Measurement Results from Wireless Battle Mesh Version 7

Type: Measurement Analysis (work in progress)
Creation date: June 15, 2014

Event:
Sublab. Leipzig, Germany
12th to 18th of May 2014
http://battlemesh.org/BattleMeshV7

Contents

1 Introduction

WBM...

2 Data and System Repositories

```
http://wibed.confine-project.eu

https://github.com/battlemesh/wibed (buildroot)
https://github.com/battlemesh/wibed-battlemesh-experiment (package)
http://wiki.confine-project.eu/wibed:start
https://github.com/axn/wbm2pdf  (this stuff, branch wbmv7 in future)

Raw measurement data:
http://wibed.confine-project.eu/resultsdir/wbmv7-axn-16_2014-05-16_19-28-43 (stationary scenar
http://wibed.confine-project.eu/resultsdir/wbmv7-axn-17_2014-05-16_20-13-20 (broken crossed st
http://wibed.confine-project.eu/resultsdir/wbmv7-axn-19_2014-05-16_21-35-33 (mobile scenarios)
```

Note 1: Testbed and experiment related code and data repositories

3 Testbed ad Experiment Description

3.1 Deployment

During the first days of the event a total of 20 wibed nodes have been deployed. 16 wibed-nodes have been spread over 3 different floors in the main event building. About 10 of these 16 nodes were located in the main event hall (approximately 300 square-meters workshop room) with highest node density in a particular corner of this room (deathroom in Note 2) and the 6 in the below and above floor of the event hall. Three more nodes have been placed in a neighboring building with wireless connectivity. One node was battery powered for allowing mobile-node scenarios. In fact not all node positions were always exactly known as nodes were sometimes moved to fulfill specific experimentation-scenario requirements. In each building 1 of the wibed-nodes were configured as GW nodes and blocked for experimental usage. The remaining 18 nodes were shared between three different experimentation groups for running tests and different scenarios (each node was used by at most one experimentation group at any time).

All experiments were performed in a single 5Ghz channel. However, due to the presence of around 50 participants with wireless laptops and several other actively used wireless equipment, also the used 5GHz channel was likely affected by non-testbed related interference.

The experiment presented in this work was lead by one of these groups and used the following 16 nodes:

NodeID	Location	exp:axn-16 (stationary)	exp:axn-19 (mobile)
164a7a	deathroom		
3b3a90	workshopRoom		
3b3d70	????		
3e9db0	deathroom??	9db0->1ab0	9db0->4174
51aac8	halleAnfang		aac8->4174
8a417e	deathroom	417e->4174	
c24174	HalleEnde (mobile)		
c2427a	deathroom??		427a->4174
ce3360	EloiTable		
e4b63a	mustiTable		
e60a62	halleMitte		
e60aac	deathroom		
e60ad6	deathroom		
e61936	axelsTable	1936->4174	1936->4174
f41ab0	kloschi (building B)	1ab0->4174	1ab0->4174

Note 2: Node locations and experiment usage

3.2 Protocol configurations and assumptions

The highly dynamic and uncontrollable interference in the measurement environment made it impossible to ensure equal conditions for sequential experiment executions. Therefore, to ensure equal (fair) environment conditions for all tested protocols, all routing protocols were running and observed in parallel on all nodes, thus all being always exposed to the exactly same environmental conditions.

The accepted downside of this approach is of course that protocol overhead introduced by one protocol or by protocol-observing tools like ping (causing total overhead in the order of few KB/s, as can be seen from later measurements) slightly affects the maximum achievable end-to-end throughput of other protocols (in the order of several MB/s).

Only netperf-tcp-based throughput probes, seeking to measure the achievable tcp performance of the end-to-end routing paths established by the individual protocols, were performed sequentially. This decision has been made because each netperf test tries to load the capacity of a given end-to-end path with a maximum of traffic, thus introducing maximum probing-traffic overhead and interference while on the other hand (given the tcp-inherent exponential back-off approach) drastically lowering the currently offered load in the presence of packet loss, leading to highly randomized results when running in parallel and making the comparison of parallel executions difficult.

All protocols were configured for routing IPv6 traffic using an individual ULA address prefix per protocol.

All protocols were configured with default parametrization, thus no environment specific customizations have been made apart from ensuring the routing of the given IPv6 address range. The exact configuration can be accessed via [**?**].

To avoid protocol-bootstrapping effects (eg unfinished neighbor-, path- or topology-discovery), all routing protocol daemons were started at least 200 seconds before any measurement.

3.3 Measurement configuration and assumptions

Follow-up measurements were executed from a singe selected node (src-node) by launching a pre-deployed test script [**?**] and given the id of a single other node (dst-node) for probing

end-to-end path characteristics. Src-node and dst-node IDs for each measurement are given in Note 2.

Each followup measurement last 200 seconds during which the following additional tools were used to observe protocol performance and overhead:

- ping6 (unix) command to dst-node ipv6 address with one-second interval and 1000 bytes icmp user data. The output of the ping6 command got logged for later end-to-end packet loss, hop-count, and round-trip time (RTT) over time analysis.
- top (unix) command for logging protocol-specific CPU and memory consumption at 1 second intervals.
- mtr (my trace route, unix) command at 1 second interval for tracing full src-to-dst protocol-established path information. Due to the difficulty to correlate or graphically represent these traces, the obtained log files were not processed further.
- netperf, executed in repeating rounds (4 rounds), each probing sequentially the maximum achievable end-to-end throughput to always the same destination node for 10 seconds via each routing protocol.
- tcpdump, passively capturing the present routing-protocol overhead of each protocol as received on the wireless channel by the src node.

3.4 Experiments

The experiment focused on measuring the overhead and performance of 5 different mesh routing protocol implementations in static and mobile scenarios. The five tested protocols were batman-advanced (batadv), bmx6, olsr, olsr2, and babel. Unfortunately, we just discovered after the measurements that the babel protocol daemon was not configured correctly, leading to broken routing decisions for multi-hop path. Therefore all babel-protocol related measurements were discarded in the following discussion.

Experiments are grouped in two different scenarios (wibed experiments with results accessible via corresponding wibed-data repositories as given in Note 1 for exp-16 and exp-19), each scenario consisted of about 4 measurement executions by using different source-destination combination (measurement results were stored on the source-node of the measurement as given by Note 2). In the following the two scenarios and selected measurement results are discussed. The complete set of obtained measurement results and graphs are in Tables 1 to 10.

3.4.1 Stationary scenario

During the stationary scenario the involved wibed nodes were not moved, however still affected by uncontrollable interference conditions from the environment itself, other parallel experiments executed on other wibed nodes, and the traffic created by the experiments itself. For each measurement a different couple of source-node and desalination-node was selected intuitively with the objective to select, in terms of network-topology, rather distant (so non-neighboring) nodes.

Each selected node couple was tested successively (with several hundreds seconds between each measurement) over 200 seconds using the wbm-test script [**?**] doing measurements as described in Section 3.3.

3.4.2 Mobile scenario

Apart from the presence of a single mobile node, always serving as destination node, all mobile scenarios were performed in the same way as the stationary scenarios. During the mobile scenario the mobile node was moved (carried) manually at slow-walking speed (approximately 1m/second) the about 100 meters back and forth along the main event hall, downstairs to the lower floor, and along the lower hall. The mobile-node carrying walker returned to its original position few seconds before the end of each 200 seconds lasting measurement.

4 Discussion of measurement results

In the following the results of only one performed measurement are discussed in more detail. Therefore the last measurement, as summarized via graphs in Table 5, was selected simply because it suited best to illustrate a number of expected and unexpected phenomenas. The results represented by this selection are by no means representative. However, looking at all performed measurements, a general ranking of protocols can only be done regarding a very few particular characteristics while for other, given the few number of measurements and the high amount of randomness involved, at most some vague tendencies may be guessed.

4.1 Protocol-traffic overhead

The protocol-traffic overhead in terms of bytes per second is given in Figure 2. The continuous lines are presenting the measured overhead over time with a one-second resolution and the dashed lines the average overhead captured by the source node over the 200-seconds measurement period. It should be noted that the captured traffic includes all received protocol traffic, the traffic created by the capturing node itself as well as the traffic created by neighboring nodes (being in transmission range of the capturing node). Only pure protocol-traffic was considered (not user-data related traffic) as created by each protocol to detect and establish the routing inside the network. For batman-advanced, encapsulating user traffic inside the same ethernet frame type as used for protocol traffic, special filtering rules were used for differentiation. From this measurement it can be seen that highest average overhead was created by batman-advanced and olsr. Significantly less overhead (about one third) was captured for by bmx6. And overhead due to olsr2 somewhere between. In fact, a similar byte-overhead related ranking can be observed in all other mobile and stationary scenarios.

