Experience with an Implementation of the *Idle Sense* Wireless Access Method

Yan Grunenberger, Martin Heusse, Franck Rousseau, Andrzej Duda
Grenoble Informatics Laboratory, France

It's about...

- How to confront simulation with implementation of a new 802.11 access method...
- ... and what experience we got from it.

2

802.11 - MAC: The Good

- Distributed algorithm for controlling Congestion Window CW
  - Binary Exponential Backoff (simple local control done by each station)
  - suitable for direct hardware implementation

3

802.11 - MAC: The Bad

- CWmin is only optimal for a given number of stations (3 or 4 for 802.11b)
- Collisions seen as failed transmissions and vice versa
  - problems with rate-adaptation

4

802.11 - MAC: The Ugly

- Short term unfairness for retrying stations
  - results from Binary Exponential Backoff
- Performance anomaly for competing stations that use different data rates

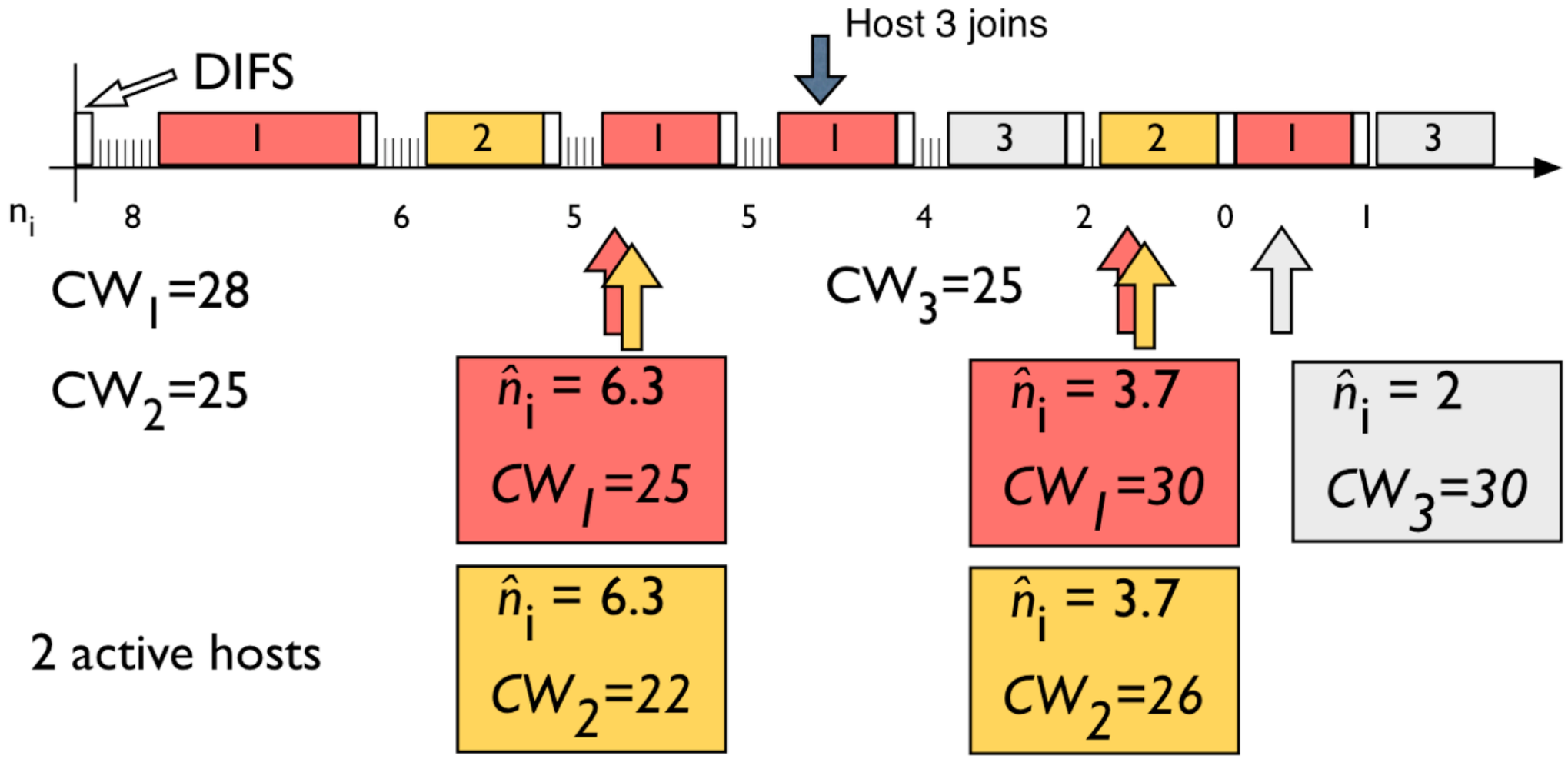
5

Idle Sense Access Method

- No BEB, dynamic control of CW instead
- Keeps distributed mode of operations
  - each station count idle slots between 2 transmissions
  - use this as an estimator of the load to adjust CW
- CW adjustment using AIMD

6

Idle Sense - Example



(α= 1/1.2, ϵ=0.01, ntrans=3, $\overline{n}_i^{target}$ = 5.7)

7

So, what do we need for its implementation?

