not be replicated so many times for assuring that no cluster crash will happen, where lightweight tasks may be replicated sufficient often to assure that for each cluster at least one HM-task is running. The *Home Management* scenario in this work is modeled to consolidate offered resources and extract the relative power consumption of local and distributed execution under a strict availability assumption.

But what influence have different repair and failure rates on availability and wattage?

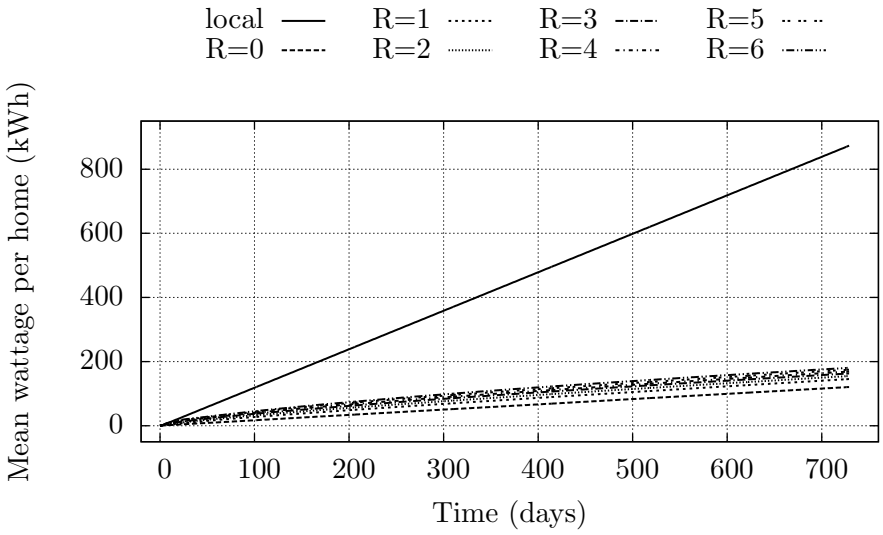

Figure 6.29: *Home Management*: Mean wattage per home for 100 homes under various replication degrees with cluster recovering.

For the next simulations, the replication degree is fixed with $R = 2$ whereas either the mean number of arrivals per week per home (A) or the number of mean failures per week per home (F) is altered. This allows to analyze the influence of the yet fixed parameters separately.

Figure 6.30 shows how different arrival rates influence the number of cluster crashes. Simulation time is 2 years, the replications degree fixed to $R = 2$, and the failure

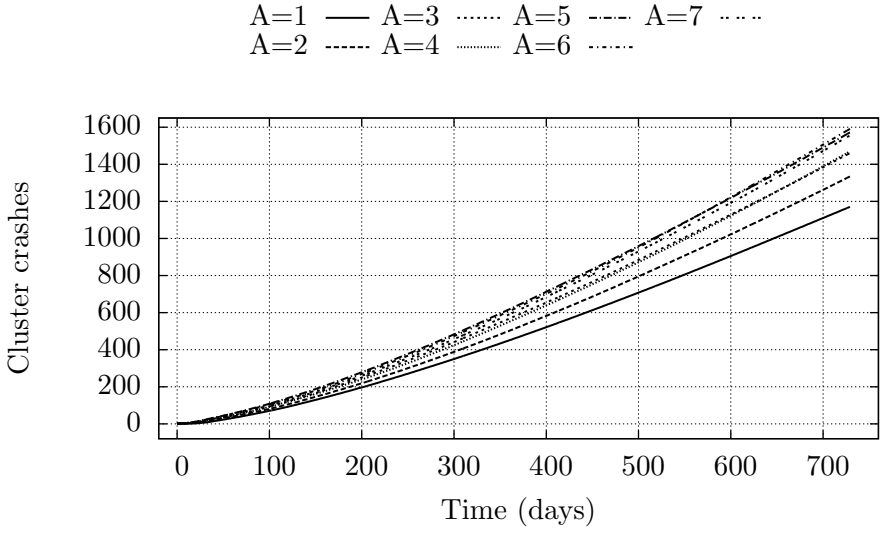

Figure 6.30: *Home Management*: Number of cluster crashes for 100 homes under various arrival rates with cluster recovering.

rate to $F = 1$ per week. The arrival (repair) rate is altered ($1 \leq A \leq 7$) and shows that more cluster crashes occur if homes become more often busy. This is because, according to cluster recovering, homes become earlier aware of crashed clusters and

recover them more quickly. If the time point of a cluster crash is T_1 and the time point of the recovery T_2, then the repair time

$$T_r = T_2 - T_1 \tag{6.11}$$

is smaller with $F = 7$ as with $F = 1$. We can think about an arrival rate of $A = 7$ how a user every day checks its home cluster and makes some adjustments. In the case of $A = 1$ it only checks the cluster once a week. For availability reasons higher arrival rates are better.

Note, that the owner home only has to recover the cluster if all replicates were crashed. In the first stage the cluster can heal itself; thus there is always one replicate *active* and the corresponding home is able to recover the cluster on-the-fly. In the second state the owner home recovers the whole cluster. For a highly available system, an owner home never should come into place to recover the own cluster fully, because this would imply that the home control was interrupted. But beware, the *Home Management* application is meant as setup central were users can configure home appliances etc. without the necessity of continuous communication on a dedicated line.

Figure 6.31 shows the number of running clusters under various arrival rates ($1 \leq A \leq 7$).

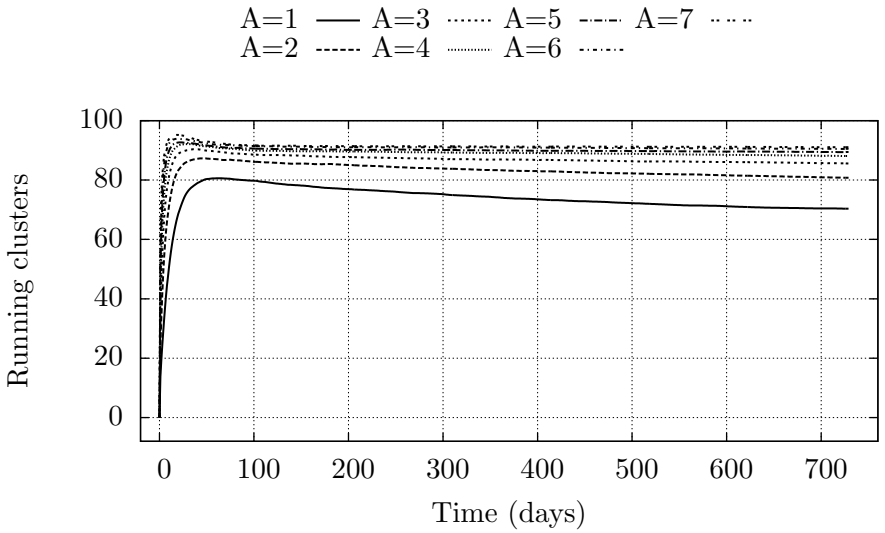

Figure 6.31: *Home Management*: Availability for 100 homes under various arrival rates with cluster recovering.

An arrival rate of $A = 1$ is not enough to ensure the availability of the system of at least 80 % while the number of running clusters is slowly decreasing. With $A = 7$ the availability is nearly stabilized at a level of 90 % and this means at least 90 % of all homes are managend continuously by their corresponding home clusters. One can recognize that the improvement of availability is smaller for higher arrival rates. The difference in terms of running cluster between $A = 6$ and $A = 7$ is much smaller than the difference between $A = 1$ and $A = 2$. The interesting point is, that the theoretically

highest arrival rate statistically enables a home to immediatly recover crashed clusters; thus the repair time T_r would be almost zero. But this is not a realistic scenario whereas the basic intention, to save power by letting homes go into a low power state P, would be violated by homes that come e.g. $A = 100$ times to state AB to push up the availbility.

The tradeoff between arrival rate and wattage can be seen in the next Figure 6.32. With $A = 1$ arrivals per week, the mean wattage per home counts up to 120 kWh

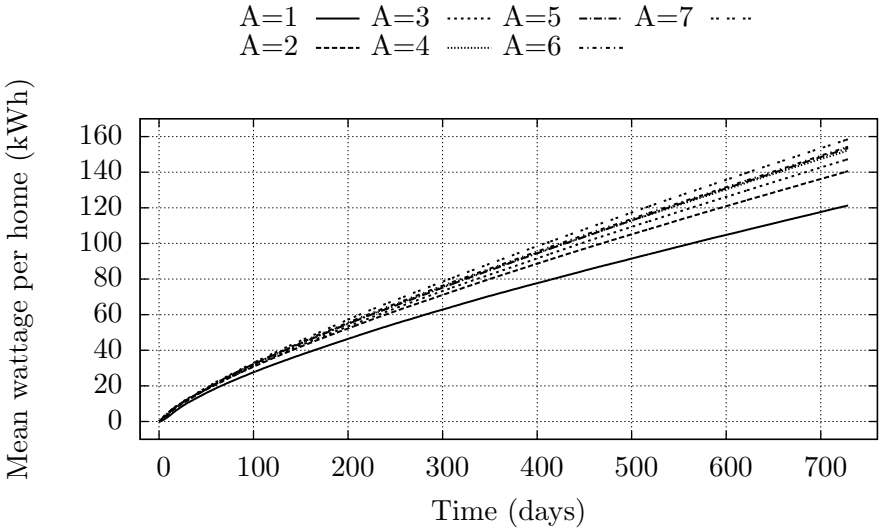

Figure 6.32: *Home Management*: Mean wattage per home for 100 homes under various arrival rates with cluster recovering.

after two years. If the home decides to become busy two times per week $A = 2$, then the wattage jumps to 140 kWh, which is a good tradeoff between higher wattage and achieved availability. Higher arrival rates ($3 \leq A \leq 7$), by now, only cost a small additional amount of power, but also do not improve the availability considerably. Concluding, whatever arrival rate of this scenario for homes is suggested, the mean wattage per home will be always clearly below the wattage of the local case where all homes host one HM-task itself and do not exchange them.

