# *Pathrate* on mobile environments
# An implementation on Android

Antonio Macrì · Francesco Racciatti · Silvia Volpe

July 8, 2013

# Contents


1. How *pathrate* works
   - Packet-pair technique
   - Capacity modes
   - Packet trains
   - Capacity estimation methodology
2. *pathrate* in mobile environments
   - Main problems
   - Interrupt Coalescence
   - IC distortions
3. SmartPathrate
   - Objectives
   - Efficiency
   - The core procedure
   - Pseudo-IC

Section 1

# How *pathrate* works

# Pathrate

## Pathrate goal

Pathrate is a tool that calculates the capacity of a path.

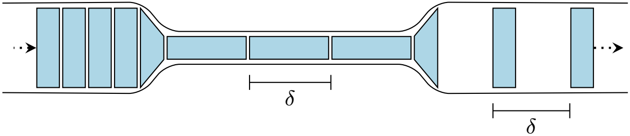
It works in 2 steps:

- phase 1: uses packet pairs to obtain a set of capacity estimations
- phase 2: uses packet trains to compute the lower bound of the capacity

# Packet-pair technique

## Scenario

Two consecutive *probing packets* leave the sender *back-to-back* and arrive at the receiver with a *dispersion* (spacing) that is determined by the *narrow link* in the path

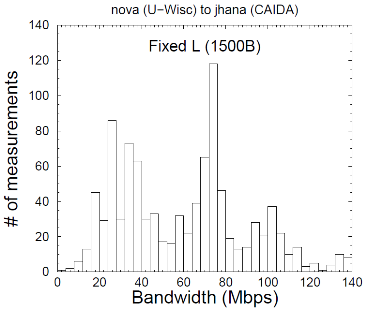



$$b = L/\delta$$

# Capacity modes

## Packet-pair bandwidth distribution

All capacities are used to compute a probability density function called the *packet-pair bandwidth distribution* $\mathcal{B}$



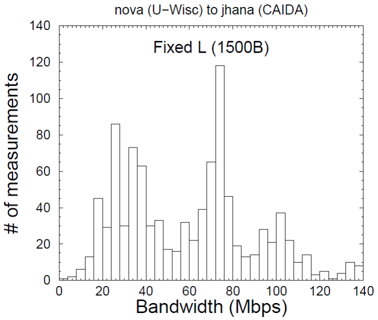
# Capacity modes (cont.)

## Problem 1

The distribution of the capacities is *multimodal*

## Problem 2

The correct path capacity is likely to be the global mode only when the path is *lightly loaded*



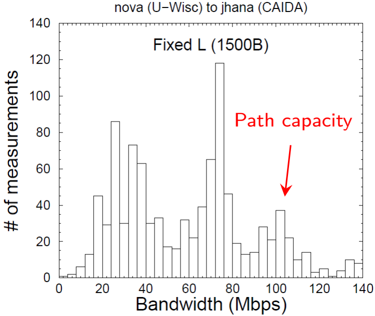
# Capacity modes (cont.)

## Problem 1

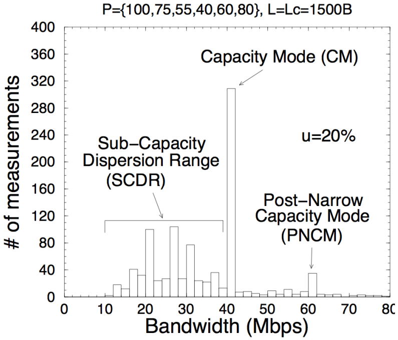
The distribution of the capacities is *multimodal*

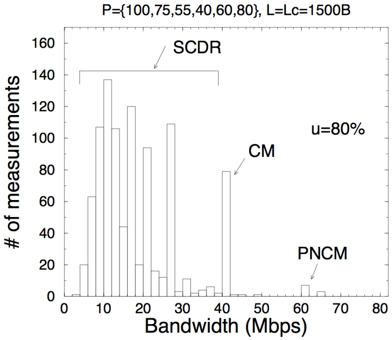
## Problem 2

The correct path capacity is likely to be the global mode only when the path is *lightly loaded*



# Capacity modes (cont.)

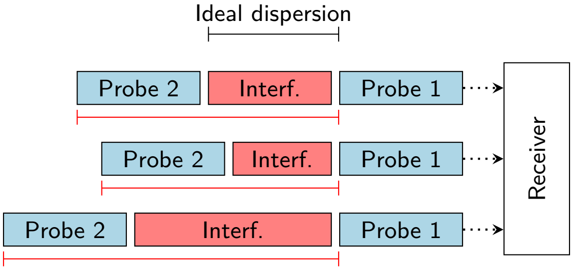


# Capacity modes (cont.)

## Wrong estimations

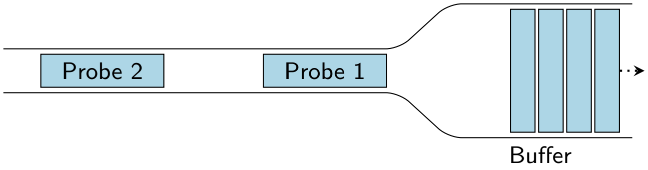
Underestimations due to cross-traffic (SCDR zone)



