Transparent Heterogeneous Mobile Ad Hoc Networks[*]

Patrick Stuedi and Gustavo Alonso

Swiss Federal Institute of Technology (ETHZ)
Departement of Computer Science
8092, ETH-Zentrum, Switzerland
{stuedip, alonso}@inf.ethz.ch

ABSTRACT

Most research on mobile ad hoc networks assumes a medium access (MAC) protocol similar to 802.11. Nevertheless, such networks can be built on top of any medium access scheme. An example are Bluetooth multi-hop networks (so called Scatternets). In practice, devices support several interfaces although the protocol stack used in each interface tends to be interface specific at the lower layers. This limits the ability of a device to switch back and forth between networks as need and opportunity dictates. As a step towards providing a more flexible handover infrastructure, in this paper we address the issue of integrating heterogeneous, mobile ad-hoc networks that use different MAC layer protocols. The goal is to provide an end-to-end communication abstraction that hides heterogeneity. In the paper, we discuss different possible designs with regard to application transparency, performance and mobility. Using these ideas, we describe an IP based heterogeneous mobile ad hoc testbed using Bluetooth and IEEE 802.11 that implements a virtual interface approach as the end-to-end abstraction. Furthermore, we also provide a detailed evaluation of the system in terms of throughput and handover-time.

1. INTRODUCTION

Mobile ad hoc networks (MANETs) are being increasingly used due to their decentralized and dynamic nature as well as the fact that they do not require any fixed, pre-existing infrastructure. The flexibility they offer opens up many possible application scenarios where devices spontaneously interact, users can create and receive arbitrary data streams (video, music, data), and a multitude of providers specialize in different network services and contents. Such a vision is supported by the many variations of ad hoc networks already in place. The drawback is that each network type typically uses its own optimized protocol stack. This is particularly true in the case of medium access. In a *personal area network* (PAN) or *wireless sensor network* (WSN) performance issues may have less priority than for example in a conference network. In contrast, battery life and low cost is critical to PANs and WSN while most probably it is not to conference networks. This leads to diverging standards for networks that will nevertheless have to co-exist in practice.

Current media access proposals for wireless networks range from Bluetooth [4] and IEEE 802.15.4 [6] up to 802.11 [5]. Already within the 802.11 group there exist many different flavors (802.11e/g/b/a/h). Nevertheless, most research on MANETs assumes such networks to have one common MAC schema. In this paper, we discuss the integration of heterogeneous mobile ad hoc networks comprising different MAC protocols. We describe and discuss the performance of an end-to-end communication abstraction that transparently hides the heterogeneity of the underlying medium access schema while still supporting node mobility, multi-hop transmission, simple addressing, etc. Such a heterogeneous mobile ad hoc networks occurs, e.g., when combining a Bluetooth PAN with an 802.11 MANET as shown in Figure 1. An example is a university campus with all sorts of personal devices ranging from mobile phones, handhelds up to laptops. Each of these devices may be equipped with different communication technologies tailored to their capabilities and intended use. Bluetooth, e.g., is an energy-saving technology commonly used in mobile phones or handhelds. Laptops on the other hand do not have such strong constraints, they might include an 802.11 as well as a Bluetooth interface. Ubiquitously combining all these devices into one heterogenous mobile ad hoc network could invite new application and services like, e.g., location based services or VoIP. In such scenarios, a personal device of one particular PAN might communicate with a personal device of another PAN in a multi-hop fashion where the underlying MAC scheme changes on a per hop basis. Another interesting application of a heterogenous mobile ad hoc network is the integration of various personal devices into a so called digital home. In a digital home, PCs and consumer electronics could work together on an ad hoc basis to different rooms in the house using a combination of networks.

Paper Organization: The next section explores various approaches to build heterogenous mobile ad hoc networks using Bluetooth and 802.11. Section 4 presents our solution

[*]The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems NCCR-MICS. a center supported by the Swiss National Science Foundation under grant number 5005-67322.

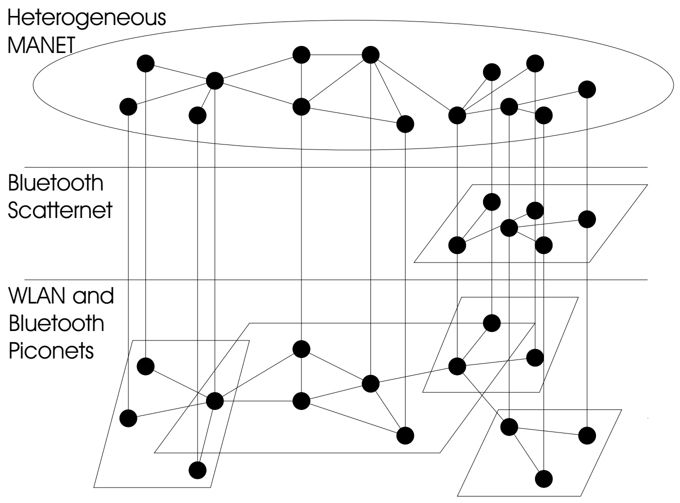


Figure 1: Heterogeneous Mobile Ad Hoc Network

which is based on virtual layer two device. In section 5 we evaluate the system and section 6 concludes the paper.