- Access to slot counting
  - Nearly “realtime” operation...
- Be able to change the backoff mechanism
  - Retransmissions should be software-based
  - Generate random values from any interval

8

What development platform?

- Hardware-based frame sending
  - Atheros-based hardware with Madwifi software : good, but not good enough
- Software-based (read : firmware) management of packet transmission
  - Intel and probably other firmware-based

9

Intel programmable cards

- Code development and debugging at Intel Labs in Cambridge
  - Many Thanks to Dina Papaggiannaki
- Output: Cards with modified firmware

10

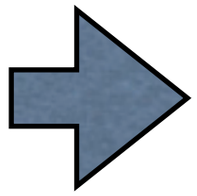
Implementation

- Programming a highly constrained device
  - memory is limited, each line counts !
  - no floating point, integers only
  - no complex arithmetic operations, only register shifting

What is problematic ?

- Our algorithm was developed on simulators
  - AIMD parameters, estimator based on floating values

- $n_{\mathrm{i}}^{\mathrm{target}} = 3.91$
- $\frac{1}{\alpha} = 1.0666$
- $\epsilon = 6.0$
- $\beta = 0.75$
- $\gamma = 4$



- $n_{\mathrm{i}}^{\mathrm{target}} = 4$
- $\alpha = 1 - 1/16$
- $\epsilon = 6$
- $\beta = 1$
- $\gamma = 4$

  - large values of CW were possible

12

... even more...

- 802.11 BEB only uses CW = 2^k
- Generating a random number is easy, but
  - Generally use a Linear Feedback Shift Register: returns a value from an interval [0, 2^k-1]
- Idle Sense requires generating random numbers from any interval
- How to generate them on the constrained device?

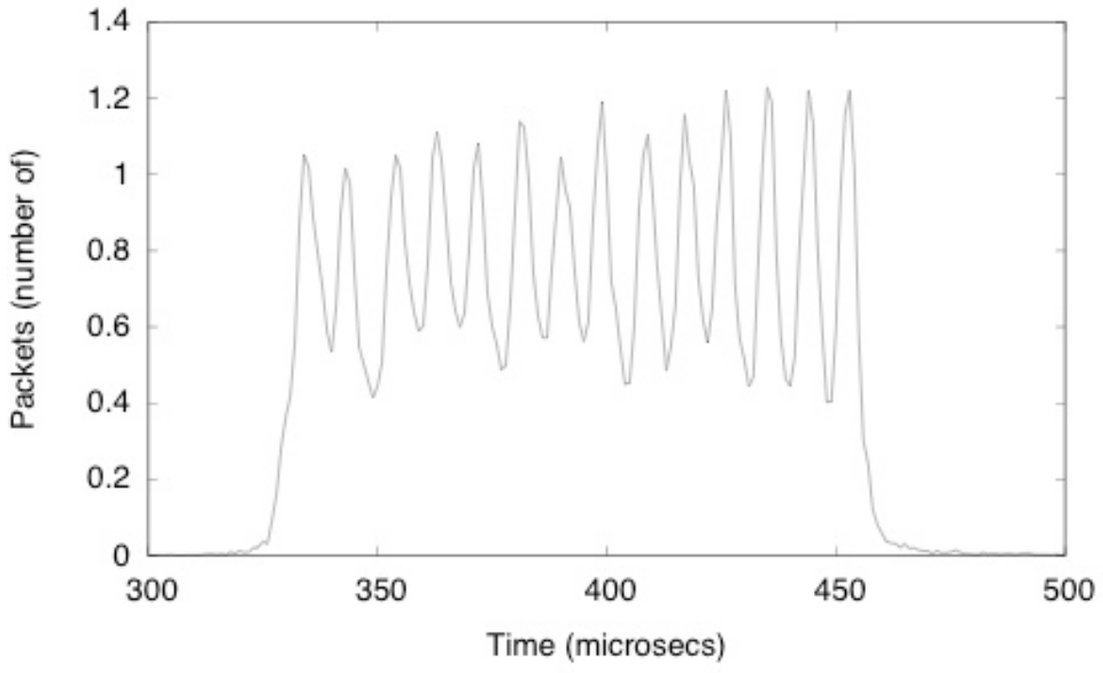
13

Random Generator

- A simple trick :

Variable	Value	Comment
Random register	**10010011** 10101110	generated
Random register	00000000 **10010011**	register shift
CW	00000000 **00001101**	CW=13
Temp	**00000111** 01110111	Random * CW
Backoff	00000000 **00000111**	Backoff = 7

