olsr.org

'Optimized Link-State Routing' and beyond

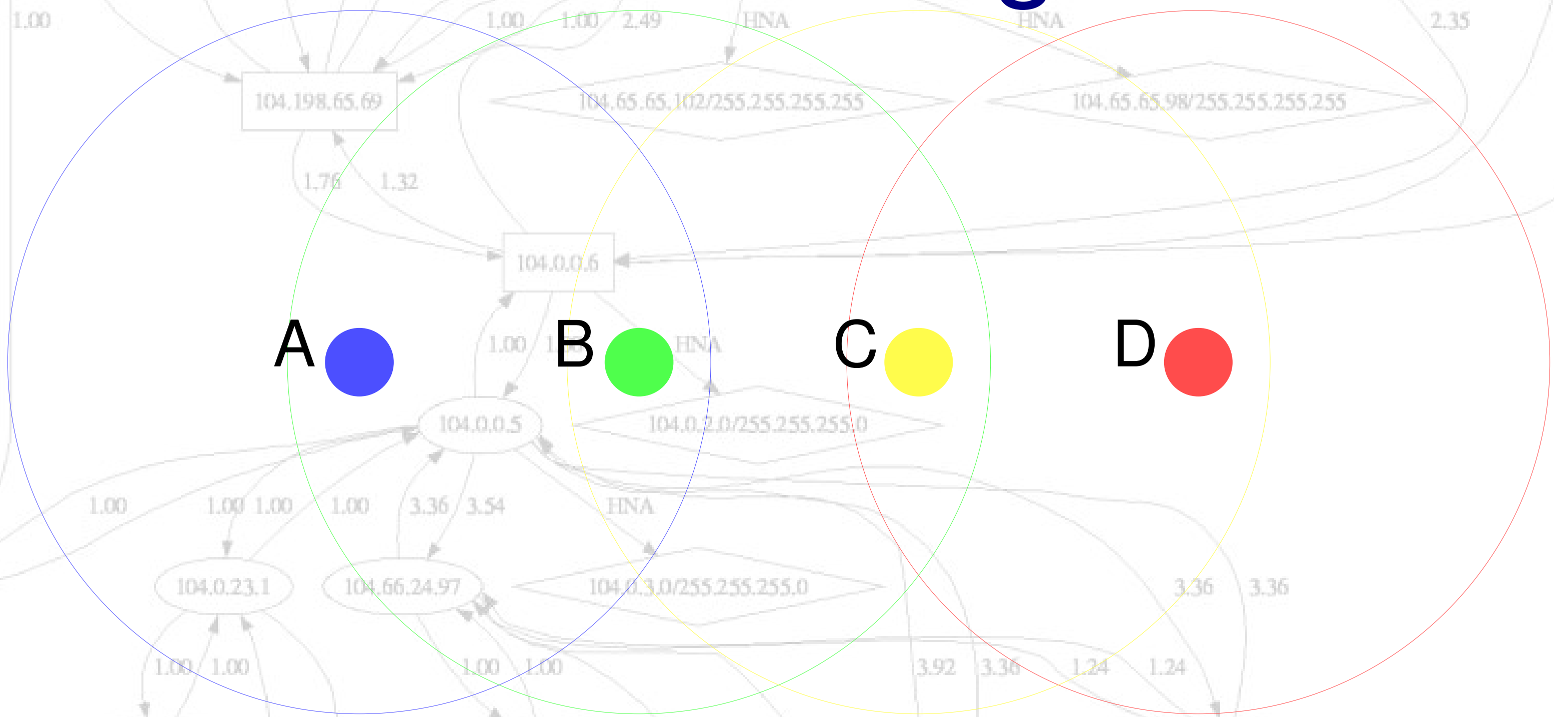
December 28th, 2005

Elektra www.open-mesh.net

Introduction

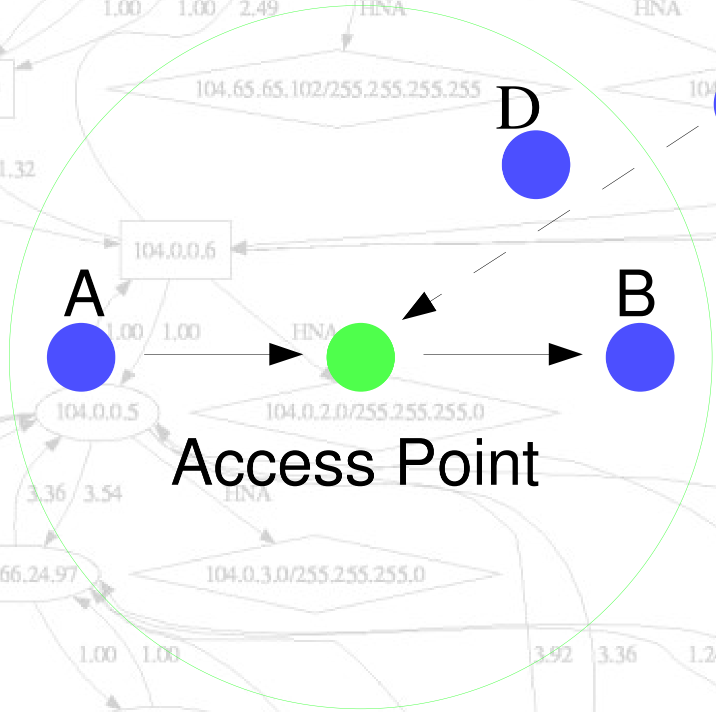
- Olsr.org is aiming to an efficient open-source routing solution for wireless networks
- Work is currently based on the Olsr-protocol suggested by RFC3626
- There is not much left from RFC3626 now, though. You'll see why...

