Approaches to an adaptive Middleware for Mobile Services

Ulrich Dümichen, Uwe Baumgarten *

Abstract—**Adaptive middleware and wireless networks are hot topics in many discussions today. Literature describes a lot of approaches which are based upon existing middleware technologies by adding functionalities for adaptive behavior. Disadvantage of this proceeding is the resulting dependence upon the interface provided by the underlying middleware. This paper will describe a whole new approach for adaptive middleware which is on the one hand able to manage and organize a complex mobile wireless network. On the other hand it interprets requirements demanded from mobile services which are running on top of it. These requirements are described within a meta information base which comes along with a mobile service. Using this additional information the middleware adapts its behavior to optimize the performance of the complete network. To understand the behavior of the middleware this paper will first give a short description of the structure of the middleware. Subsequently it will classify four different abstraction layers starting with a view onto the entire network up to the fine-tuning mechanisms within the middleware itself. Afterward it will describe in detail a set the different adaption approaches within the four abstraction layers and show how these approaches are integrated into the middleware.**

Keywords: Adaptive middleware, Bluetooth Networks, Meshnets, Mobile Services, BlueSpot System

1 Introduction

Interest in wireless networks has increased dramatically in the last few years. But in contrast to wired networks, wireless networks need to be managed more elaborately. In many cases it is very useful to have a middleware which takes control over the connection management, fault management and communication management. Most of today's approaches concentrate on a subset of topics to be determined in this environment. This paper will take a closer look at the *BlueSpot System*[4], which is a project that develops an adaptive middleware to handle complete wireless network scenarios. The middleware is used to control wireless Meshnet infrastructures. There are mobile services on top of the middleware, which are implemented in either native C code or as Java MIDlets. The middleware itself is available for different architectures, like Linux, Symbian OS or Windows Mobile and for different devices like Smartphones or PDAs. Built up in this manner, the BlueSpot System is used to demonstrate todays existing wireless network management mechanisms and to check these mechanisms against each other. Additionally, because of the well-defined interfaces the middleware provides, it is very easy to integrate new concepts and to gain benchmarking results which are comparable to existing ones, made before with the BlueSpot System. The middleware is build up modular, so that it can be extended or shifted very easily. In contrast to other projects, the BlueSpot System can be adapted to a very wide field of problem areas, without loosing exiting results from other use cases. The underling network can be divided in communication nodes. Each communication node is classified as either an infrastructure node, which is helping to establish the network or a client node, which is a user of the infrastructure. The used hardware for infrastructure nodes are Gumstix[1]. While developing the system the main aspects were self-configuration, self-healing and self-organization of wireless networks with the ability to support nearly every kind of mobile service. Services are equipped with a meta information base, which describes the demands the service makes on the middleware. The middleware itself adapts to these demands by changing its behavior dynamically. In the following this paper will have a closer look at the architecture of the middleware and the contained parameters of the meta information base of a service. Afterward it will describe the application range divided into different abstraction layers. These are needed to consider different adaption behaviors. In the next section this application range will be used to classify different for one thing existing approaches, for another thing new approaches. It will describe how the network and the middleware adapt to an example parameter, a mobile service demands. At the end the paper will give a comparison of other approaches of adaptive middleware and will point out the main differences.

*Institut für Informatik, Technische Universität München, 85747 Garching, Germany, {duemiche, baumgaru}@in.tum.de

[1]Gumstix are XScale based embedded computers in the size of a chewing gum stick. For additional informations see `http://www.gumstix.com/`

2 Middlware description

The middleware consists of different layers. Figure 1 shows an overview of its structure. To be able to integrate

