



UiO • **Faculty of Mathematics and Natural Sciences**
University of Oslo



A TCP person's ns-3 experience report

Michael Welzl



WNS3 2023
28. June 2023

Please bear with me

- I'm a TCP insider ... but an ns-3 outsider ☹️
 - As announced, this is a newbie's experience report
 - Please don't get mad, as I will criticize your tool....
 - Instead, take it as: “aha, to make it idiot-proof, even for fools like him, we should...”
- ... and feel free to ask me TCP questions instead 😊

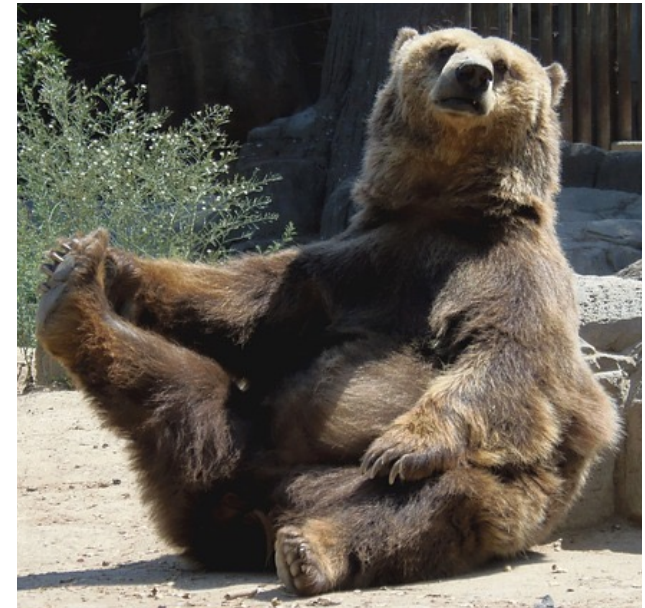


Image by daniel alonso from Pixabay

Outline

1. There are some problems with ns-3 and TCP...
 - Hello world

2. ... but when it works, it's impressive!
 - An example: something cool that we could do with ns-3 + TCP

3. And now what?
 - The role of ns-3 in TCP (and related) research

Part 1: Hello world

“Hello world” in TCP land

No, this is not it:

