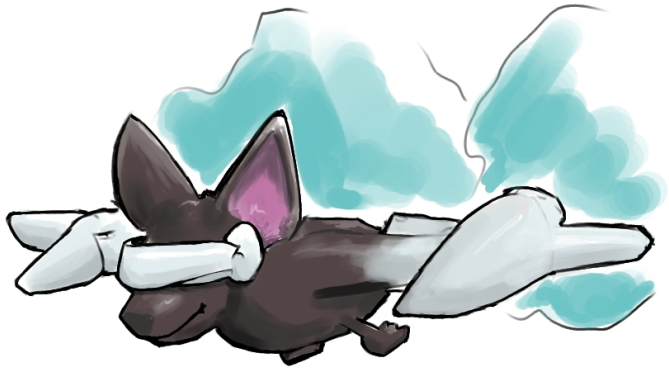
Wireless Kernel Tweaking

or how B.A.T.M.A.N. learned to fly

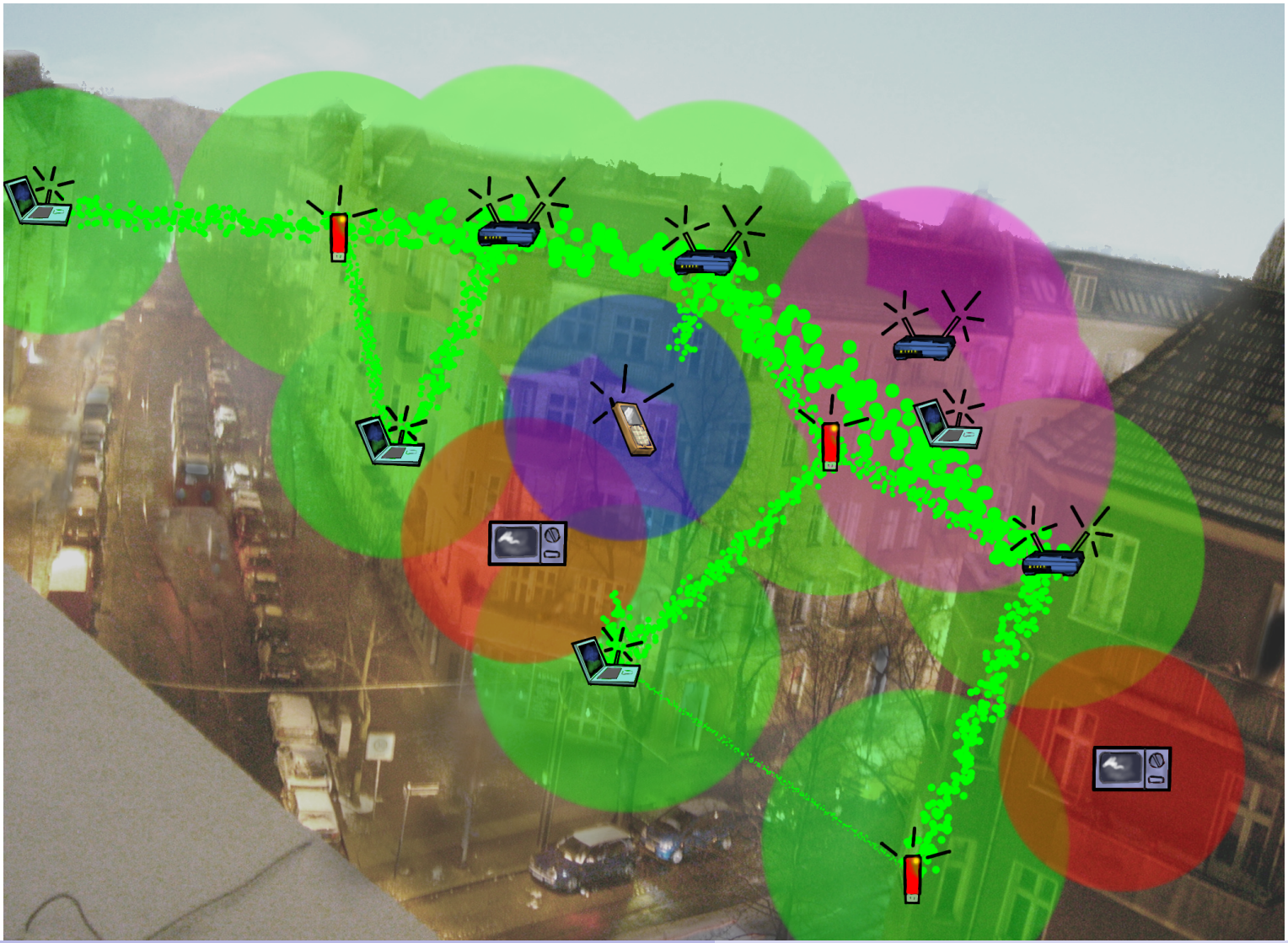
Marek Lindner, Simon Wunderlich

December 28, 2007