The last experiment varies the failure rate F under fixed replication degree of $R = 2$ and also fixed repair rate $A = 5$. Therefore, each home is served by a cluster of three members; the master task and the two replicates. Figure 6.33 shows this case. We clearly see how the number of failures per week F has a strong impact on the number of cluster crashes in the system. With on average one failure per week per home $F = 1$, almost 1548 cluster crashes occurred system-wide after two years. This are 86.1 % less failures per week per home as with $F = 7$. We can further see that the stepwide between high failure rates is smaller which verifies one aspect of the simulated model that with higher occurence of cluster crashes the owner homes must more often recover the whole cluster instead of recovering one failed HM-task of a running cluster. Frequently fully recoverd clusters crash more rarely as clusters where cluster members

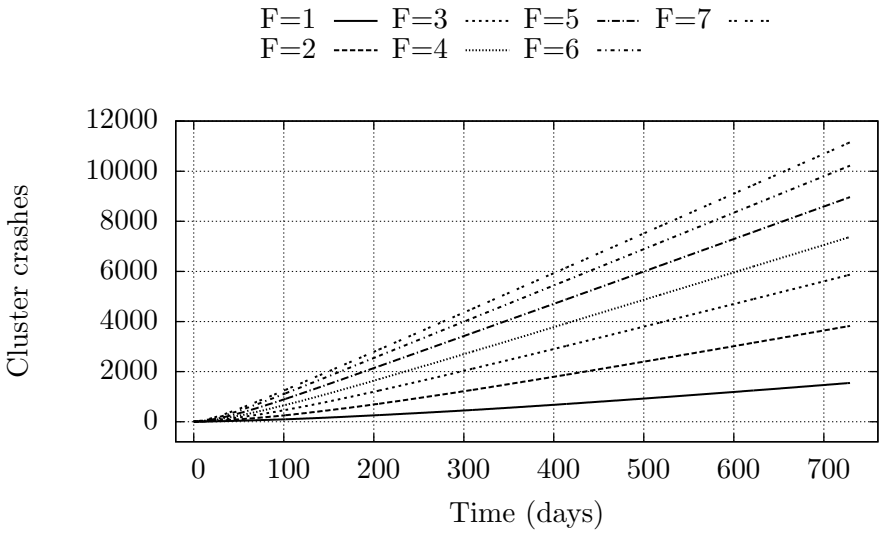

Figure 6.33: *Home Management*: Number of cluster crashes for 100 homes under various failure rates with cluster recovering.

must react within some delay to first detect the failed HM-task, then recover it and finally migrate it to another home. In this time, other HM-tasks of the cluster may fail and the cluster crashes earlier.

Figure 6.34 shows the availability in terms of running clusters. First of all, we see

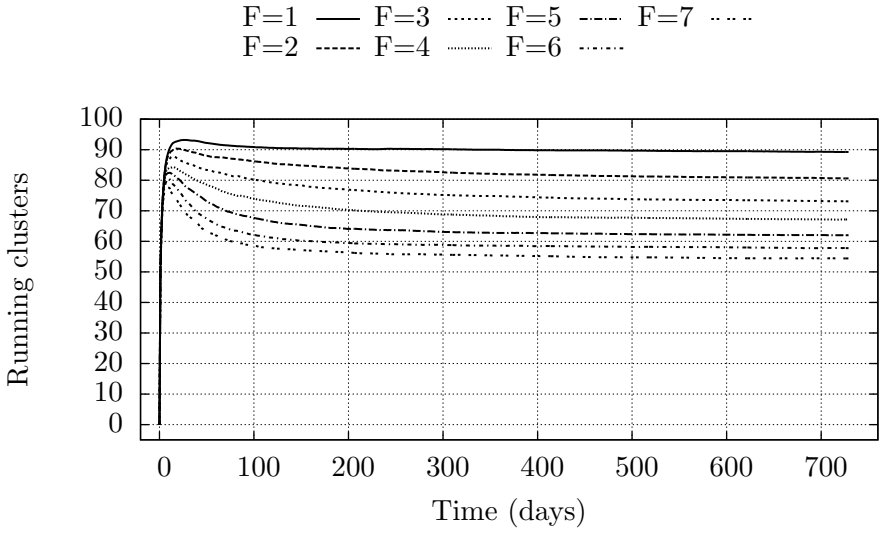

Figure 6.34: *Home Management*: Availability for 100 homes under various failure rates with cluster recovering.

under a failure rate of one failure per week per home $F = 1$, that the system holds a level around 90 % of all homes being managed by their correspondig home cluster. Higher failure rates ($2 \leq F \leq 7$) result in a slightly decreasing availability and lie clearly below the desired 80 % mark of runnig clusters. This can be improved by more frequent cluster recoveries by increasing the arrival or repair rate A.

Which impact higher failure rates have on the wattage per home is shown in Figure 6.35. This outlines the tradeoff between availability and power consumption. With

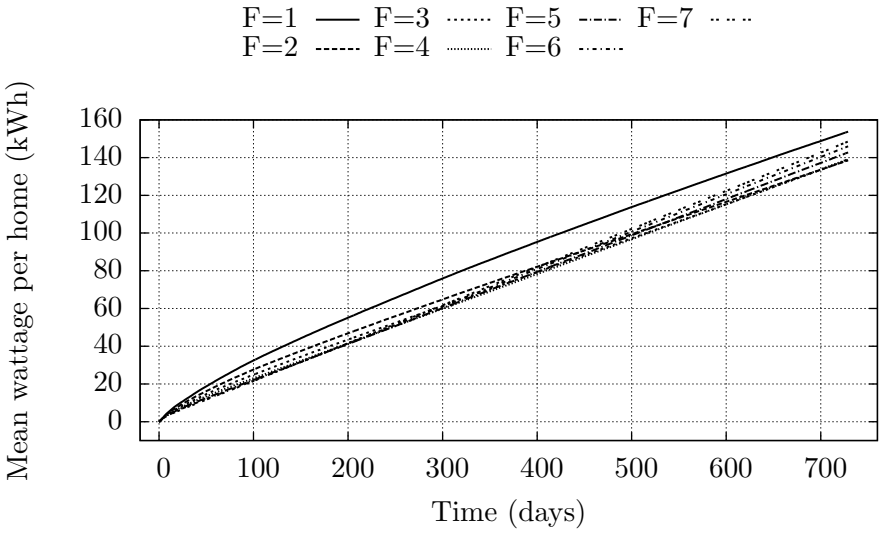

Figure 6.35: *Home Management*: Mean wattage per home for 100 homes under various failure rates with cluster recovering.

$F = 1$ one home caused a wattage of 153.8 kWh, and with $F = 7$ 148.5 kWh. That are 3.4 % less wattage per home with $F = 7$ because clusters are more rarely fully up due to frequent home failures. In the same time, the availability goes down by 39 % as we have seen in the previous Figure 6.34. This implies that the additional wattage caused by less home failures does not harm the power optimization goal and must be accepted for highly available systems. The main advantage comes from executing the load in a distributed manner but with replication.

Summarizing, *Home Management*, as modeled in this work, is in principle suitable for power saving. The distributed case with replication always outperforms the local case in two points. First, distribution of the load allows to aggregate it on a small part of the network and therefore consolidate the offered resources in a way that a big amount of power can be saved which results also in less wattage per home. This also holds with additional load caused by replication, which satisfies the second optimization goal, the availability.

6.4 Combined Scenario

The final analysis in terms of power saving is a combined scenario where all three previous introduced applications *Download Sharing* (DS), *Video Encoding* (VE) and *Home Management* (HM) are evaluated together. Now, homes start one HM-task and many DS- and VE-tasks to create a load mix; homes have to cope with this three different types of load at once.

The HM-task represents a task with moderate resource and availability requirements. The DS-task is a kind of lightweight task which at most requires some network band-

width whereas a VE-task causes high CPU utilization but has also considerably network bandwidth requirements. The simulation parameters for the following experiment are listed in Table 6.7.

Parameter	**Unit**	**Value**
numHomes	Number	100
numSuperHomes	Number	4
α	Factor	0.6
β	Factor	0.6
load (A)	Number	$1 \leq A \leq 35$
simPeriod	s	604800
serviceTime	s	28800
dsBwDn	kbit/s	2500
dsBwUp	kbit/s	200
dsCpu	Mhz	10
dsMem	MB	700
veSrcLength	min	100
veSrcVidRate	kbit/s	5000
veSrcAudRate	kbit/s	448
veTarVidRate	kbit/s	1000
veTarAudRate	kbit/s	128
hmBwDn	kbit/s	500
hmBwUp	kbit/s	500
hmCpu	Mhz	300
hmMem	MB	500
hmRep (R)	Number	$R = 0$
hmHbeat	s	300
hmCheck	s	60
hmTimeout	s	600

Table 6.7: Simulation parameters for the combined scenario.

The network consists of 100 homes with 4 clusters, each of them controlled by a super home. The performance factor α and also the wattage factor β are set to 0.6 indicating that the half of homes are 40 % slower in terms of the CPU as the other part of the network, but consume under load 40 % less power. Homes generate 1 to 35 DS- and VE-tasks per week ($simPeriod$), but only one HM-task. Since the HM-task has the highest priority, the other tasks are executed under best effort. This denotes that once the HM-task is started by each home, the system will be filled up with DS- and VE-task as many as homes can handle with shared resources. This implies a given loss rate and ensures that eventually all homes are consolidated. This in turn allows the analysis of a fully loaded network.

The service time is 8 hours and indicates that *active* homes must accept remote

tasks for this period. The resource requirements of a DS- or a VE-task are the same as defined previously in Section 6.1 or Section 6.2 respectively. Also the resources required for a HM-task is taken from Section 6.3, but this time without replication ($R = 0$) and without failures for showing the maximal achievable power saving.

Figure 6.36 gives us the mean number of busy (either in state A, AB or ABC) homes dependent on the load.