2. POLYMAC IN MANETS USING BLUETOOTH AND 802.11

As mentioned we aim at the integration of heterogeneous mobile ad hoc networks comprising different MAC protocols. Although we look for a generic solution we take a network as shown in Figure 1, including 802.11 and Bluetooth, as a base for our investigations.

802.11 is currently the most common MAC protocol used in MANETs as it has achievable data rates of up to $54Mbits/s$. From the two modes of operation only the decentralized *contention based* mode is suitable for MANETs. Medium access is then provided by a CSMA/CA protocol. Bluetooth on the other hand is optimized for low cost and only offers data rates of $\sim 1Mbit/s$. Media Access in Bluetooth is centrally organized. A master node manages the time slots within a *piconet* (collaboration between maximum 8 nodes), thereby avoiding collisions. Several piconets can be combined to a so called *scatternet*. In a scatternet packets no longer follow a predefined way towards their destination. Instead they have to be routed on a per master basis. From a more abstract perspective we could also say, a scatternet is a MANET where the piconets are the nodes. Besides these characteristics Bluetooth and 802.11 differ with regard to message broadcast. While 802.11 inherently supports broadcast, Blutooth does not support it at all [1].

After having described the specific properties of 802.11 and Bluetooth we now want to define the requirements which needs to be fulfilled by a heterogenous mobile ad hoc network. Basically we think such a network should at least satisfy the following criteria:

1. **Transparency:** The network should provide transparent end-to-end communication.
2. **Mobility:** Node mobility has to be supported.
3. **Addressing:** Addressing should be independent of how many interfaces are attached to a particular node.
4. **Configuration:** Configuring a node should be possible without any knowledge of the underlying MAC technology.

So in a nutshell, what we are looking for is an end-to-end communication abstraction that supports MAC-switching[2], node mobility and multihoming [3]. Certainly two issues to be solved are broadcast emulation and handover. Broadcast emulation since it is not directly supported in Bluetooth (nor on nodes comprising both Bluetooth and 802.11). Handover since in the case of heterogeneous mobile ad hoc networks a handover might include a change in how the medium is accessed. A handover can be caused by node mobility, a change in user preferences (the user chooses to save energy and use Bluetooth instead of 802.11), or performance reasons. We will come to that point later in section 4.

3. RELATED WORK

In principle, a communication abstraction like the one proposed in the previous section can be provided on any layer above MAC. SCTP [16], or Stream Control Transmission Protocol, is a transport protocol defined by the IETF providing similar services to TCP. It ensures reliable, in-sequence transport of messages. While TCP is byte-oriented, SCTP deals with framed messages. A major contribution of SCTP is its multi-homing support. One (or both) endpoints of a connection can consist of more than one IP addresses, enabling transparent fail-over between hosts or network cards. Since both, 802.11 and Bluetooth have IP bindings, one could think of using a common MANET routing protocol.

[1]Bluetooth 2.0 specifies an Active Slave Broadcast (ASB). But up to our knowledge there is no implementation supporting it.

[2]Refers to the fact that the used MAC technology may change along a source/destination path

[3]A node having multiple network interfaces

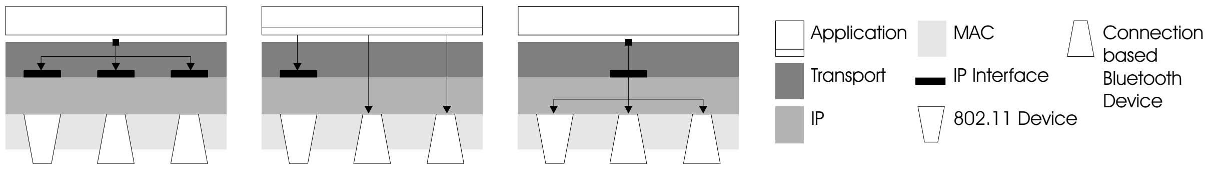


Figure 2: Communication Abstraction: (a) Transport Layer (b) Application Layer (c) IP Layer