Idea: Multipoint to Multipoint Networking



- A.k.a. Mesh-Networking
- Wireless network based on 802.11 nodes, operating in Ad-Hoc-Mo
- Cover large areas: A and D talk via B and C

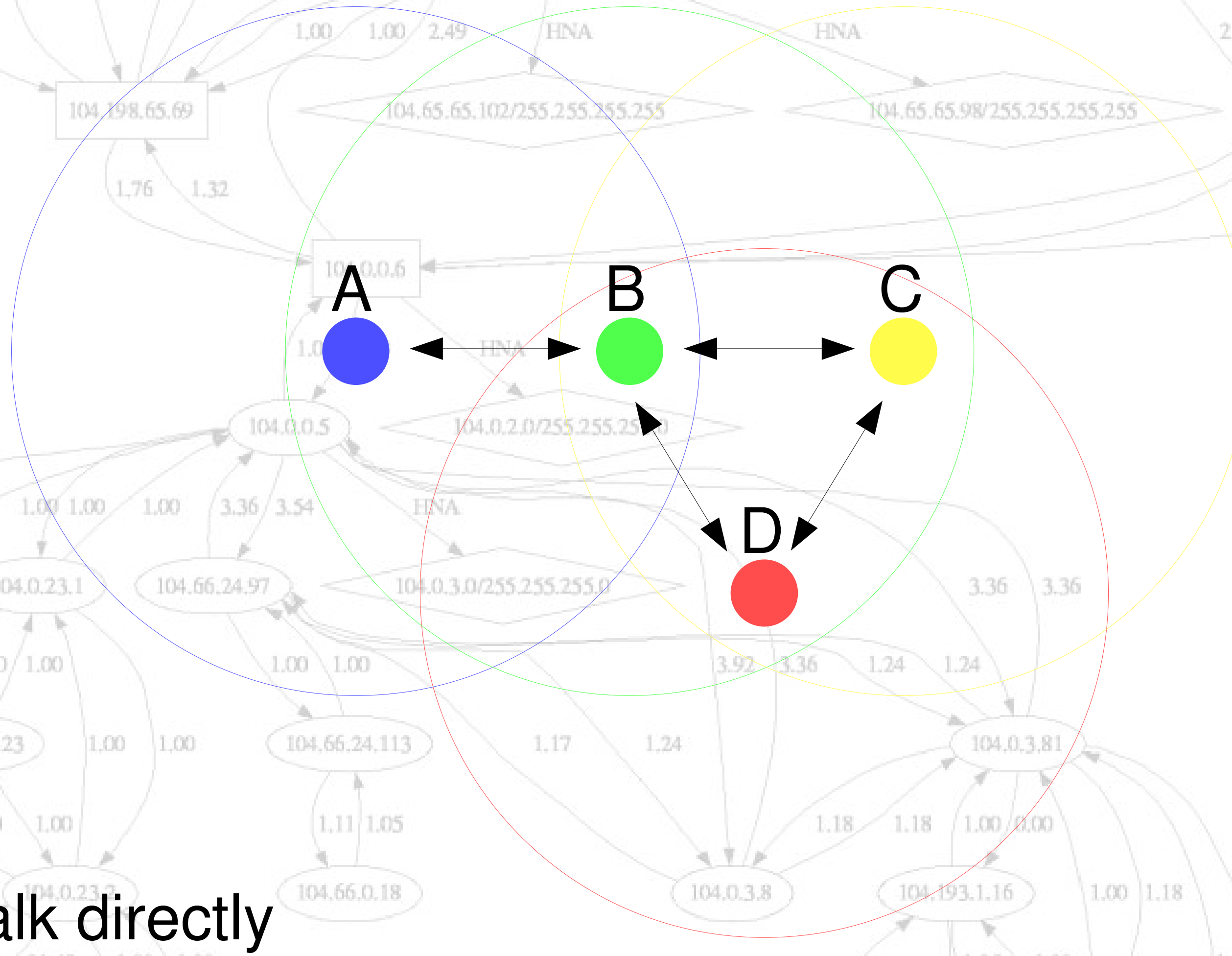
802.11 Managed Mode doesn't allow this*



- A talks to B via central Access Point
- C cannot talk to B or A – although B would be in range of C's Wifi
- D and B have to use the AP as relay, thus speed is only 50%

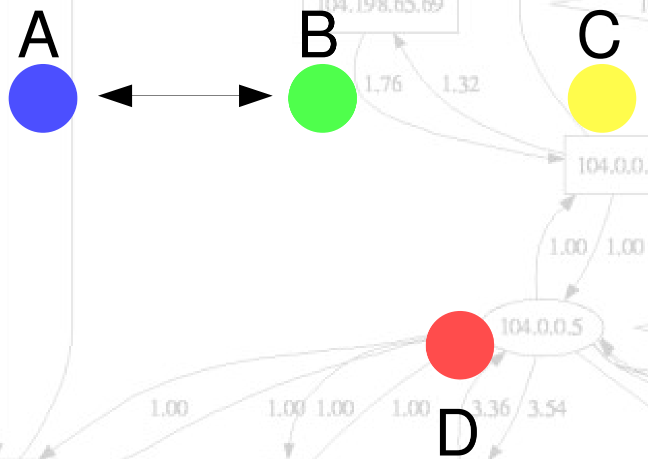
*) when operating with a single interface