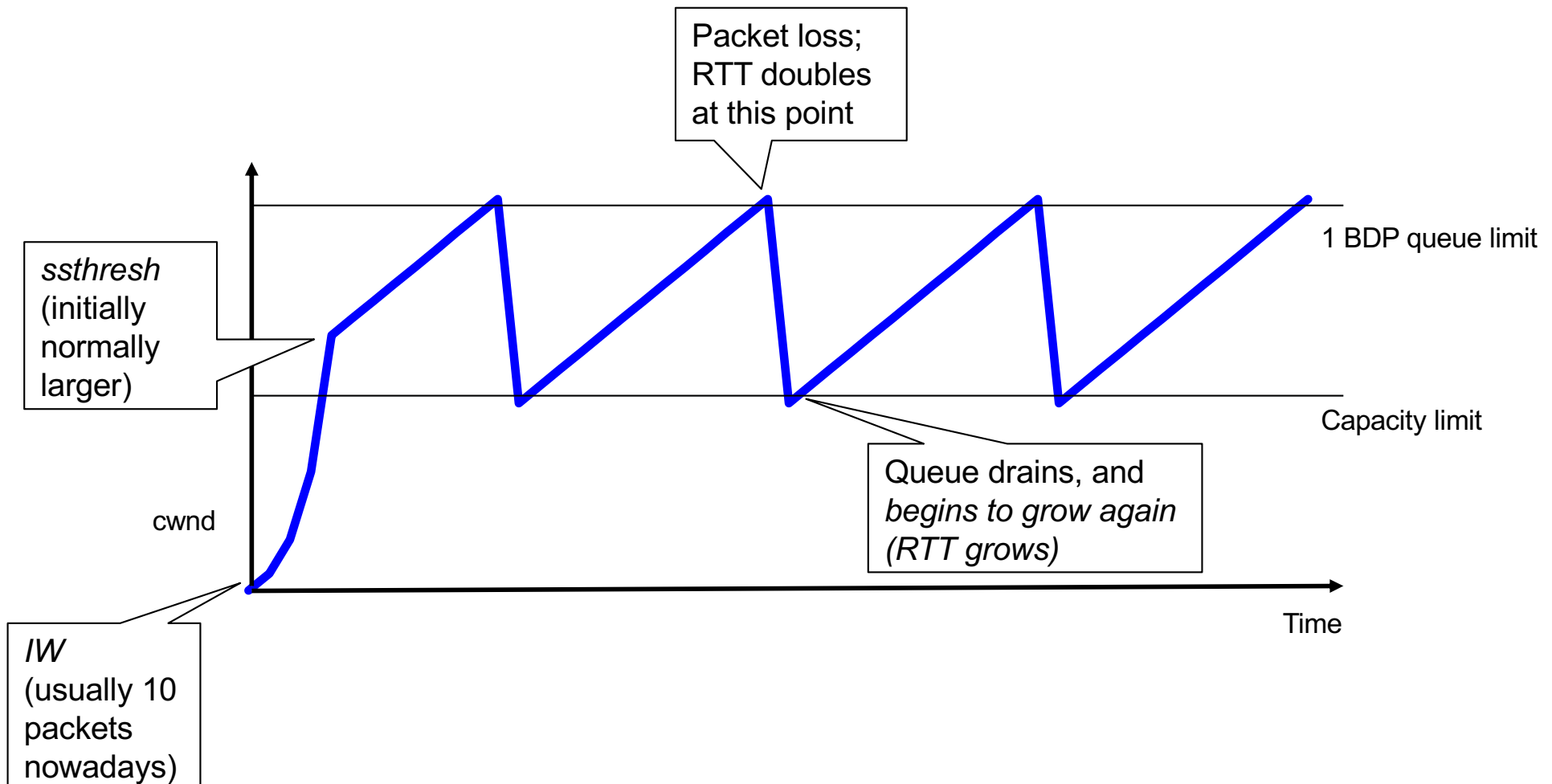
```
./ns3 run hello-simulator
```

Hello Simulator

The point of “hello world” is to confirm that the necessary things are in place and working, such that we can take the next steps (in coding)

- I can write to the console; probably, I can now write various programs with input and output

Instead...



A simple cwnd plot with TCP Reno

Hello world in ns-2

```
set ns [new Simulator]
set cwndf [open cwnd.tr w]

set n0 [$ns node]
set n1 [$ns node]

$ns duplex-link $n0 $n1 5Mb 30ms DropTail
$ns queue-limit $n0 $n1 25

set tcp0 [new Agent/TCP/Linux]
$tcp0 set window_ 1000
$tcp0 set packetSize_ 1460
$ns attach-agent $n0 $tcp0

set sink0 [new Agent/TCP/Sink/Sack1]
$ns attach-agent $n1 $sink0

$ns connect $tcp0 $sink0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
```

```
proc plotcwnd {tcpSource outfile} {
    global ns

    set now [$ns now]
    set cwnd [$tcpSource set cwnd_]
    puts $outfile "$now $cwnd"
    $ns at [expr $now+0.01] "plotcwnd $tcpSource $outfile"
}

#Define a 'finish' procedure
proc finish {} {
    global cwndf
    close $cwndf
    exit 0
}

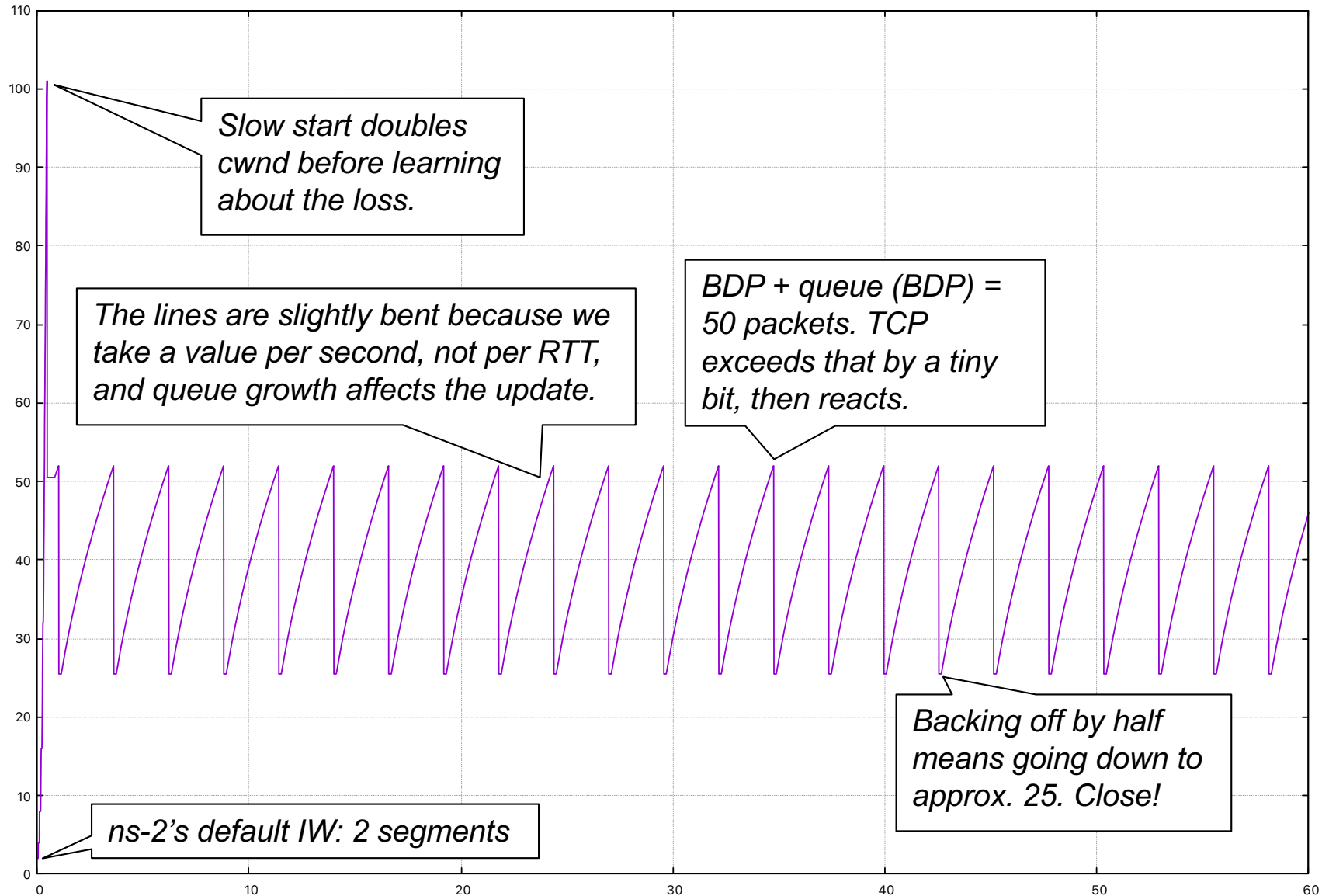
$ns at 0.0 "plotcwnd $tcp0 $cwndf"
$ns at 0.0 "$ftp0 start"
$ns at 60.0 "finish"

$ns run
```