# Capacity modes (cont.)

## Wrong estimations

Overestimations due to non-empty buffers in some post-narrow node (PNCMs)



# Packet trains

Generalization: using packet trains ($N > 2$ back-to-back packets of the same size $L$) we can calculate the bandwith as:

$$b(N) = \frac{(N-1)L}{\Delta(N)}$$

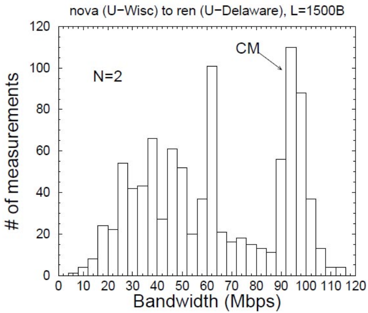
# Packet trains

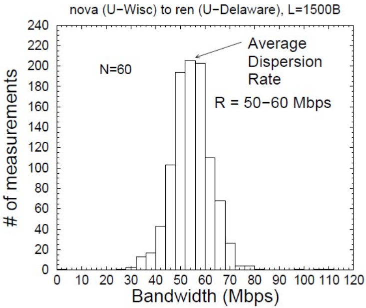
Generalization: using packet trains ($N > 2$ back-to-back packets of the same size $L$) we can calculate the bandwith as:

$$b(N) = \frac{(N-1)L}{\Delta(N)}$$

When the train length $N$ is sufficiently large, bandwidth measurements tend toward a single value leading to a unimodal distribution that becomes *independent of N*: this value is called the *Average Dispersion Rate* (ADR)

# Average Dispersion Rate



# Capacity estimation methodology

## SCDR and PNCMs countermeasures

Varying probing packet size among different packet pairs makes SCDR modes wider and weaker. Probing packets as large as possible (but not too much) reduce the creation of PNCMs

# Capacity estimation methodology

## SCDR and PNCMs countermeasures

Varying probing packet size among different packet pairs makes SCDR modes wider and weaker. Probing packets as large as possible (but not too much) reduce the creation of PNCMs

## ADR as lower bound

The ADR is a *lower bound* of the capacity of the path

# Capacity estimation methodology

### SCDR and PNCMs countermeasures

Varying probing packet size among different packet pairs makes SCDR modes wider and weaker. Probing packets as large as possible (but not too much) reduce the creation of PNCMs

### ADR as lower bound

The ADR is a *lower bound* of the capacity of the path

### Final capacity estimate

In the packet pair bandwidth distribution, ignore modes below the ADR mode and choose the one with the maximum *figure of merit*

Section 2

# *pathrate* in mobile environments

# Main problems

Why do not port *pathrate* to Android?

- long running time ($15 \div 30$ mins)

# Main problems

Why do not port *pathrate* to Android?

- long running time ($15 \div 30$ mins)
- no care for consumed traffic ($100 \div 180$ MB)

# Main problems

Why do not port *pathrate* to Android?

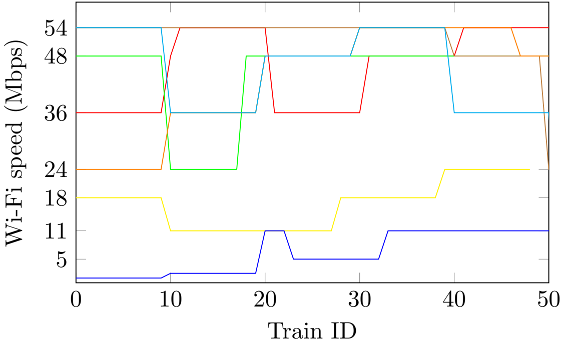
- long running time ($15 \div 30$ mins)
- no care for consumed traffic ($100 \div 180$ MB)
- not suitable for energy-constrained devices

# Not suitable for energy-constrained devices

- CPU frequency is scaled based on battery level or current load

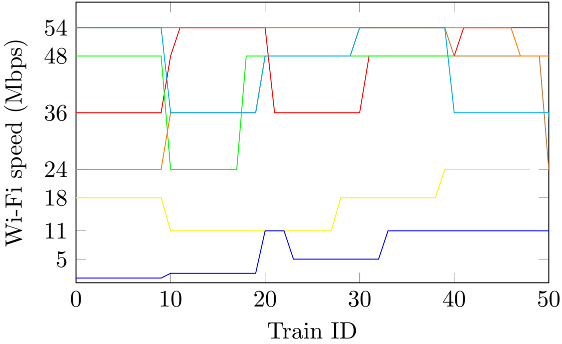
# Not suitable for energy-constrained devices

- CPU frequency is scaled based on battery level or current load
- NIC rate depends on channel conditions