Protocol-traffic overhead in terms of packets per second is given in Figure 1. Here batman advanced showed significantly more (in the order of x10) created packets per second than all other protocols. Allowing the conclusion that batman-advanced packet size must be rather small compared to other protocols. Least amount of packets were captured from olsr. Thus, olsr packets must be quite big given the similar amount of bytes-per-second

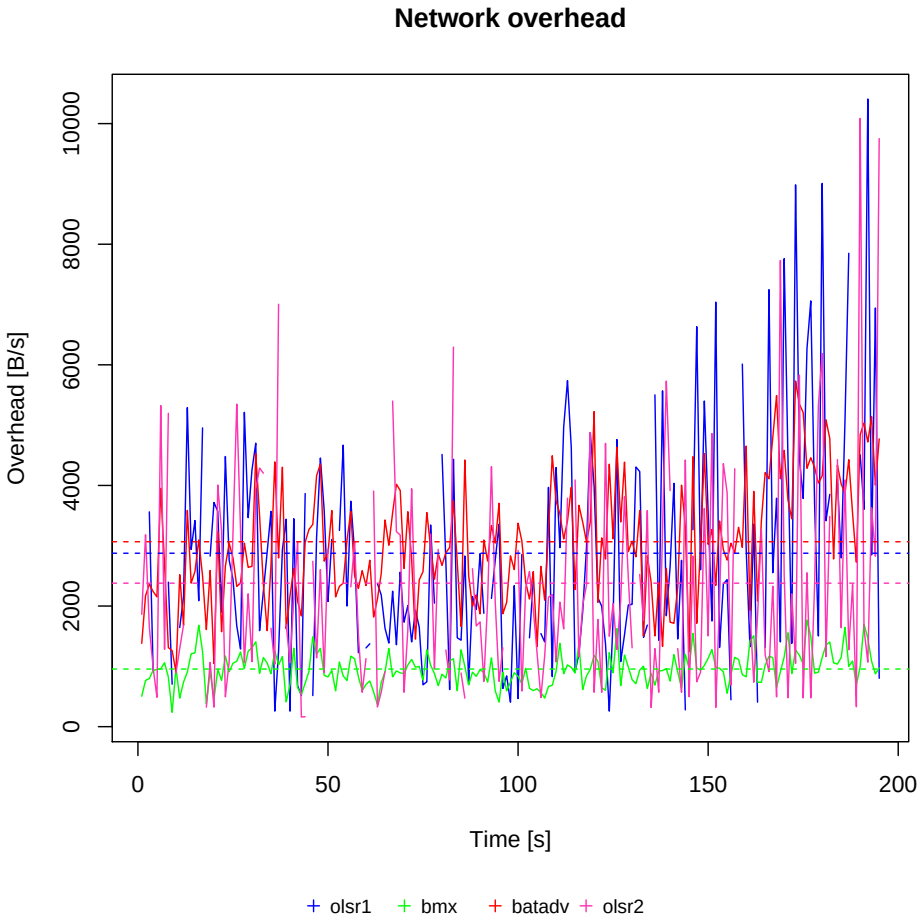

Figure 1: Protocol traffic overhead in bytes/second over time

overhead that the olsr and batadv protocols created. Also this ranking repeats in all other measurements.

4.2 Memory and CPU consumption

Memory and CPU consumption over time are given in Figure 3 and 4 respectively. Both characteristics are not captured for batman advanced, which, running in kernel space, does not allow an easy profiling of this data. It can be seen that none of the captured protocols changed its memory requirements during this (nor any later) measurement. Also, the observed differences seem rather marginal (lowest seen for olsr with about 1.15 MBytes and highest for olsr2 with about 1.7 MBytes) compared to the total available memory of 128 MBytes.

CPU consumption is given relative to the total available CPU processing capabilities (typically characterized by CPU architecture, number of cores, and speed). Here olsr2 showed significantly highest CPU usage (consuming about 10% of the total available CPU capacities and about 10 time more compared to other protocols). However, it should be noted that the olsr2 code from olsr.org is still under heavy development, probably containing many CPU-consuming debug code that will be cleaned up in later versions. In fact, as stated by olsr2 developers, it was the first time this code was tested in a real and reasonable diverse testbed environment. It is also interesting (even strange) that for two

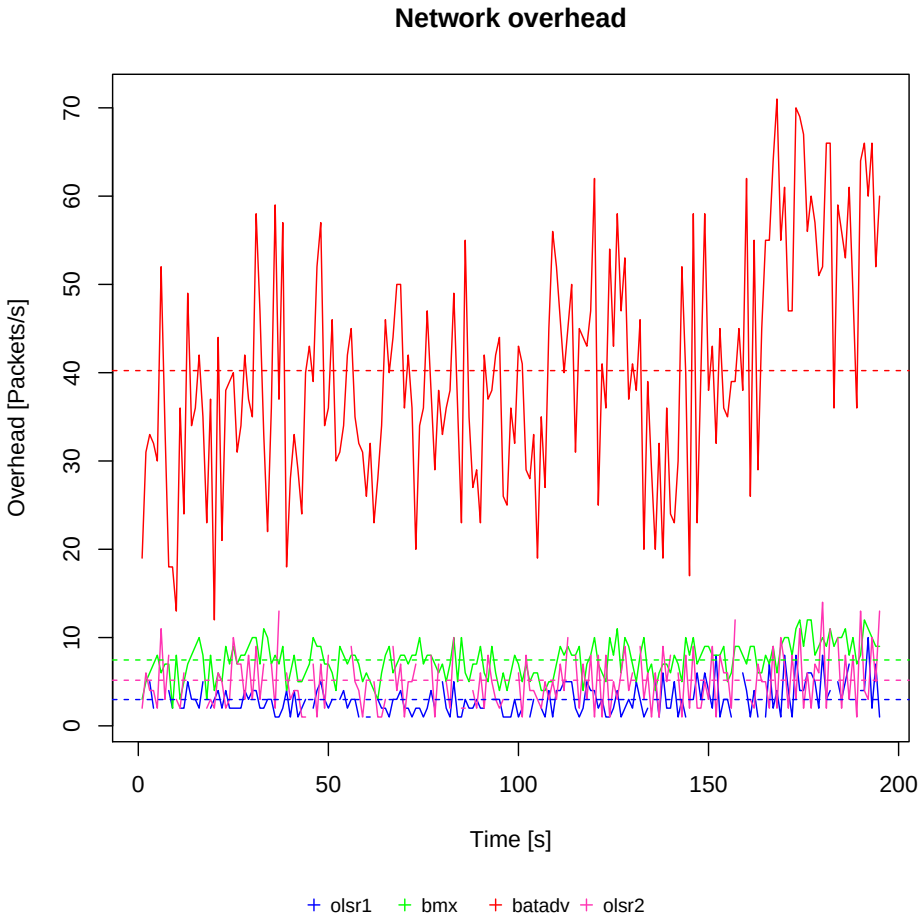

Figure 2: Protocol traffic overhead in packets/second over time

of the total of 8 measurements olsr2 showed only about 4% CPU usage.

4.3 Ping6 tests: path RTT, hops, delivery rate

The analysis of established path characteristics as probed using the ping6 tool is analyzed from different perspectives in Figures 5, 6, and 7 respectively.

Figure 5 shows the resulting RTT and involved hops for the established end-to-end path of each protocol over time (Hops given for batman-advanced are incorrect due to its layer-2 routing).

It can be seen that the path (in terms of hops) changed various times for olsr and bmx6, choosing pathes with 2 to 6 hops, despite the fixed locations of nodes in this experiment. On the other hand, the olsr2 path seemed quite constant, choosing a continuous three-hop path most of the time which (as can be seen from later discussed Figures 7 and **??** also lead to the best measured performance). As expected, the RTT shows smaller values for less hops and higher latencies for more hops. The tendency of olsr2 choosing pathes with less hops repeats in other stationary measurements. So does the tendency of olsr and bmx6 for more frequent path changes and choosing pathes with more hops. It can also be noted (especially by looking also at graphs from the other stationary measurements) that path length of olsr and bmx6 often starts and ends with less hops than observed for the majority and in-between time of the measurement duration. This phenomenon may be