BDP: 5 Mbit * 60 ms, in packets:
 $5\ 000\ 000 * 0.06 / 8 / 1500 = 25$

Gnuplot output

set grid; set nokey; plot "cwnd.tr" w l



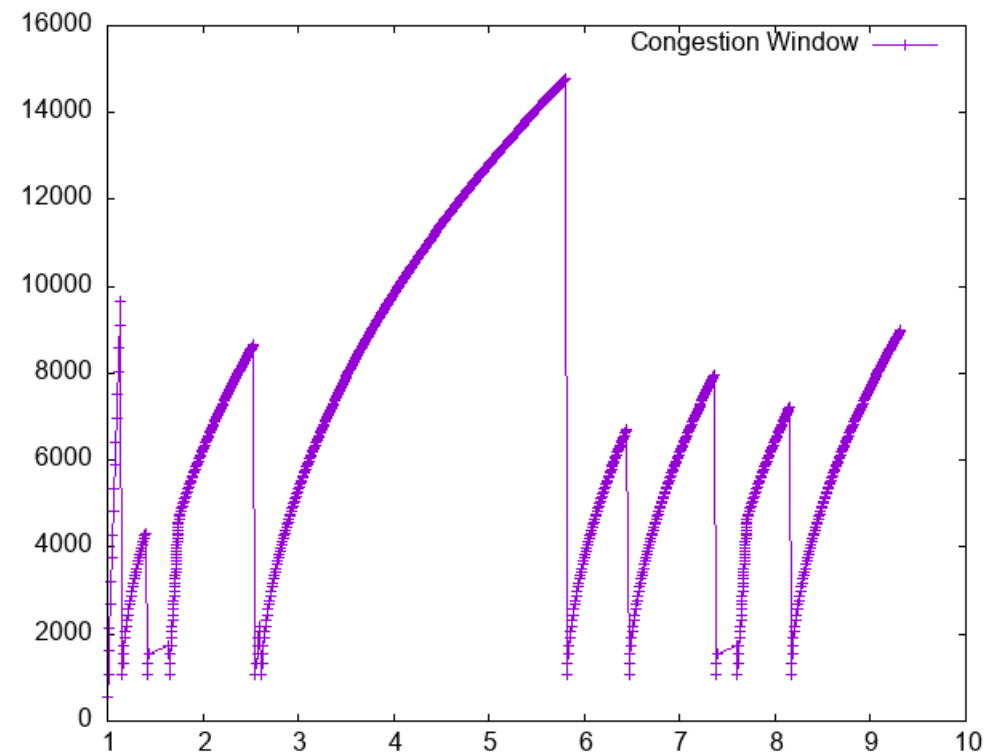
A TCP "hello world" in ns-3

- ... begins with: looking for TCP related code in the tutorial

- **fifth.cc**

- There's even a nice cwnd plot right there!