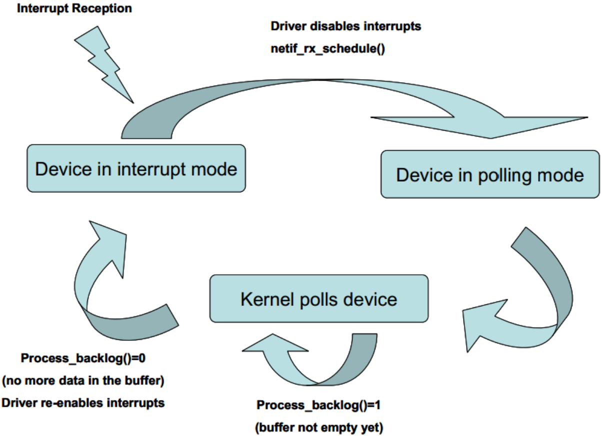
# Not suitable for energy-constrained devices

- CPU frequency is scaled based on battery level or current load
- NIC rate depends on channel conditions

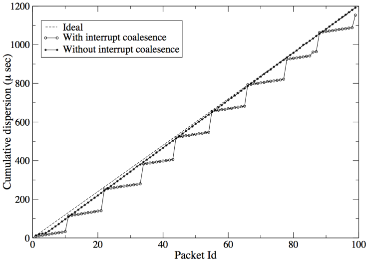



- Network device driver may activate *Interrupt Coalescence* (IC)

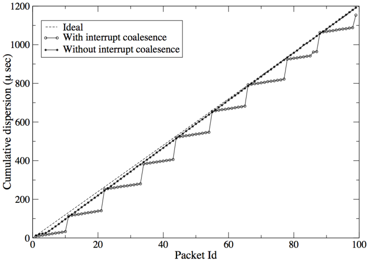
# Interrupt Coalescence



# Interrupt Coalescence in wired environment



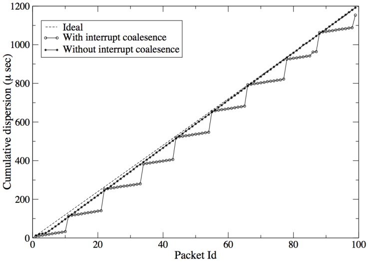
# Interrupt Coalescence in wired environment




## Detect IC

*pathrate* detects coalescence by sending a back-to-back packet train and comparing (at the receiver) measured dispersions with the kernel-to-user latency $\delta_{\text{k-u}}$

# Interrupt Coalescence in wired environment




## Detect IC

*pathrate* detects coalescence by sending a back-to-back packet train and comparing (at the receiver) measured dispersions with the kernel-to-user latency $\delta_{\text{k-u}}$

## Capacity from IC

In case of IC, the capacity of the path can be calculated by dividing the plateau length by the jump height

# Interrupt Coalescence in mobile environment

## Problems

- In the NIC buffer there are packets for other applications

# Interrupt Coalescence in mobile environment

## Problems

- In the NIC buffer there are packets for other applications
- Kernel can adapt CPU frequency to computational load and battery level

# Interrupt Coalescence in mobile environment

## Problems

- In the NIC buffer there are packets for other applications
- Kernel can adapt CPU frequency to computational load and battery level
- Timer resolution can be insufficient (30 $\mu$s)

# Interrupt Coalescence in mobile environment

## Problems

- In the NIC buffer there are packets for other applications
- Kernel can adapt CPU frequency to computational load and battery level
- Timer resolution can be insufficient (30 $\mu$s)
- Height of jump depends on many factors, not only packet buffering (e. g. context switches, scheduling mechanisms)

# Interrupt Coalescence in mobile environment

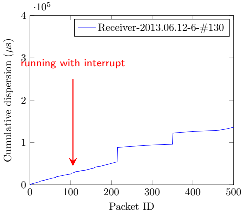
## Problems

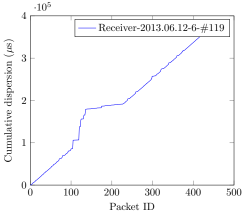
- In the NIC buffer there are packets for other applications
- Kernel can adapt CPU frequency to computational load and battery level
- Timer resolution can be insufficient (30 $\mu s$)
- Height of jump depends on many factors, not only packet buffering (e. g. context switches, scheduling mechanisms)

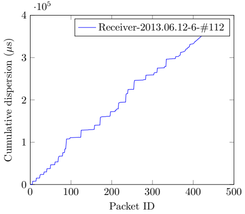
## Pseudo-IC

Due to these factors the observed behavior is highly variable. Then, in mobile environment it is more correct to talk about pseudo-IC rather than simple IC.