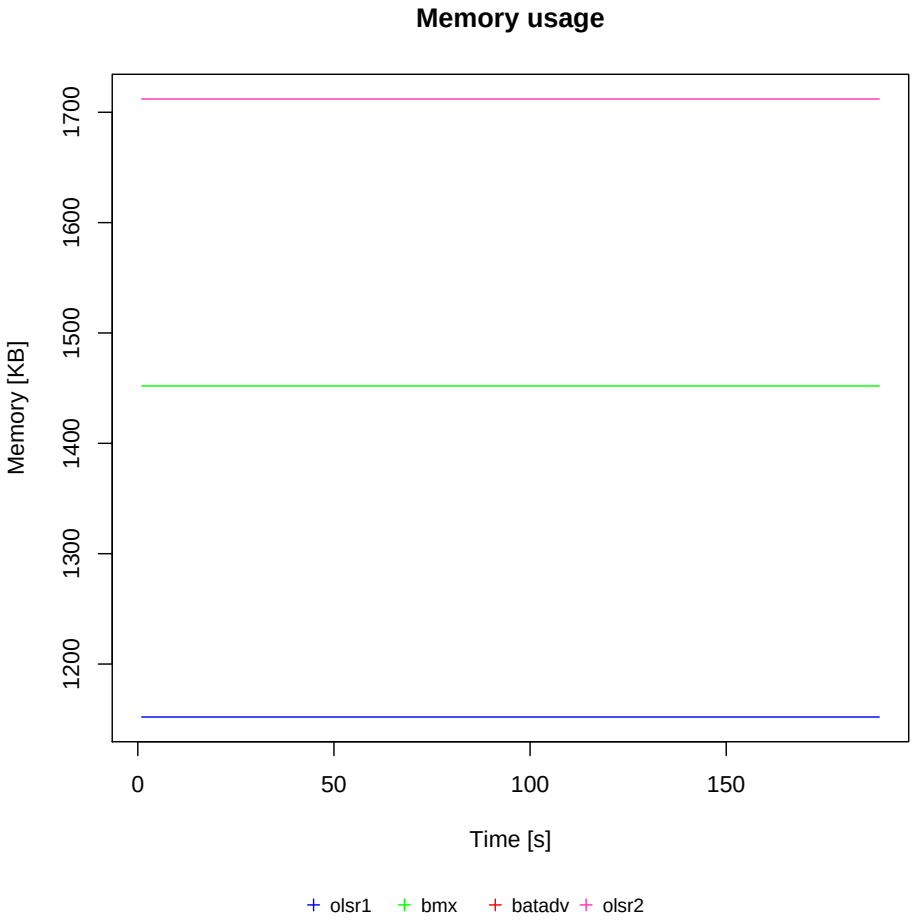

Figure 3: Protocol memory consumption over time

caused by the rather heavy interference introduced by the parallel netperf measurement which, starting roughly at the same time than the ping measurement, last only 160 seconds (4 rounds, 4 protocols, 10 seconds). As a consequence, routing protocols may react to the new interference conditions (as perceived by decreasing link qualities especially for long-distant links) with some delay by routing traffic over a new end-to-end path, using more hops but short-distant links. Once the netperf probes and related interference finishes, the protocols converge back to paths with less hops but links covering longer distances.

Figure 6 shows the distribution of observed path hops.

Figure 7 shows the ECDF of the observed RTTs during the whole 200 seconds measurement...

4.4 TCP throughput

Figure 8 shows the TCP throughput, measured successively for each protocol and round...