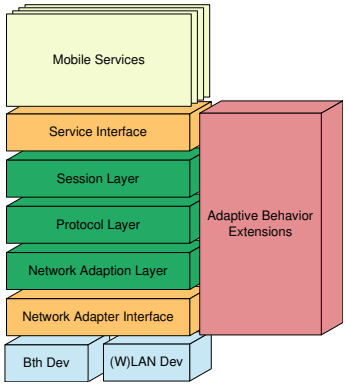

Figure 1: Middleware structure

different wireless network technologies, the BlueSpot System provides an interface, which enables the connection to underlying network drivers. This interface is located at the bottom of the stack and is called the *Network Adapter Interface*. It is implemented by the different available network adapters and connects them to the middleware. Support for Bluetooth and (W)LAN devices is already implemented. The *Network Adapter Interface* represents the consistent abstraction of any used network technology, but the main used technology at the moment is Bluetooth. Above the Network Adapter Interface is the *Network Abstraction Layer*, which abstracts the used communication technology to the rest of the middleware stack. On this way the top adjacent layer does not need to know anything about the kind of the underlying network. Additionally all relevant network attributes are collected to a defined set of parameters and are provided to the upper layers of the stack. The *Protocol Layer* sits on top of the Network Abstraction Layer. It handles the forming of message headers for communication. Additionally it is responsible for controlling available communication partners detected by the network adapters. To abstract communication from connections there is the *Session Layer* above. As the name implies, it is responsible for session handling and end-to-end communication. Especially using Bluetooth as underlying network technology, the Session Layer is very important. In this case the communication is often disturbed because of lost connections. After the reconnect, accomplished by the Protocol Layer, the connection is immediately assigned the the still existing session again and communication can be continued. This happens completely transparent to the on top running mobile services. In order to be able to dock mobile services to the middleware, there is the *Service Interface*. It defines the entry point for the execution of the services. Additionally it enables the management of the mobile services with functions e.g. needed to start or stop a service. As mentioned before, services can be implemented either in native C code or as Java MIDlets. Services implemented in native C are integrated via dynamic library linking whereas MIDlets are started in a new process. In this case the data exchange between service and middleware is done via socket communication. This is necessary because Java Native Interfaces (JNI) are not supported by Java MIDlets yet.

In general the are two ways to add adaptive behavior to a software. One way is to directly insert the corresponding code into the source code of it. This leads to a mishmash of the software's functionalities and adaptive functionalities within the same source code, which affect enormous the readability of it. The second approach is to define interfaces within each software module. The adaptive behavior is gained by implementing the interface in a separate module and connecting this module to the corresponding interfaces. This approach has the big advantage, that the software's functionalities and the adaptive behavior are strictly separated from each other. Additionally later it is easy to add further adaptive behavior without the need of editing the original code. A disadvantage of this approach is, that in some cases it is nearly impossible to separate the adaptive functionalities from the software's ones. This is why within this middleware both approaches are used. Whereby the second was strictly preferred wherever it was possible. The adaptive extensions for this middleware is implemented in the adaption module beside the main stack. It includes the defined interfaces for each layer. E.g. there is the *Protocol Interface*, which enables the binding of different replaceable wireless routing protocols to the Protocol Layer. The classification of different adaptive extensions assigned to the different middleware layers will be discussed in detail in section 5.

Each node of the network is equipped with the middleware. As part of the self-configuration, every node automatically tries at startup to connect to its neighbors to build up a network. At the moment the topology of the network is defined by a connection list which is given to every node. This was necessary to handle the complexity of the network construction, but the replacement of this list and thus the atomization of network forming will be part of future work. When a connection drops, the concerned nodes automatically try to reconnect to each other. After a defined time out without successful reconnection, the currently used routing protocol is notified and has to build up a different communication path as part of the self-healing behavior. Available services are announced and distributed by the *Service Discovery Service* (SDS), which is also used to find a communication partner, equipped with the same mobile service within the entire network. If a node does not provide a required service, this service is sent by the SDS to this node automatically. Afterward it is started immediately. The binaries of the service are transmitted in form of a zip archive which contains the several different binaries for the differ-

ent possible hardware architectures. To be able to run the service on each node in the entire network it is necessary to have the service in different versions. One the one hand as Java version and on the other hand as precompiled binaries. It is needful to have these binaries compiled for each hardware platform. Within the zip archive there is a folder for each supported hardware platform with the corresponding binaries in it. Within the BlueSpot System this procedure is called *service mobility*. The service mobility leads to an overhead of transferred data when the service is delivered to an other node but this is the only way to support all different devices. After the delivery the SDS searches the received zip archive an extracts the preferred binaries. Subsequently it is able to start the new service. With these efforts of self-management the usability of the complete system is simplified enormously.