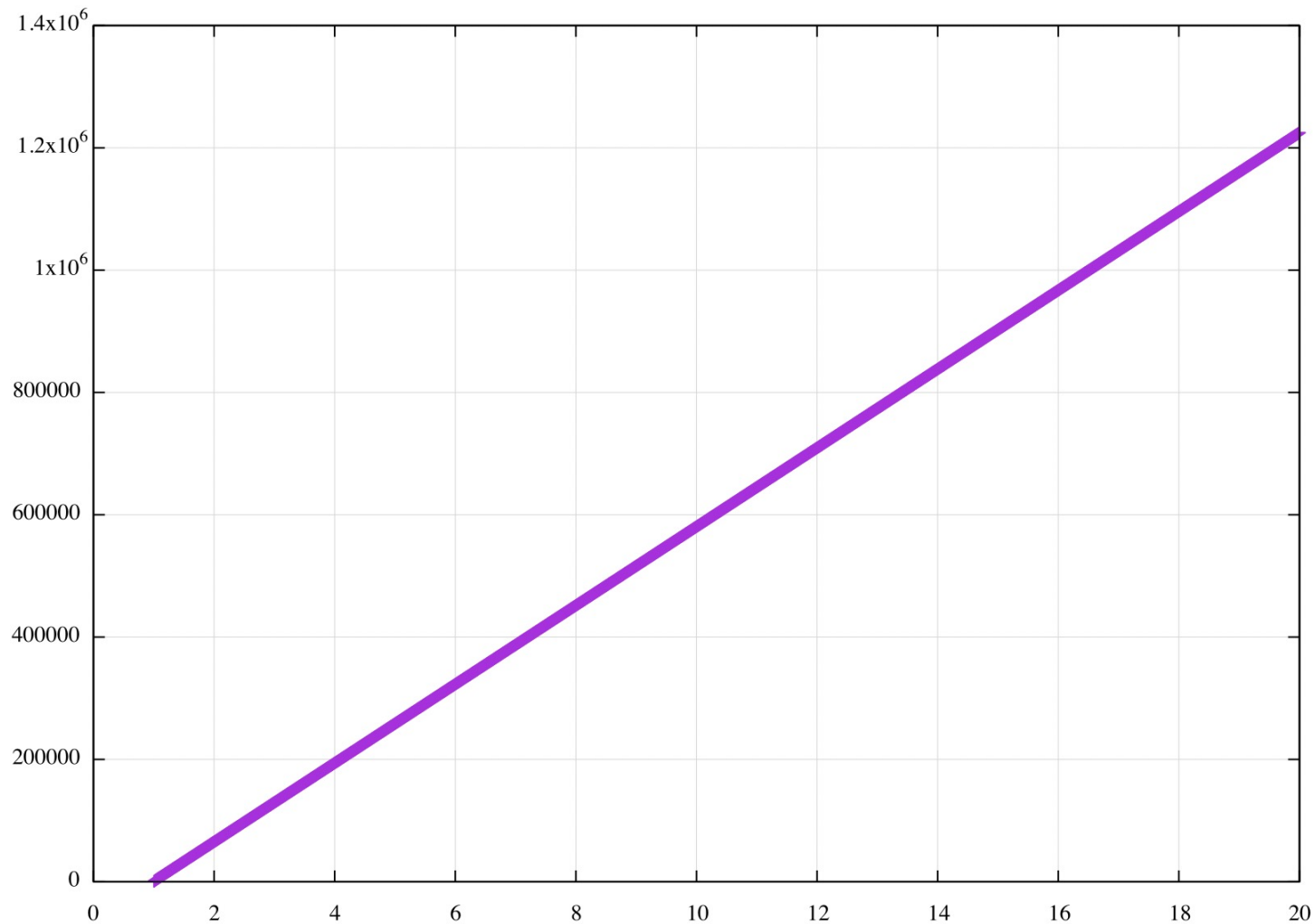
... Not quite like the previous page: this uses an error model



*Side note about the tutorial: with fifth.cc and tutorial-app.cc, I get a linker error.
(trouble finding the .o file)*

Removing the error model...

```
em->SetAttribute("ErrorRate", DoubleValue(0.0));
```



This must be: a huge capacity or queue

- How to fix this?
 - Tutorial chapter "Building Topologies"

```
PointToPointHelper pointToPoint;  
pointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));  
pointToPoint.SetChannelAttribute("Delay", StringValue("2ms"));
```
- No effect. So, it must be the queue...
 - Subsection "Queues in ns-3"

```
p2p.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("50p"));
```
- No effect either.