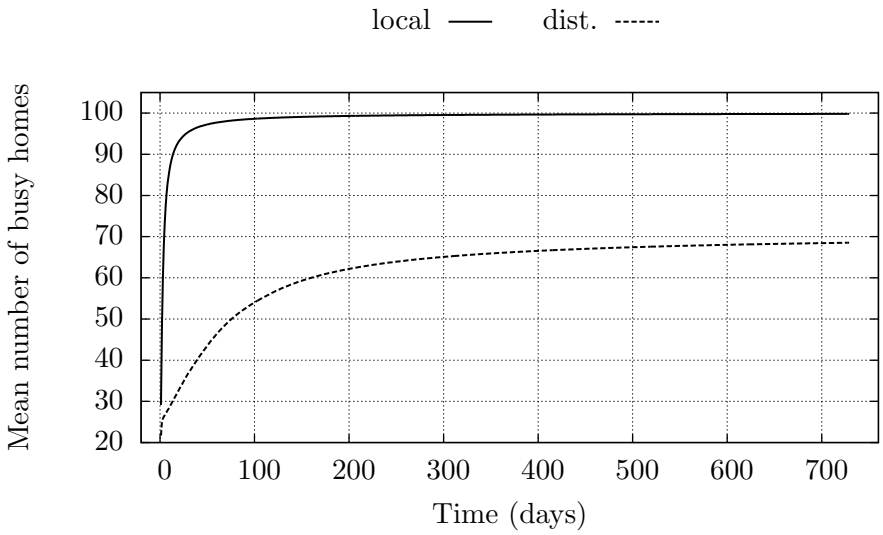

Figure 6.36: Combined Scenario: Mean number of busy homes.

As we see in the local case, the mean number of busy homes approaches quickly the natural maximum of 100 homes since this is the networks size. In this case no power can be saved in the sense that certain homes can be powered down. Instead, the only potential of power saving is to aggregate and distribute the load in a way that homes are consolidated, thus their shared resources are on average fully utilized. This can be done by ensuring an appropriate task mix for homes being busy and considering aggregation of CPU intensive tasks in homes with a good performance per watt ratio. In the distributed case the system requires 31.4 % less homes to cope with the same load as in the local case. In the distributed case on average only 68.5 homes must be *active* at the same time to work out the same load as in the local case where all homes must be busy.

The difference between local and distributed case is smaller in terms of wattage as in terms of busy homes as shown in Figure 6.37. In the local case, a home causes a wattage of 932.3 kWh after two years where most of this goes back to *Home Management* as mentioned in Section 6.3. In comparison, in the distributed case each home causes 690.8 kWh which is a saving of 25.9 % or 241.5 kWh per home after two years. The whole system will save 12075 kWh after one year. The single home therefore has a monthly power saving of 10.1 kWh.

The relation between the mean number of busy homes and mean wattage per home

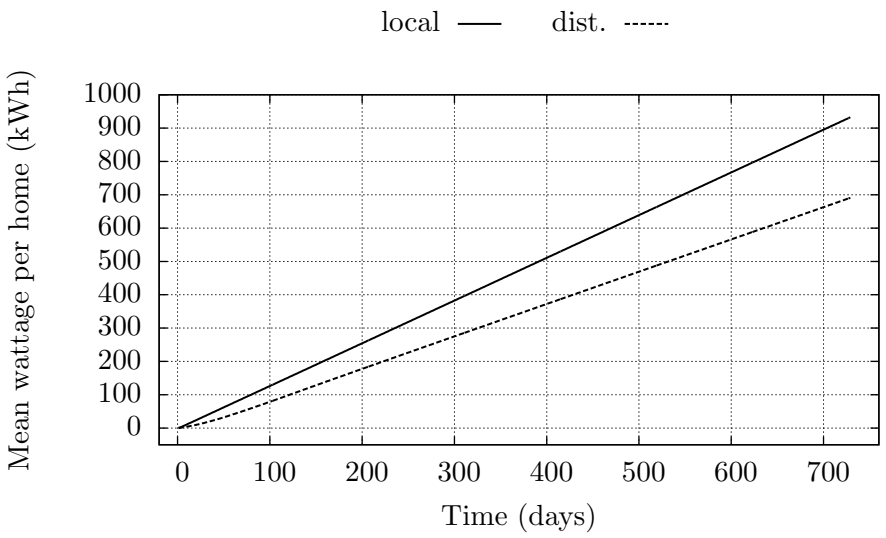

Figure 6.37: Combined Scenario: Mean wattage per home.

can be seen best in the following Figure 6.38. Again we see the two planes for the local

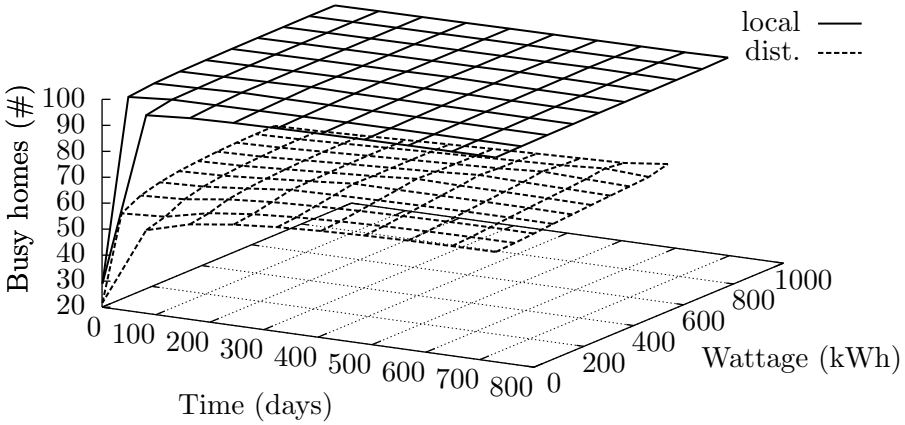

Figure 6.38: Combined Scenario: Joint plot of mean busy homes and mean wattage per home.

and the distributed case. In the local case the mean number of busy homes approaches the network size quicker where also the wattage is always above the distributed case. It can be postulated, that the outcome of this work is the fact that for certain applications is it possible to lower the wattage by nearly a third. It has been also shown that larger networks can further increase power saving.

6.5 Traffic Study

In this section the application *Download Sharing* (DS) is used to investigate the traffic produced under different organization schemes. Power saving is not addressed at all,

but the aim now is to know how much traffic originate on those homes which maintain statistics, the super homes.

A DS-task is a description of desired content (music, video, etc.) and maximum allowed downlink bandwidth allocatable for its download. Homes play two roles; as owner they send out DS-tasks to other homes and receive results later or as executer they receive DS-tasks, download the desired content, and then transfer the completed DS-task back to owners. There is a communication protocol between homes that can be divided into a state phase and a resource phase; in the state phase state information about homes is exchanged, whereas during the resource phase available resources are located and allocated. State information about homes include the current state of the home and the amount of free resources available for executing DS-tasks.

Conceptually, DS is based on *Virtual Machines* (VMs), encapsulating all necessary parts for migrating a DS-task before and after execution. These VMs are rather small on owner side and grow to considerable file sizes on executer side, because they include now the download content. Therefore, a constraint is the possible usable bandwidth between owner and executer.

Only for investigating the signaling traffic, a server-based approach is compared to two unstructured hybrid overlays in terms of traffic overhead, caused by different strategies of information management. Initially, a simplified centralized *Server*-based approach is simulated. Each *Home* (H) sends state information to one single H that acts as *Super Home* (SH), which has a global view on the system. In case of task-migrations, all resource requests must be sent to that SH.

The second approach (called *Overlay 1*), shown in Figure 6.39, is based on an unstructured and hybrid overlay inspired by Overnet as discussed in Chapter 3. A number of SHs are defined, which cluster the network. A SH is a H that additionally acts as a server for a cluster of Hs (clusters are illustrated as squares in Figure 6.39). Every state change information within a cluster is replicated to all other SHs (SHs have a global view). Resource requests of Hs are answered by their corresponding SH. The arrows in Figure 6.39 illustrate two independent communication flows. With message $S1$ a H sends its state to the SH. The SH forwards the state message to all other SHs ($S2$). At this point, the state information is replicated amongst SHs. Each state change triggers a state message, therefore a considerable amount of messages is generated. To gain resources, a resource request ($R1$) is sent to to a corresponding SH. The SH replies with a list of currently *active* Hs ($R2$).

Both of these approaches, *Server* and *Overlay 1*, create a global view on the overlay and cause overhead. The resource management is mainly done by lightweight always-on gateways as described in Chapter 3 and the overhead needs to be reduced. To achieve this, a third approach, *Overlay 2* is suggested (again Overnet inspired), that has modified communication patterns (shown in Figure 6.40). It has the aim of keeping resource information as local as possible. Hs send state information to their

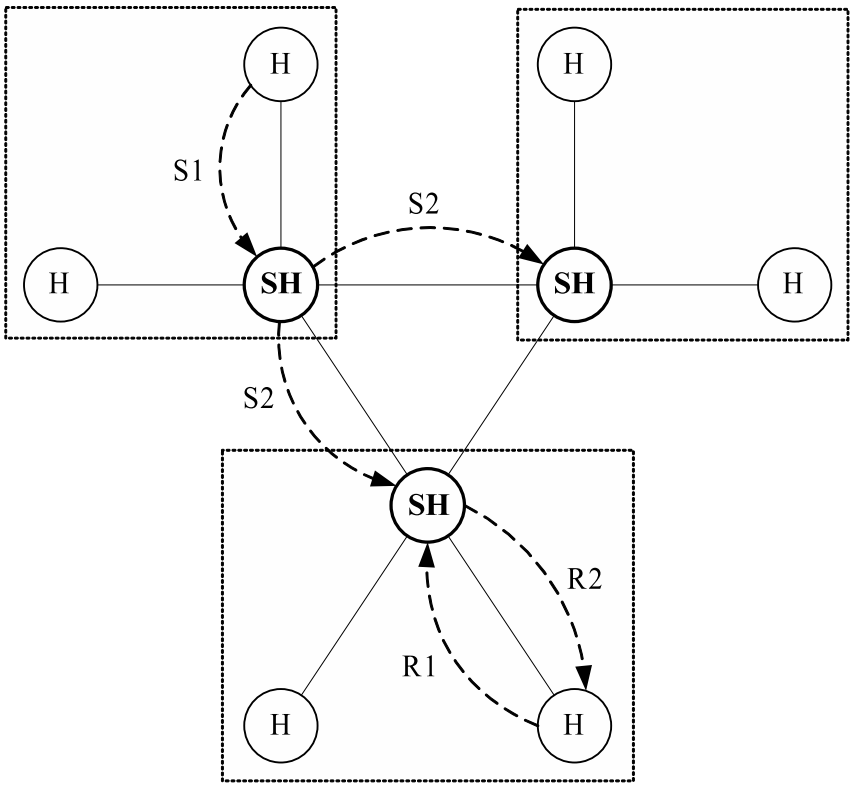

Figure 6.39: Topology and information flow of *Overlay 1*.