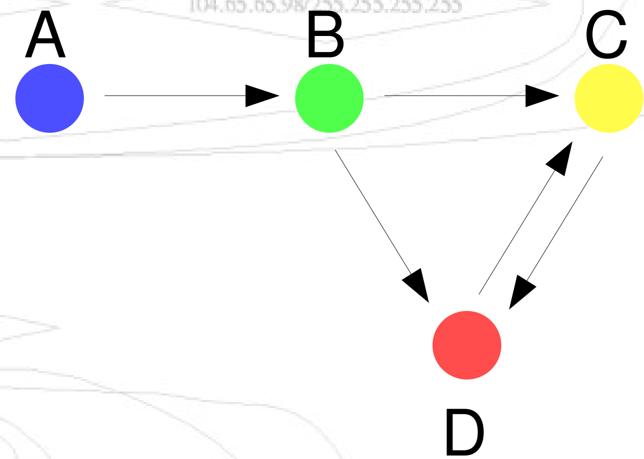
802.11 Ad-Hoc Mode



- Nodes talk directly
- Decentralized & scalable when routing is applied

Proactive Link-State Routing

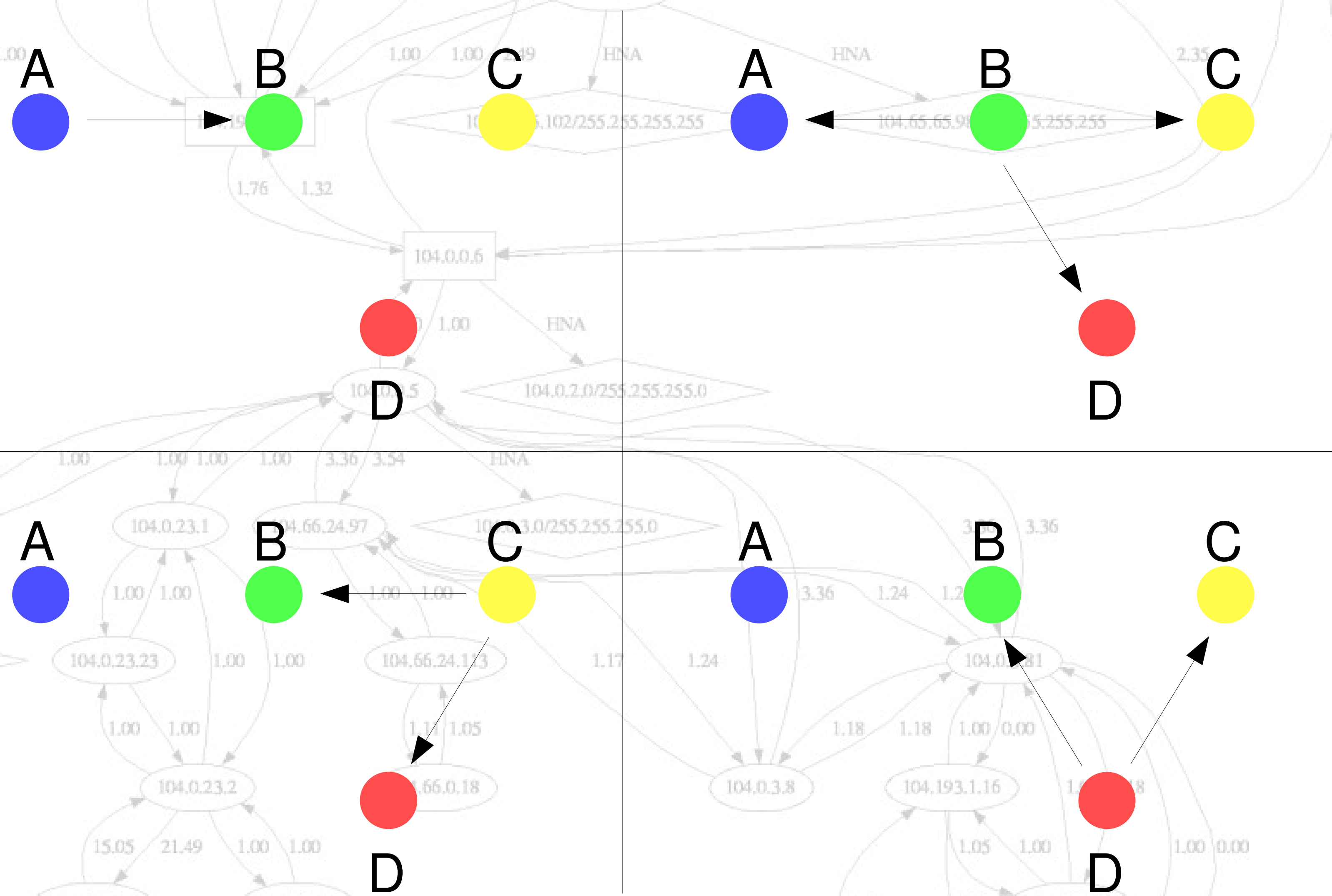




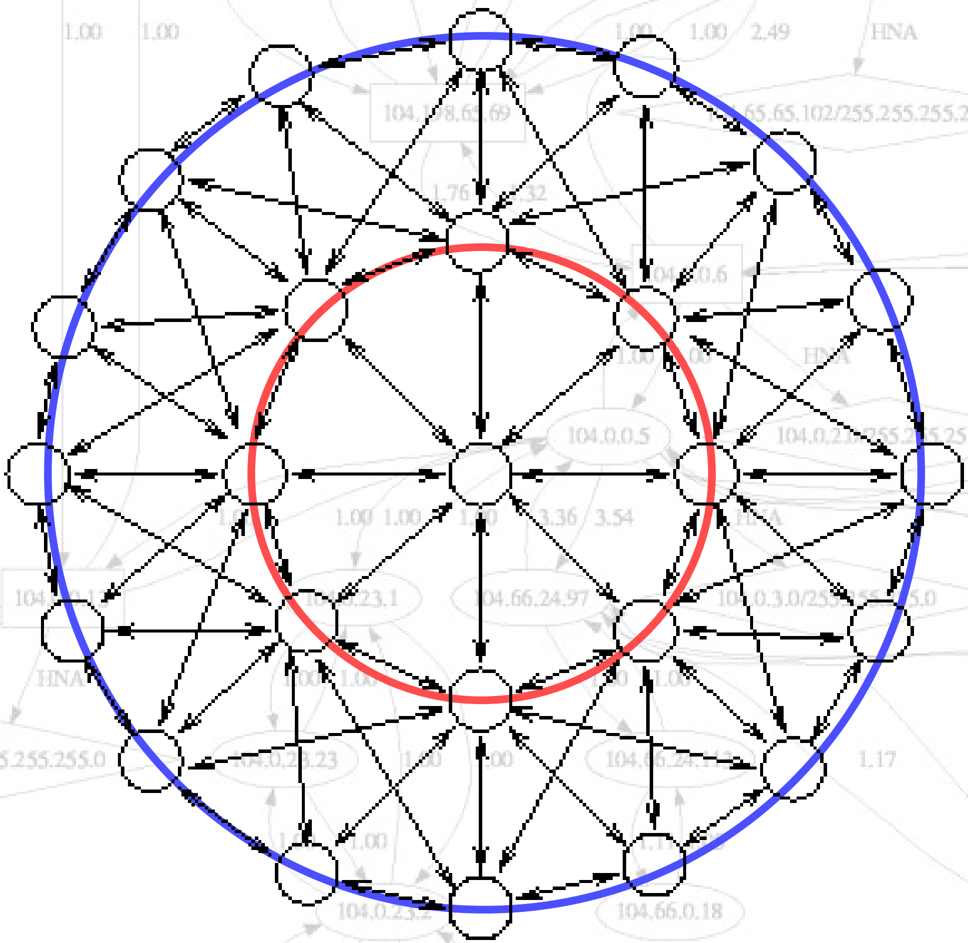
- Link-Detection via Hello-Broadcasts
- A and B notice each other

- Topology Information flooding