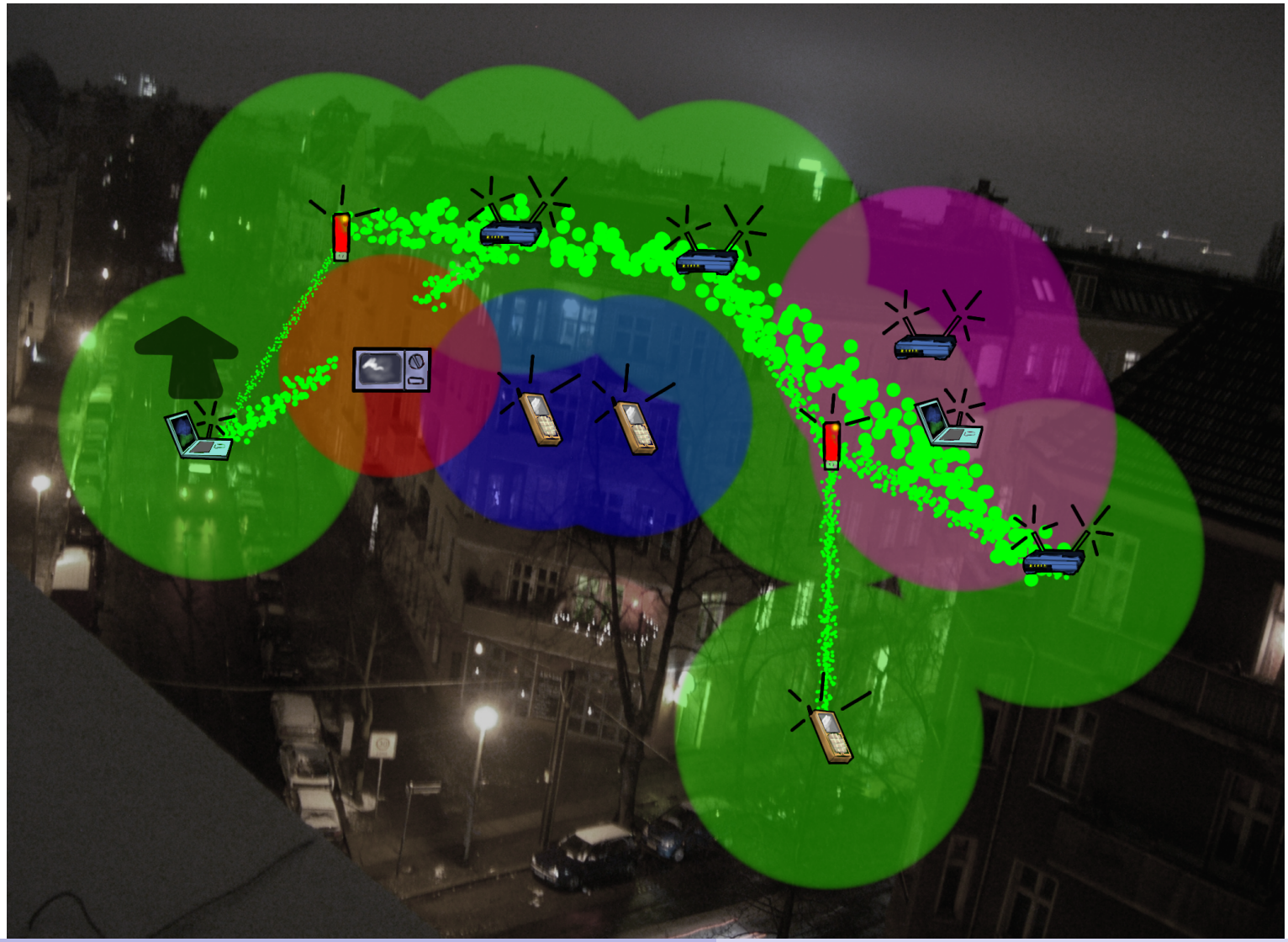


Outline

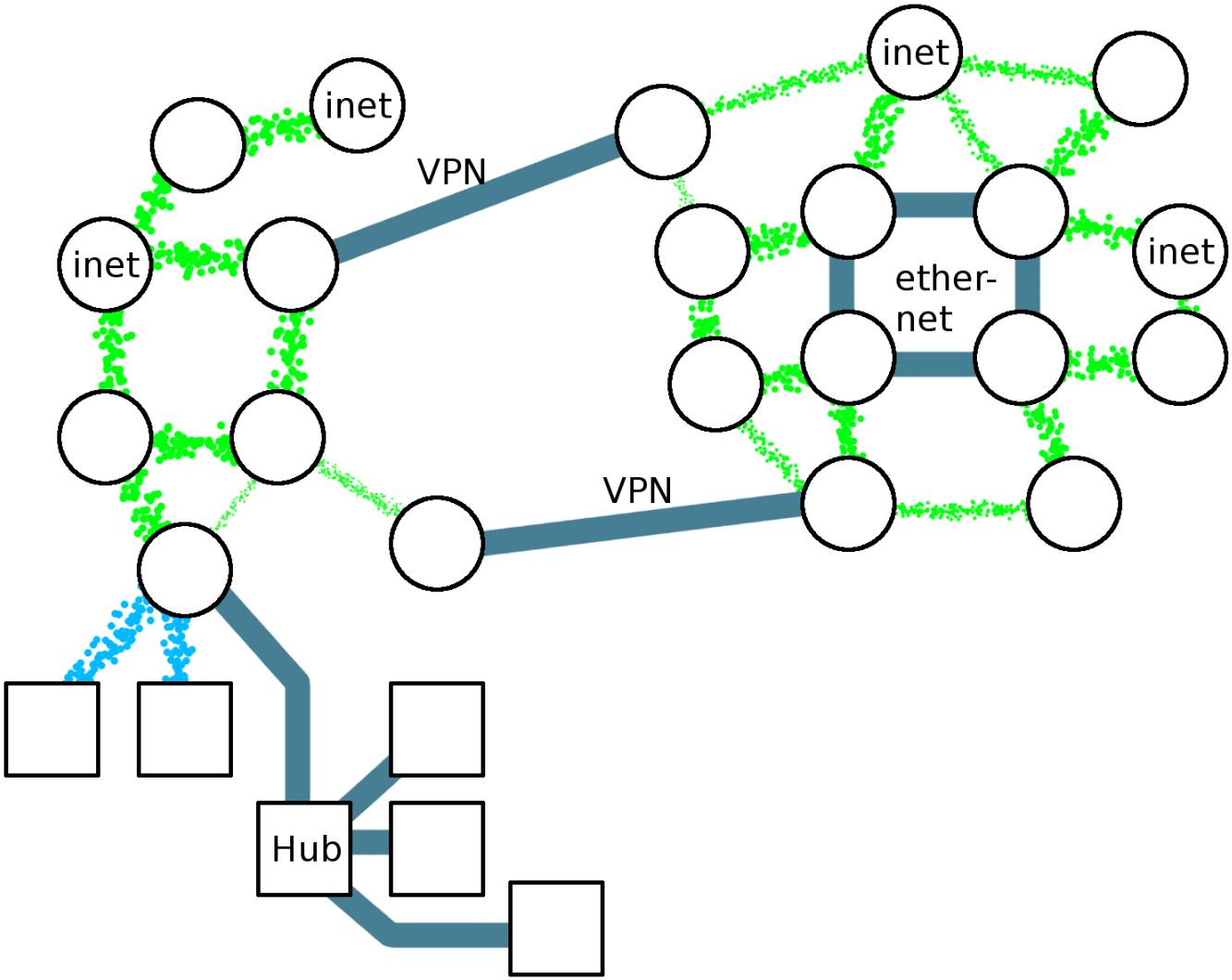
Example scenario - 6:00



Example scenario - 23:00



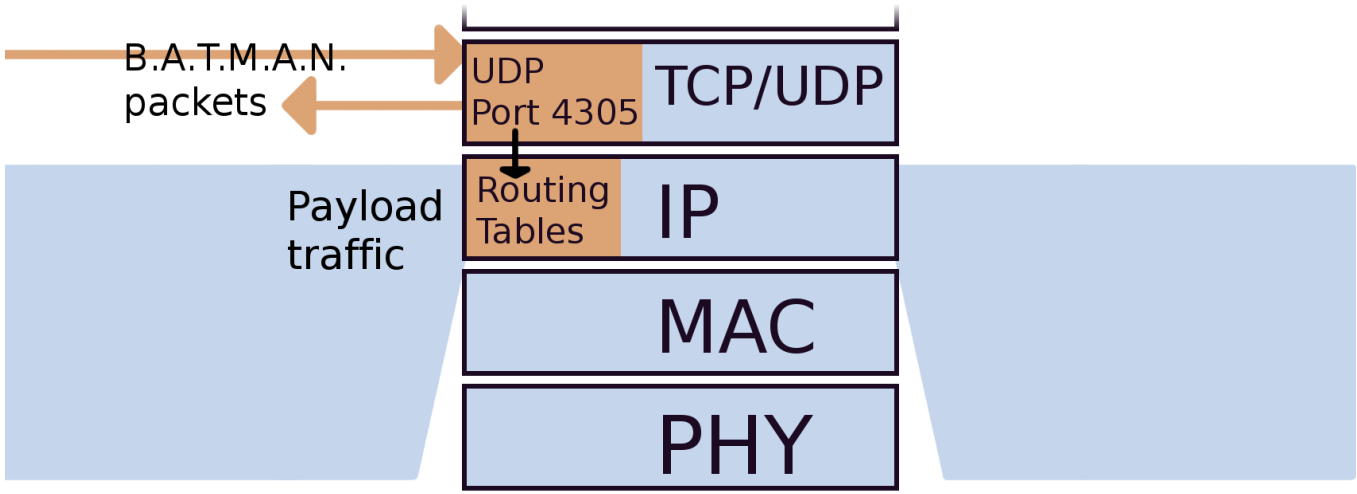
Example scenario (2)



Introduction to B.A.T.M.A.N.

- B.A.T.M.A.N. = better approach to mobile adhoc networks
- only decide next neighbour, not whole route
- topology is not used or known by nodes
- routing decisions are distributed by the nodes
- designed for lossy networks
- routing protocols internal is out of scope, we just assume it works ;)

Layer 3 - isn't that enough?