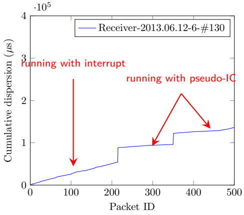
# Pseudo-IC

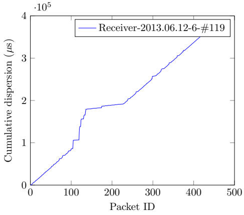

$\cdot 10^5$
Receiver-2013.06.12-6-#130
running with interrupt
Cumulative dispersion ($\mu s$)
Packet ID

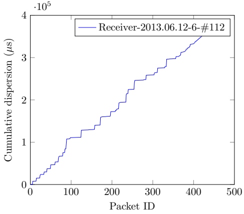

$\cdot 10^5$
Receiver-2013.06.12-6-#119
Cumulative dispersion ($\mu s$)
Packet ID


$\cdot 10^5$
Receiver-2013.06.12-6-#112
Cumulative dispersion ($\mu s$)
Packet ID

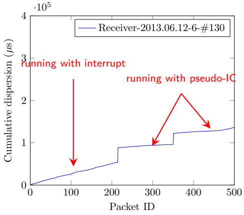
# Pseudo-IC

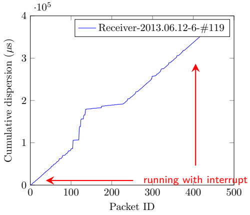

$\cdot 10^5$
Receiver-2013.06.12-6-#130
running with interrupt
running with pseudo-IC
Cumulative dispersion ($\mu s$)
Packet ID
0
1
2
3
4
0
100
200
300
400
500

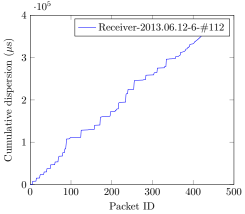

$\cdot 10^5$
Receiver-2013.06.12-6-#119
Cumulative dispersion ($\mu s$)
Packet ID
0
1
2
3
4
0
100
200
300
400
500


$\cdot 10^5$
Receiver-2013.06.12-6-#112
Cumulative dispersion ($\mu s$)
Packet ID
0
1
2
3
4
0
100
200
300
400
500

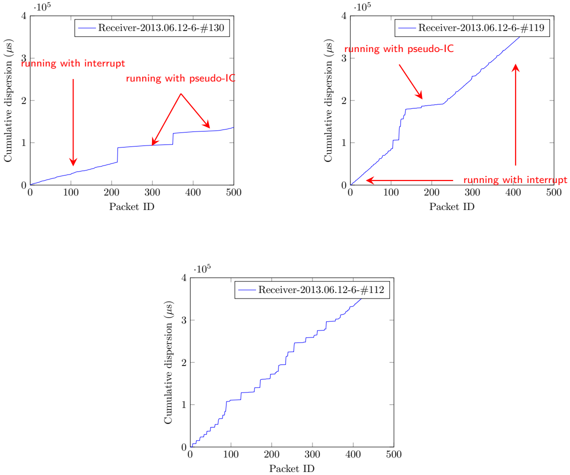
# Pseudo-IC


·10^5
Receiver-2013.06.12-6-#130
running with interrupt
running with pseudo-IC
Cumulative dispersion (μs)
Packet ID

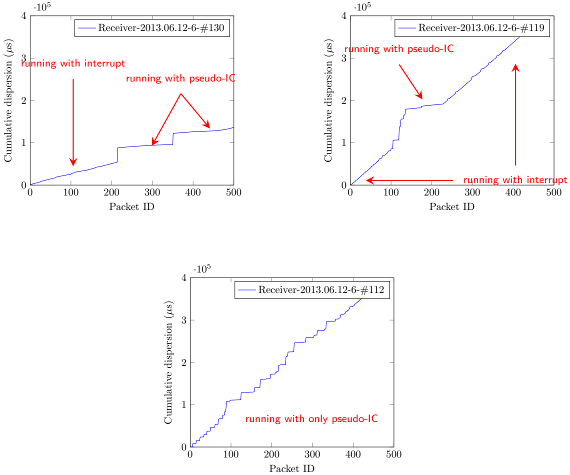

·10^5
Receiver-2013.06.12-6-#119
running with interrupt
Cumulative dispersion (μs)
Packet ID


·10^5
Receiver-2013.06.12-6-#112
Cumulative dispersion (μs)
Packet ID

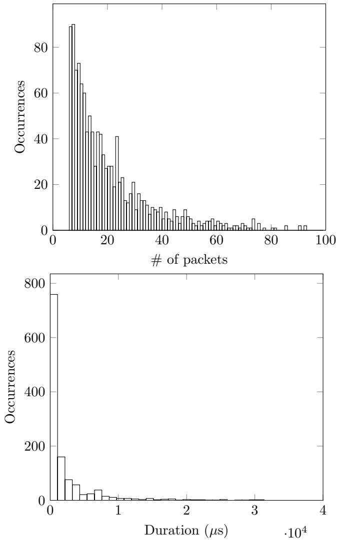
# Pseudo-IC


Receiver-2013.06.12-6-#130
running with interrupt
running with pseudo-IC
Cumulative dispersion ($\mu s$)
Packet ID
$\cdot 10^5$
Receiver-2013.06.12-6-#119
running with pseudo-IC
running with interrupt
Cumulative dispersion ($\mu s$)
Packet ID
$\cdot 10^5$
Receiver-2013.06.12-6-#112
Cumulative dispersion ($\mu s$)
Packet ID
$\cdot 10^5$