Each interface could be separately configured and maintained (AODV-UU [8], e.g., supports multiple interfaces). This solution (see Figure 2a) seems to be quite promising in terms of performance since SCTP optimizes the transmission over multiple links. In fact, if one particular node can be reached through several interfaces, SCTP switches transmission from one interface to another after a predefined number of missing acknowledgements. Unfortunately, the solution lacks transparency. Applications running traditional unix sockets would have to be changed to use SCTP sockets instead. Another problem arises with the connection oriented nature of Bluetooth. In Bluetooth, interfaces appear and disappear dynamically depending on whether the connection to the specific node is currently up or down. Therefore, this is something that both the ad hoc routing protocol as well as SCTP would have to cope with. Recently, the Network Working Group at the IETF proposed ASCONF [11], a feature that allows dynamic reconfiguration of IP addresses during an SCTP session. On the routing layer, however, we are not aware of any protocol that supports dynamic interfaces. Typically, these protocols expect the network interfaces to be configured statically within the system. And even if routing modules were able to handle dynamic interfaces, still the mapping between the nodes and IP addresses would have to be solved. Instead of using an all-IP communication based on SCTP and a MANET routing protocol, one could as well think of an application layer solution (see Figure 2b). In [13], the authors propose a communication framework based on JXTA [3, 10] that is in charge of transporting packets end-to-end by choosing the right transport mechanism below, i.e 802.11/IP or raw Bluetooth. This allows the use of separate optimized routing algorithms for IP and Bluetooth respectively. Unfortunately, this solution goes at the expense of performance. How should the application layer framework decide which subsystem best to send the packet to without using flooding? Furthermore, application transparency is not given at all. Rather, transparency can be provided when multihoming is implemented on a lower layer. Imagine a single IP interface (virtual interface) handling different physical interfaces (see Figure 2c). Neither the application nor the routing would be affected by the fact that eventually a node consists of multiple interfaces belonging to different MAC technologies. But how can such an interface know about the real physical interface to send the packet to? How feasible is such a solution? The following sections not only shows that such a solution is feasible, but also that it performs well.

4. A MAC LAYER APPROACH

As discussed, a *virtual interface* approach is the most convenient solution for heterogeneous mobile ad hoc networks in terms of transparency. The question now is how to implement it and how to make it perform well.

4.1 Linux Ethernet Bridge

The solution we propose here is inspired by the linux ethernet bridge [7, 1]. Similar to a physical bridge device, the linux ethernet bridge ties separate layer-two-networks together, only that it is purely software. The bridge includes much of the functionality we would like to see in a *virtual interface* as described at the end of section 2. Appearing to the operating system as a regular layer-two-device one can easily assign IP addresses to bridges. The bridge is supposed to work in combination with 802.x devices, therefore not including Bluetooth. Fortunately, the Bluetooth personal area network (PAN) profile specifies BNEP [2] which itself defines a packet format to transport common networking protocols over the Bluethooth media. BNEP supports the same networking protocols that are supported by IEEE 802.3/Ethernet encapsulation, therefore enabling Bluetooth to be used within the bridge. Layer-two-bridges have the well known property of forwarding packets to the correct physical network corresponding to the destination MAC address. So the bridge's decision for a frame is one of these:

- bridge it, if the destination MAC address is on another side of the bridge
- flood it over all the forwarding bridge ports, if the position of the box with the destination MAC is unknown to the bridge
- pass it to the higher protocol code (IP) if the destination MAC address is that of the bridge or of one of its ports
- ignore it if the destination MAC address is located on the same side of the bridge

As the bridge learns about the neighboring nodes and their MAC addresses, it is capable of forwarding the packet to the right physical interface instead of just flooding the packet. It does so by keeping a neighborhood database (NDB) mapping destination MAC addresses to physical interfaces. Entries are refreshed by incoming packets, otherwise they time out.

4.2 The virtual Interface

Since we are looking for a virtual interface, we are not interested in packet forwarding, but the idea of storing a MAC/interface mapping based on incoming packets is also suitable for local traffic. We have implemented a virtual interface (*vi*) that adopts this mechanism. Like the linux ethernet bridge, the *vi* represents a regular layer-two-device and can be configured accordingly. The *vi* allows to plug in any 802.x compatible network device, like e.g a wireless LAN card or a BNEP/Bluetooth connection, while hiding

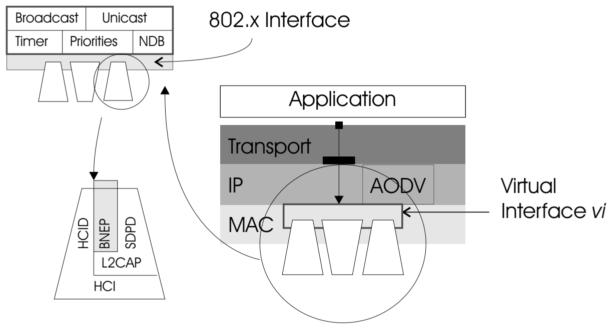
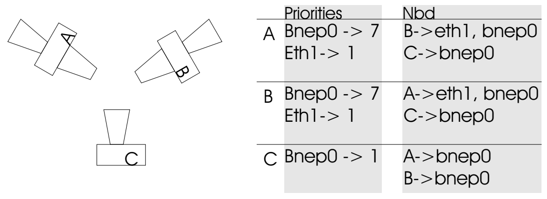