~ 6 years ago, Google helped me...

<https://stackoverflow.com/questions/38951115/tcp-congestion-window-graph-ns-3>

I am trying to plot the graph of the congestion window size of a TCP sender.

I am working with the following example and using ns-3's development branch.

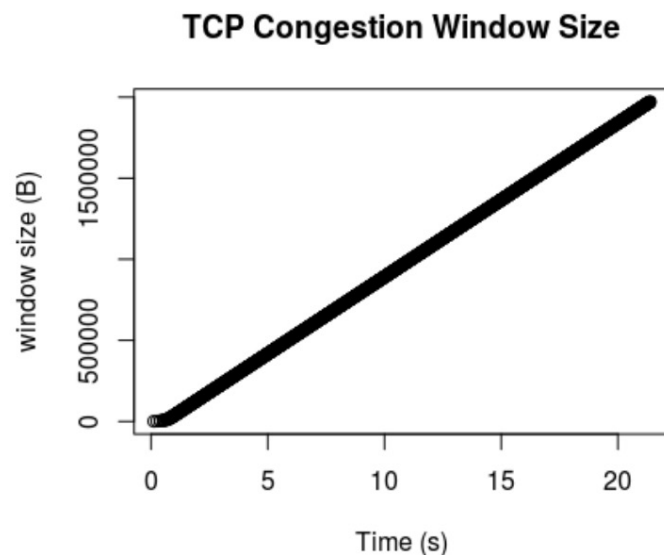
<http://intronetworks.cs.luc.edu/current/html/ns3.html>

This example implements a simple PointToPoint topology between 2 nodes A and C passing as in the following diagram.

A ----- B ----- C

10Mbps ----- 800Kbps

When I run this example I get the following graph.



- For me, this was ns-3.27

This tutorial shows cwnd plots... so it was once easier!

- We two were not the only ones - see:
<https://groups.google.com/g/ns-3-users/c/M7YqWnvrFJg>

"You have, somewhere, an infinite (or very large) buffer. No drops, and the cwnd grows like a young whale in the first 6 months of life."

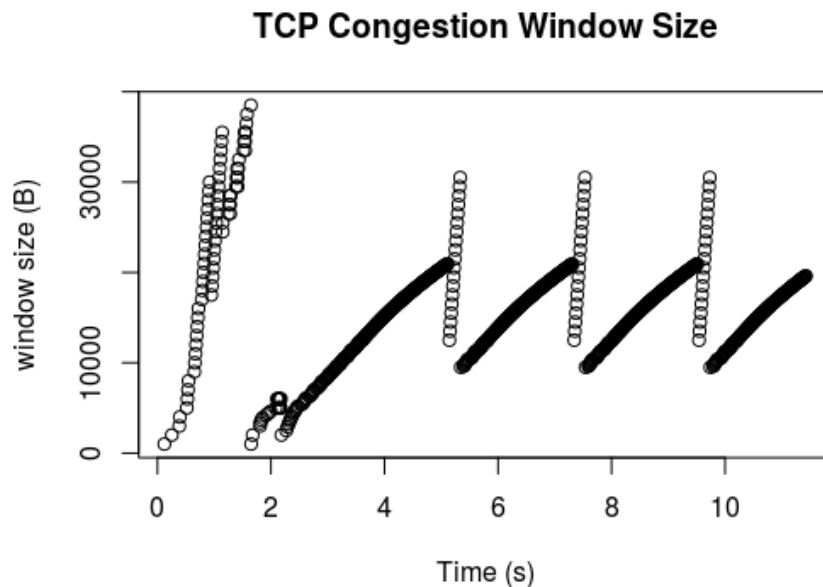
... and this solution worked for me, then:

EDIT:

Upon further investigation this might be due to the TrafficControl (see 1.) put in place by the Ipv4AddressHelper. When commenting the following lines of code in src/internet/helper/ipv4-address-helper.cc

```
Ptr<TrafficControlLayer> tc = node->GetObject<TrafficControlLayer> ();  
if (tc && DynamicCast<LoopbackNetDevice> (device) == 0 && tc->GetRootQueueDisc  
{  
    NS_LOG_LOGIC ("Installing default traffic control configuration");  
    TrafficControlHelper tcHelper = TrafficControlHelper::Default ();  
    tcHelper.Install (device);  
}
```