# Pseudo-IC


·10^5
Receiver-2013.06.12-6-#130
running with interrupt
running with pseudo-IC
Cumulative dispersion (μs)
Packet ID
·10^5
Receiver-2013.06.12-6-#119
running with pseudo-IC
running with interrupt
Cumulative dispersion (μs)
Packet ID
·10^5
Receiver-2013.06.12-6-#112
running with only pseudo-IC
Cumulative dispersion (μs)
Packet ID

# Pseudo-IC (cont.)



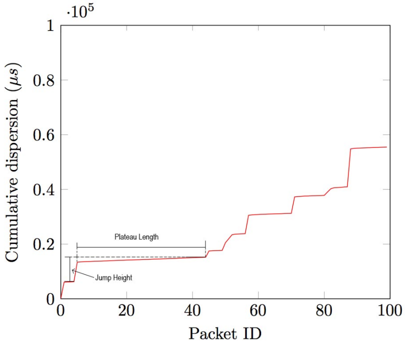

## Plateaus length & jumps height

If the behavior of pseudo-IC was regular then all the occurrences would be concentrated around a single value. The same speech applies to jumps height.

# Pseudo-IC (cont.)

## Calculation of path capacity

Pseudo-IC patterns are very irregular then it is not possible to calculate capacity with jumps and plateaus.



Section 3

# SmartPathrate

# Our objectives

The execution of our application

- should not take a long time
  - user wants to have a result quickly
  - battery life should not be significantly affected

# Our objectives

The execution of our application

- should not take a long time
  - user wants to have a result quickly
  - battery life should not be significantly affected
- should not require too much resources
  - processing complexity of mathematical computations
  - memory usage for buffering partial results

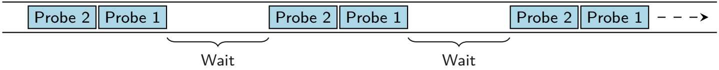
# Our objectives

The execution of our application

- should not take a long time
  - user wants to have a result quickly
  - battery life should not be significantly affected
- should not require too much resources
  - processing complexity of mathematical computations
  - memory usage for buffering partial results
- should use as less data as possible
  - mobile data plan may have traffic limitations
  - more data means more processing

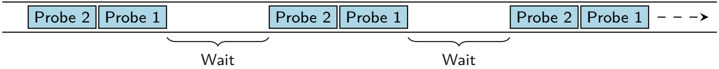
# Packet pairs vs packet trains

- Pathrate: spacing between consecutive pairs
  - to avoid late packets interfere with subsequent pairs
  - drop pair in case of interference or packet losses



# Packet pairs vs packet trains

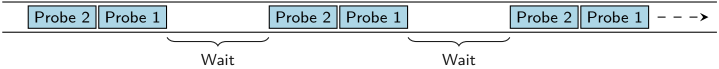
- Pathrate: spacing between consecutive pairs
  - to avoid late packets interfere with subsequent pairs
  - drop pair in case of interference or packet losses




  - total time in wait state is at least about $12 \div 13$ minutes

# Packet pairs vs packet trains

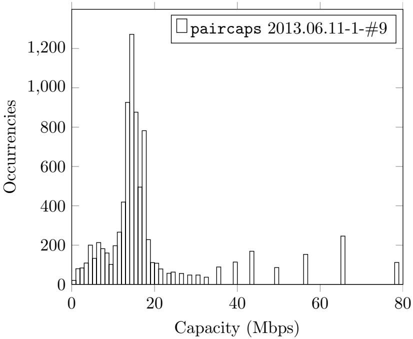
- Pathrate: spacing between consecutive pairs
  - to avoid late packets interfere with subsequent pairs
  - drop pair in case of interference or packet losses




  - total time in wait state is at least about $12 \div 13$ minutes

- SmartPathrate: smaller spacing between trains
  - treat packets from previous trains as cross traffic
  - drop only in case of packet losses