- B.A.T.M.A.N. alters routing tables
- kernel manages routing of payload traffic
- this works only for IP, no IPv6, DHCP, IPX ...
- users have to make sure that everyone has an unique IP
- routing into/outside other networks is quite complex

Let's try layer 2

- write userspace proof-of-concept, then go to kernelspace
- instead of IPs, use MAC-addresses as identifiers (should be[TM] unique per design)
- we provide a virtual switch-port "bat0" to the user
- virtual Ethernet interface (TAP), all other nodes are just one (virtual) hop away
- IP, IPv6, DHCP, IPX already works on Ethernet, we have nothing to do
- can be used as bridge over multiple interfaces (e.g. WiFi and Ethernet)

Usage

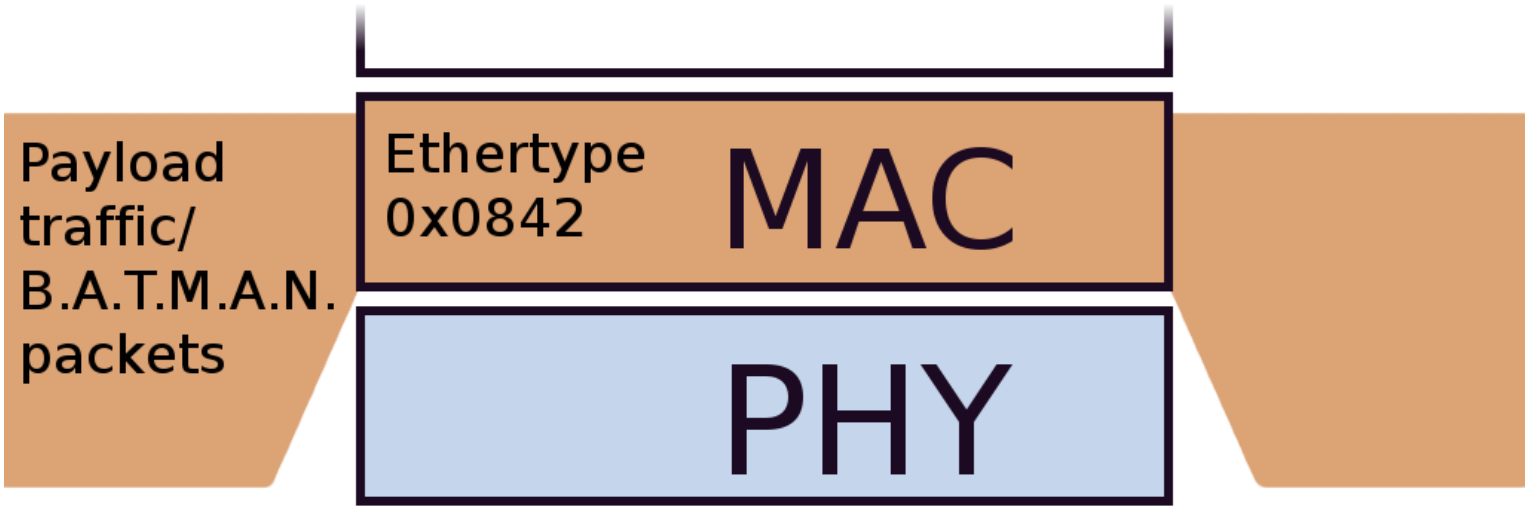
- provide a virtual switch-port "bat0" to the user

```
kero:/# ifconfig bat0
bat0      Link encap:Ethernet  HWaddr 00:13:37:91:42:37
          inet6 addr: fe80::217:13ff:fe37:4237/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1472  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:328 (328.0 B)
```