- A says 'I see B', B says 'I see A, C, D' a.s.o.

Flooding of Topology Information

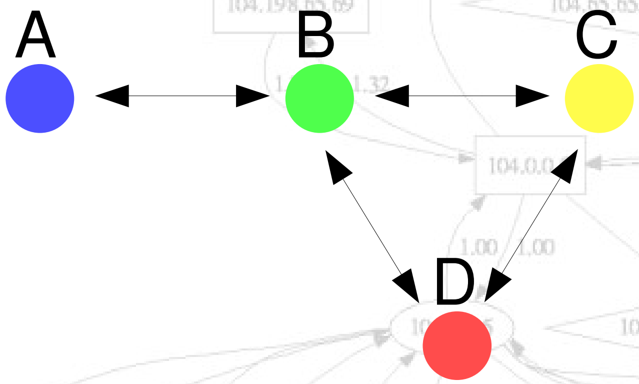


Topology Message Flooding



- All neighbors retransmit messages all over the network
- Bandwidth usage
- Wasting CPU-Cycles
- Collisions

Dijkstra's Algorithm



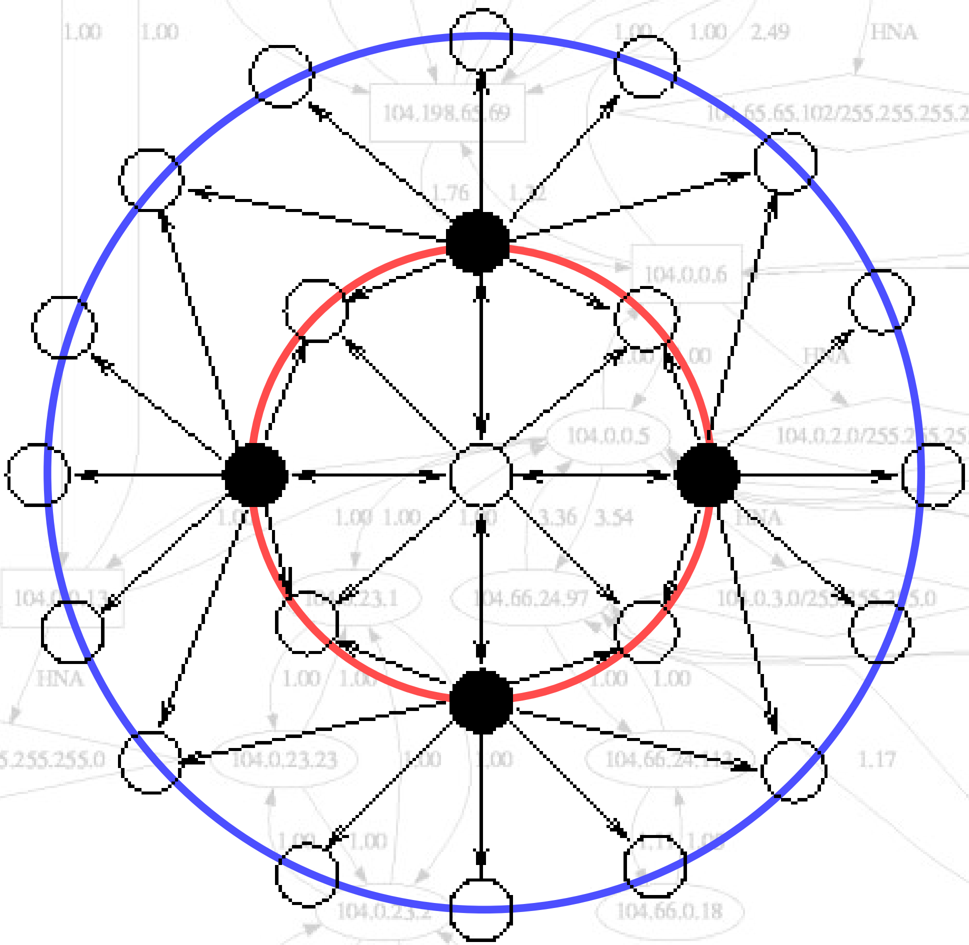
	A	B	C	D
A	*	*	B	B
B	*	*	*	*
C	B	*	*	*
D	B	*	*	*