# Processing complexity and memory usage




**Mode calculation**

$$3 \cdot O(n\omega)$$
$$\Downarrow$$
$$O(n\omega) + 2 \cdot O(n \log \omega)$$

**Main change**

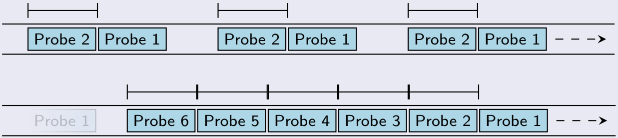
Linear search

$\Downarrow$

Binary search

# Data complexity

## Number of capacity estimates

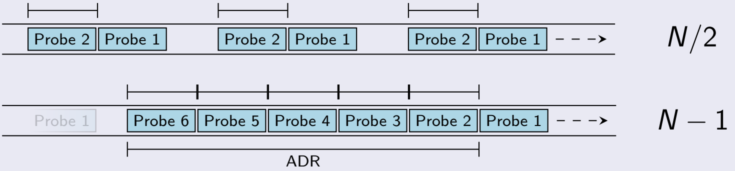



$N/2$

$N-1$

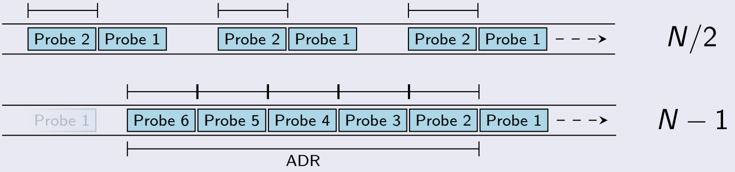
# Data complexity

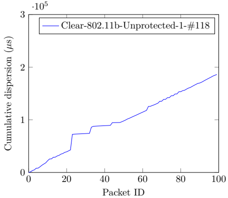
## Number of capacity estimates



# Data complexity

## Number of capacity estimates

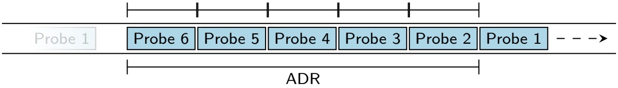



## Data efficiency

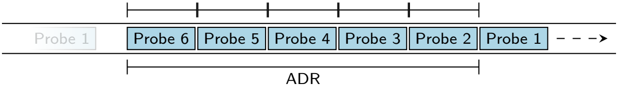
Given a train, try to split it into separate zones, remove coalescence and extract as much information as possible

# The core procedure

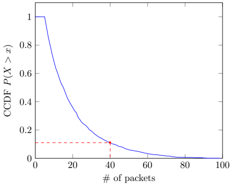



- Send maximum-size packets
- Start with trains of 40 packets
- Gradually increase train length

# The core procedure




- Send maximum-size packets
- Start with trains of 40 packets
- Gradually increase train length

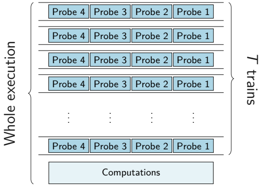


# The core procedure (cont.)

- Proceed round by round
    - Why rounds?

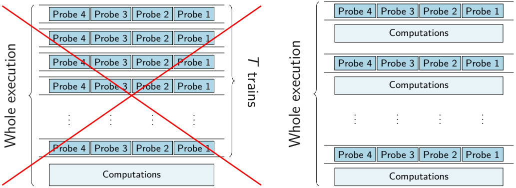
# The core procedure (cont.)

- Proceed round by round
  - Why rounds?



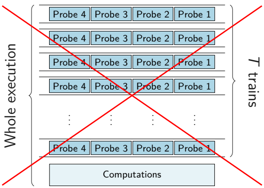
# The core procedure (cont.)

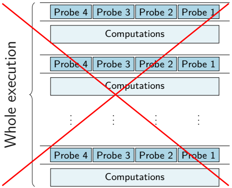
- Proceed round by round
  - Why rounds?

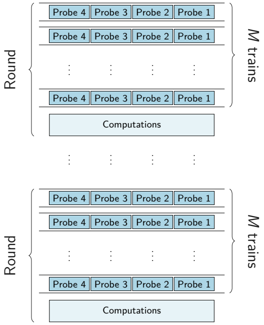


# The core procedure (cont.)

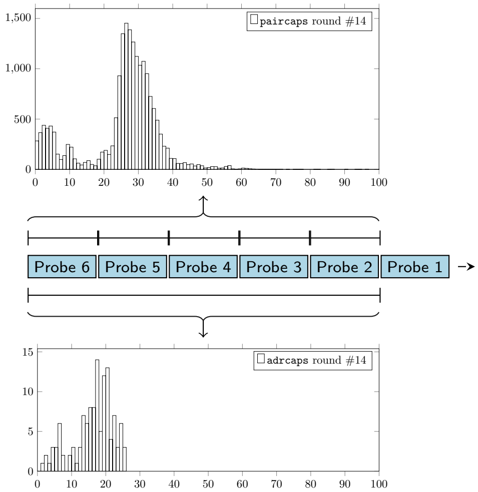
- Proceed round by round
  - Why rounds?



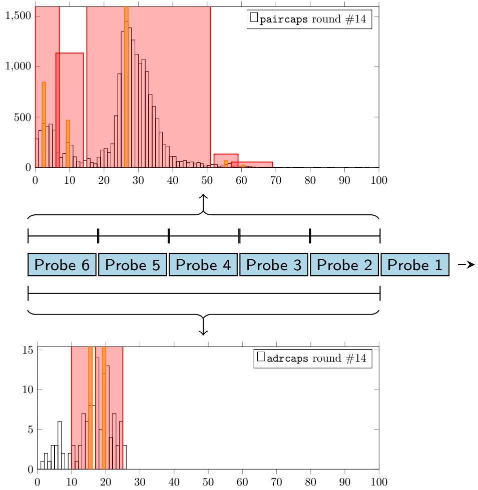
# The core procedure (cont.)