3 Meta information base

The meta information base (MIB) contains a set of parameters which are adjustable according to the requirements of the service. An example parameter described in the meta information base is the minimum bandwidth the service needs run properly. Beside the minimum bandwidth there are several additional parameters supported which will be described in the following. This is a list of currently considered parameters:

- minimum bandwidth
- maximum bandwidth
- maximum allowed latency time
- priority of the service
- connection orientated or connection less service
- streaming or message based communication
- preferred routing algorithm

In addition to the minimum bandwidth requirement, also the maximum bandwidth, a service will use, is considered. This simplifies the arrangement of the available throughput within a domain. A routing algorithm is able to more easily plan the allocation of network resources, if it has this information. Another main parameter is the maximum allowed latency time. In case of a telephony service a long latency time would cause bad speech quality because of long pauses while waiting for an answer of the communication partner. The forth parameter is the predefined priority for a service. This priority, which is handled as an initial value, is used to determine which of the competitive services will be preferred. There are two approaches available for handling services priority. The simple one uses the parameter from the meta information base as a constant value. The second approach changes this parameter by considering the bandwidth the service has used within a defined time span. When it communicated a lot of data, the service is downgraded and thus its priority is decreased. In opposite, when the service has used less bandwidth its priority is increased again. This mechanism is useful to achieve a fair allocation of available bandwidth to services, where many services have to share a connection with very less bandwidth in total. The parameters described next are boolean values. The first one determines if the communication occurs in connectionless or connection orientated manner. This is used to disable sending acknowledgment packets in order to safe bandwidth and calculation time if a indication for successful transmission of data is not needed. If the service is set to streaming mode the middleware provides memory space for buffering received packets. The last parameter is a string value and names the routing algorithm which fits best to the properties of the service. If there are running more than one service, the used routing algorithm is elected by the middleware. Before starting a new service it sends a request to all already running services in order to receive their preferred algorithm and their priority value. These two values are used to calculate a mean value for each routing algorithm over the complete network. The one with the highest mean value is chosen as the next algorithm.

4 Application range

To be able to describe the adaptive behavior of the system, it is necessary to have a look at its application range. Figure 2 shows the different abstraction layers that have to be considered. On the right hand side there is the mobile service with its MIB. Within it the service contains

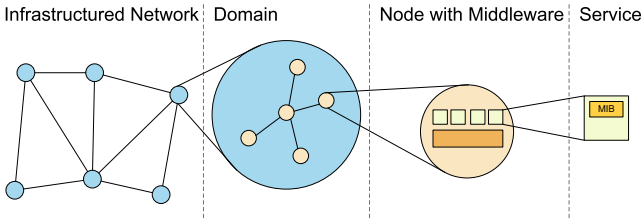

Figure 2: Application range

the demands the service makes to the network in order to run properly. This makes the service to the point of origin in order to observe the adaptive behavior of the middleware. Before starting a service, the middleware has to analyze the requirements of the service. When the demands can't be met by the middleware, it has to free resources or to trigger a change of network behavior. If so, neighbor nodes will get involved into the adaption handling. All nodes of the network are grouped into different domains to simplify the network management. In case of the BlueSpot System, the domain is equal to a *Bluetooth Piconet* [2], while in other cases it may make

sense to group the nodes into a hierarchical or geographical order. Most of the time a domain is built up either in a star layout or as a tree. As a result there is only one path to a designated node. The connection to other domains is done by bridging nodes. In case of the BlueSpot System these nodes belong to two different Piconets and hence form a *Scatternet* [3]. These nodes are called *Slave-Slave Nodes* because they are slaves in both Piconets. As part of the adaption handling, the middleware examine the situation within the domain at first. If the demands can't be met it has to expand its activities over the borders of the domain, onto the entire network.

5 Adaptive Behavior

In the following, this paper will describe the adaptive extensions of the middleware. It will explain how the system is able to react to demands described in the MIBs, i.e. demands on bandwidth for communication. For this reason it will consider the behavior of the system by means of the abstraction layers described in the section above in opposite direction. Additionally, it will point out the special requirements to the middleware within its architecture.