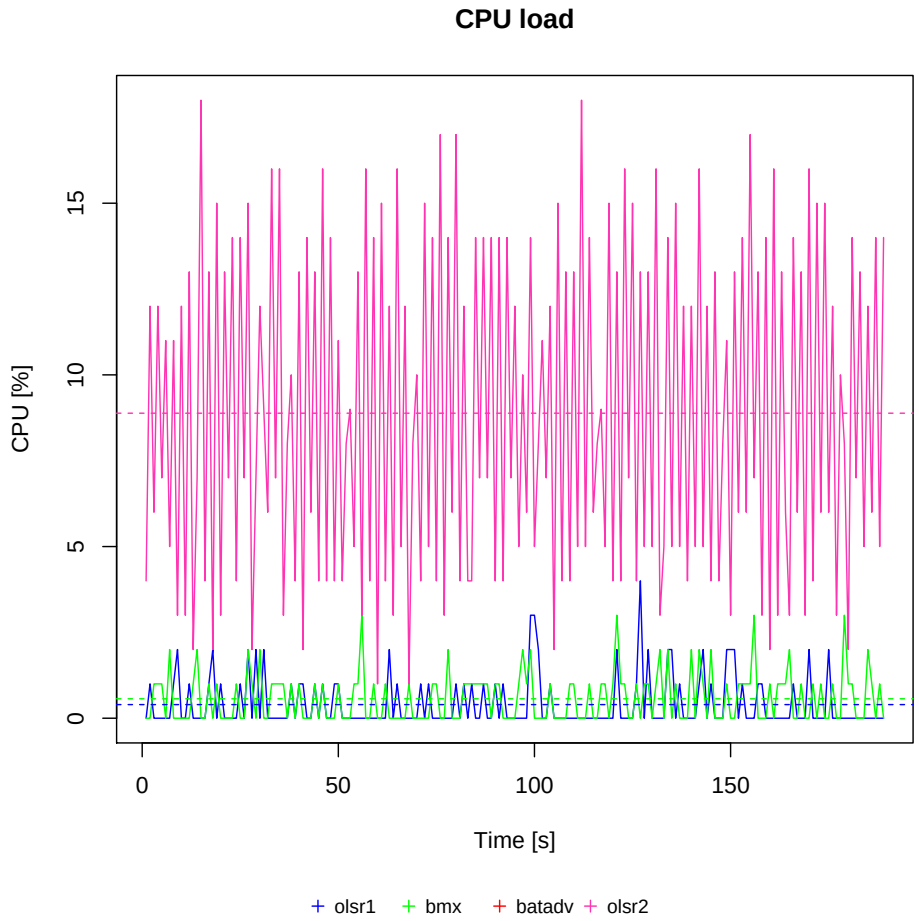

Figure 4: Protocol CPU consumption (relative to total system CPU) over time

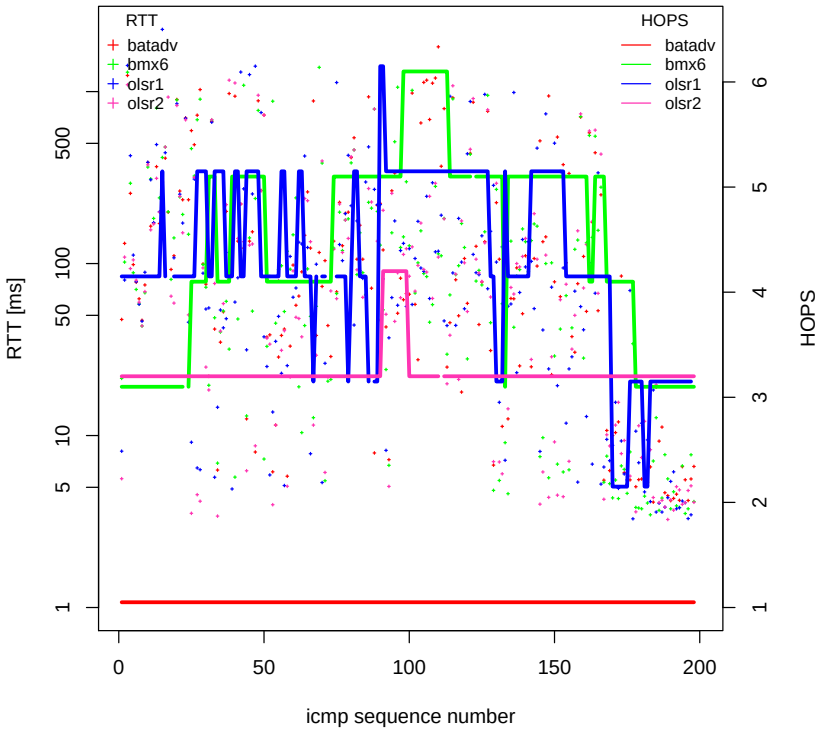

Figure 5: Observed path RTTs and hops over time

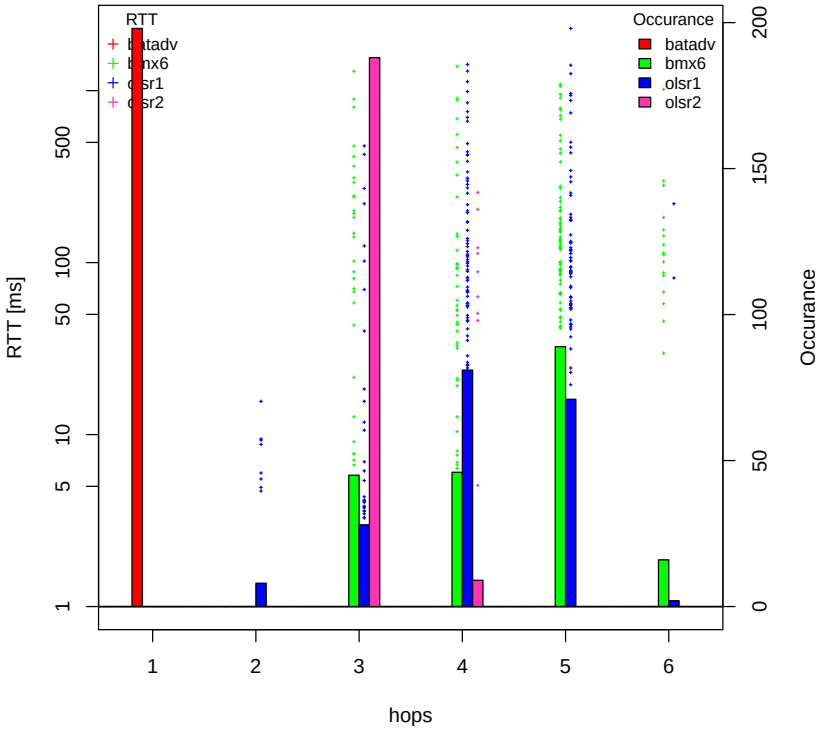

Figure 6: Observed path RTTs and probing occurances over time

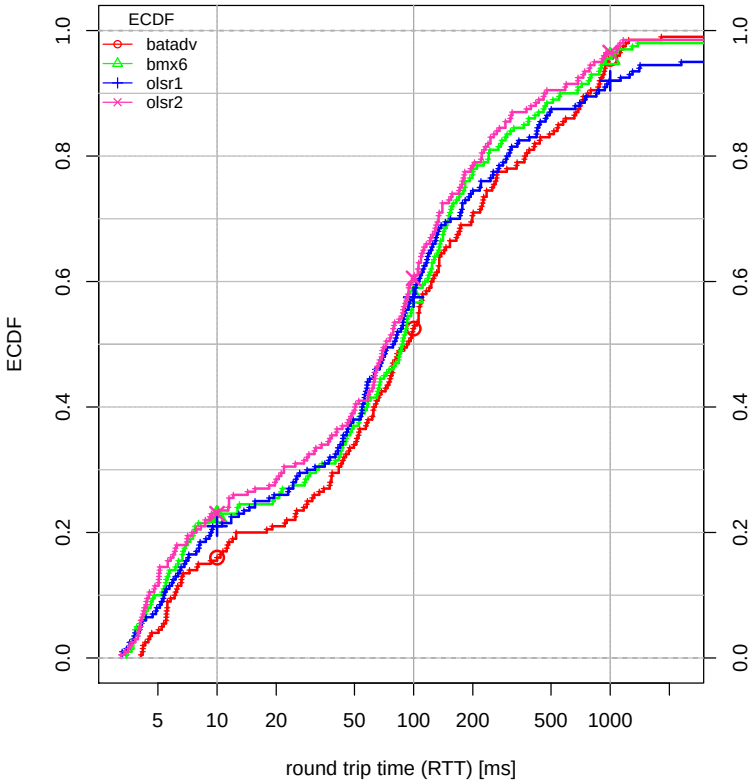

Figure 7: ECDF of success rates over RTT

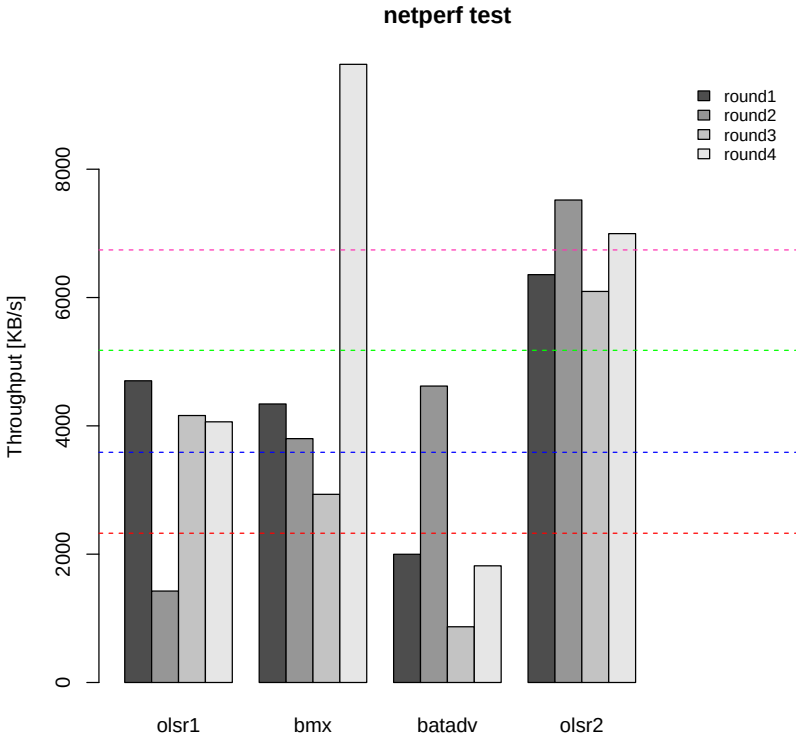

Figure 8: TCP throughput, measured successively for each protocol and round

4.5 Summary

Given the few number of measurements and the high amount of randomness involved, the results gathered during this measurement campaign are by no means representative. Looking at all performed measurements, as given in Tables 1 to 10, a general ranking of protocols can only be done regarding some few and particular characteristics. For other, if at all, at most some vague tendencies may be guessed. Main conclusion here is that much more measurements would be needed for concluding representative results. Thus, observations summarized in the following MUST be taken carefully.

OLSR showed varying and in average medium results for measured path throughput, RTT, and total packet loss (the four performed throughput measurements showed it in all positions, from best- to least-well performing protocol). Its observed CPU and memory consumption was always notable low. Regarding protocol-traffic overhead, it typically ranged within the most expensive (bandwidth consuming) protocols.

OLSR2 also showed varying and in average medium results for measured path throughput, RTT, and total packet loss (OLSR2 driven path throughput was typically lowest but once notable high). Its observed CPU and memory consumption was always notably higher than any other protocol. Regarding protocol-traffic overhead, it showed average cost (bandwidth consuming more than bmx6 but less than olsr and batman advanced).

BMX6 showed rather good results for measured path throughput, RTT, and total packet loss (all four throughput measurements resulted in first or second position). Its observed CPU and memory consumption showed average cost. Regarding protocol-traffic overhead, it always showed least cost.

Batman advance also showed varying and in average medium results for measured path throughput, RTT, and total packet loss. Memory and CPU usage could not be measured. Regarding protocol-traffic overhead, it typically ranged within the most expensive (bandwidth consuming) protocols.

5 Recommendations for next battlemesh

- Traceroute and mrt often show high packet for intermediate nodes. This is due to a kind of denial-of-service mechanism enabled by default in Linux kernel. WIth this mechanism the kernel simply discards frequent icmp responses (eg due to exceeded TTL values). This behavior can be disabled by lowering the default net.ipv6.icmp.ratelimit=1000 setting, eg via: sysctl -w net.ipv6.icmp.ratelimit=10
- Use batctl for batman-advanced and ping related measurement foo.
- Do also protocol execution sequentially, so that protocol overheads do not affect other protocols path performance.

6 Appendix

6.1 All stationary-scenario measurement graphs

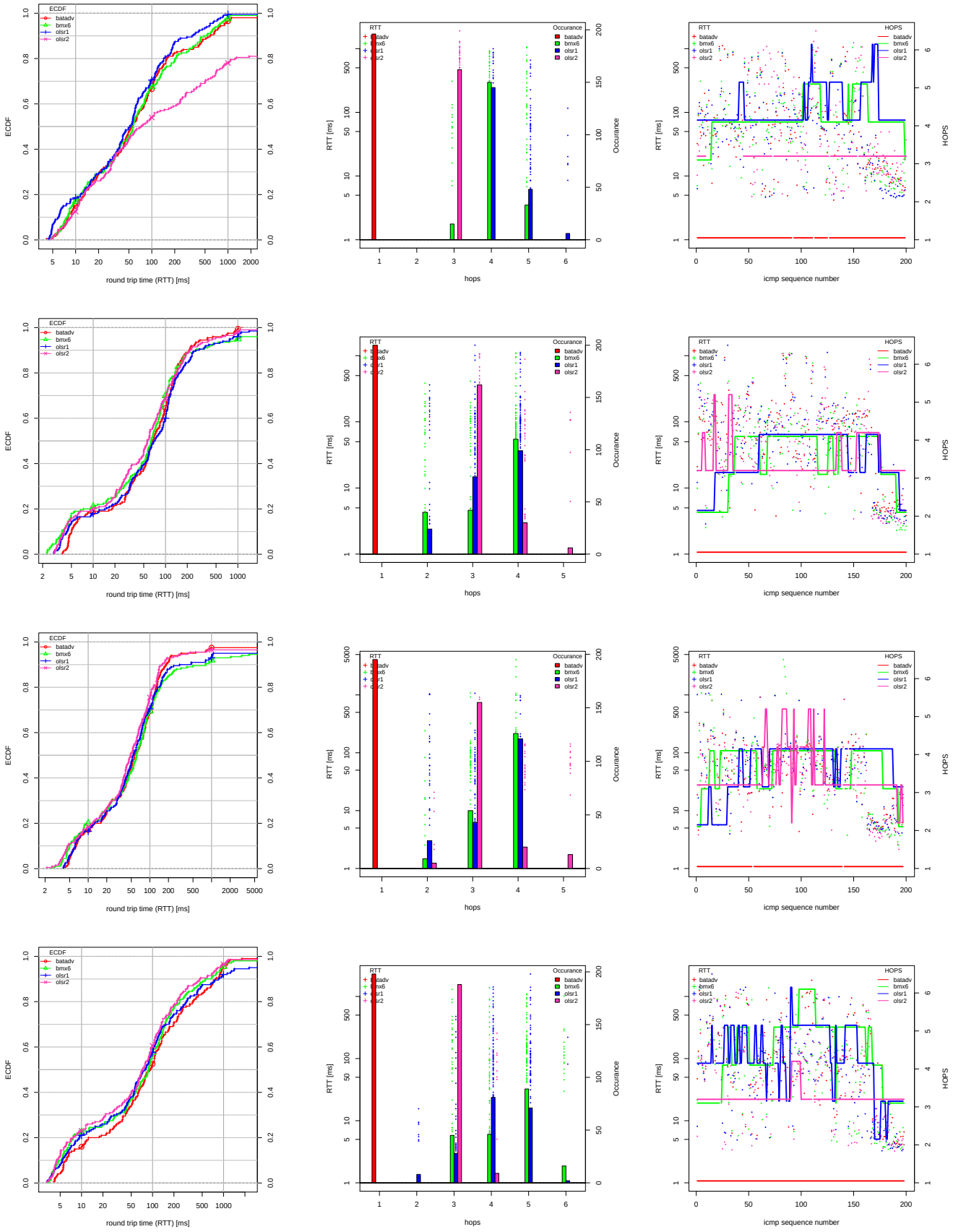

Table 1: End-to-end ping6 performance between two stationary nodes: 9db0-1ab0, 417e-4174, 1936-4174, 1ab0-4174