## Calculations at each round

- Calculate pair capacities
- Calculate ADR capacities
- Calculate distribution
- Calculate bin width

# The core procedure (cont.)




## Calculations at each round

- Calculate pair capacities
- Calculate ADR capacities
- Calculate distribution
- Calculate bin width
- Extract modes
- Estimate (temporary) capacity

# Detect pseudo-IC

**Filtering capacities from IC**

Detecting interrupt coalescence is a critical task

# Detect pseudo-IC

## Filtering capacities from IC

Detecting interrupt coalescence is a critical task

## Idea

Why do not reduce the receiver's buffer size by means of `setsockopt`, to reduce or completely suppress IC?

# Detect pseudo-IC

## Filtering capacities from IC

Detecting interrupt coalescence is a critical task

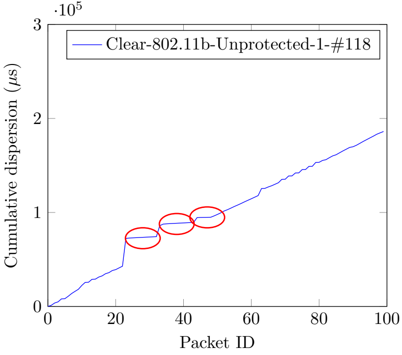
## Idea

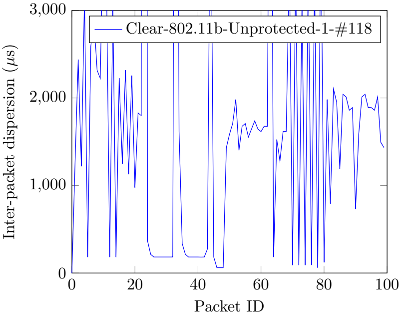
Why do not reduce the receiver's buffer size by means of `setsockopt`, to reduce or completely suppress IC?

## Wrong

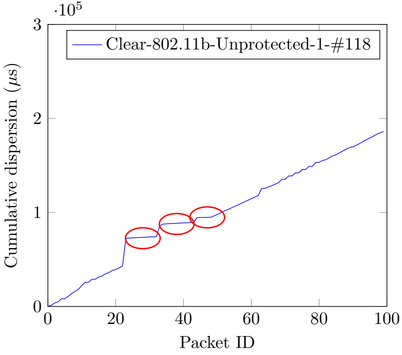
Option `SO_RCVBUF` affects only a buffer related to the *UDP socket*, and has nothing to do with the kernel's buffer used by NIC.
In fact, the receiver's buffer should be as large as possible, to reduce (late) packet drops.

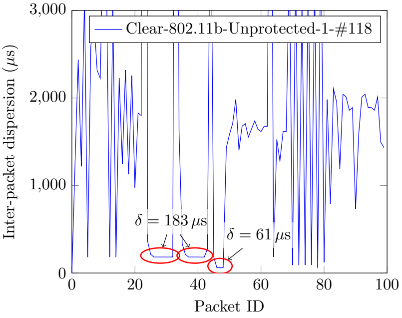
# Detect pseudo-IC (cont.)



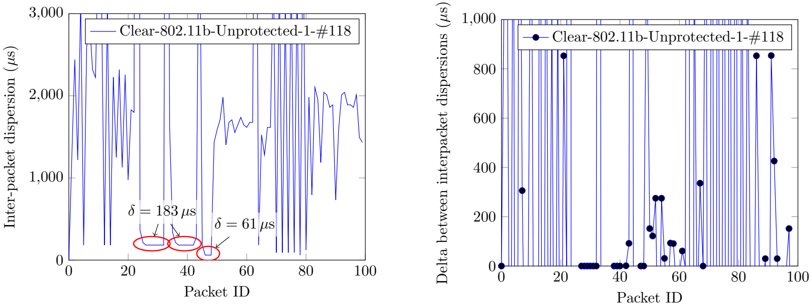
# Detect pseudo-IC (cont.)



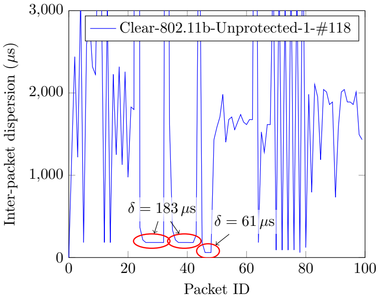



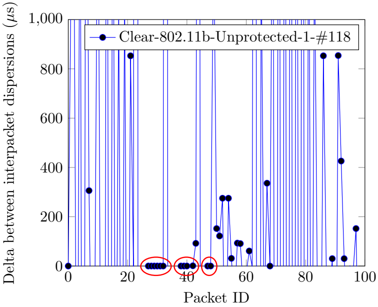

$$\frac{1500 \times 8\ \text{bit}}{183\ \mu s} \approx 65.6\ \text{Mbps}$$

# Detect pseudo-IC (cont.)

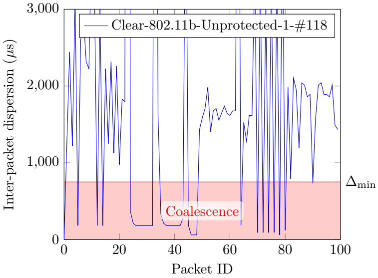


# Detect pseudo-IC (cont.)



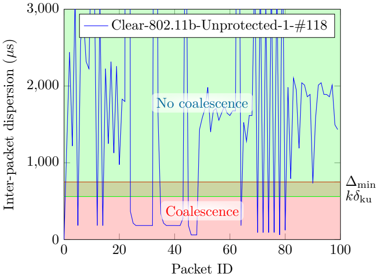
# Detect pseudo-IC (cont.)