5.1 Network

The most common way to handle QoS parameters is to change the behavior of the network. This is achieved by adjusting the parameters of the routing algorithm. E.g. by replacing a reactive routing algorithm with a proactive one, less control messages are caused and thus less bandwidth is used[2]. In the case of the BlueSpot System there are three different algorithms implemented at the moment. The first is a simple flooding routing protocol with duplicate detection. The second is a modified DSR[3] protocol [5] which is an example for a reactive protocol. The last one is a modified DSDV[4] routing protocol [9] as an proactive protocol. Both protocols, the DSR and the DSDV, has to be modified to fit to the requirements of the BlueSpot System. If a node asks for an exchange of the routing protocol and the rest of the BlueSpot System agrees, a broadcast message is sent by this node to all other nodes of the entire network. Every node stops its activities and performs the exchange. After the new protocol has been started successfully, every node brings up its connections to its neighbors again and continues to communicate. The messages to be sent while the exchange were buffed and send after the successful reconnect.

If the exchange of the routing protocol does not free enough bandwidth, the routing algorithm itself has to react. Therefore it triggers a route search in oder to find a new communication path as shown in figure 3. In this example the Service S_1 on Node *Src* is communicating with Node *Dest* via the nodes A_1 and A_2. After the additional Service S_2 has started its communication on the same route, the available bandwidth is falling under the required threshold quoted in the meta information base of Service S_1. Now the routing algorithm reacts by changing the route to Node *Dest* via nodes B_1 and B_2. As a result, the network is more load-balanced and the Service S_1 obtains its needed bandwidth. If bandwidth of the new path is still not enough the routing algorithm bundles two different paths. So the evolving bandwidth is calculated by adding the available bandwidth of the two paths minus the bandwidth used for the required controlling messages.

The BlueSpot System provides three different ways for

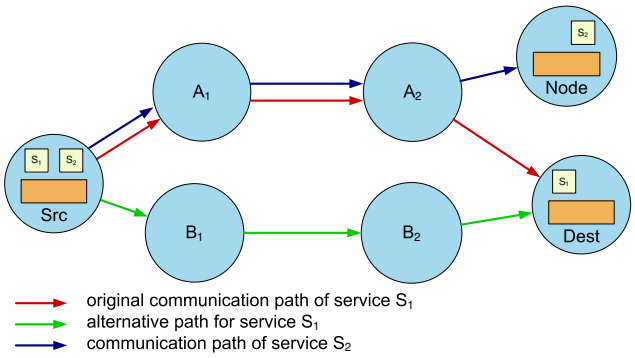

Figure 3: Alternative path finding

different path routing. The first approach is the known *Best Effort* proceeding first introduced by *GÉANT*[5]. Best Effort is not configurable and tries to find the best load-balancing on its own. A second approach is the use of priorities. Therefore we implemented a modified *DiffServ* protocol [8] which supports four different measures of quality, called *quality classes*. Each connection is associated to such a quality class and will be preferred by a node if its class is higher than the class of an other inbounding connection. The third approach is based on resource reservation. This approach pick-ups the idea of the *IntServ* proceeding [7]. It reserves bandwidth for every connection. In opposite to the DiffServ a connection is not associated to a quality class but gets a absolute priority. This is the big advantage of resource reservation, because once a connection is established the system is able to guarantee the requested resources. As a disadvantage the administration overhead for the resource reservation is very high. Each node contained in a path has to be informed to reserve the requested resources. Whereas using the DiffServ protocol every node just has to handle its own connection requests and there is no need for it to know the other involved nodes of a communication path.

When a service indicates in his meta information base

[2]Reactive algorithms fit better for scenarios in which clients move in higher speed through the entire network. In this case it is necessary that routing updates are made more frequently.

[3]Dynamic Source Routing

[4]Destination-Sequenced Distance-Vector

[5]see http://www.geant.net

that it is running in connectionless mode, it does not need the acknowledgments of arrived packets at the receiver side[6]. Thus again bandwidth can be saved by omitting the *ack* packets.
Considering the middleware architecture, the involved layers are the Protocol Layer and the Session Layer. The latter has to handle the different path routing while the Protocol Layer is responsible for the routing algorithm exchange.