- Everybody knows everybody else and their links
- Routing table: Dijkstra's Algorithm for shortest paths

OLSR Basics

- INRIA-Draft specified by RFC 3626
- Proactive, using Dijkstra's Algorithm
- Communication via UDP broadcasts
- Multiple OLSR messages per UDP packet
- Validity time in OLSR messages
- Information discarded by timeouts
- Introduced new ideas that were meant to reduce protocol overhead and increase stability: Hysteresis, MultiPointRelays

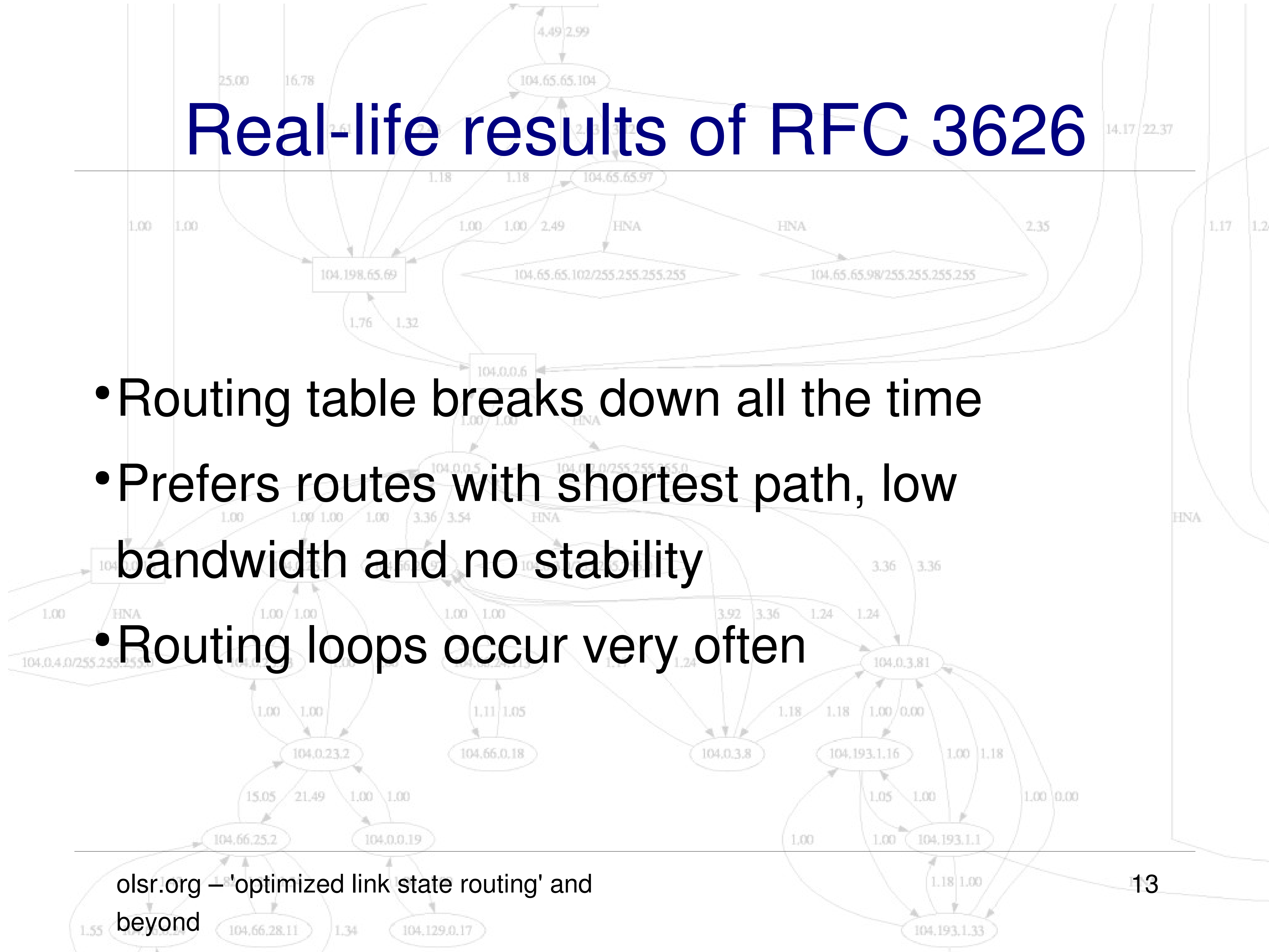
RFC3626 Idea: Reducing Overhead



- Only selected neighbors (Multi-Point Relays, MPRs) retransmit TC-messages
- Select MPRs such that they cover all 2-hop neighbors
- 2-hop neighbors taken from neighbors' HELLO messages
- Does not work in real-life! Reduces redundancy <u>and</u> stability!

Issues in the INRIA-Draft

- Adds new and unnecessary message class of MPRs
- Still optimizes for lowest Hop-Count
- Discards links to neighbors by Hysteresis
- Reduces topology information redundancy
- Every node floods the whole network (at least all MPRs)
- Breaks the KISS-Attitude!