- participants set IP adresses (etc.) on their bat0 interface

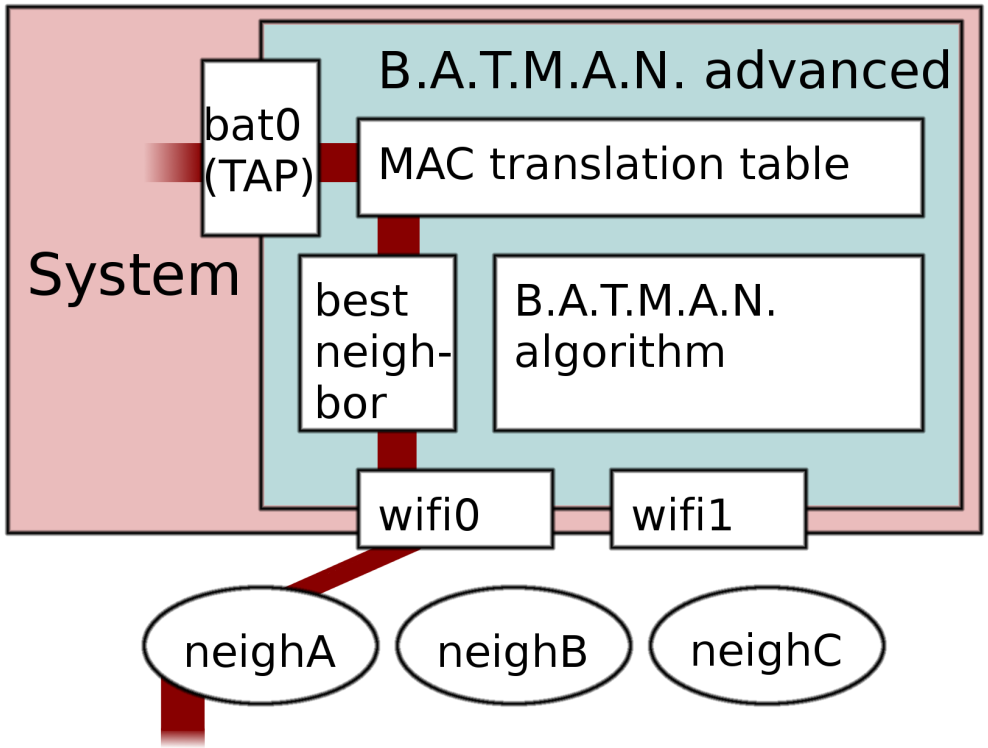
```
kero:/# ifconfig bat0 inet 192.168.10.23
kero:/# route add default gw 192.168.10.23
(or even better:)
kero:/# dhclient bat0
```

All the layer 2 belong to us!



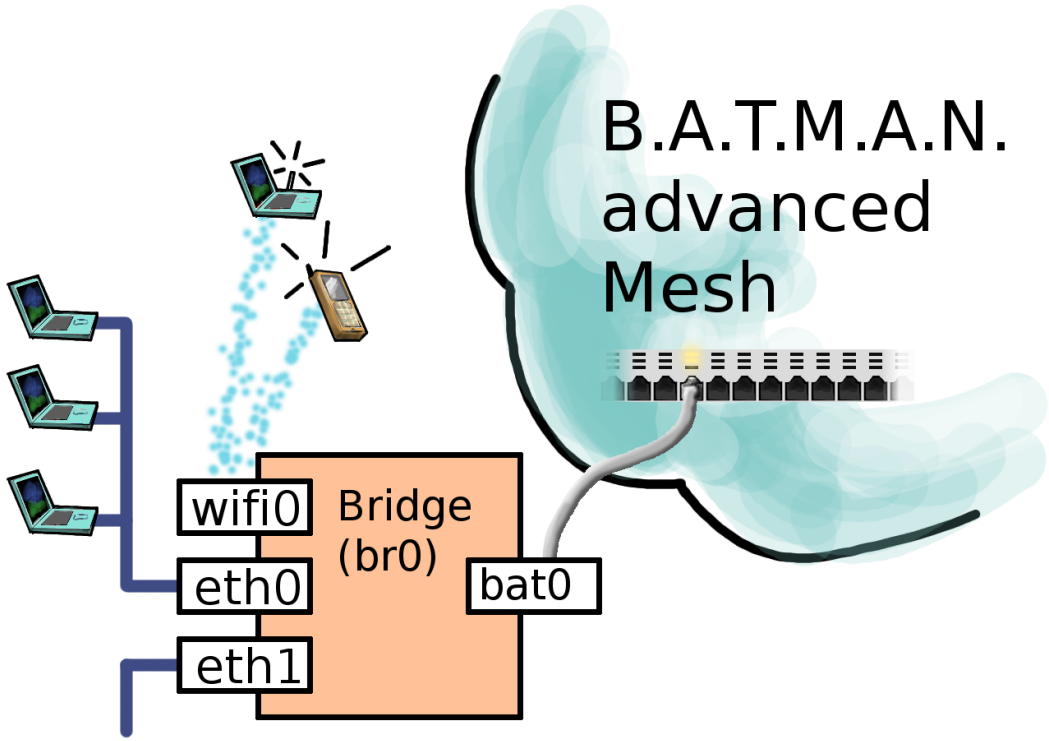
- B.A.T.M.A.N. transports the Ethernet-Frame to the node with the destination MAC
- it does not care about IP-adresses etc, just as your switch
- OGMs and payload are encapsulated in our own Ethernet-Frames (Ethertype 0x0842)