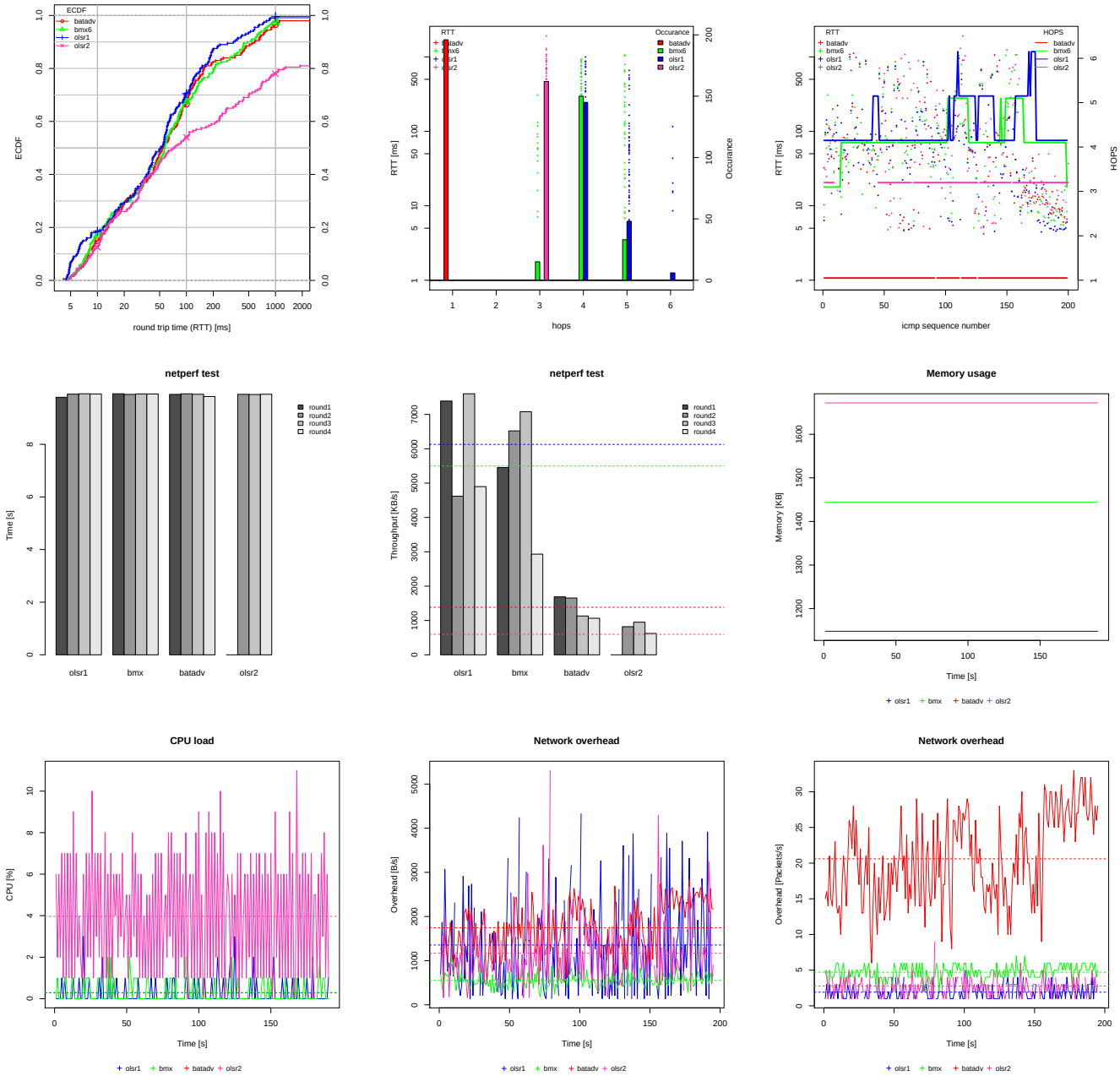

Table 2: Overhead and end-to-end performance between two stationary nodes: 3e9db0 and 1ab0

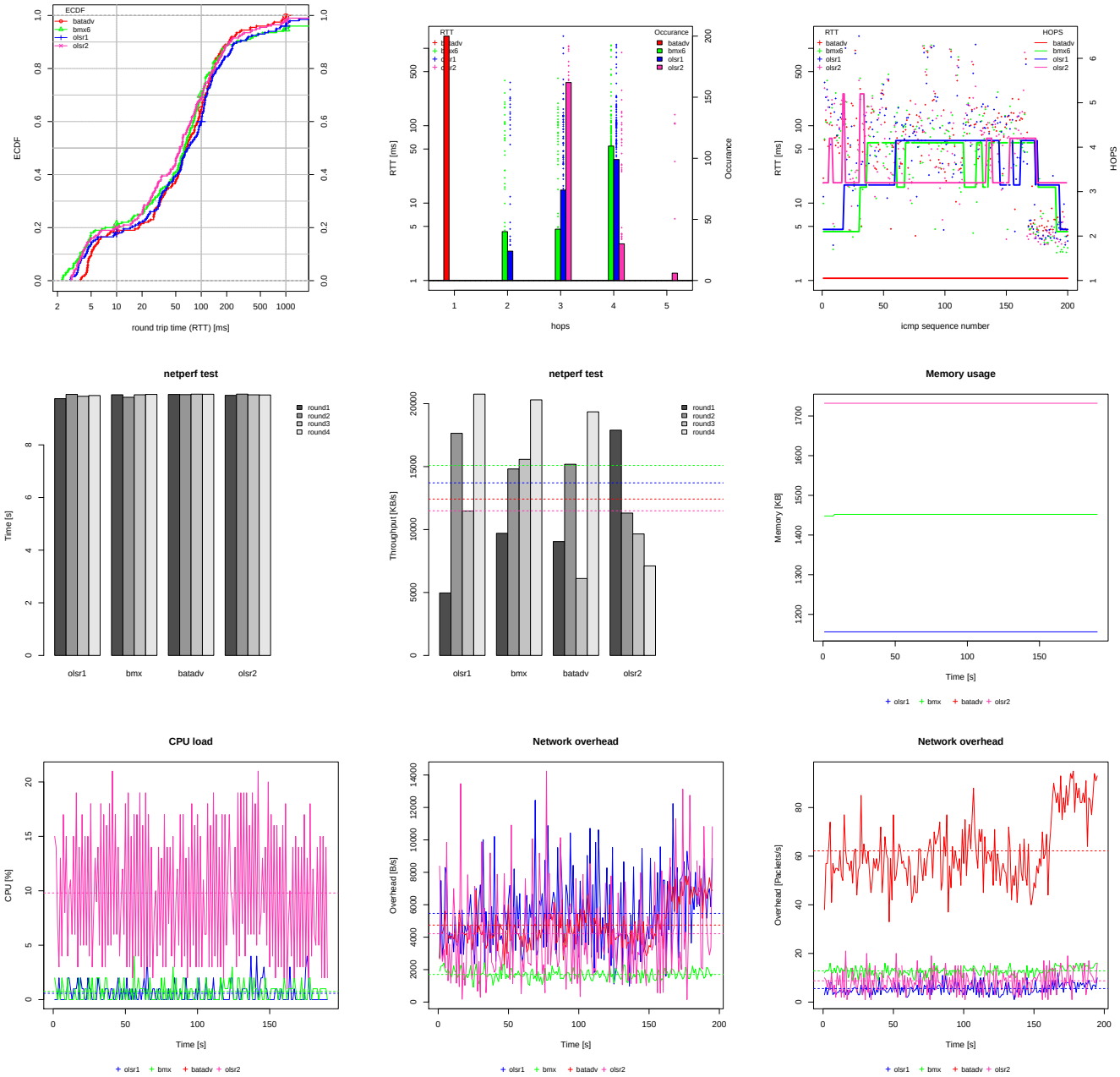

Table 3: Overhead and end-to-end performance between two stationary nodes: 8e417e and c24174