I get this new graph



- IPv4AddressHelper?
- Not exactly where I'd expect "tc" related code...
- But now, with ns-3.38, this no longer seems to work !
 - IPv4AddressHelper code block is still there, and has evolved
 - Removing it: no effect

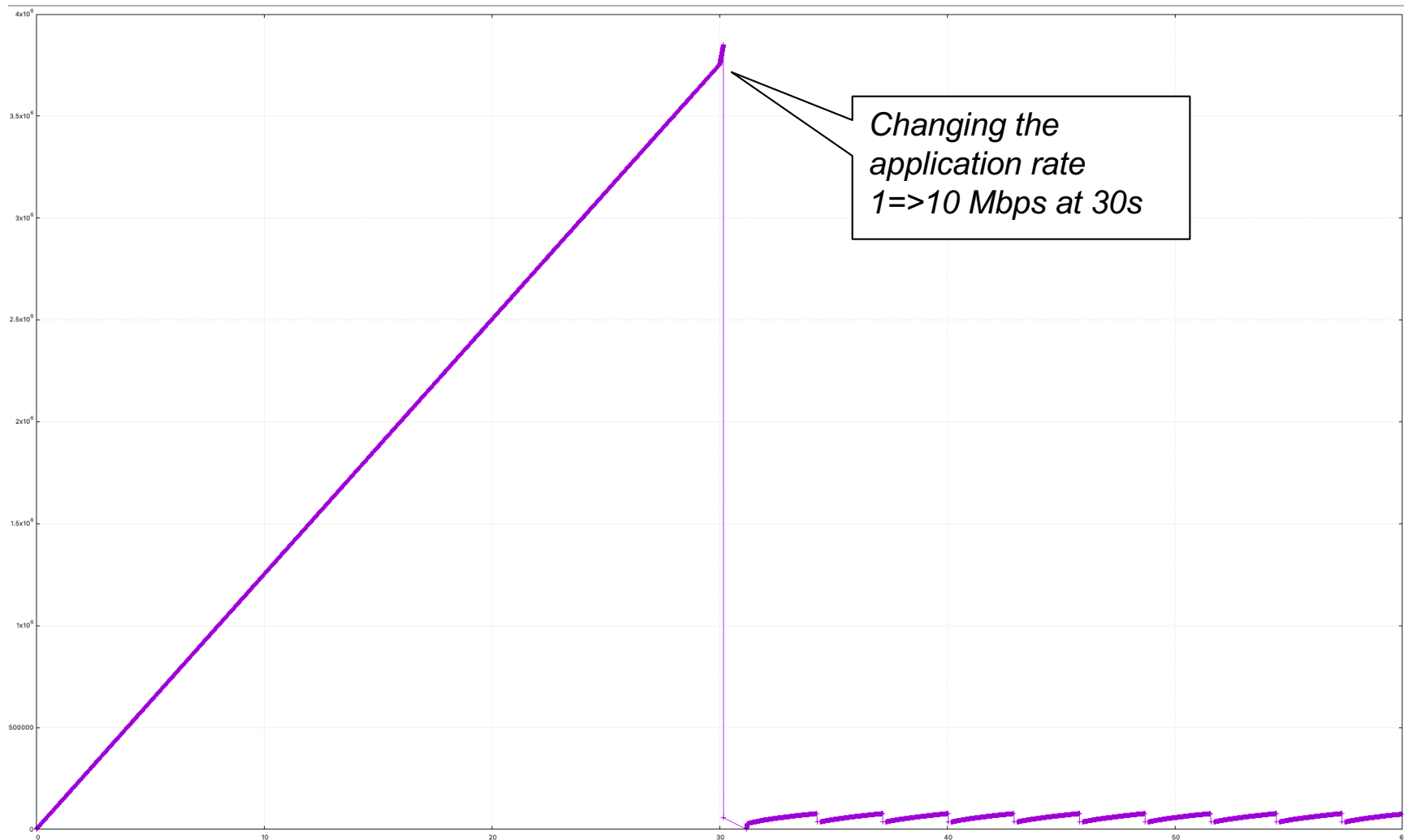
Proceeding with ns-3.38...

- The culprit was this line, no need for changes in IPv4AddressHelper:

```
app->Setup(ns3TcpSocket, sinkAddress, 1040, 10000,  
DataRate("1Mbps"));
```

- 1 Mbps is less than my bottleneck's capacity
 - so, the transfer was application-limited.
- But, is the ever-growing cwnd in the app-limited case "real" (in ns-3)??
 - If so, that's not good
 - It would mean: an application, after sending too little for many seconds, could immediately "jump" to a large cwnd value
 - Else, it's still a little confusing, as an output...