Real-life results of RFC 3626

- Routing table breaks down all the time
- Prefers routes with shortest path, low bandwidth and no stability
- Routing loops occur very often

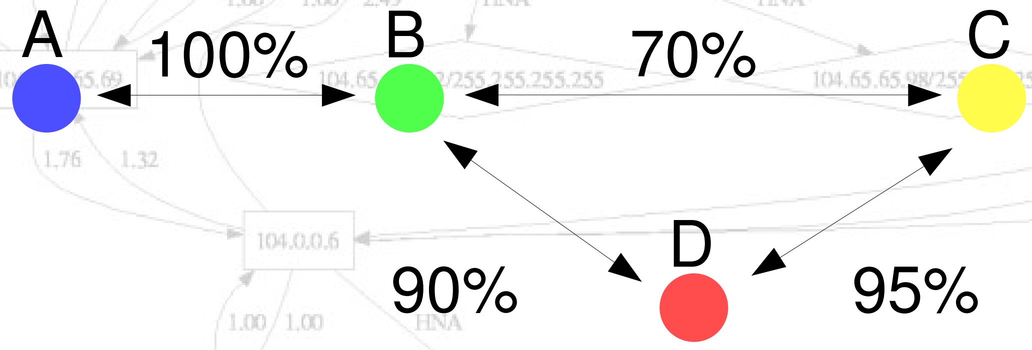
Lessons learned by using RFC3626

- A mesh is a big chaos with interference and collisions
- Theoretical solutions are unlikely to work in real life scenarios.
- Make it work. Make it stable. Worry later about optimizations routing the whole universe in one subnet...
- **Linkstate routing algorithms depend on synchronized information.**
- Transmissions must be redundant (when using Linkstate protocols...)
- New message types introduce new headaches.

What we did...

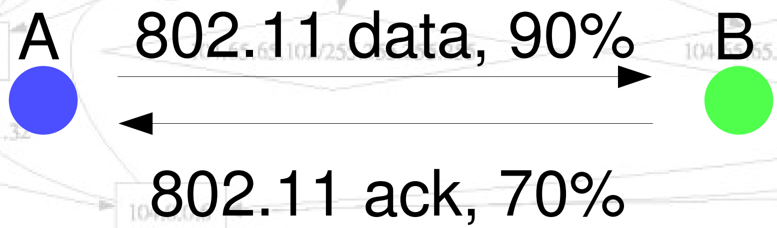
- Disable Hysteresis in the configuration file
- Disable MultiPointRelay selection
- Implement route calculation depending on packet loss (LQ-ETX)
- Implement fish-eye mechanism for forwarding of topology information (Link-Quality-Fish-Eye. New in olsr-0.4.10)

Link Quality I



- OLSR minimizes hop count, hence favors longer (lossier) links
- Alternative – minimize packet loss
  - A – B – C with 70% path quality
  - A – B – D – C with 85% path quality
- Other metrics – latency, throughput, ...