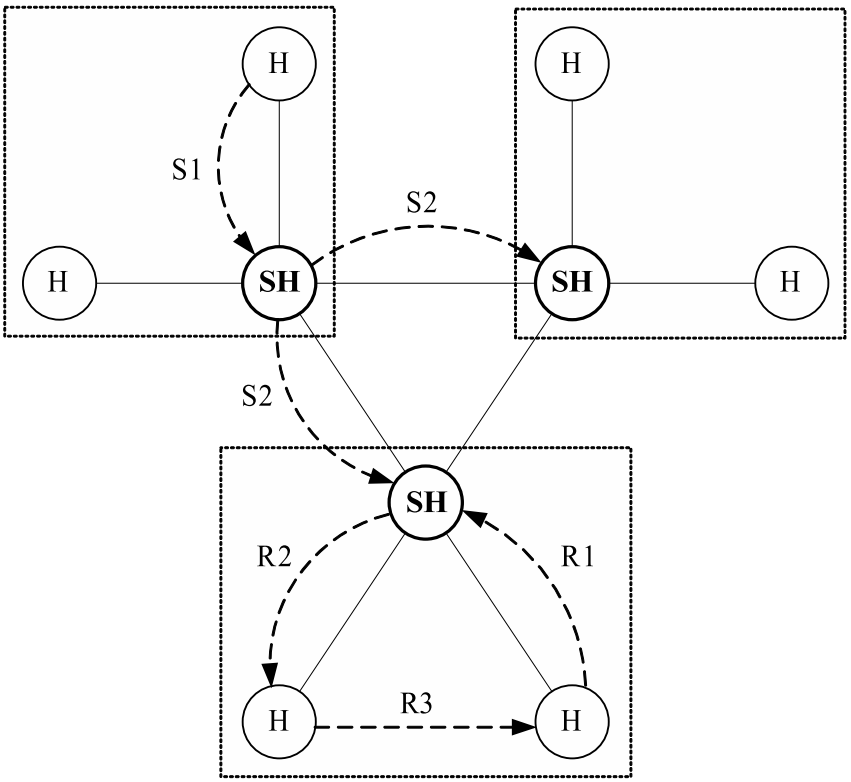

Figure 6.40: Topology and information flow of *Overlay 2*.

corresponding SH (message $S1$). SHs do not replicate this information, but exchange meta information about free hardware resources within the own cluster in configurable time-intervals ($S2$). In contrast to *Overlay 1*, none of the SHs has a global view on available resources. Resource information is kept local within each SH's cluster, only meta information is exchanged. This results in a different resource request scheme. The resource request first goes to the responsible SH ($R1$). The SH checks its own cluster and forwards the request to a H with free resources if possible ($R2$); this H directly answers the requester ($R3$). Otherwise, if there is no host within the own cluster that can process the task, the SH contacts other SHs based on the available meta information. If another SH has enough idle resources in its cluster, the initial SH forwards the resource request.

Costs of resource management in the system investigated is mainly based on the number and size of signaling messages within the network. To understand the message complexity of the presented overlays the sizes of modeled messages according to the introduced communication protocol are explained. The messages are specified in Table 6.8. $S = 60$ byte is the size of a state message, $R = 24$ byte is the size of a resource message, and L is the number of entries in the list of *active* Hs within a SH's cluster in *Overlay 1*. State update messages in byte/s for *Server* can be approximated by

Message type	**Overlay 1**	**Overlay 2**
State update	$S \quad (S1, S2)$	$S \quad (S1, S2)$
Resource request	$R \quad (R1)$	$R \quad (R1, R2b)$
Resource response	$L \times S \quad (R2)$	$R \quad (R2a, R3)$

Table 6.8: Signaling messages applied in *Overlay 1* and *Overlay 2*.

$$\frac{NMS(T+E)}{Y} \tag{6.12}$$

for *Overlay 1* by

$$\frac{NMS(T+E)}{Y} \times \left(2 - \frac{C}{N}\right) \tag{6.13}$$

and for *Overlay 2* by

$$\frac{\frac{Y}{U_H}C + 2\frac{Y}{U_{SH}}(\frac{N}{C} - 1)}{Y}S \tag{6.14}$$

where N is the number of Hs (network size), M is the assumed load (number of DS-tasks per H per year), T is the number of state transitions per H if homes staying in state A for executing tasks or staying in state P for saving power, E is the number of state events per task as defined in the communication protocol of *Overlay 1*, Y is the simulation time in seconds (one year), and C is the cluster size (each SH manages C Hs). U_H & U_{SH} indicate the delay between state updates from Hs and SHs as defined in the communication protocol of *Overlay 2*.

Accordingly, the traffic of resource messages in byte/s per SH for *Server* can be

approximately calculated by

$$\frac{M(R+LS)}{Y}N, \tag{6.15}$$

for *Overlay 1* by

$$\frac{M(R+LS)}{Y}C, \tag{6.16}$$

and for *Overlay 2* by

$$\frac{2(P_aMC + P_bMC(\frac{N}{C}-1))}{Y}R \tag{6.17}$$

which shows that the critical parameter N dramatically increases the message complexity for *Server*, whereas *Overlay 1* only relies on the number of clusters C which is a predefined fraction of N. For *Overlay 2*, additional parameters P_a & P_b are necessary. Those parameters express the probability if a task can be processed within the requester's cluster (P_a), or if it is forwarded to another cluster (P_b).

Table 6.5 presents state update traffic in a network with $C = 25$ with N increasing. The number of tasks per week per home (λ) is fixed to $\lambda = 10$, one year is considered

N	**Server**	**Overlay1**	**Overlay2**
100	0.69	1.21	1.65
1000	6.93	13.68	6.45
10000	69.25	138.34	54.45
100000	692.54	1384.91	534.45

Table 6.9: Expected state update traffic per SH in byte/s

where $U_H = 1200$ s and $U_{SH} = 900$ s. As clearly shown, the traffic is correlated with the network size N.

For verification and comparison all overlays are simulated. Figure 6.41 shows state and resource traffic in byte/s with regard to N. Generally, it can be seen that state traffic is much more critical than resource traffic. For state traffic, *Server* and *Overlay 2* clearly outperform *Overlay 1*. For $N > 700$, *Server* is also outperformed by *Overlay 2*; however state traffic is still linear and a matter of scalability. Further the resource traffic of *Server* grows with N.

Figure 6.42 shows the same for fixed N and increasing λ. Again, the state traffic is the more critical one. *Overlay 2* clearly outperforms *Overlay 1* for $\lambda > 6$ and *Server* for $\lambda > 12$. Overall, *Overlay 2* exhibits the good property of being invariant to load. Even the overhead caused by *Overlay 1* does not constrict Hs with high synchronous access bandwidth like 50 Mbit/s to act as SH.

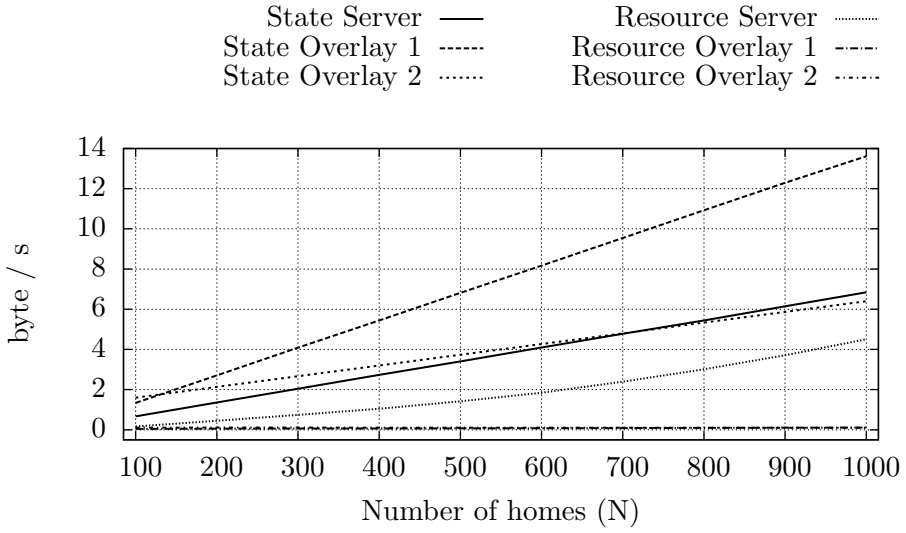

Figure 6.41: Traffic in terms of network size with a fixed load of 10 tasks per week.

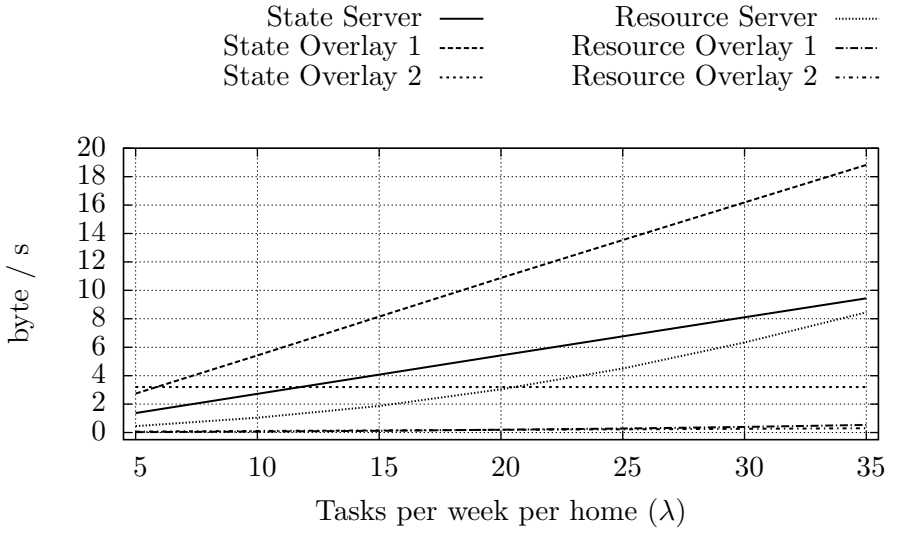

Figure 6.42: Traffic in terms of load with a fixed network size of 400 Hs.