Figure 3: Virtual Interface Approach



	Priorities	Nbd
A	Bnep0 -> 7 Eth1-> 1	B->eth1, bnep0 C->bnep0
B	Bnep0 -> 7 Eth1-> 1	A->eth1, bnep0 C->bnep0
C	Bnep0 -> 1	A->bnep0 B->bnep0

Figure 4: The Neighborhood Database (NDB)

the heterogeneity of the used devices from the upper layers. For every neighboring node, the *vi* holds an array of possible outgoing interfaces. The number of entries in the array follows directly from the intersection of the two sets of interfaces. An entry contains a timestamp and is created upon receiving the first packet (i.e a routing broadcast message or a route reply) of the associated neighbor/interface pair. Every consecutive incoming packet refreshes the timestamp. If the *vi* receives a packet from the upper layer for delivery, it first checks the packet type. In case the packet is a broadcast packet, it will be sent through all available interfaces. Therefore, the *vi* also acts as a broadcast emulation layer for Bluetooth. This is particularly important since BNEP is point-to-point. However, if the packet is unicast, the *vi* looks for the corresponding entry in the neighborhood database mentioned above and retrieves the information about the interface the packet has to be sent to (entries are periodically checked for expiration). If there is more than one option, the *vi* makes use of another feature, the so called priority table. The priority table specifies some sort of ranking among the interfaces, meaning that whenever a given neighbor can be reached through several interfaces, the interface with the lowest priority is taken. Figure 3 illustrates the architecture of the virtual interface and how it is embedded within the network stack. The state of the neighborhood database and priority table in a scenario where three nodes form a neighborhood (they are within transmission range of each other) is shown in Figure 4. Apart from the priority table, another mechanism is needed to prevent links from becoming stalled. Imagine two nodes, each of them containing an 802.11 as well as a Bluetooth interface. Obviously, there are two ways for the two nodes to exchange packets, therefore the corresponding neighborhood database contains two entries. Further, assume the Bluetooth interface to have the highest priority on the sender node (the node might be battery driven and tries to minimize energy consumption per packet transmission). If the Bluetooth connection temporarily suffers from very bad link properties, the node probably wants to switch to the 802.11 interface at the expense of a higher energy consumption. This will not be possible under the current setup as the *vi* always uses the interface with the highest priority to transmit the packet. To overcome this unfavorable situation, a cleanup timer, which periodically removes all expired entries, could be used (an entry is expired if it has not been refreshed for a certain amount of time), but this would unnecessarily increase routing traffic since the next packet in the queue right after the cleanup is triggering a route request. Instead of a cleanup timer, we therefore use a different mechanism that is sort of routing-friendly. We introduce a new parameter that is associated with a *vi*, the so called *maxdiff* threshold. The *maxdiff* threshold unit is 10ms and it decides how much two single entries within the neighborhood database may differ in terms of timestamps to keep the priority policy up. So a higher ranked interface entry can be replaced by a lower priority interface if the timestamp differs for more than *maxdiff*. For a more detailed look see Algorithm 1.

Algorithm 1 Local packet processing

```
receiving_packet(vi, p);
if p.type = BROADCAST then
    interfaces := get_interfaces(vi);
    for all $i$ in interfaces do
        transmit($i$, $p$);
    end for
else
    current := ndb[hashval(p.mac.dst_address)];
    outgoing := NULL;
    tmp = current;
    while current $\neq$ NULL do
        if outgoing = NULL then
            outgoing := current;
        else if current.priority < outgoing.priority then
            $\Delta$ := outgoing.ts - current.ts;
            if $\Delta$ < vi.maxdiff then
                outgoing := current;
            end if
        else
            $\Delta$ := current.ts - outgoing.ts;
            if $\Delta$ > vi.maxdiff then
                outgoing := current;
            end if
        end if
        current := current.next;
    end while
end if
```

4.3 Configuration

The virtual interface consists of a kernel module and a user-mode command-line tool [4]. The kernel module immediately gets loaded when creating a new virtual interface using the *victl addvi* command. There are also commands to add and remove devices to/from the virtual interface as well as to set the interface priorities. Table 6 shows how a virtual interface containing Bluetooth and WLAN is typically set up. Like a regular layer-two-interface, the IP configuration of vi can be performed with *ifconfig*. The question that remains is how to dynamically add Bluetooth devices to the *vi* (remember that they appear on a connection basis). Apparently, we need some help from Bluez [9] here. A script is called whenever a Bluetooth interface is created. We then simply let the *vi*

[4]Furthermore a C API is provided for the case where the application directly wants interact with the *vi*.