Implementation



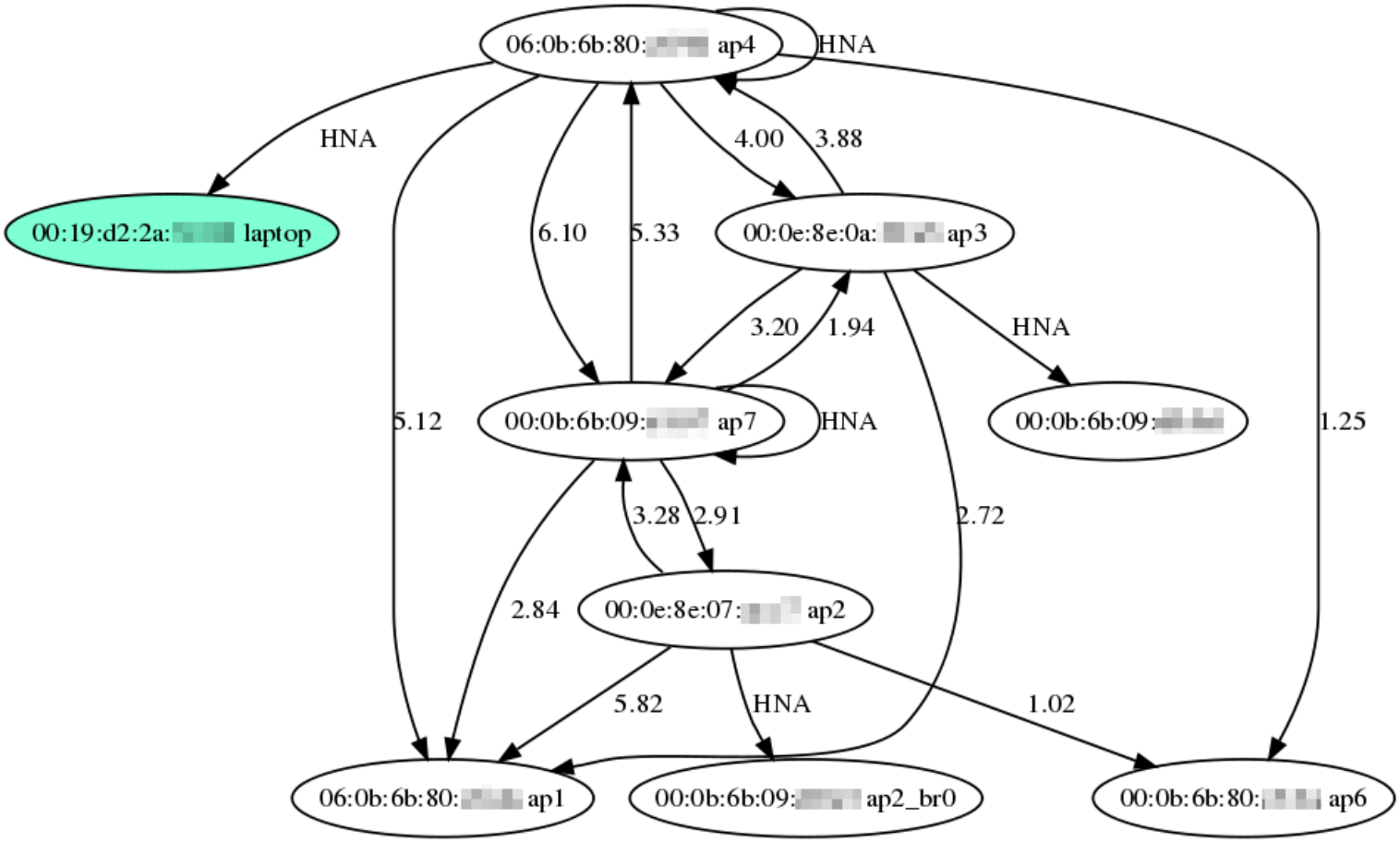
- TAP-interface bat0 receives/sends Ethernet-Frames from the user
- we decide which neighbour should receive it, based on the B.A.T.M.A.N. algorithm