7 Summary

This thesis addresses energy efficient resource sharing for networked homes. First, the current power usage of personal computers is identified and an energy efficiency goal formulated. A chapter about related work introduces similar works besides resource sharing with Grid and Cloud computing, Virtualization and corresponding technologies, and the concept of Virtual Home Environments. Then, an architecture describing a P2P-overlay and home network states is introduced. Three applications, each of them with different resource requirements, are suggested and investigated analytically. A simulation model is presented and comprehensive evaluation shows the possible gain of power saving.

7.1 Conclusion

The three sample applications *Download Sharing (DS)*, *Video Encoding (VE)* and *Home Management (HM)* show the power saving potential of resource sharing in the end user or home domain under two assumptions. First, a synchronous access bandwidth is necessary to gain advantage of transferring load from home to home. Second, heterogeneous homes in terms of shared resources or energy efficiency are necessary for CPU intensive tasks. With large networks, the possibility of many different configurations rise, and by the way also the distribution of load becomes better. *Download Sharing* demonstrates a good sharing performance in terms of distribution and wattage. *Video Encoding* does not reach that power saving as *Download Sharing* but still outperforms its local case. Finally, *Home Management* is, despite of replication and considerable CPU demands, very suitable for distributed execution which is clearly evinced by the power saving potential of the distributed case compared to the local case. Considering the sharing behavior of participating homes has shown, that fairness is oppositional to energy efficiency in the sense, that a part of the advantage of distributed execution is compensated. But the fair distributed case still outperforms the local case without sharing.

7.2 Future work

After finishing this thesis interesting questions came up:

- Is a pure distributed solution for managing the network of homes always better than a centralistic or even a hybrid approach and may the overhead of a fully distributed network outweigh the advantage in comparison to a server-assisted network topology?

- Makes replication for all applications sense and to which extend?
- Which economic model fits best with the energy efficiency goal and how the underlying cost function looks like?
- Are there completely other applications with power saving potential?
- Exists a general communication pattern between future homes or even residential locations conceptually below the application layer with optimization potential towards energy efficiency?
- Is it possible to create virtual machines small enough for sending them, but functional enough to contain a whole execution environment?
- Which concrete virtualization technology is suitable for task migration as suggested in this thesis?
- Would a prototype implementation of a software client, implementing the virtualization and the networking part, provide deeper insights as already done in this work?

Abbrevations

A	Active
AB	Active Blocked
ABC	Active Blocked Content
DS	Download Sharing
HM	Home Management
P2P	Peer-to-Peer
P	Passive
VE	Video Encoding
VHE	Virtual Home Environment
VMM	Virtual Machine Monitor

List of Figures

2.1 a) Grid virtualization and b) server virtualization. 16
2.2 The basic principle of a Virtual Machine Monitor. 18
2.3 Virtualization in PlanetLab. 23
2.4 The generic VHE roaming model. 29

3.1 Elemens of an architecture for energy efficient resource sharing. 36
3.2 An architecture for distributed energy efficient resource sharing. 39
3.3 State cycle of homes. 42
3.4 Case driven virtualization. 44

4.1 Failures modeled as birth-death process for *Home Management*. 56
4.2 Failures modeled as birth-death process for *Home Management* with 15 homes. 56
4.3 *Home Management*: Number of *active* homes. 59
4.4 *Home Management*: Achieved availability. 60
4.5 Birth-death process for local DS-tasks of one single home. 61
4.6 Birth-death process for simultaneous DS-tasks of N homes. 62
4.7 Number of *active* homes. 64
4.8 Energy efficiency η. 65
4.9 Number of *active* homes for $N = 100$ homes. 66
4.10 Number of *active* homes for $N = 1000$ homes. 66
4.11 Resource utilization (CPU, memory, network) of a download scenario. Fat lines denote results for a 8 Mbit/s access network (i.e. saturation occurs for $\lceil 8/2.4 \rceil = 4$ parallel downloads), thin lines for a 50 Mbit/s access network. 69

5.1 The rudimentary simulator architecture. 76
5.2 Simulated network. 78
5.3 State cycle of nodes. 80
5.4 An overlay for 100 homes and 4 super homes. 81
5.5 State information replication. 83
5.6 Default communication pattern of task sharing. 84
5.7 Communication pattern of task sharing if the task can not be outsourced. 85
5.8 Behavior classification for service time modification. 90

6.1 *Download Sharing*: Mean number of busy homes with 95 % confidence intervals (CI). 96
6.2 *Download Sharing*: Mean wattage per home. 96

6.3 *Download Sharing*: Joint plot of mean busy homes and mean wattage per home. 97
6.4 *Download Sharing*: Mean number of busy homes for 1000 and 100 homes. 98
6.5 *Download Sharing*: Mean wattage per home for 1000 and 100 homes. . 98
6.6 *Download Sharing*: Load plot for 1 to 35 DS-tasks per week. 99
6.7 *Download Sharing*: Wattage plot for 1 to 35 DS-tasks per week. 100
6.8 *Download Sharing*: Mean number of busy homes with fairness. 100
6.9 *Download Sharing*: Mean service time per home with fairness. 101
6.10 *Download Sharing*: Mean wattage per home with fairness. 102
6.11 *Video Encoding*: Mean number of busy homes with 95 % confidence intervals (CI). 104
6.12 *Video Encoding*: Mean wattage per home. 104
6.13 *Video Encoding*: Joint plot of busy homes and wattage. 105
6.14 *Video Encoding*: Mean number of busy homes for 1000 and 100 homes. 106
6.15 *Video Encoding*: Mean wattage per home for 1000 and 100 homes. . . . 106
6.16 *Video Encoding*: Load plot for 1 to 35 VE-tasks per week. 107
6.17 *Video Encoding*: Wattage plot for 1 to 35 DS-tasks per week. 108
6.18 *Video Encoding*: Mean number of busy homes with fairness. 108
6.19 *Video Encoding*: Mean service time per home with fairness. 109
6.20 *Video Encoding*: Mean wattage per home with fairness. 109
6.21 *Home Management*: Mean busy homes under various replication degrees with 95 % confidence intervals (CI). 112
6.22 *Home Management*: Mean wattage per home under various replication degrees. 112
6.23 *Home Management*: Home cluster. 114
6.24 *Home Management*: Number of cluster crashes for 100 homes under various replication degrees. 115
6.25 *Home Management*: Number of running clusters for 100 homes under various replication degrees. 116
6.26 *Home Management*: Mean wattage per home for 100 homes with varying replication degrees. 117
6.27 *Home Management*: Number of cluster crashes for 100 homes under various replication degrees with cluster recovering. 118
6.28 *Home Management*: Availability for 100 homes under various replication degrees with cluster recovering. 119
6.29 *Home Management*: Mean wattage per home for 100 homes under various replication degrees with cluster recovering. 120
6.30 *Home Management*: Number of cluster crashes for 100 homes under various arrival rates with cluster recovering. 120
6.31 *Home Management*: Availability for 100 homes under various arrival rates with cluster recovering. 121
6.32 *Home Management*: Mean wattage per home for 100 homes under various arrival rates with cluster recovering. 122

6.33 *Home Management*: Number of cluster crashes for 100 homes under various failure rates with cluster recovering. 123
6.34 *Home Management*: Availability for 100 homes under various failure rates with cluster recovering. 123
6.35 *Home Management*: Mean wattage per home for 100 homes under various failure rates with cluster recovering. 124
6.36 Combined Scenario: Mean number of busy homes. 126
6.37 Combined Scenario: Mean wattage per home. 127
6.38 Combined Scenario: Joint plot of mean busy homes and mean wattage per home. 127
6.39 Topology and information flow of *Overlay 1*. 129
6.40 Topology and information flow of *Overlay 2*. 129
6.41 Traffic in terms of network size with a fixed load of 10 tasks per week. . 132
6.42 Traffic in terms of load with a fixed network size of 400 Hs. 132

List of Tables

1.1 Mean power consumption of typical end user computers in watt (W) in 2008. . . . 2
1.2 Mean power consumption of typical end user computers in watt (W) in 2009. . . . 3

2.1 Typical differences between resource sharing concepts. . . . 15
2.2 Typical differences between virtualization technologies. . . . 20

3.1 Possible home states with two computers inside. . . . 43

4.1 Characteristics of tasks. . . . 48
4.2 *Download Sharing*: Minimum transfer bandwidth required. . . . 51
4.3 *Video Encoding*: Minimum transfer bandwidth required. . . . 52
4.4 *Home Management*: Minimum transfer bandwidth required. . . . 54
4.5 Estimated example sensor sampling/sending rates for *Home Management* services. . . . 55

5.1 Simulation parameters. . . . 87

6.1 Simulation parameters for the network topology. . . . 93
6.2 Simulation parameters for homes. . . . 94
6.3 Simulation parameters for *Download Sharing*. . . . 94
6.4 Simulation parameters for the default *Video Encoding* Scenario. . . . 103
6.5 Simulation parameters for the default *Home Management* scenario. . . . 110
6.6 Calculated fraction of busy homes. . . . 111
6.7 Simulation parameters for the combined scenario. . . . 125
6.8 Signaling messages applied in *Overlay 1* and *Overlay 2*. . . . 130
6.9 Expected state update traffic per SH in byte/s 131

Bibliography

[AA06] Adams and Agesen. A Comparison of Software and Hardware Techniques for x86 Virtualization. In *12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XII)*, San Jose, California, USA, 2006.