## Two *features*

Packet pair dispersion $\Rightarrow$ Minimum possible dispersion

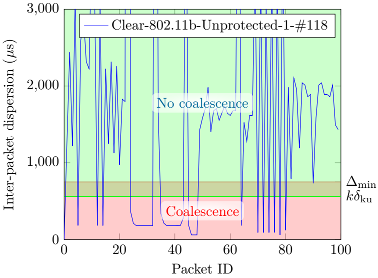
# Detect pseudo-IC (cont.)

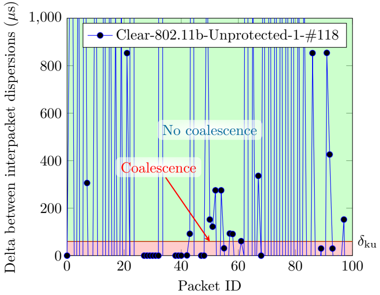



## Two *features*

| | | |
|---|---|---|
| Packet pair dispersion | $\Rightarrow$ | Minimum possible dispersion |
| Packet pair dispersion | $\Rightarrow$ | Kernel-to-user latency |

# Detect pseudo-IC (cont.)



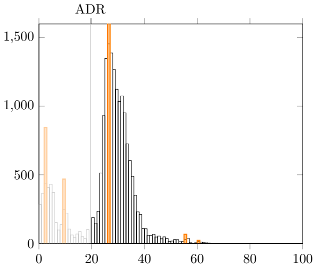

## Two *features*

| | | |
|---|---|---|
| Packet pair dispersion | $\Rightarrow$ | Minimum possible dispersion |
| Packet pair dispersion | $\Rightarrow$ | Kernel-to-user latency |
| Delta of packet pair dispersions | $\Rightarrow$ | Kernel-to-user latency |

# Final capacity estimate

Procedure:

- Proceed round by round
- Do calculations at the end of each round
- Try to determine the path capacity
- Check for convergence of partial results



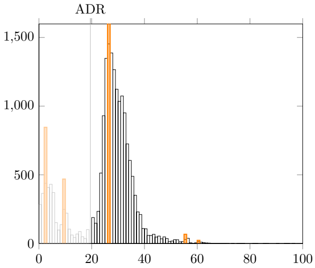

Typical results:

- Execution time: $15 \div 30$ mins
- Generated traffic: $100 \div 180\,\text{MB}$

# Final capacity estimate

Procedure:

- Proceed round by round
- Do calculations at the end of each round
- Try to determine the path capacity
- Check for convergence of partial results




Typical results:

- Execution time: 15 ÷ 30 mins ⇒ 1 ÷ 2 mins
- Generated traffic: 100 ÷ 180 MB ⇒ 15 ÷ 30 MB

# Questions?

## Questions

# References I


URL: http://vger.kernel.org/~davem/skb_sk.html.

Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux Device Drivers*. Third edition. O'Reilly, 2005. URL: http://lwn.net/images/pdf/LDD3/ch17.pdf.

Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. "Packet-dispersion techniques and a capacity-estimation methodology". In: *IEEE/ACM Transactions on Networking*. Vol. 12. 6. Dec. 2004, pp. 963–977.

Jamal Hadi Salim, Robert Olsson, and Alexey Kuznetsov. "Beyond Softnet". In: *Proceedings of the 5th Annual Linux Showcase & Conference*. Oakland, California, USA, 2001. URL: http://static.usenix.org/publications/library/proceedings/als01/full_papers/jamal/jamal.pdf.

Van Jacobson. "Congestion Avoidance and Control". In: *Proceedings of ACM SIGCOMM*. Sept. 1988, pp. 314–329.

# References II


- Richard Kelly and Joseph Gasparakis. *Common Functionality in the 2.6 Linux Network Stack*. Tech. rep. 2010. URL: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/linux-2-6-network-stack-paper.pdf.
- Ravi Prasad, Manish Jain, and Constantinos Dovrolis. "Effects of Interrupt Coalescence on Network Measurements". In: *Passive and Active Network Measurement*. Vol. 3015. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 247–256.
- Miguel Rio et al. *DataTAG – A Map of the Networking Code in Linux Kernel 2.4.20*. Tech. rep. URL: http://datatag.web.cern.ch/datatag/papers/tr-datatag-2004-1.pdf.
- Rami Rosen. *Linux Kernel Networking – advanced topics*. 2009. URL: http://www.haifux.org/lectures/217/netLec5.pdf.

# References III

Klaus Wehrle et al. *The Linux Networking Architecture: Design and Implementation of Network Protocols in the Linux Kernel*. Prentice Hall, 2004. URL: http://www.6test.edu.cn/~lujx/linux_networking/0131777203_ch25lev1sec3.html.