Link Quality II

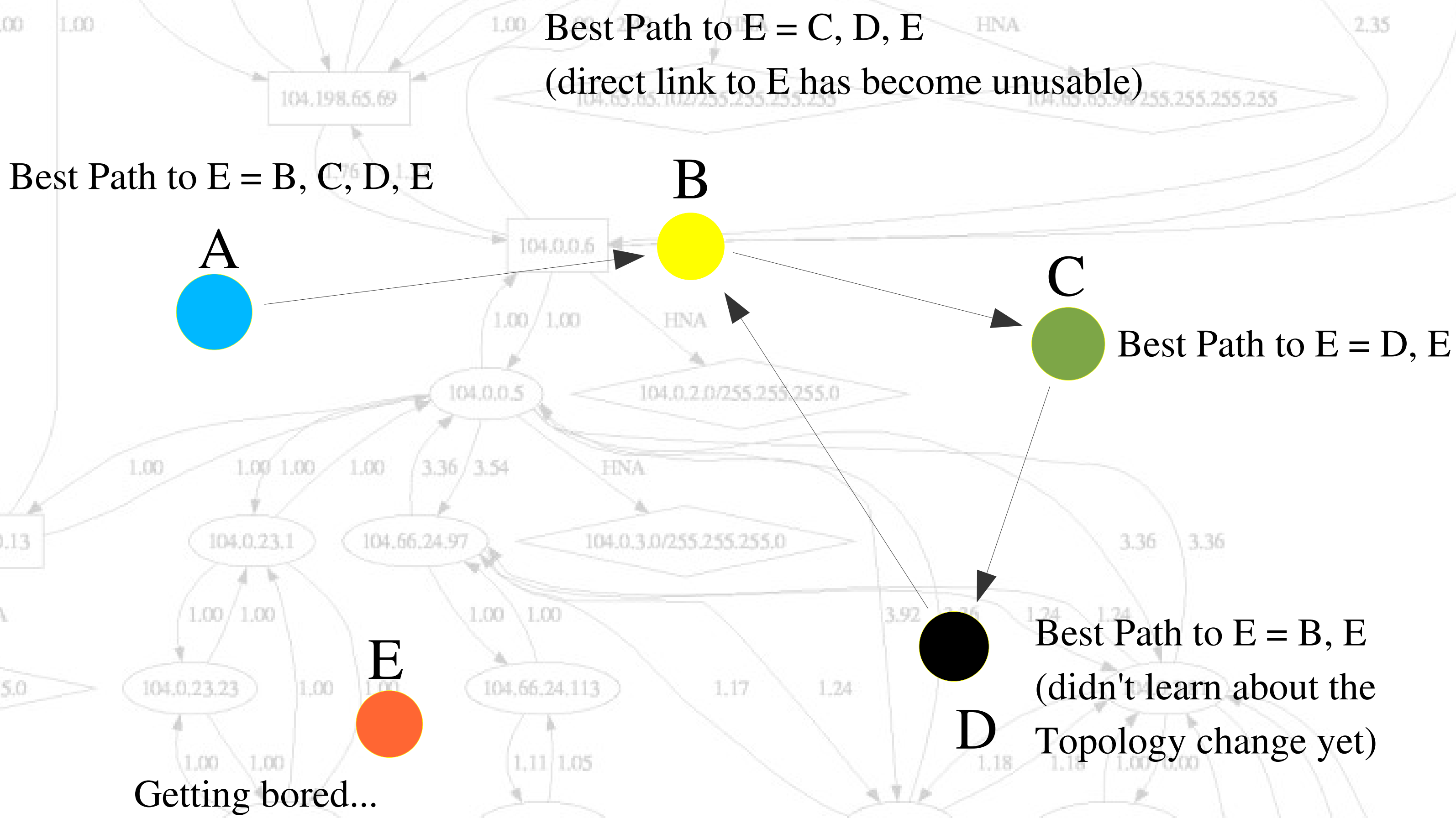


- Minimize Expected Transmission Count (ETX)
- Retransmission – packet or acknowledgment lost
- Packet loss among recent *x* HELLO messages
- $LQ_1 = 90\%$, $LQ_2 = 70\%$
- $ETX = 1 / (LQ_1 \times LQ_2) = 1 / 0.63 = 1.59$

Result: Olsr.org works

- Europe: Many people successfully share DSL-Lines with their mesh.
- Networks up to 150 nodes work well (2008: 800!)
- Still issues under high traffic load – as links saturate routing loops occur. (Almost completely solved with Fisheye)
- Networks that don't saturate their WiFi-Links are not affected.
- The Berlin mesh with more than 250 routes pushes small CPUs to the limit

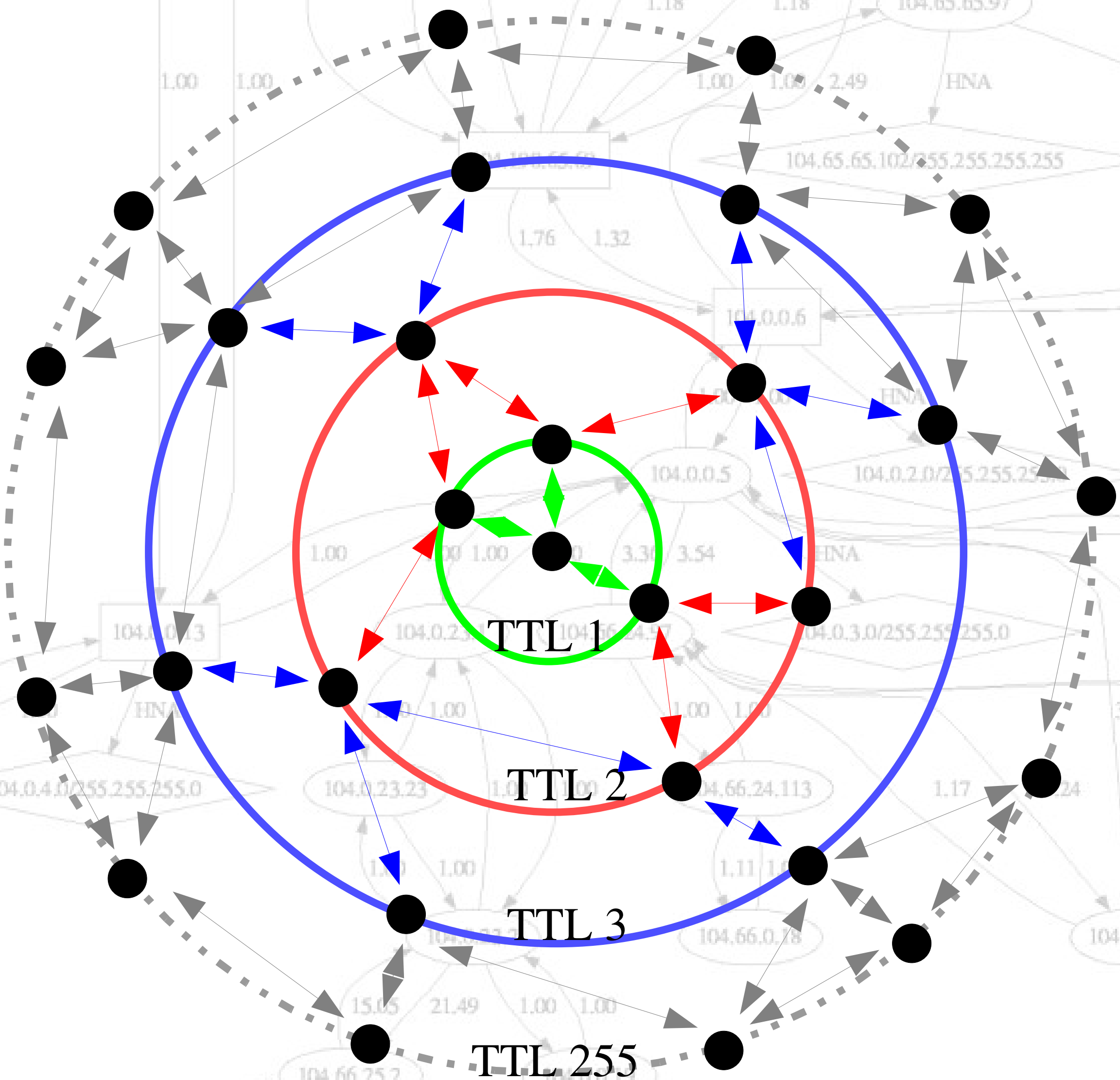
A typical routing loop



Addressing the routing-loop issue

- Occurs when topology information is not in sync
- Loops happen amongst adjacent nodes
- Interference causes topology information loss
- Payload traffic causes interference
- Topology information must be redundant and sent often, more often then Hello-messages to provide information timely
- MultiPointRelays don't help

Link Quality Fish Eye



- Broadcast topology messages with small TTL often
- Send messages with large TTL seldom
- Distant nodes have hazy view – sufficient
- Saving CPU-Cycles
- Saving Collisions

Implementation

- olsrd 0.4.10 – www.olsr.org
- Linux, *BSD, Mac OS X, Windows
- Reasonably stable – Berlin and Amsterdam (More than 200 Nodes in Berlin)
- Plug-in interface (OLSR Flooding)
- Web-based monitoring
- Link Quality Fish Eye Algorithm

OLSR-NG

- Austrian NIC foundation gave funds for performance improvements and code-cleanup
- Huge performance improvements in Dijkstra algorithm

Results from the grid at Meraka

- Olsr-RFC failed the tests (provided only two of four routes)
- Olsr with ETX works well, apart from routing loops if links are saturated.
- B.A.T.M.A.N.-Experimental does not loop. Ever.