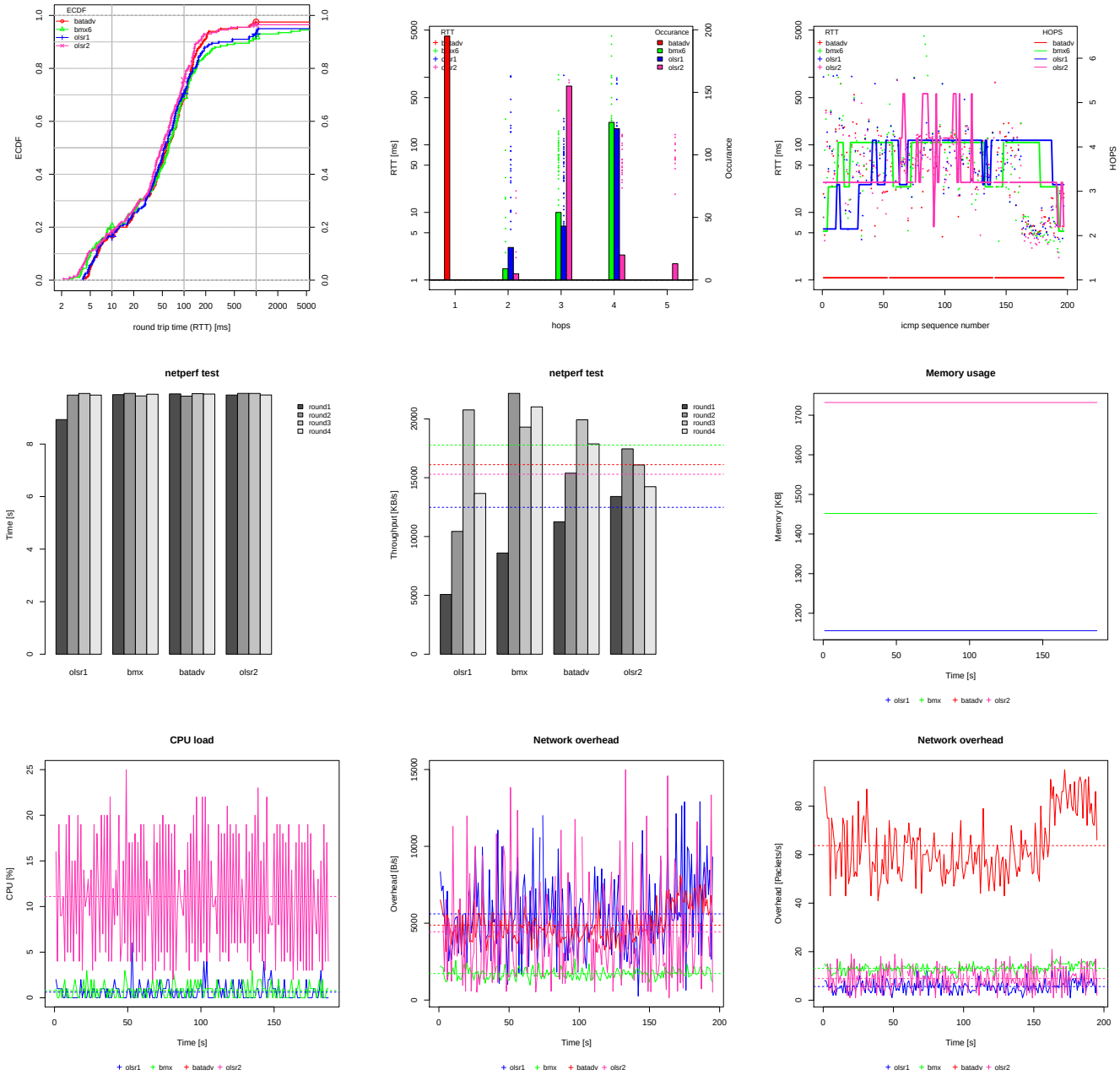

Table 4: Overhead and end-to-end performance between two stationary nodes: e61936 and c24174

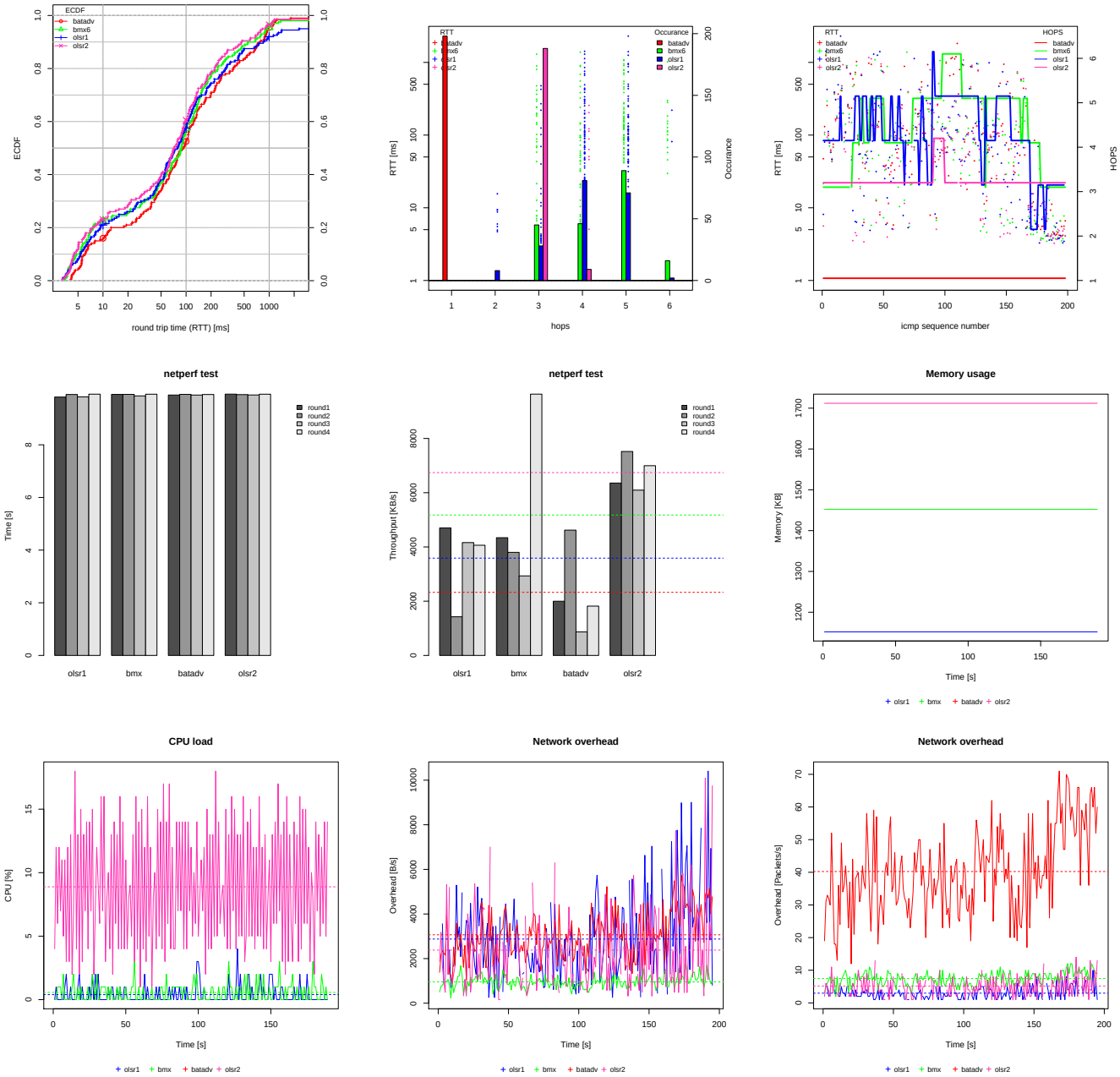

Table 5: Overhead and end-to-end performance between two stationary nodes: f41ab0 and c24174