5.2 Domain

The contemplations done before are only valid if the communication happens between different domains. Within a domain the adaption occurs in a different way. Here it is not possible to change communication paths because of the described properties of a domain. So the middleware reacts in two different ways. The first way is to move a node from one domain into another one. This is only possible if the concerned node is a client node and it is within range of a second domain. In this case the middleware on the client node initiates a handover to the new infrastructure node in the second domain. Now the service can use the available bandwidth in the new domain. The handover is implemented in the Session Layer of the middleware. The second way is to throttle the bandwidth used by other services. Because of the underling Bluetooth technology a domain represents a Piconet which is buildup in a star topology with the master in the center. For this reason in most cases the master is the point of origin for bandwidth throttling within a domain. The BlueSpot middleware provides two different approaches to support bandwidth throttling. The first approach is the usage of a token. Each master of a Piconet commands a token which it allot to the slaves in a predefined order. This approach is similar to standard token routing protocols, known from the network routing sector. The second approach is a credit system. Each slave obtains an predefined amount of credits which it can employ for communication. If its credits are used up the slave is not able to send data anymore. After a defined round time or after all other slaves have used up their credits, every slave obtains a new amount of credits and is able to continue communication. Advantage of the credit system is, that all nodes are allowed to communicate simultaneously. While using the token approach only one node a time is able to send data. But the token approach needs less overhead for managing the communication. Also fairness is easier to handle. The credit system and the token approach are implemented on hight of the Protocol Layer within the adaption module.

[6]This may happen i.e. within a sensor network where a sensor does not need to know if its sent information receives the listening destination node

5.3 Node with middleware

Looking at a node, it is not possible to directly manage bandwidth issues. Thus all approaches need to be indirect. A main starting point is the observation of the CPU load. If the load is over a predefined threshold it is predictably that the throughput of the network adapter will drop. This high CPU load is avoided by the middleware in two ways. At first it disables a running service with low priority or it prevents the start of a new one. If this approach does not succeed, the middleware tries to swap services to other nodes. Therefore it is necessary to determine if a service at all can be moved to another node. In most cases this is only possible if the service is a worker and does not need direct user interaction. In the BlueSpot System this *service mobility* is implemented by splitting the functionality of the service into two seperated services: one which handles the interaction with the user and another which is responsible for the whole work. An implemented example within the BlueSpot project is a service which steers a radio controlled (RC) miniature car. As described before the service is divided into to different services. One does the user interaction and enables the user to choose a predefined route the car has to drive. The other is calculating each track section. If the car (itself a client to the network, but with very low resources) comes within range of a node, the service running on it is already awaiting the connection. It immediately takes over the steering of the car until the car again leaves the range of its radio. While steering the service duplicates itself onto the node which will control the next track section. To simplify the positioning of the car, every node knows its own coordinates and the top velocity of the car. By knowing the start position of the car and interpreting the steering commands the steering service is able to calculate the position of the car at every time. Of course this is not (and never will be) an exact driving but it works still fine enough for demonstration issues of service mobility. A second indicator to monitor is the available memory. If the node runs out of memory the middleware falters. In this case it is not possible anymore to guarantee any bandwidth value. These approaches are implemented in the Session Layer and the Service Interface.

5.4 Service

Considering the situation on service level the application of a service should be reviewed by the usage of a simulator. Before the service is allowed to start running on the real system, its processing is simulated within a testing environment. To handle service testing and verification the BlueSpot System is supplemented with an additional network adapter. This adapter implements the binding to a NS2 simulator[7], which is able to simulate a com-

[7]For additional information about NS2 see the website `http://www.isi.edu/nsnam/ns/`

plex network topology. The simulator runs through different programmed network situations which will cause the middleware to react in order to fulfill the demands of the service. If these testings showed positive, the service is verified to run properly on the system. Otherwise the implementation of the service has to be reviewed or the middelware has to be added by extra functionality for the requirements of the service.