[Agr02] Alan Agresti. *Categorical Data Analysis*. Wiley-Interscience, 2nd edition, July 2002.

[AGR05] Anceaume, Gradinariu, and Ravoaja. Incentive for P2P fair resource sharing. In *5th IEEE International Conference on Peer-to-Peer Computing (P2P 2005)*, Konstanz, Germany, 8 2005.

[AIS04] John Augustine, Sandy Irani, and Chaitanya Swamy. Optimal Power-Down Strategies. In *45th Annual IEEE Symposium on Foundations of Computer Science*, Washington, DC, USA, 2004.

[ALM04] Amin, Laszewski, and Mikler. Grid Computing for the Masses: An Overview. *Lecture Notes in Computer Science; Grid and Cooperative Computing*, 3033:464–473, 2004.

[And04] Anderson. BOINC: A System for Public-Resource Computing and Storage. In *5th IEEE/ACM International Workshop on Grid Computing (GRID '04)*, Pittsburgh, USA, 11 2004.

[AR06] Anderson and Roscoe. Learning from PlanetLab. In *3rd Conference on USENIX Workshop on Real, Large Distributed Systems (WORLDS'06)*, volume 3, Seattle, Washington, USA, 2006.

[ATS04] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)*, 36(4):335–371, 2004.

[BA07] Bertoldi and Atanasiu. Electricity Consumption and Efficiency Trends in the Enlarged European Union - Status report 2006 - Institute for Environment and Sustainability. Scientific Publication EUR 22753EN, 2007. `http://ie.jrc.ec.europa.eu`.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Operation Systems Review*, 37(5):164–177, 2003.

[BFdM09] Andreas Berl, Andreas Fischer, and Hermann de Meer. Using System Virtualization to Create Virtualized Networks. In *Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen (WowKiVS2009), Kassel, Germany, March 2-6, 2009*, volume 17 of *Electronic Communications of the EASST*. EASST, 3 2009.

[BGdMT06] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. WileyBlackwell, 2nd edition, 5 2006. `http://www4.informatik.uni-erlangen.de/QNMC`.

[BH06] P. Bull and M. Harrison. Managing broadband home networks. *BT Technology Journal*, 24(1):79–85, 2006.

[BH07a] Ying-Wen Bai and Huang-Te Hsu. Design and implementation of an instantaneous turning-on mechanism for PCs. *IEEE Transactions on Consumer Electronics*, 53(4):1595–1601, 2007.

[BH07b] Barroso and Hölzle. The Case for Energy-Proportional Computing. *Computer*, 40(12):33–37, 2007. `http://www.computer.org/portal/site/computer`.

[BHK+07] Bolioli, Hamlin, Kardach, Korn, Walker, and Wong. ENERGY STAR* System Implementation. Published by Intel with technical collaboration from the U.S. Environmental Protection Agency, 2007.

[BL07] Andreas Blinzenhoefer and Kenji Leibnitz. Estimating Churn in Structured P2P Networks. *Managing Traffic Performance in Converged Networks*, 4516/2007:630–641, 2007.

[BQ06] Bustamante and Qiao. Designing less-structured P2P systems for the expected high churn. *IEEE/ACM Transactions on Networking (TON)*, 16(3):617–627, 2006.

[BWS+09] Berl, Weidlich, Schrank, Hlavacs, and de Meer. Network Virtualization in Future Home Environments. In *International Workshop on Distributed Systems: Operations and Management (DSOM09)*, Venice, Italy, 10 2009.

[BYV08] Buyya, Yeo, and Venugopal. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In *10th IEEE International Conference on High Performance Computing and Communications (HPCC-08)*, Los Alamitos, CA, USA, 9 2008.

[CB08] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A Survey of Network Virtualization. Technical report, University of Waterloo, 10 2008.

[CEEC08] Chan, Estve, Escriba, and Campo. A review of smart homes Present state and future challenges. *Computer Methods and Programs in Biomedicine*, 91(1):55–81, 2008.

[CFH+05] Clark, Fraser, Hand, Hansen, Jul, Limpach, Pratt, and Warfield. Live Migration of Virtual Machines. In *2nd Symposium on Networked Systems Design & Implementation (NSDI'05)*, Boston, Massachusetts, USA, 2005.

[CKC+09] Kyung Choi, Mi-hui Kim, Ki-Joon Chae, Jong-Jun Park, and Seong-Soon Joo. An Efficient Data Fusion and Assurance Mechanism using Temporal and Spatial Correlations for Home Automation Networks. *IEEE TRANSACTIONS ON CONSUMER ELECTRONICS*, 55(3):1330–1336, AUG 2009.

[CSP04] Ci, Sharif, and Peng. An Effective Scheme for Energy Efficiency in Mobile Wireless Sensor Networks. In *IEEE INternational Confernece on Communications*, pages 3468–3490, 2004.

[DM02] Daoud and Mohan. Strategies for Provisioning and Operating VHE Services in Multi-Access Networks. *IEEE Communications Magazine*, 40(1):78–88, 1 2002.

[EGH+07] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerdt, Laurent Mathy, and Tim Schooley. Evaluating XEN for Router Virtualization. In *16th International Conference on Computer Communications and Networks (ICCCN 2007)*, pages 1256–1261, 8 2007.

[FC05] Michal Feldman and John Chuang. Overcoming free-riding behavior in peer-to-peer systems. *SIGecom Exch.*, 5(4):41–50, 2005.

[FGR07] Nick Feamster, Lixin Gao, and Jennifer Rexford. How to lease the internet in your spare time. *SIGCOMM Comput. Commun. Rev.*, 37(1):61–64, 2007.

[FH09] Reza Zanjirani Farahani and Masoud Hekmatfar. *Facility Location - Concepts, Models, Algorithms and Case Studies, Chapter 8: Median Location Problem*. Contributions to Management Science. Physica-Verlag HD, July 2009. Facility Location - Concepts, Models, Algorithms and Case Studies.

[FHL+01] Fraser, Hand, Limpach, Warfield, Magenheimer, Nakajima, and Mallick. Xen 3.0 and the Art of Virtualization, 2001. `http://www.cl.cam.ac.uk/netos/papers/2005-xen-ols.ppt`.

[FJS08] Feigenbaum, Johnson, and Syverson. A Model of Onion Routing with Provable Anonymity. *Lecture Notes in Computer Science; Financial Cryptography and Data Security*, 4886:57–71, 2008.

[For06] Forte. Advances in Onion Routing: Description and backtracing \ investigation problems. *Digital Investigation*, 3(2):52–88, 2006.

[FWB07a] Fan, Weber, and Barroso. Power provisioning for a warehouse-sized computer. In *34th annual international symposium on Computer architecture (ISCA'07)*, San Diego, California, USA, 2007. `http://www.cse.ucsd.edu/isca2007`.

[FWB07b] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andr Barroso. Power Provisioning for a Warehouse-sized Computer. In *Proceedings of the ACM International Symposium on Computer Architecture*, San Diego, 6 2007.

[FXY+04] Fang, Xin, Yang, Huimin, and Ping. A policy based multi-dimension adaption framework in the Virtual Home Environment. In *2004 IEEE 60th Vehicular Technology Conference (VTC2004-Fall)*, Los Angeles, USA, 9 2004.

[GBH+08] García, Berl, Hummel, Weidlich, Houyou, Hackbarth, de Meer, and Hlavacs. An Economical Cost Model for Fair Resource Sharing in Virtual Home Environments. In *4th EURO-NGI Conference on Next Generation Internet Networks (NGI-2008)*, Krakow, Poland, 4 2008.

[GCN05] Chamara Gunaratne, Ken Christensen, and Bruce Nordman. Managing energy consumption costs in desktop PCs and LAN switches with proxying, split TCP connections, and scaling of link speed. *International Journal of Network Management*, 15(5):297–310, 2005.

[Gee08] Geelan. Twenty-One Experts Define Cloud Computing, 8 2008. `http://virtualization.sys-con.com/node/612375`.

[Geu01] Geuna. UMTS Network Aspects, 1 2001.

[GM05] Gilbert and Malewicz. The Quorum Deployment Problem. *Lecture Notes in Computer Science; Principles of Distributed Systems*, 3544:316330, 2005.

[GSS06] Godfrey, Shenker, and Stoica. Minimizing Churn in Distributed Systems. In *2006 Conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM'06)*, Pisa, Italy, 9 2006.

[HC07] Hautakorpi and Camarillo. Evaluation of DHTs from the Viewpoint of Interpersonal Communications. In *6th International Conference on Mobile and Ubiquitous Multimedia (MUM'07)*, Oulu, Finland, 12 2007.

[HCW05] Daniel Hughes, Geoff Coulson, and James Walkerdine. Free Riding on Gnutella Revisited: The Bell Tolls? *IEEE Distributed Systems Online*, 6(6):1, 2005.

[HH09] Stuart Hacking and Benoît Hudzia. Improving the live migration process of large enterprise applications. In *VTDC '09: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, pages 51–58, New York, NY, USA, 2009. ACM.

[HHS08] Hong, Hilt, and Schulzrinne. Evaluation of Control Message Overhead of a DHT-Based P2P System. *Bell Labs Technical Journal*, 13(3):79–86, 2008. `http://www3.interscience.wiley.com/journal/97517143/home`.

[HHW+07] Hlavacs, Hummel, Weidlich, Houyou, and de Meer. Energy Efficiency in Future Home Environments: A Distributed Approach. In *1st Home Networking Conference*, Paris, France, 12 2007.

[HHWdM10] Hlavacs, Hummel, Weidlich, and de Meer. Modeling Energy Efficiency in Distributed Home Environments. *International Journal of Communication Networks and Distributed Systems (IJCNDS)*, 4(2):161–182, 2010.

[Hou08] Houy. A characterization of majority voting rules with quorums. *Theory and Decision*, 67:295–301, 2008.

[HWH+08] Hlavacs, Weidlich, Hummel, Houyou, Berl, and de Meer. Distributed energy efficiency in future home environments. *Annals of Telecommunications - Home networking: performance and architecture challenges*, 63(9-10):453–541, 2008.