No, this cwnd growth is not “real”. *Phew.*

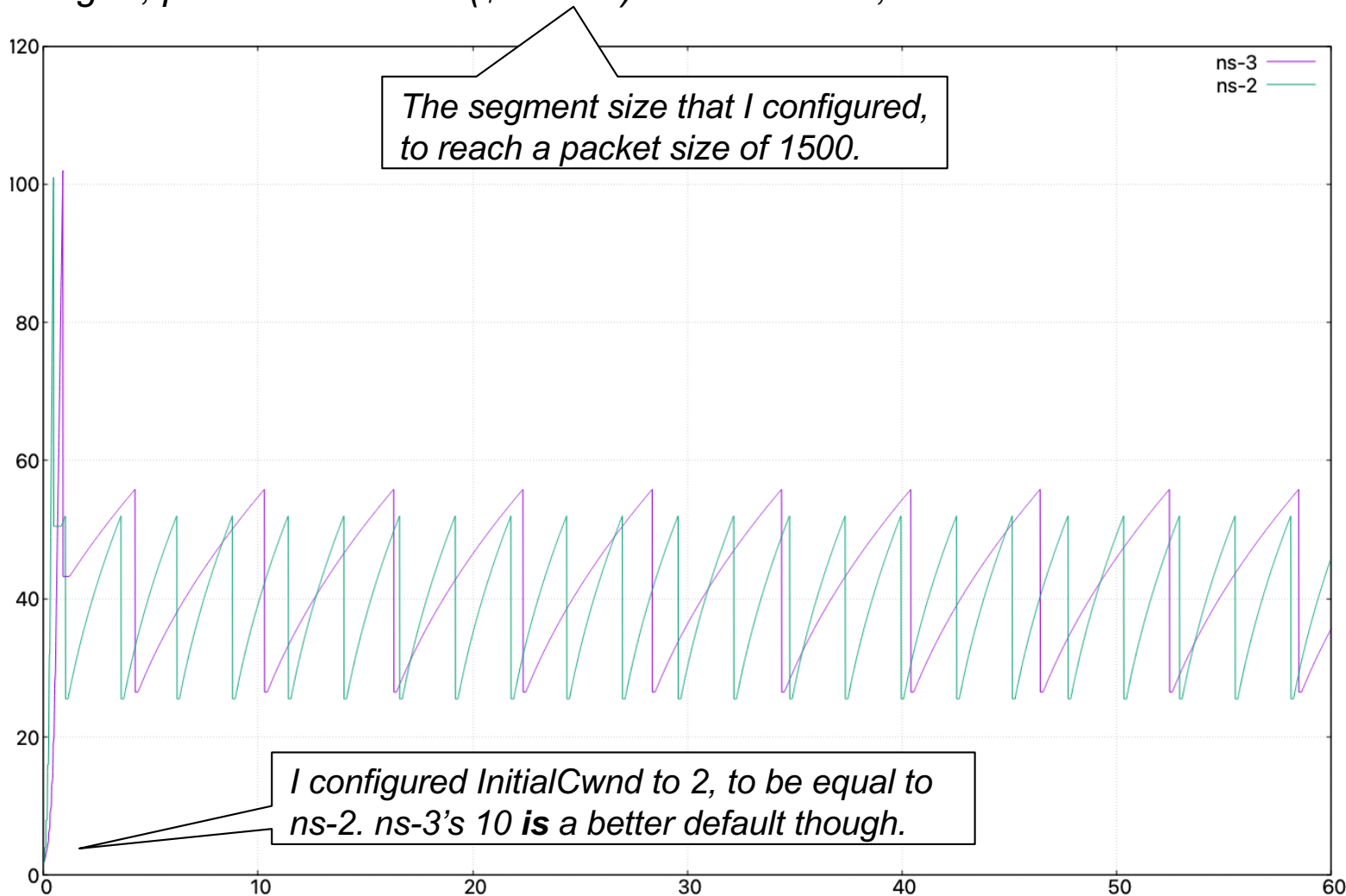


About TCP's default segment size

- Next trouble: Wireshark shows me a packet size of 590, irrespective of how I set the MTU on my pointToPoint interface
 - A bit of digging yields:
https://www.nsnam.org/docs/doxygen/d3/dea/classns3_1_1_tcp_socket.html
- **SegmentSize**: TCP maximum segment size in bytes (may be adjusted based on MTU discovery)
 - Set with class: **ns3::UIntegerValue**
 - Underlying type: uint32_t 0:4294967295
 - Initial value: 536
 - Flags: `construct` `write` `read`
- RFC 9293 (TCP standard): “TCP endpoints *MUST* implement both sending and receiving the MSS Option (MUST-14).”
 - ...but I see no MSS option in the SYN packet.
 - “The MSS value to be sent in an MSS Option should be equal to the effective MTU minus the fixed IP and TCP headers.”
 - “RFC 1191 discusses this implication of many older TCP implementations setting the TCP MSS to 536 (corresponding to the IPv4 576 byte default MTU) for non-local destinations, rather than deriving it from the MTUs of connected interfaces as recommended.”