Bridging support



- B.A.T.M.A.N. collects MACs of participants behind the Bridge
- These lists are announced via HNA-Messages and flooded to all B.A.T.M.A.N. nodes
- With this, we have a decentralized MAC Translation Table

Visualization



- Nice side effect: with the HNA information, the whole topology with the nodes behind the APs becomes visible

Great - and now?

- proof-of-concept implementation in the userspace works quite well
- the problem is: performance!
- it should also run well on minimal embedded systems (Access Points, Cell Phones)
- typical path is:
  - select(): wait for a packet
  - read() it
  - find next hop, update tables etc. (pretty fast)
  - write() it
- System Calls for read/write take very long time (switch to kernel mode and back, copy overhead)
- becomes a problem with high bandwidth usage, peak performance of the NICs can't be reached

Put it into kernelspace

- No useless message copy (recycle kernel buffers)
- no Syscalls and no user/kernel mode switch
- kernel works asynchronous and preemptive
- asynchronous packet handling possible

Living in the kernelspace

- the proc filesystem

```
# ls /proc/net/batman-adv/
gateways
interfaces
log
log_level
originators
orig_interval
```

- activating batman-adv

```
# echo wlan0 > /proc/net/batman-adv/interfaces
```

- deactivating batman-adv

```
# echo "" > /proc/net/batman-adv/interfaces
```

Logging merits special attention

- the log level

```
# cat /proc/net/batman-adv/log_level
[x] critical (0)
[ ] warnings (1)
[ ] notices (2)
[ ] batman (4)
[ ] routes (8)
```

- setting the log level

```
# echo 3 > /proc/net/batman-adv/log_level
# cat /proc/net/batman-adv/log_level
[x] critical (0)
[x] warnings (1)
[x] notices (2)
[ ] batman (4)
[ ] routes (8)
```

- reading the log

```
# cat /proc/net/batman-adv/log
[       626] B.A.T.M.A.N. Advanced 0.1-alpha (compability version 1)
[       971] Changing log_level from: 0 to: 3
```

Kernel development

- don't be scared
- the kernel is a big library for all your hacking needs
- debugging techniques:
  - clean programming - think before you insmod
  - printk - tells you what's up
  - kernel oops - gives you the stack trace
  - UML - safer debugging
- again: don't panic! :-)

Battool

- there is no ICMP on Layer 2
- we still want to ping, traceroute etc to debug the network
- implement own ICMP protocol into batman-adv protocol
- battool provides ping, traceroute and raw packet dump
- injects and receives special packets into unix socket (userspace) or device (kernelspace)

Links

- http://open-mesh.net/
- https://dev.open-mesh.net/batman

Thank you!