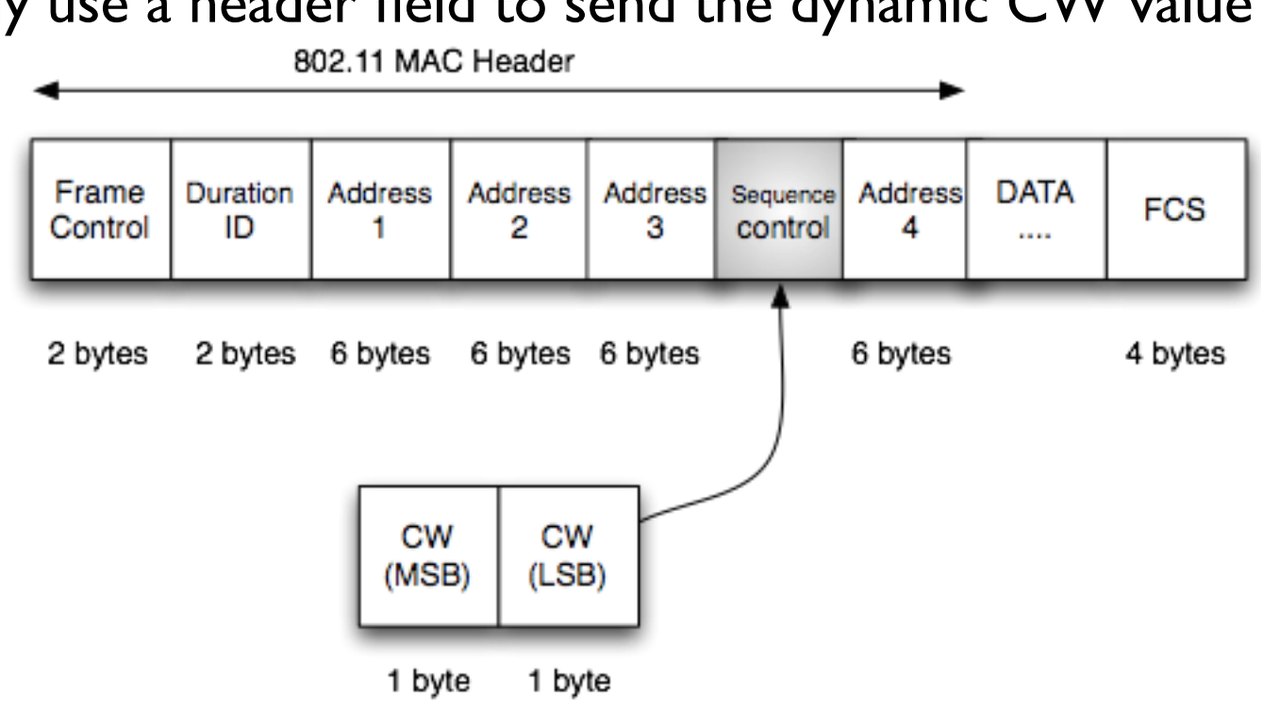
- Check randomness :



14

... but how to check CW evolution?

- Debugging a new method on off-the-shelf hardware
  - simply use a header field to send the dynamic CW value