6.2 All mobile-scenario measurement graphs

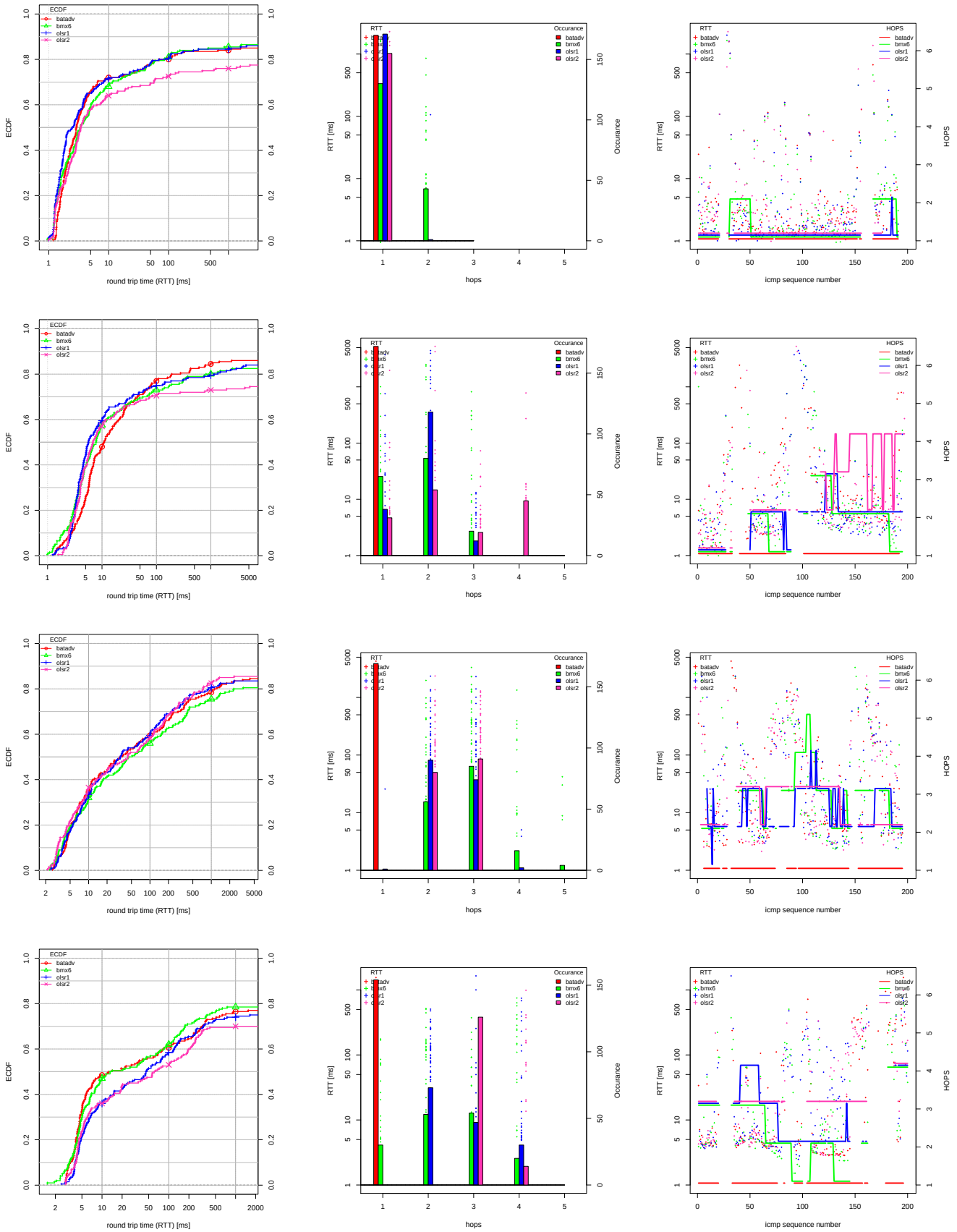

Table 6: End-to-end ping6 performance to mobile node 4174 from aac8, 1936, 1ab0

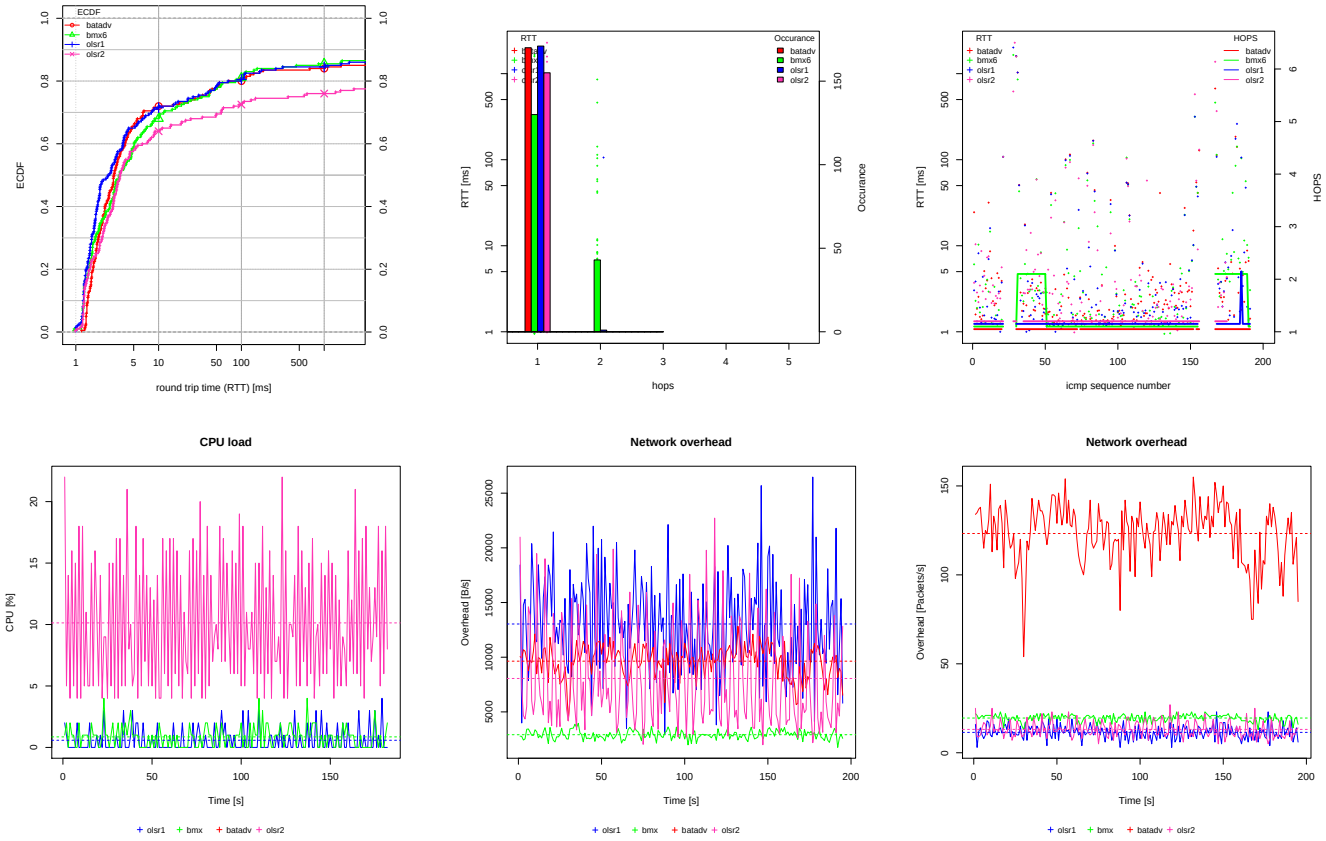

Table 7: Overhead and end-to-end performance to mobile node c24174 from stationary node 51aac8