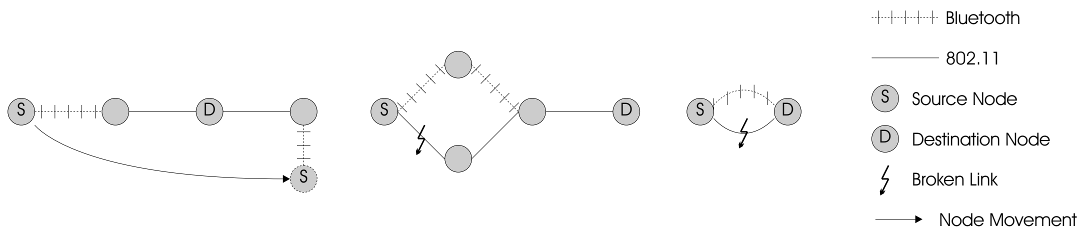


Figure 5: Horizontal (1), Diagonal (2), and Vertical (3) handover

maintenance be triggered by this event, as shown in Figure 7. Devices are added and removed to/from the *vi* every time they appear or disappear. Another issue to take care of is the MAC address of the *vi*. The MAC address can manually be set using the *victl addr* command. This address can be totally virtual or correspond to a physical interface.

```
[root@somedesk]# victl help
commands:
    help                            command list
    addvi       <vi>                add vi
    delvi       <vi>                delete vi
    addif       <vi> <device>       add interface
    delif       <vi> <device>       del interface
    setdiff     <vi> <maxdiff>      set maxdiff
    setprio     <vi> <port> <prio>  set priority
    addr        <vi> <mac-addr>     set mac address
    show        <vi>                list of vi's
[root@somedesk]# victl add vi0
[root@somedesk]# victl addif vi0 eth1
[root@somedesk]# victl setprio vi0 eth1 80
[root@somedesk]# victl addif bnep0
[root@somedesk]# victl setprio vi0 bnep0 100
[root@somedesk]# ifconfig eth1 0.0.0.0
[root@somedesk]# ifconfig vi0 192.168.220.2
[root@somedesk]# victl addr vi0 00:02:72:B2:78:DC
[root@somedesk]# victl setdiff 200
[root@somedesk]# ifconfig
bnep0 Link encap:Ethernet  HWaddr 00:02:72:B2:78:D2
      UP BROADCAST RUNNING MULTICAST  MTU:1500 METRIC:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:4 errors:0 dropped:0 overruns:0
      collision:0 txqueuelen:100
      RX bytes:104 (104.0 b) TX bytes:88 (88.0 b)

eth1  Link encap:Ethernet  HWaddr 00:02:2D:7B:88:D1
      UP BROADCAST RUNNING MULTICAST  MTU:1500 METRIC:1
      RX packets:1093 errors:277 dropped:0 overruns:0
      TX packets:51 errors:0 dropped:0 overruns:0
      collision:0 txqueuelen:100
      RX bytes:65778 {64.2 Kb} TX bytes:11386 {11.1 Kb}
      Interrupt:11 Base address:0x100

vi0   Link encap:Ethernet HWaddr 00:02:72:B2:78:DC
      inet addr:192.168.220.2 Bcast:192.168.220.255
      UP BROADCAST RUNNING MULTICAST  MTU:1500 METRIC:1
      RX packets:0 errors:0 dropped:0 overruns:0
      TX packets:0 errors:0 dropped:0 overruns:0
      collisions:0 txqueuelen:0
      RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)

[root@somedesk]# victl show vi0
virtual interface maxdiff interfaces  priority
vi0               200     bnep0       100
                          eth1        80
[root@somedesk]# □
```

Figure 6: Setting up a virtual interface

```
#!/bin/sh
victl addif vi0 $1
victl setportprio vi0 $1 80
ifconfig $1 0.0.0.0
```

Figure 7: /etc/bluetooth/pan/dev-up

Outgoing packets always adopt the MAC address of the *vi*, regardless of the interface they leave. Some WL cards do not allow MAC addresses to be overwritten, so the address for the *vi* is best chosen to correspond to the one of the WL card. On the contrary in a *vi* comprising Bluetooth interfaces only, a virtual MAC address is more suitable since otherwise, communicating nodes would have to adapt their ARP binding every time the MAC address changes.