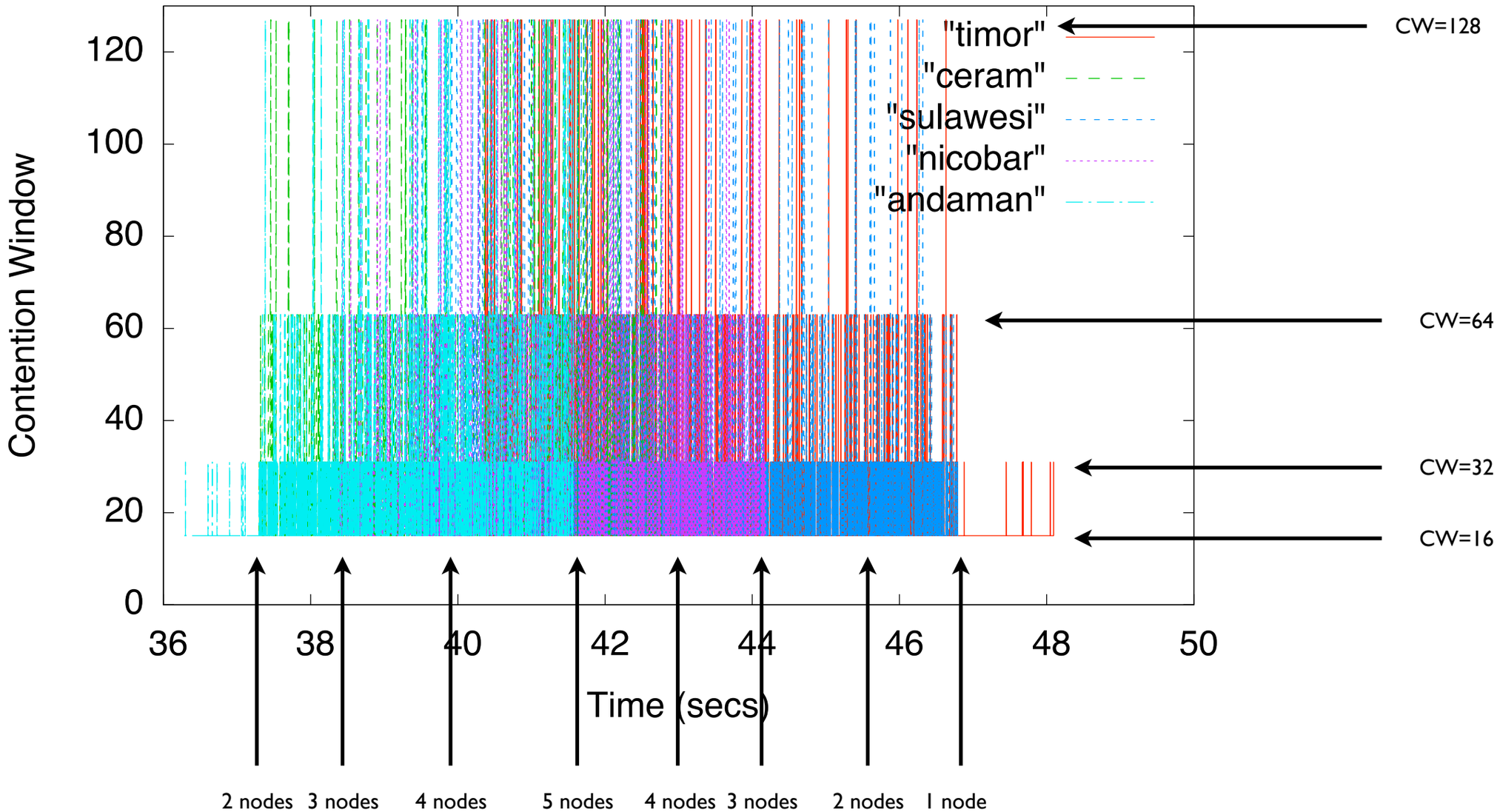
15

So, does it work?

- Yes, but we could measure it for a small number of stations
  - 6 stations hardly compare with 20 simulated nodes
  - Idle Sense benefits for a cell with many stations
- Validation - we compared standard DCF microcode and our new method...