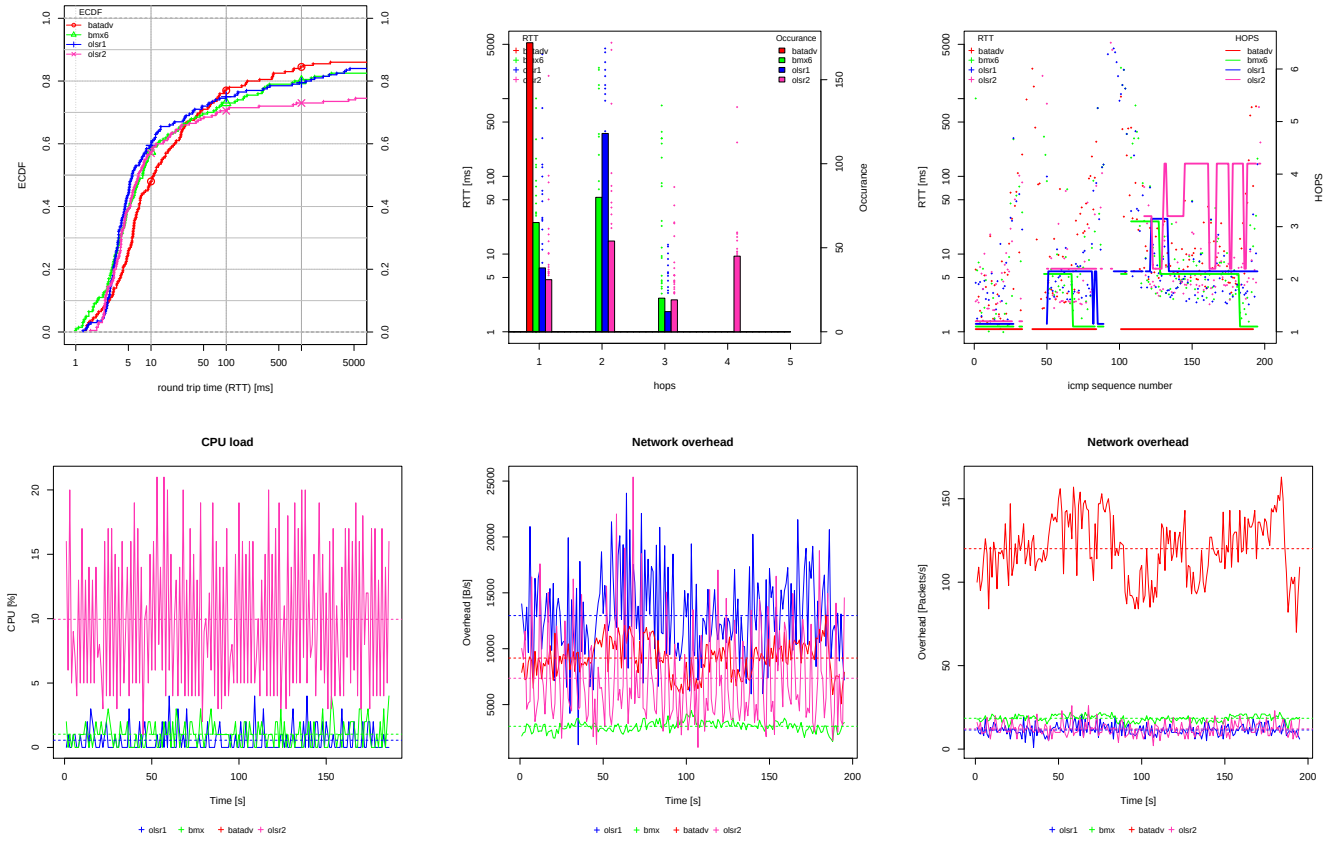

Table 8: Overhead and end-to-end performance to mobile node c24174 from stationary node e61936

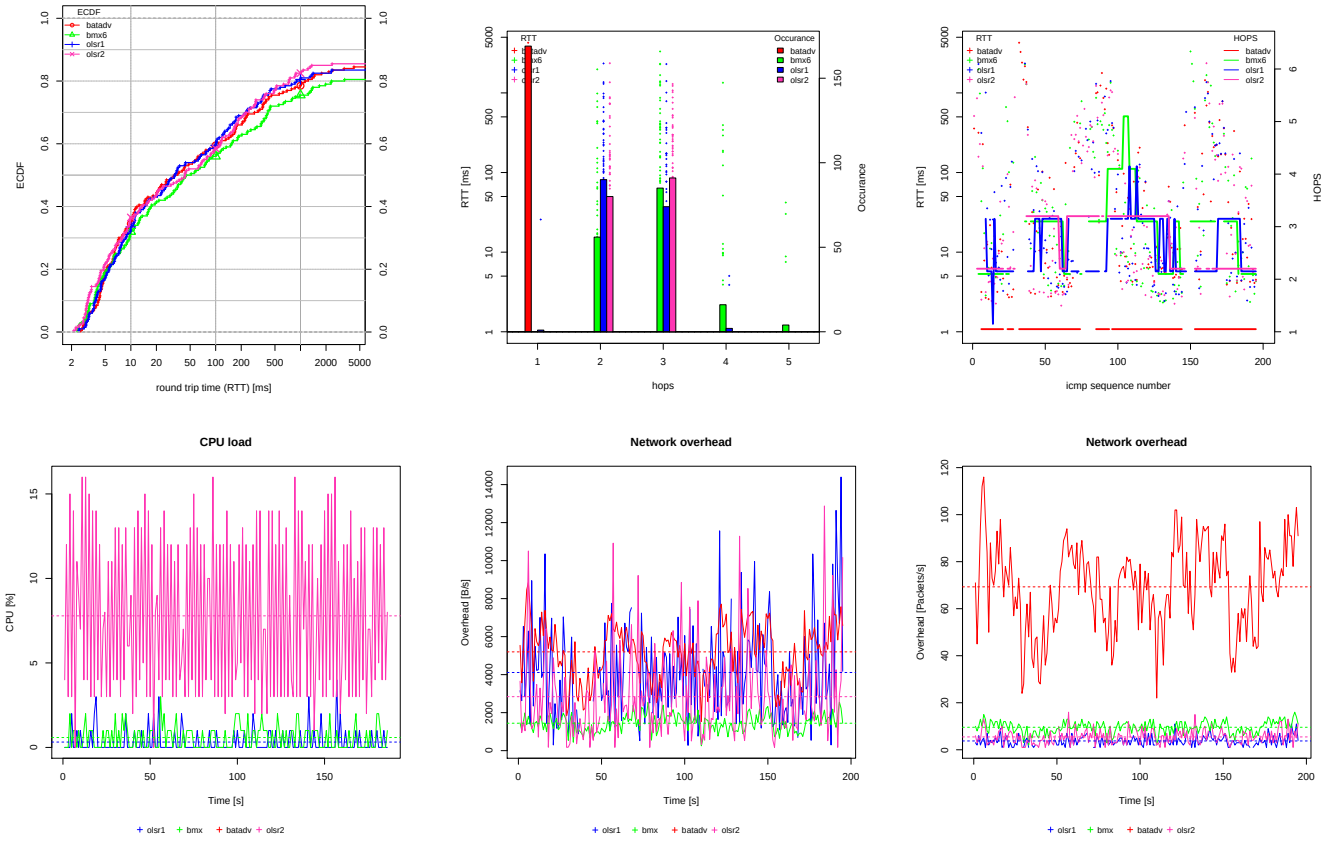

Table 9: Overhead and end-to-end performance to mobile node c24174 from stationary node f41ab0

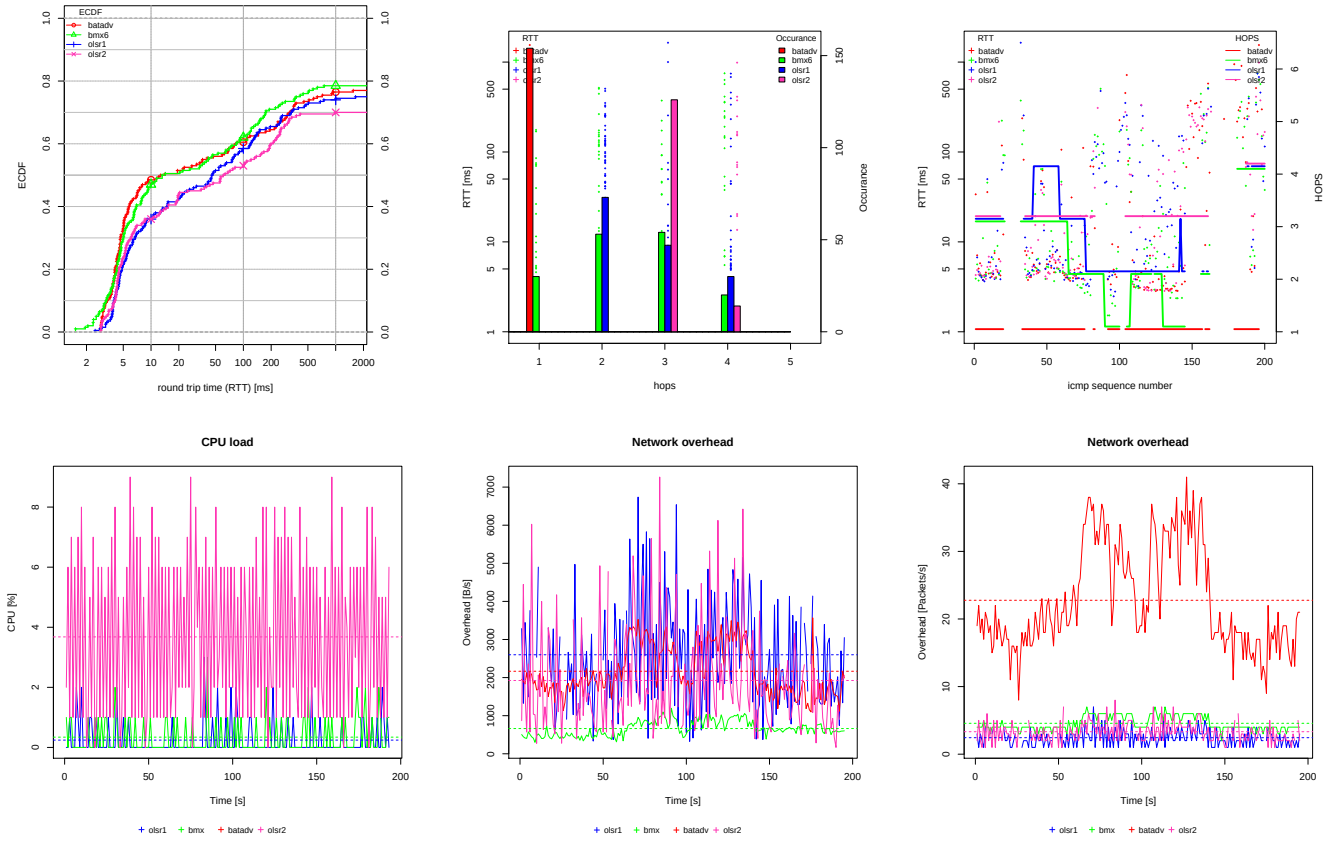

Table 10: Overhead and end-to-end performance to mobile node c24174 from stationary node c2427a