Performance comparison of

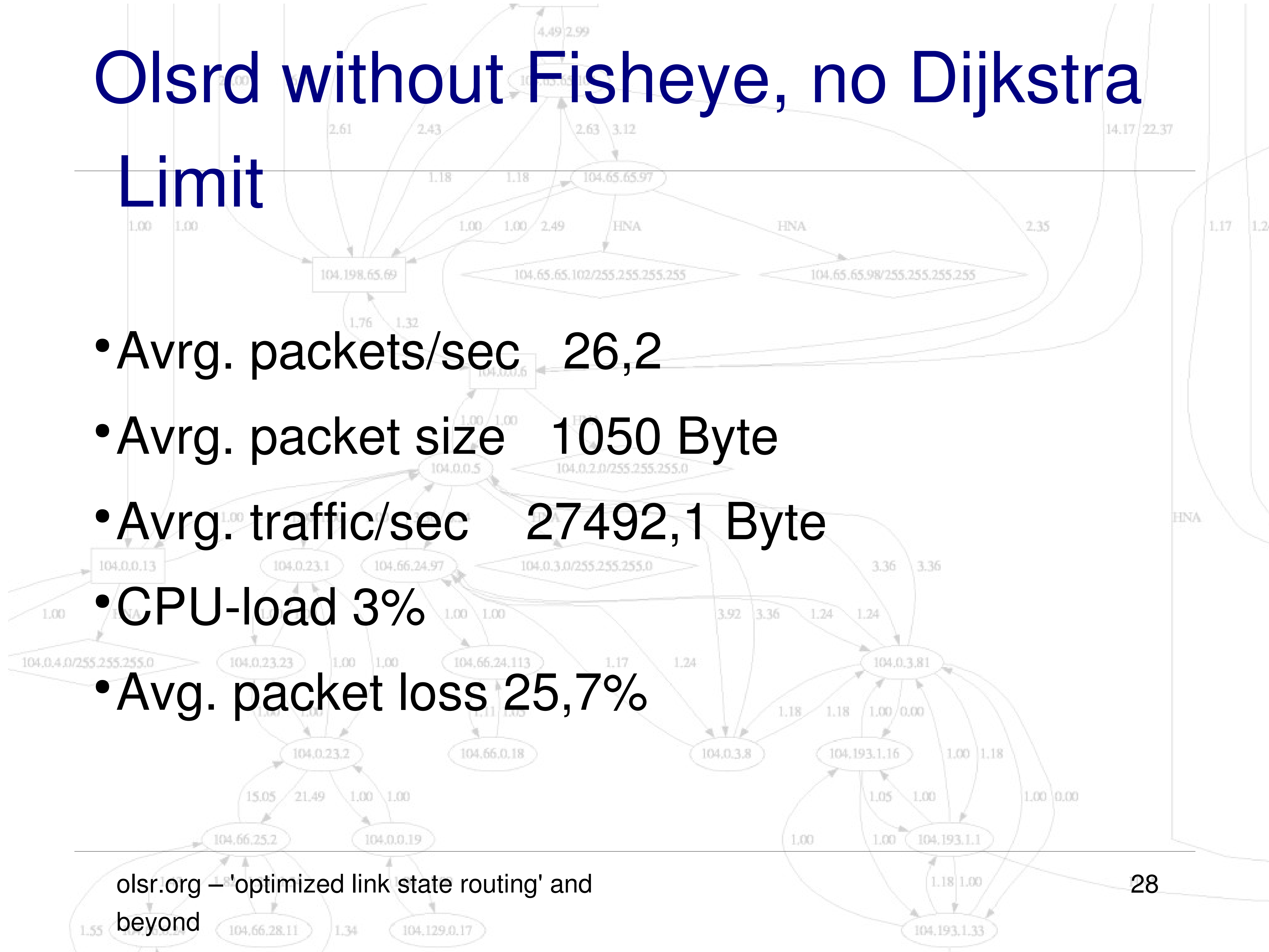
- Batman-Experimental
-
- Olsrd with LQ/ETX with Fisheye, with Dijkstra Limit
-
- Olsrd with LQ/ETX no Fisheye, no Dijkstra limit

Batman-Experimental

- Avg. packets/sec 12,3
- Avg. packet size 204 Byte
- Avg. traffic/sec 2525,4 Byte
- 0,8 % CPU load
- Avg. packet loss 12,7%

Olsrd with Fisheye, with Dijkstra Limit

- Avg. packets/sec 17,7
- Avg. packet size 828,2 Byte
- Avg. traffic/sec 14666,8 Byte
- CPU-load 0.3%
- Avg. packet loss 15,9%



Olsrd without Fisheye, no Dijkstra Limit

- Avrg. packets/sec   26,2
- Avrg. packet size   1050 Byte
- Avrg. traffic/sec   27492,1 Byte
- CPU-load 3%
- Avg. packet loss 25,7%

Thanks

for your

attention.

Questions?