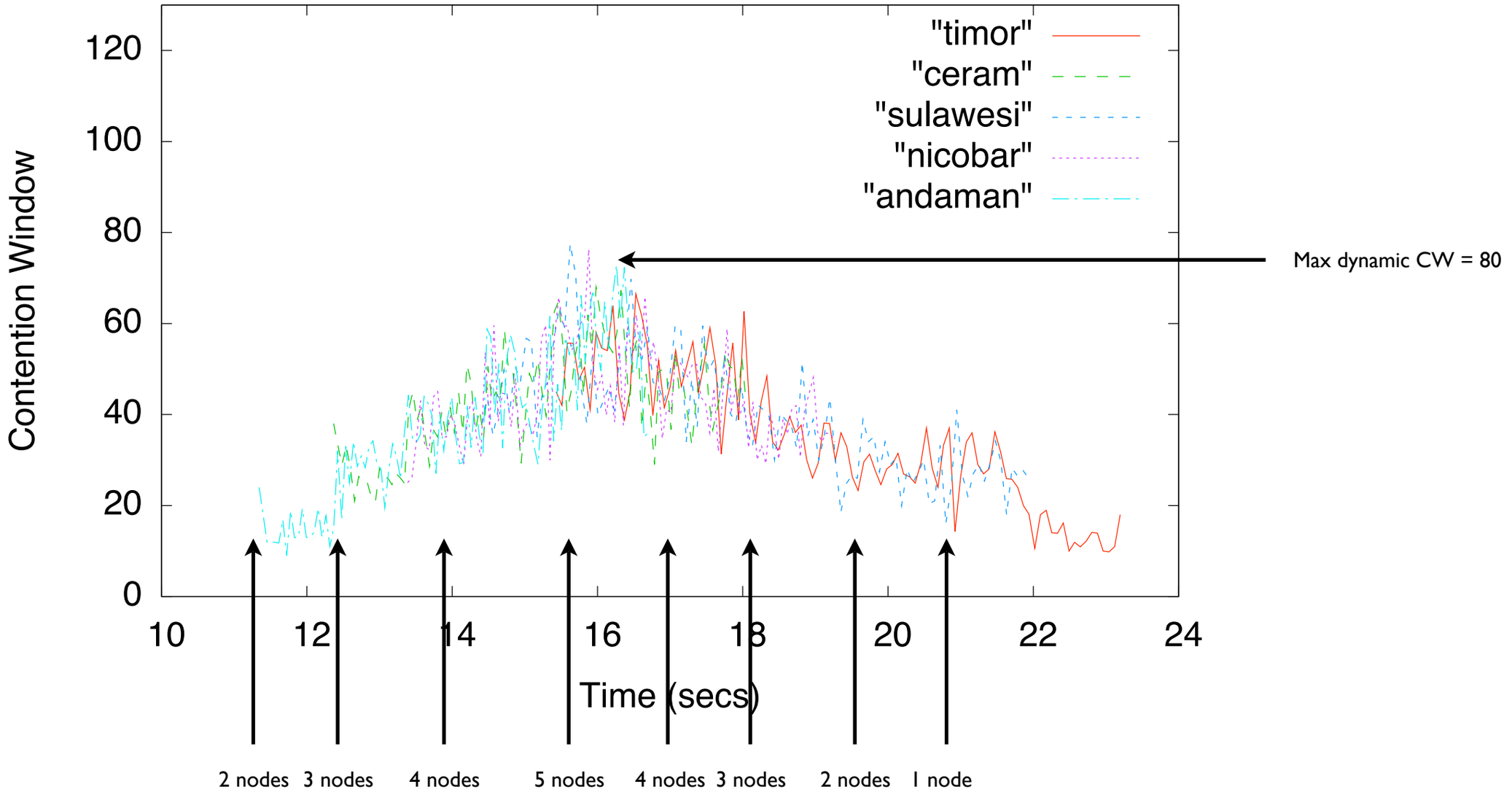
16

Evolution of contention window - DCF

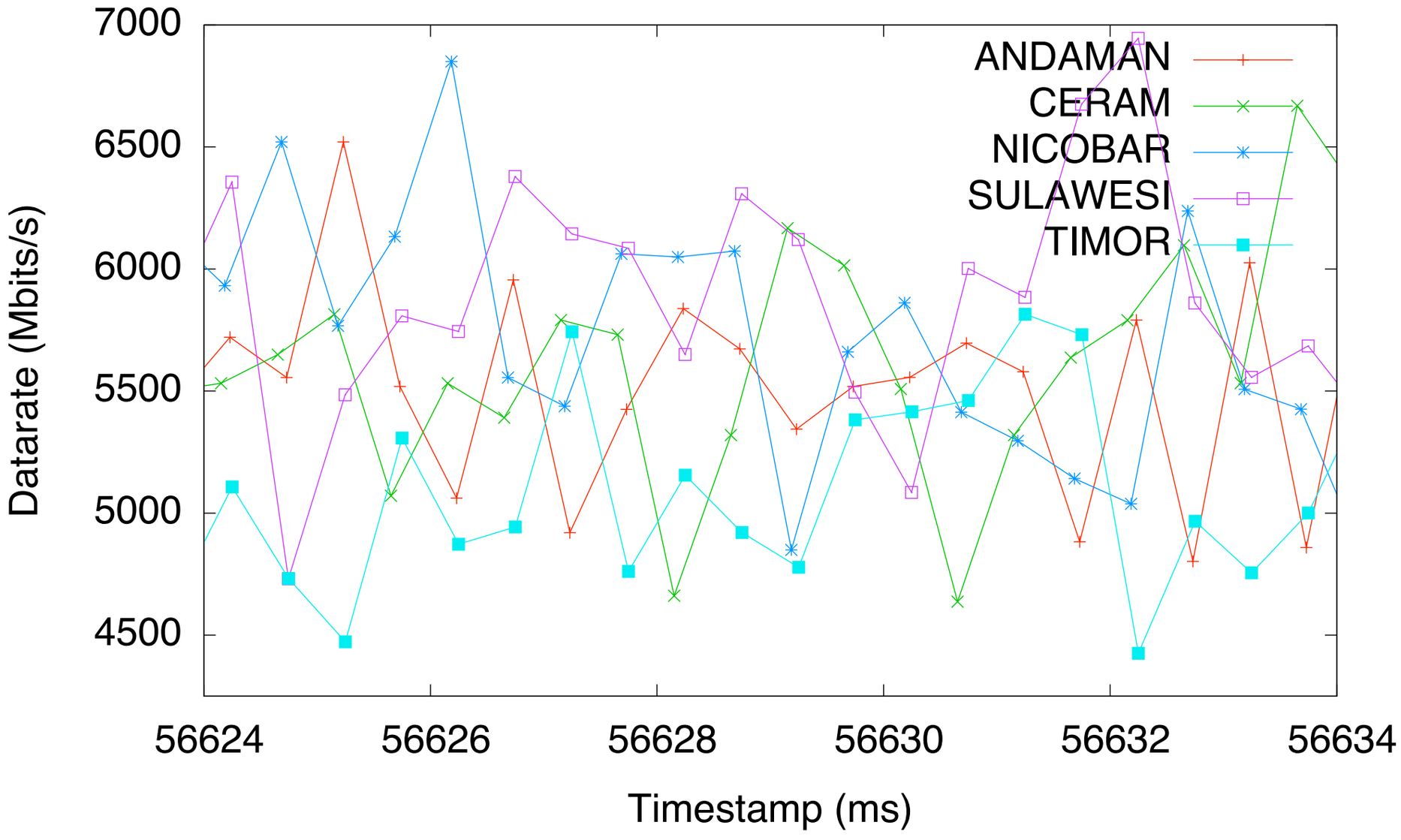


17

Evolution of contention window - IdleSense