4.4 Handover

A handover includes route changes as well as MAC switching. In principle, there are three possible scenarios (see also Figure 5):

1. **Horizontal Handover:** the route changes while the underlying MAC technology remains the same
2. **Vertical Handover:** the route does not change but the given neighbor is now reached through a new physical interface
3. **Diagonal Handover:** MAC technology and route change simultaneously

Let us now take a closer look at how routes get established in a heterogeneous mobile ad hoc network when using virtual interfaces. In the case of a reactive routing protocol, it is the application that triggers a path setup. Since there is no route available yet, the routing protocol typically first broadcasts a route request. At the very beginning the neighborhood database contains no entries but the transmission of a broadcast packet does not need any neighborhood information anyway (see section 4). After the route request has passed several hops, a route reply eventually returns back to the origin. The route reply not only establishes the route but also creates an entry within the neighborhood database, providing the *vi* with information on the interface to which the packets to the given neighbor have to be transmitted. In the case of a pro-active routing protocol, things are slightly different. Here nodes periodically broadcast their neighboring information and therefore are also creating entries within neighborhood databases. In both cases - proactive and reactive - the NDB entry is established in combination with the new route, regardless of whether the MAC technology changes or not. In a vertical handover on the other hand, the NDB is affected differently. Either the priority table changes due to user preferences or certain entries expire, meaning their timestamp difference exceeds the threshold *maxdiff* (bad link quality). Obviously, the smaller *maxdiff*, the more agile the virtual interface is for handover. But a small *maxdiff* threshold can also be unfavorable since it is a counter movement to the priority policy. If the priorities reflect the bandwidth of the interfaces, then a small *maxdiff* threshold may lead to a diminishing throughput, as we will see in section 5.

5. SYSTEM EVALUATION

This section describes some of the experiments in a heterogeneous environment based on the *virtual interface*. Our setup consists of up to five DELL Latitude laptops. We used cisco aironet as 802.11 network cards and Acer Bluetooth dongles implementing the Bluetooth 1.1 specification.

5.1 Handover

In a first set of experiments we want to explore the handover quality. An exact measurement of the handover-time is difficult since it requires detailed information about the time the link disappears and the new routing entry is inserted. To simplify the measurement process we used the *ping* command to measure the lost packets. As *ping* triggers a packet transmission every second and as it is UDP based, there is almost a one-to-one relationship between the lost packets and the handover time. In order to take the impact of the routing module on the neighborhood database into account, we measured along three tracks. Once using AODV [15] representing the reactive class, once OLSR [12] as a proactive routing protocol and once without routing module at all. Measurements represent average values taken from 10 samples and the standard deviation is shown as well. Throughout the measurements we use *raw* to describe scenarios not including the virtual interface. On the contrary we use *vi/x* to indicate scenarios including the virtual interface with a *maxdiff* threshold of *x*. The results are shown in Table 1. Apart from measurements marked as 'priority driven', the priority table reflects the bandwidth proportions of the associated interfaces, therefore 802.11 always has the highest priority.

Let us look at the horizontal handover first. We trigger a route change by physically detaching the interface of a node. Under our setup we measured a packet loss of roughly 2 packets when the route changed from one hop to another, regardless of which routing protocol was used and whether a *vi* was included or not. But note that OLSR is not as stable as AODV (its standard deviation is higher). This is due to the fact that proactive routing protocols (like OLSR) do not react to route failures directly but broadcast their routing information on a fixed schedule instead. This behavior is even amplified in a pure Bluetooth network [5]. Here we observe a difference in packet loss between the two protocols of 10 packets (34.6 for AODV and 44.3 for OLSR). Again, the standard deviation of OLSR is higher than the one of AODV. For the diagonal handover - we assume the Bluetooth connection already being established on the MAC level - a similar picture appears, OLSR is always a little behind AODV while being less stable. However, changing the route simultaneously with the MAC technology does not seem to introduce additional overhead in the AODV case. For the vertical handover (no route change, but interface switching only) we now also take the *maxdiff* threshold into account since it influences the agility in terms of the handover decision. Furthermore, we introduce *priority driven* handover. While in all other experiments a handover is triggered by the environment (in our scenario by physically detaching an interface), *priority driven* means the interface ranking is changed (using *victl setportpriority*) during a *ping* session. From the results given for AODV and OLSR in Table 1 we see that packet loss increases with increasing *maxdiff* threshold and almost disappears for priority driven handover. The former is reasonable because the bigger the *maxdiff* value, the more the priority policy gets enforced, and a pure priority driven MAC switching would not lead to any switching at all. The latter approves our idea of implementing MAC-switching transparent at the lowest layer possible. We also tested the impact of *maxdiff* values on priority driven handover. As expected, the smaller *maxdiff* gets the less stable the handover becomes. However, in our scenario a *maxdiff* value of 5-10 was sufficient to guarantee stable handover while changing interface priorities. The very last part in Table 1 refers to vertical handover measurements without a MANET routing module. Here packet loss does not increase with increasing *maxdiff* threshold and no value is assigned to priority driven handover. This can be explained simply as follows. If no routing module exists, populating the Neighborhood Database (NBD) will occur as a side-effect of ARP (Address Resolution Protocol). After a link gets broken, the next upcoming ARP request, or to be concrete, the following ARP reply message creates a neighborhood entry within the NDB. Therefore, the handover time is a direct function of ARP request cycles, which is roughly 20 seconds in our example. The reason why priority driven handover doesn't work without routing module, is because no broadcast messages are received by the sending node. Typically, the NDB then contains only one entry.