[HWT08] Hlavacs, Weidlich, and Treutner. Energy Saving in Future Home Environments. In *2nd Home Networking Conference at IFIP Wireless Days*, Dubai, United Arab Emirates, 11 2008.

[HWT10] Hlavacs, Weidlich, and Treutner. Energy Saving in Future Home Environments. *Under review for the First ACM SIGCOMM Workshop on Green Networking*, 8 2010.

[HZCS09] Tzu-Chi Huang, Sherali Zeadally, Naveen Chilamkurti, and Ce-Kuen Shieh. Design, implementation, and evaluation of a Programmable Bandwidth Aggregation System for home networks. *Journal of Network and Computer Applications*, 32(3):741–759, 2009.

[IAA07] Iyilade, Aderounmu, and Adigun. Incentives for Resource Sharing and Cooperation in Grid Computing System. In *The 2007 Conference on International Next Generation Mobile Applications, Services and Technologies (NGMAST'07)*, Cardiff, Wales, UK, 9 2007.

[Int06] Intel. Leap Ahead (online) - White Paper - Data Center Optimization - Increasing Data Center Density while Driving down Power and Cooling Costs, 2006. `http://www.intel.com/business/bss/infrastructure/enterprise/power_thermal.pdf`.

[Int07] Intel. Virtualizatin View (online) - Virtualization Can Help Power Efficiency, 2007. `http://www-03.ibm.com/systems/virtualization/news/view/011607.html`.

[ISG03] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Transactions on Embedded Computing Systems (TECS)*, 2(3):325–346, 2003.

[Jai91] R. K. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 4 1991. `http://www.cse.wustl.edu/~jain/books/perfbook.htm`.

[JSAC01] Christine E. Jones, Krishna M. Sivalingam, Prathima Agrawal, and Jyh Cheng Chen. A Survey of Energy Efficient Network Protocols for Wireless Networks. *Wireless Networks*, 7(4):343–358, 2001.

[Kat08] Katzer. Debunking Power Supply Myths, 9 2008. `http://www.anandtech.com/printarticle.aspx?i=3413`.

[KBN+06] Koomey, Belady, Nordman, Lange, Tipley, Darnell, Accapadi, Rumsey, Kelley, Tschudi, Moss, Greco, and Brill. Server Energy Measurement Protocol - Version 1.0. In *Energy Efficiency Server Benchmark Technical Workshop*, Santa Clara, CA, USA, 2006.

[kE07] El khatib and Edwards. A Survey-based Study of Grid Traffic. In *GridNets 2007*, Lyon, France, 10 2007.

[KES+07] Ryota Kawamoto, Takumi Emori, Shiro Sakata, Kazunori Furuhata, Kouhei Yuasa, and Seiichiro Hara. DLNA-ZigBee gateway architecture and energy efficient sensor control for home networks. In *2007 PROCEEDINGS OF THE 16TH IST MOBILE AND WIRELESS COMMUNICATIONS, VOLS 1-3*, pages 821–825, 2007. 16th IST Mobile and Wireless Communications, Budapest, HUNGARY, JUL 01-05, 2007.

[KKSK09a] Saad A. Khan, Fahad A. Khan, Arslan Shahid, and Zubair A. Khan. Load balanced clustering algorithm for energy efficient home area networking. In *SAS 2009 - IEEE Sensors Applications Symposium*, pages 284–289, 2009. 4th IEEE Sensors Applications Symposium, New Orleans, LA, Feb. 17-19, 2009.

[KKSK09b] Saad Ahmad Khan, Fahad Ahmad Khan, Arslan Shahid, and Zubair Ahmad Khan. Zigbee Based Reconfigurable Clustered Home Area Network. In *SENSORCOMM '09: Proceedings of the 2009 Third International Conference on Sensor Technologies and Applications*, pages 32–37, Washington, DC, USA, 2009. IEEE Computer Society.

[KKU08] Karakaya, Körpeŏglu, and Ulusoy. Counteracting free riding in Peer-to-Peer networks. *Computer Networks*, 52(3):675694, 2008.

[KLS08] Kontogiannis, Lewis, and Smith. Supporting A Service-Oriented Architecture. In *2nd international workshop on Systems development in SOA environments (SDSOA'08)*, Leipzig, Germany, 5 2008.

[KM08] Kobayashi and Mark. *System Modeling And Analysis - Foundations of System Performance Evaluation*, volume 1. Prentice Hall, 1st edition, 2008. `http://www.princeton.edu/kobayashi/Book/book.html`.

[Knu97] Knuth. *The Art of Computer Programming - Seminumerical Algorithms*, volume 2. Addison-Wesley Professional, 3rd edition, 1997. `http://www-cs-faculty.stanford.edu/~knuth/taocp.html`.

[Koo07] Jonathan G. Koomey. Estimating Total Power Consumption by Servers in the U.S. and the World, 2007. `http://www.koomey.com`.

[KSTT04] Krishnan, Smith, Tang, and Telang. The Impact of Free-Riding on Peer-to-Peer Networks. In *37th Hawaii International Conference on System Sciences (HICSS'04)*, Big Island, HI, USA, 2004.

[Lau08] Lau. Verteiltes Rechnen übers Internet mit BOINC & Co. *c't Magazin für computer technik*, 21:140–145, 2008.

[LCP+05] Lua, Crowcroft, Pias, Sharma, Lim, and Microsoft Asia. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Communications Surveys and Tutorials*, 7(2):72–93, Second Quarter 2005.

[Lig05] Ligneris. Virtualization of Linux based computers: the Linux-VServer project. In *19th International Symposium on High Performance Computing Systems and Applications (HPCS 2005)*, Guelph, Ontario, Canada, 2005.

[LK99] Law and Kelton. *Simulation Modeling and Analysis*. McGraw-Hill Science/Engineering/Math, 3rd edition, 1999. `http://www.mhhe.com/engcs/industrial/lawkelton`.

[LKP+09] Jong-Hoon Lee, Jung-Tae Kim, Eui-Hyun Paik, Intark Han, and Kwang-Roh Park. Design and implementation of a service oriented power saving system for the home network. *International Conference on Computers in Education*, 0:1–2, 2009.

[LMBE03] Löser, Müller, Berger, and Eikerling. Peer-to-Peer Networks for Virtual Home Environments. In *36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, Hawaii, USA, 1 2003.

[LYBP02] Liotta, Yew, Bohoris, and Pavlou. Supporting Adaptation-aware Services through the virtual home environment, 2002.

[MCZ06] Aravind Menon, Alan L. Cox, and Willy Zwaenepoel. Optimizing Network Virtualization in Xen. In *USENIX Annual Technical Conference*, pages 15–28, 5 2006.

[MN98] Matsumoto and Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998. `http:`/www.linklings.net/tomacs.

[MRPM08] Meshkova, Riihijärvi, Petrova, and Mähönen. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Computer Networks*, 52(11):2097–2128, 2008. `http://www.elsevier.com/locate/comnet`.

[MSHK07] H.-S. Mok, S.-Y. Son, J.H. Hong, and S. Kim. An approach for energy-aware management in ubiquitous home network environment. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4761 LNCS:293–300, 2007.

[NUT+02] Nakajima, Ueno, Tokunaga, Ishikawa, Satoh, and Aizu. A Virtual Overlay Network for Integrating Home Appliances. In *2002 Symposium on Applications and the Internet (SAINT'02)*, Nara, Japan, 7 2002.

[NWD03] Ngan, Wallach, and Druschel. Enforcing Fair Sharing of Peer-to-Peer Resources. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, USA, 2 2003.

[OBC05] Hayoung Oh, H. Bahn, and Ki-Joon Chae. An energy-efficient sensor routing scheme for home automation networks. *Consumer Electronics, IEEE Transactions on*, 51(3):836–839, Aug. 2005.

[OFGG06] Osrael, Froihofer, Gladt, and Goeschka. Adaptive Voting for Balancing Data Integrity with Availability. *Lecture Notes in Computer Science; On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, 4278:1510–1519, 2006.

[OHP08] Hyun Woo Oh, In Tark Han, and Kwang Roh Park. A power saving system based on energy-aware control elements in ubiquitous home network. In *Consumer Electronics, 2008. ISCE 2008. IEEE International Symposium on*, pages 1–4, April 2008.

[Owe07] Owen. Fun with onion routing. *Network Security*, 2007(4):8–12, 2007.

[PBFM06] Peterson, Bavier, Fiuczynski, and Muir. Experiences building PlanetLab. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*, Seattle, Washington, USA, 2006.

[PCLP08] Wan-Ki Park, Chang-Sic Choi, Haeryong Lee, and Kwang-Roh Park. Energy Efficient Home Gateway Based on User Service Traffic in Always-On Home Network Environment. *International Conference on Advances in Electronics and Micro-electronics*, 0:121–125, 2008.

[PM02] Pope and Meredith. The Virtual Home Environment, Release 5. Technical Report TR 22.121 V5.3.1, 3GPP, 7 2002.

[PR06] Peterson and Roscoe. The design principles of PlanetLab. *ACM SIGOPS Operating Systems Review (SOSP'06)*, 40(1):11–16, 2006.

[PRSBM+09] Milad Pastaki Rad, Ali Sajedi Badashian, Gelare Meydanipour, Morteza Ashurzad Delcheh, Mahdi Alipour, and Hamidreza Afzali. A survey of cloud platforms and their future. In *ICCSA '09: Proceedings of the International Conference on Computational Science and Its Applications*, pages 788–796, Berlin, Heidelberg, 2009. Springer-Verlag.

[PRV09] H. Pensas, H. Raula, and J. Vanhala. Energy Efficient Sensor Network with Service Discovery for Smart Home Environments. In *Proceedings - 2009 3rd International Conference on Sensor Technologies and Applications, SENSORCOMM 2009*, pages 399–404, 2009.

[PvdH07] Papazoglou and van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The International Journal Very Large Databases (VLDB Journal)*, 16(3):389–415, 2007.

[QNC06] Quétier, Neri, and Cappello. Scalability Comparison of Four Host Virtualization Tools. *Grid Computing*, 5(1):83–98, 2006.