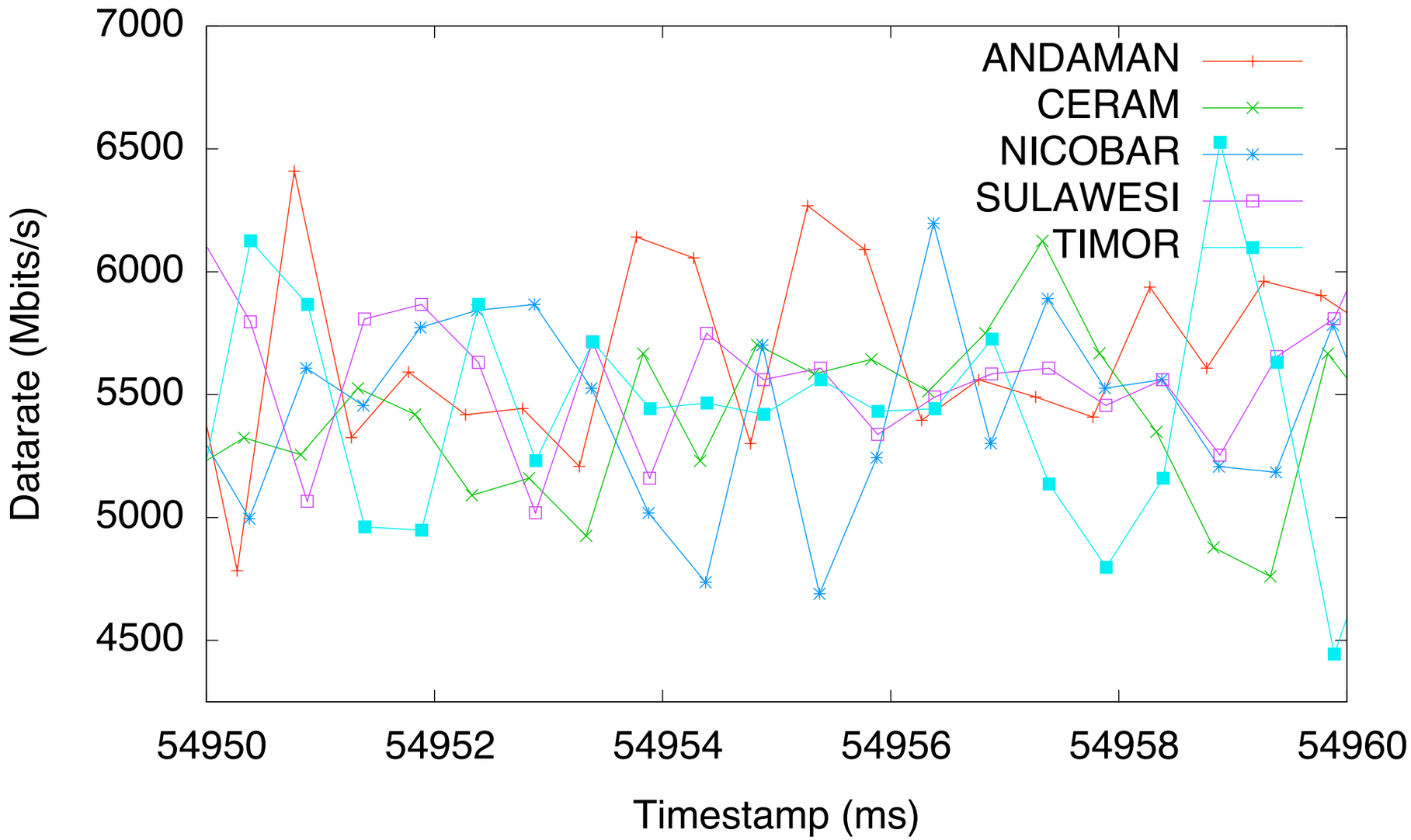


Throughput on DCF



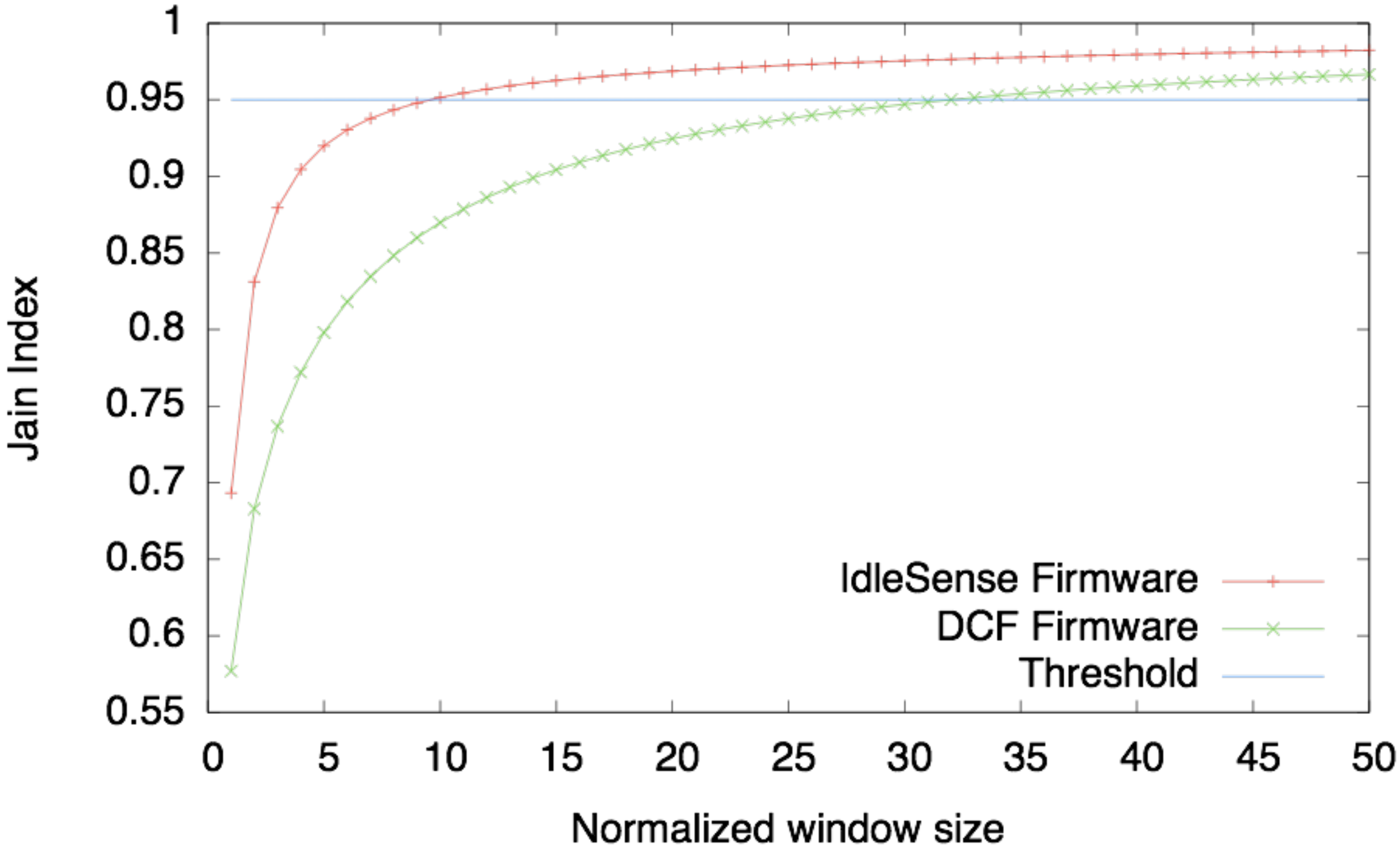
19

Throughput on Idle Sense



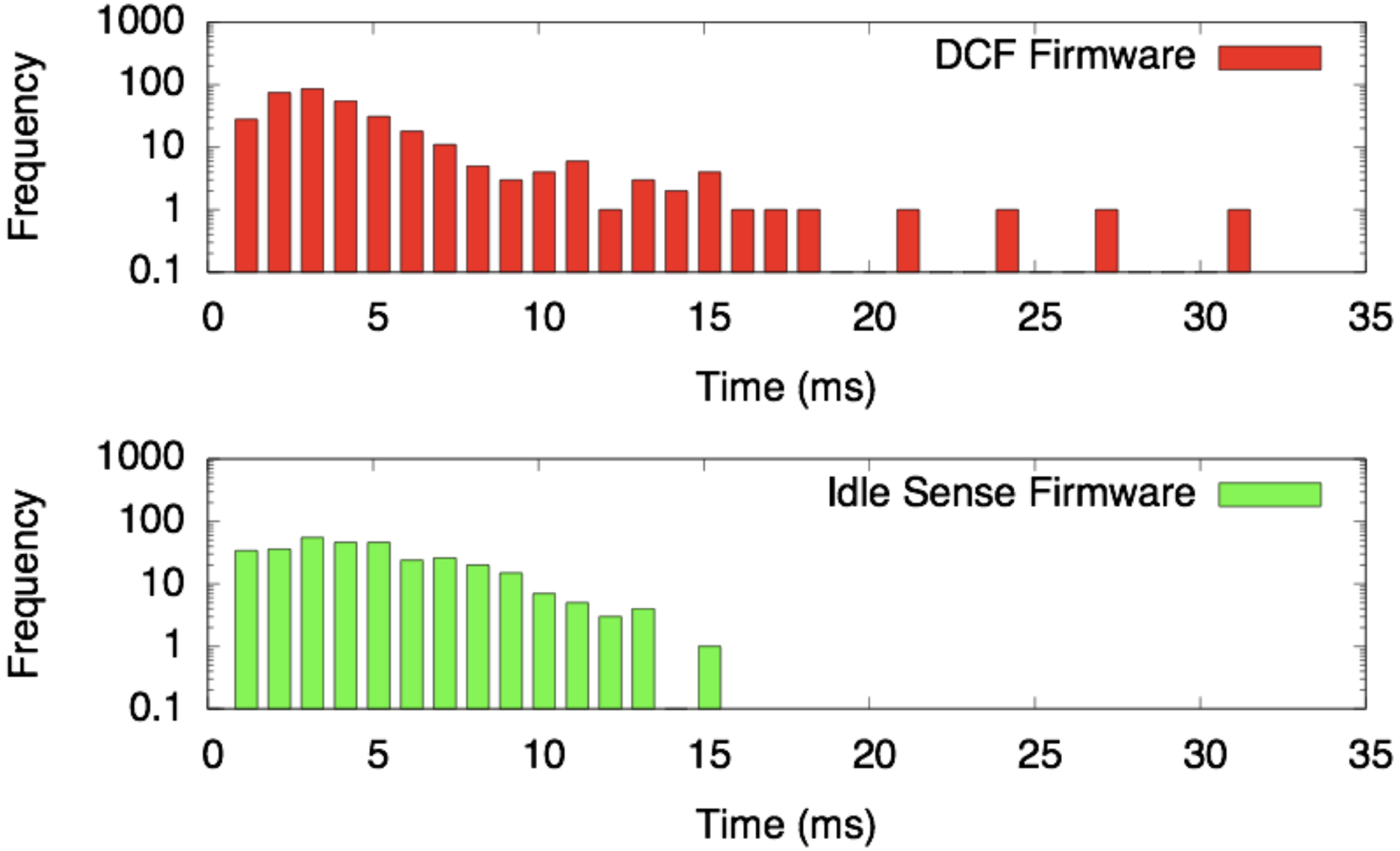
20

Fairness


Jain Index
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0
5
10
15
20
25
30
35
40
45
50
Normalized window size
IdleSense Firmware
DCF Firmware
Threshold