5.2 Throughput

In the second experiment we measured the overhead produced by the virtual interface in terms of throughput. Our setup consists of two laptops, both equipped with Bluetooth (∼1Mbit/s) and a Cisco Aironet 350 Wireless LAN card (11Mbit/s). We measure both UDP and TCP throughput using the *iperf* [14] bandwidth measurement tool. From the table 2 we see that a pure 802.11 connection achieves a throughput of up to 5Mbit/s for TCP traffic and around 6Mbit/s for UDP. UDP is known to have a better performance in wireless networks since TCP's congestion control algorithm does not handle the unstable wireless link well. The same experiment, but including the *vi* on top, shows a comparable result. There seems to be almost no overhead produced by *vi*. This also applies to the Bluetooth case. An interesting result can be observed for the case where both machines are multihomed (WL/BT). While a low *maxdiff* threshold does not affect TCP traffic - neither for OLSR nor for AODV - it does so for UDP. The explanation is simple, TCP is acknowledgement based and therefore periodically refreshes the high priority entry within the NBD

[5]Note that after the route gets broken, a bluetooth inquiry is triggered in order to establish a new connection, which is a quite time consuming process.

Type	From	To	Routing	Mode	Avg. Packet Lost	Stddev
Horizontal	WL	WL	AODV	raw	2	0
				vi	2	0
			OLSR	raw	2.1	0.7416
				vi	2.1250	0.8539
	BT	BT	AODV	vi	34.6	2.67
			OLSR	vi	44.3	8.06
Diagonal [§]	WL	BT	AODV	vi	2.2	0.42
			OLSR	vi	4.5	1.28
Vertical [§]	WL	BT	AODV	priority driven	< 1	0
				vi/1	0.6	0.2108
				vi/200	1.4	0.3944
				vi/500	4.3	0.2581
				vi/1000	9.6	0.2108
			OLSR	priority driven	< 1	0
				vi/1	< 1	0
				vi/200	2.2	0.5374
				vi/500	4.3	0.7888
				vi/1000	10.25	0.6770
			—	priority driven	–	–
				vi/1	20.1	0.316
				vi/200	20.2	0.421
				vi/500	20	0
				vi/1000	20.1	0.316

Table 1: Handover-time

Routing	From	To	Mode	Protocol	Throughput [Mbit/s]	Stddev
AODV	WL	WL	raw	TCP	5.029	0.0190
				UDP	6.019	0.008
			vi	TCP	5.019	0.0056
				UDP	5.984	0.066
	BT	BT	raw	TCP	0.4854	0.053
				UDP	0.4791	0.0677
			vi	TCP	0.4403	0.0538
				UDP	0.4830	0.05
	WL/BT	WL/BT	vi/1	TCP	5.018	0.0044
			vi/25	TCP	5.02	0.0044
			vi/200	TCP	5.021	0.0042
			vi/1	UDP	2.928	0.621
			vi/25	UDP	5.249	0.46
			vi/200	UDP	6.024	0.005
OLSR	WL/BT	WL/BT	vi	TCP	5.01	0.0105
			vi/1	UDP	2.034	0.667
			vi/25	UDP	5.154	0.538
			vi/200	UDP	5.816	0.224

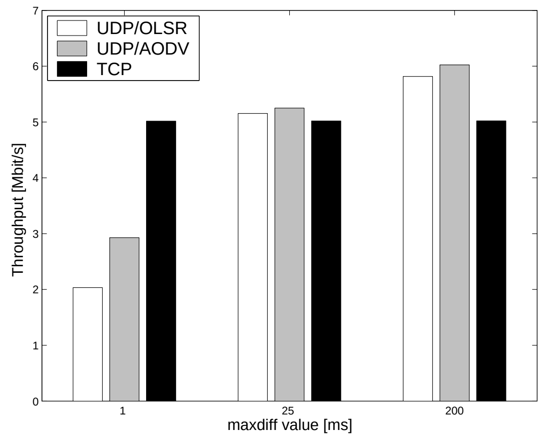
Table 2: Throughput

[§]The results were taken from measurement where medium access changes from 802.11 to Bluetooth. However, the Bluetooth-to-802.11 case has shown similar results