[RCXA03] Roussaki, Chantzara, Xynogalas, and Anagnostou. The Virtual Home Environment roaming perspective. In *IEEE International Conference onCommunications (ICC'03)*, 3 2003.

[RG05] Rosenblum and Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. *Computer*, 38(5):39– 47, 2005.

[RJX+02] Roussaki, Jormakka, Xynogalas, Laikari, Chantzara, and Anagnostou. Multi-terminal and multi-network access to virtual home environment. In *IST Mobile and Wireless Telecommunications*, Thessaloniki, Greece, 6 2002.

[RL02] L. Ramaswamy and Ling Liu. Free Riding: A New Challenge to Peer-to-Peer File Sharing Systems. In *36th Hawaii International Conference on System Sciences (HICSS'03)*, Big Island, HI, USA, 2002.

[RMD+07] Ricquebourg, Menga, Durand, Marhic, Delahoche, and Log. The Smart Home Concept: our immediate future. In *9th International Conference on Advanced Communication Technology (ICACT)*, Gangwon-Do, Korea, 2 2007.

[RSO01] Roque, Soares, and Oliveira. VESPER Project- Validation of VHE Concept, 2001.

[RSRK07] Suzanne Rivoire, Mehul A. Shah, Parthasarathy Ranganathan, and Christos Kozyrakis. JouleSort: a balanced energy-efficiency benchmark. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 365–376, New York, NY, USA, 2007. ACM.

[Sch07] Schmidt. A Survey of Desktop Grid Applications for E-Science. *International Journal of Web and Grid Services*, 3(4):354–368, 2007.

[SEP05] Vassos Soteriou, Noel Eisley, and Li-Shiuan Peh. Software-directed power-aware interconnection networks. In *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 274–285, New York, NY, USA, 2005. ACM.

[SHM08] Sanders, Hamilton, and MacDonald. Supporting A Service-Oriented Architecture. In *The 2008 Spring simulation multiconference)*, Ottawa, Canada, 4 2008.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[SR06] Stutzbach and Rejaie. Understanding Churn in Peer-to-Peer Networks. In *6th ACM SIGCOMM conference on Internet measurement (IMC'06)*, Rio de Janeiro, Brazil, 10 2006.

[SS07] Gven Sahin and Haldun Sral. A review of hierarchical facility location models. *Computers & Operations Research*, 34(8):2310 – 2331, 2007.

[Sto07] Heinz Stockinger. Defining the grid: a snapshot on the current view. *The Journal of Supercomputing, Special issue on "Grid Technologies"*, 42(1):3–17, 3 2007.

[SW05] Ralf Steinmetz and Klaus Wehrle. *Peer-to-Peer Systems and Applications (Lecture Notes in Computer Science)*. Springer-Verlag New York, Secaucus, NJ, USA, 2005.

[TCdM02] Tomarchio, Calvagna, and di Modica. Virtual Home Environment for multimedia services in 3rd generation networks. In *NETWORKING 2002. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; and Mobile and Wireless Communications: Second International IFIP-TC6 Networking Conference*, Pisa, Italy, 5 2002.

[TdMV+02] Tomarchio, di Modica, Vecchio, Hovanyi, Postmann, and Portschy. Code mobility for adaption of multimedia services in a VHE environment. In *Seventh International Symposium on Computers and Communications (ISCC'02)*, Taormina, Italy, 1 2002.

[TLT09] Chang-Chun Tsai, Cheng-Jung Lee, and Shung-Ming Tang. The web 2.0 movement: mashups driven and web services. *W. Trans. on Comp.*, 8(8):1235–1244, 2009.

[Tut04] Kurt Tutschku. A measurement-based traffic profile of the eDonkey filesharing service. *Lecture notes in computer science*, 3015/2004:12–21, 2004.

[Vou08] Vouk. Cloud Computing - Issues, Research and Implementations. In *30th International Conference on Information Technology Interfaces (ITI 2008)*, Cavtat, Croatia, 6 2008.

[WCC+08] Walters, Chaudhary, Cha, Guercio Jr., and Gallo. A Comparison of Virtualization Technologies for HPC. In *22nd International Conference on Advanced Information Networking and Applications (AINA 2008)*, GinoWan, Okinawa, Japan, 2008.

[Whi91] Ward Whitt. The Efficiency of One Long Run Versus Independent Replications in Steady-State Simulation. *MANAGEMENT SCIENCE*, 37:645–666, 1991.

[Win07a] Windeck. Energy Star 4.0. *c't Magazin für computer technik*, 14:52–53, 2007.

[Win07b] Windeck. Spar-o-Matic. *c't Magazin für computer technik*, 15:200–207, 2007.

[WSG02] Whitaker, Shaw, and Gribble. Denali: Lightweight virtual machines for distributed and networked applications. In *5th Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, Massachusetts, USA, 2002.

[YGM02] Yang and Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *International Conference on Distributed Computing Systems (ICDCS 2002)*, Vienna, Austria, 2002. `http://www.eecg.utoronto.ca/icdcs07`.

[YHJ+08] Younwoo, Hyunsu, Junghwan, Changhwanand, and Ik. Design of a Multi-middleware Bridge for Supporting Interoperability in Home Network Environments. In *International Conference on Advanced Language Processing and Web Information Technology (ALPIT'08)*, Dalian Liaoning, China, 7 2008.

[YWL+06] H. Yan, S.A. Watterson, D.K. Lowenthal, K. Li, R. Krishnan, and L.L. Peterson. Client-Centered, Energy-Efficient Wireless Communication on IEEE 802.11b Networks. *Mobile Computing, IEEE Transactions on*, 5(11):1575–1590, Nov. 2006.

[ZHvdA06] Zdun, Hentrich, and van der Aalst. A survey of patterns for Service-Oriented Architectures. *International Journal of Internet Protocol Technology*, 1(3):132–143, 2006.

[Zuk09] Moshe Zukerman. *Introduction to Queueing Theory and Stochastic Teletraffic Models*. Zukerman, 2009. `http://www.ee.cityu.edu.hk/~zukerman/classnotes.pdf`.

Abstract

This work addresses power saving for home networks. It shows how power saving is possible and clearly unveils the power saving potential of resource and task sharing through cooperation. The global wattage of a network of homes as well as the local wattage of a single home can be reduced by load concentration. The energy efficiency of the remote case, in which tasks are distributed among homes, is compared to a local case without sharing for the same load. An architecture is proposed based on concepts of resource sharing, virtualization and virtual home environments. For this architecture, applications with typical resource requirements are tested to show under which circumstances power can be saved. Analytical models and simulation models are developed to figure out the benefit of the remote case under various situations.

Zusammenfassung

Diese Arbeit beschäftigt sich mit Energieeffizienz für Heimnetzwerke. Sie zeigt inwiefern Stromsparen mittels Ressourcen- und Aufgaben-Sharing durch Kooperation möglich ist. Der globale Stromverbrauch eines Netzwerkes von Heimen, und auch der lokale Stromverbrauch eines einzelnen Heimes, können durch Lastkonzentration reduziert werden. Die Energieeffizienz des verteilten Falls, in welchem Aufgaben unter den Heimen verteilt werden, wird mit der Energieeffizienz des lokalen Falls ohne Verteilung für die gleiche Last verglichen. Eine Architektur, basierend auf Konzepten des Ressourcen-Sharings, der Virtualisierung und virtuellen Heimumgebungen, wird vorgestellt. Für diese Architektur werden Applikationen mit typischen Ressourcen-Anforderungen erforscht um aufzuzeigen unter welchen Umständen Strom gespart werden kann. Analytische Modelle und Simulationsmodelle sind entwickelt worden um den Vorteil des verteilten Falls unter verschiedenen Aspekten darzustellen.

Curriculum Vitae

Personal: Mag.rer.soc.oec Roman Weidlich
Vienna, 29 July 1977
Austrian, married, no children yet

Education: 6.1999
School leaving examination of Commercial Academy

10.1999 – 12.2004
Economics and Computer Science at Technical University of Vienna

10.2006 – 5.2010
PhD in Computer Science at University of Vienna

Projects: Euro-NF, Network of Excellence, Network of the Future

Softnet Austria, Formal Methods in Software Engineering of Mobile Applications

Virtual Home Environments (VHE)

Decentralized and Self-* Networked Systems – Distributed Algorithms for Resource Sharing

Languages: German (native)
English (fluent)
Italian (advanced)
French (basic)

Publications:

Weidlich; P2P-Technologie - Design und Implementierung einer P2P- Anwendung unter JXTA, Diploma thesis, 2005.

Hlavacs, Hummel, Weidlich, Houyou, and de Meer. Energy Efficiency in Future Home Environments: A Distributed Approach. In 1st Home Networking Conference, Paris, France, 12 2007.

García, Berl, Hummel, Weidlich, Houyou, Hackbarth, de Meer, and Hlavacs. An Economical Cost Model for Fair Resource Sharing in Virtual Home Environments. In 4th EURO-NGI Conference on Next Generation Internet Networks (NGI-2008), Krakow, Poland, 4 2008.

Hlavacs, Weidlich, Hummel, Houyou, Berl, and de Meer. Distributed energy efficiency in future home environments. Annals of Telecommunications - Home networking: performance and architecture challenges, 63(9-10):453-541, 2008.

Hlavacs, Weidlich, and Treutner. Energy Saving in Future Home Environments. In 2nd Home Networking Conference at IFIP Wireless Days, Dubai, United Arab Emirates, 11 2008.

Berl, Weidlich, Schrank, Hlavacs, and de Meer. Network Virtualization in Future Home Environments. In International Workshop on Distributed Systems: Operations and Management (DSOM09), Venice, Italy, 10 2009.

Hlavacs, Hummel, Weidlich, and de Meer. Modeling Energy Efficiency in Distributed Home Environments. International Journal of Communication Networks and Distributed Systems (IJCNDS), 4(2):161-182, 2010.

Hlavacs, Weidlich, and Treutner. Energy Saving in Future Home Environments. Under review for the First ACM SIGCOMM Workshop on Green Networking, 8 2010.