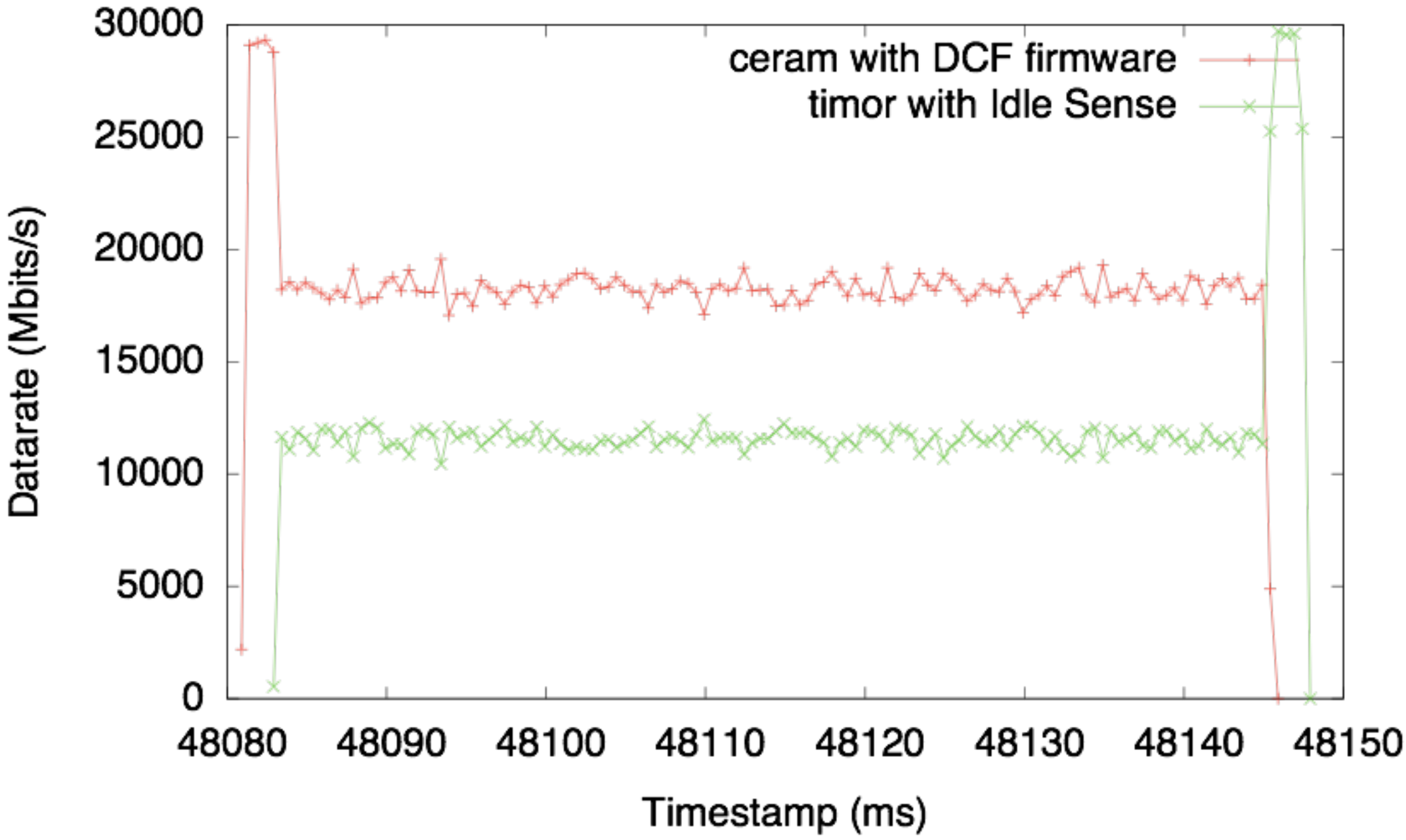
21

Latency


Frequency
1000
100
10
1
0.1
0
5
10
15
20
25
30
35
DCF Firmware
Time (ms)
Frequency
1000
100
10
1
0.1
0
5
10
15
20
25
30
35
Idle Sense Firmware
Time (ms)

22

Interoperability



23

... so yes, it works.

- Off-the-shelf hardware can be useful for implementing new MAC methods
  - but some hacks are needed
- Some thoughts about simulation vs. implementation
  - simulators were needed to tune parameters and predict behavior
  - implementation has revealed some limitations: fixed values, random generation
  - now, we need simulators that give results close to measurements

24

That’s all folks!

- Questions ?
  - yan.grunenberger@imag.fr

25