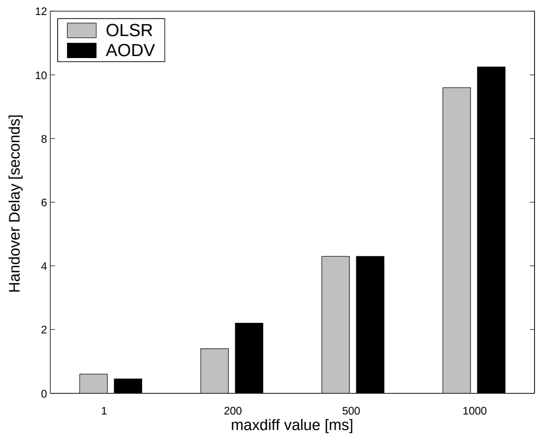
(WL in this case) during sending. UDP on the other hand does not know about acknowledgements and thus the entry within the NDB is only refreshed when it receives routing broadcast packets or ARP replies. This is causing the *vi* to use the lower priority interface occasionally which results in decreased throughput, especially if the *maxdiff* threshold is low.

5.3 Summary

The measurements show that the system performs well. There is almost no overhead when using the additional *vi* on top of a physical interface. Generally, one can say that the *vi* performs better in combination with AODV than with OLSR. Except for the case of priority driven handover, the *vi* also works without a MANET routing module. However, a routing module increases the performance in terms of packet loss during handover. In the case of multiple physical interfaces there is a trade-off between agility in terms of vertical handover and throughput for UDP traffic. A big *maxdiff* threshold is favorable for throughput but at the expense of handover agility, see Figure 8. This is due to the interface decision mechanism shown in Algorithm 1 that selects the outgoing interface for a given packet according to the interface priority and the up-to-dateness of neighborhood



(a) Throughput



(b) Handover Time

Figure 8: The *maxdiff* trade-off

database entries. The weighting of these two factors is determined by the *maxdiff* value.

6. CONCLUSIONS

In this paper, we have proposed an end-to-end communication abstraction that can be used in heterogeneous mobile ad hoc networks. Such networks are characterized by different MAC technologies used among the nodes. The proposed solution is based on a *virtual interface* (vi) approach. In the paper we have shown how the virtual interface can be used to transparently integrate Bluetooth and 802.11 into one single IP-based MANET. Note that our solution is not bound to 802.11 or Bluetooth, but works together with any 802.x-compatible MAC Layer. The *vi* in combination with a MANET routing protocol supports multihoming, dynamic reconfiguration and node mobility. The experiments presented demonstrate the feasibility of the abstraction and its potential in building heterogeneous ad-hoc wireless networks.

7. REFERENCES

[1] Linux ethernet bridge. http://bridge.sourceforge.net.

[2] Bluetooth Network Encapsulation Protocol, June 2001. http://grouper.ieee.org/groups/802/15/Bluetooth/-BNEP.pdf.

[3] JXTA v2.0 Protocols Specification, Oct. 2003. http://spec.jxta.org/nonav/v1.0/docbook/-JXTAProtocols.html.

[4] Specification of the Bluetooth System, Core Package version 1.2, Nov. 2003. https://www.bluetooth.org/spec/.

[5] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, June 2003. http://standards.ieee.org/getieee802/download/802.11-1999.pdf.

[6] Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), Oct. 2003. http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf.

[7] Media Access Control (MAC) Bridges, June 2004. http://standards.ieee.org/getieee802/802.1.html.

[8] AODV-UU. http://core.it.uu.se/AdHoc/AodvUUImpl.

[9] Bluez, official linux bluetooth protocol stack. http://www.bluez.org.

[10] D. Brookshier, D. Govoni, and N. Krishnan. *JXTA: Java P2P Programming*. SAMS, Mar. 2002.

[11] M. J. Chang, M. J. Lee, and S. J. Koh. Address management for mobile sctp handover. Internet Draft, October 2004.

[12] T. Clausen, P. J. (editors), C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L.Viennot. Optimized link state routing protocol (olsr). RFC 3626, October 2003. Network Working Group.

[13] A. Frei and G. Alonso. A dynamic lightweight architecture. *Proceedings of the 3rd International Conference on Pervasive Computing and Communications*, 2005.

[14] Iperf Project. http://dast.nlanr.net/Projects/Iperf.

[15] C. Perkins, N. R. Center, E. Belding-Royer, S. B. S. Das, and U. of Cincinnati. Ad hoc on-demand distance vector (aodv) routing. RFC 3561, July 2003. Network Working Group.

[16] R. Stewart, Q. Xie, and K. Morneault. Stream control transmission protocol. *RFC: 2960*, Oct 2000.