6 Related Work

There is currently considerable ongoing research in the area of adaptable middleware. Most approaches aim at extending existent middleware technologies by own attempts. [1] describes how to extend CORBA to gain better network adaptivity for multimedia applications. The approach is to obtain a look into black-box systems to e.g. add special algorithms for different network bindings. [11] extended CORBA for context-sensitive communication in ubiquitous environments. This approach aims at the special requirements within ad hoc networks combined with the perception of context-sensitive sensors. But both approaches have in common that they use an existing CORBA implementation and thus are not able to directly change the behavior of more than one node in a network. Their point of view is to examine only one node and to make the best efforts to optimize its situation. The approach in this paper is to move beyond the boundaries of one single node. The complete system is involved into adaption behavior and hence there are much more possibilities to meet the demands of a single service.
Concerning the classification of the BlueSpot middleware, [6] gives a detailed overview of different proceedings in composing adaptive software. To classify the kind of adaption it describes two different proceedings: the *parameter adaption* and the *compositional adaption*. The first one focuses on a advancement of the performance by changing predefined parameters. Like described in section 5.2 bandwidth can be gained by increasing the priority of a service. If the middleware triggers a routing protocol exchange, new code is loaded and thus new algorithms are included into the middleware. This is what [6] is calling compositional adaption. Further [6] examines the different constituents of an adaptive middleware. This idea goes back to [10] which decomposes a middleware into four layers: *Host-infrastructure middleware*, *Distribution middleware*, *Common middleware* and *Domain-specific middleware*. [6] says, that these four layers bridge the gap between an application program and the underlying operating systems, network protocols and hardware devices. The BlueSpot middleware stack described in section 2 also fits into this decomposition. The Network Adapter Interface and the Network Adaption Layer represent the Host-interface middleware. They hide the heterogeneity of the underlying network devices. The Protocol Layer and the Session Layer fit to both the Distribution middleware and the Common middleware. They handle fault tolerance as well as high-level programming abstraction for enabling developers to write distributed applications in a way similar to stand-alone applications. The Service Interface implements parts of the Distribution middleware as well as of the Domain-specific middleware. Services are able to connect to the middleware by implementing the interface the Service Interface provides. With the possibility to implement services in both native C code and as Java MIDlets, each service can be specialized to the demands made to the service.

7 Future Work

The further activities at the BlueSpot project will be the development of a monitoring tool. This tool will be used to illustrate the behavior of the complete system by dint of a graphical user interface (GUI). At the moment the complete code is added by hooks which send UDP massages via a LAN device to a predefined destination address. The monitor running on the destination device will record all incoming messages and present them in the GUI. It is necessary to use an extra LAN device for monitoring to influence the systems performance as less as possible. With the monitoring tool the adaptive behavior of the middleware will be obvious. Additionally different approaches i.e. the usage of different routing algorithms can be easily compared to each other and new approaches can be verified quickly. Supplementary it is necessary to do more benchmarking on the complete system especially to evaluate the different adaptive behavior efforts. These benchmarks will be presented in future publications. The only benchmarks which exists at the moment are values about the throughput and the latency times of the underlying Bluetooth Scatternets with the use of the BlueSpot Bluetooth device driver. However, these values are still not comparable to other implementations, so they will be also published after an exact verification in future. As described in section 2 further work will be done to automate the network forming. A main task is, that all nodes manage their connections on their own. The difficult in this task is to build up the preferred domain based network with the Slave-Slave nodes as bridges. E.g. a node has to realize that it is a bridge between two domains and has to turn itself into a special bridging mode. In this mode no messages are reviewed but only forwarded to the other domain to save resources and gain latency time.

References

[1] G. Blair, G. Coulson, N. Davies, P. Robin, and T. Fitzpatrick. Adaptive middleware for mobile multimedia applications. In *Proceedings of the IEEE 7th International Workshop on Network and Operating*

System Support for Digital Audio and Video, 1997., pages 245–254. IEEE, IEEE, May 1997.

[2] Bluetooth SIG. *Bluetooth Core Specification v2.0 + EDR - Piconets*, chapter 4, pages 51–58. Bluetooth SIG, November 2004.

[3] Bluetooth SIG. *Bluetooth Core Specification v2.0 + EDR - Scatternet*, chapter 8, pages 177–178. Bluetooth SIG, November 2004.

[4] U. Dümichen and U. Baumgarten. BlueSpot System Infrastructure for Mobile Services. In *Proceedings of 49. Internationales Wissenschaftliches Kolloquium (IWK) Ilmenau*, volume 1, pages 375–380. Shaker Verlag, September 2004.

[5] IETF MANET Working Group. The dynamic source routing protocol for mobile ad hoc networks (dsr), July 2004.

[6] McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, and B.H.C. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

[7] Network Working Group. Rfc 2210: The use of rsvp with ietf integrated services, September 1997.

[8] Network Working Group. Rfc2474: Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers, December 1998.

[9] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *SIGCOMM 94*. ACM, August 1994.

[10] D. C. Schmidt. Middleware for real-time and embedded systems. *Commun. ACM*, 45(6):43–48, 2002.

[11] S. S. Yau and F. Karim. An adaptive middleware for context-sensitive communications for real-time applications in ubiquitous computing environments. *Real-Time Systems*, 26(1):29–61, January 2004.