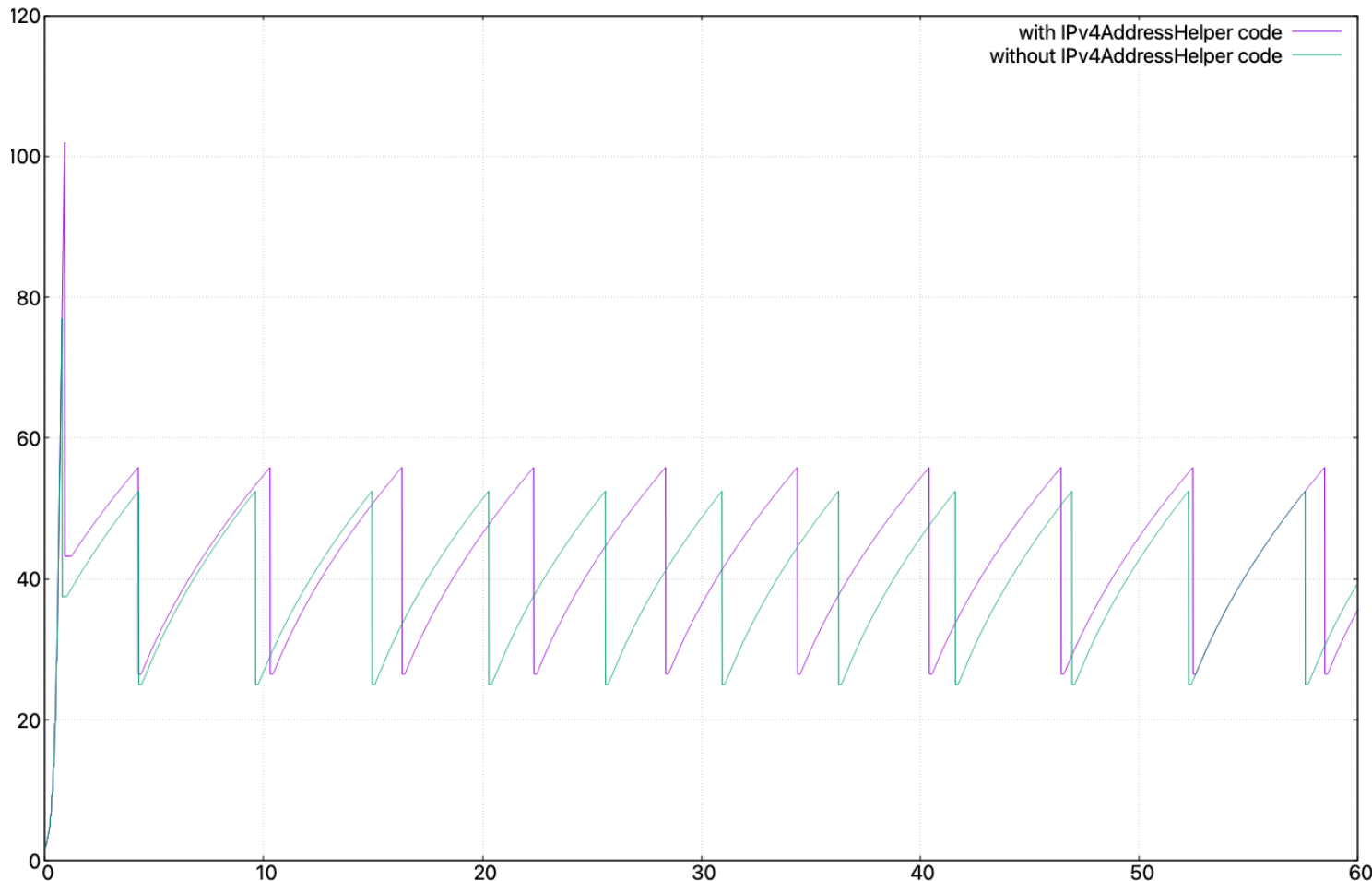
We're slowly getting there...

```
set grid; plot "ns-3.txt" u 1:($2/1446) w l title "ns-3", "ns-2.txt" u 1:2 w l title "ns-2"
```



- Half the sawteeth: this must be delayed ACKs
- But, also, and more concerning: ns-3 goes higher...

Let's try commenting the tc-related IPv4AddressHelper code block again...



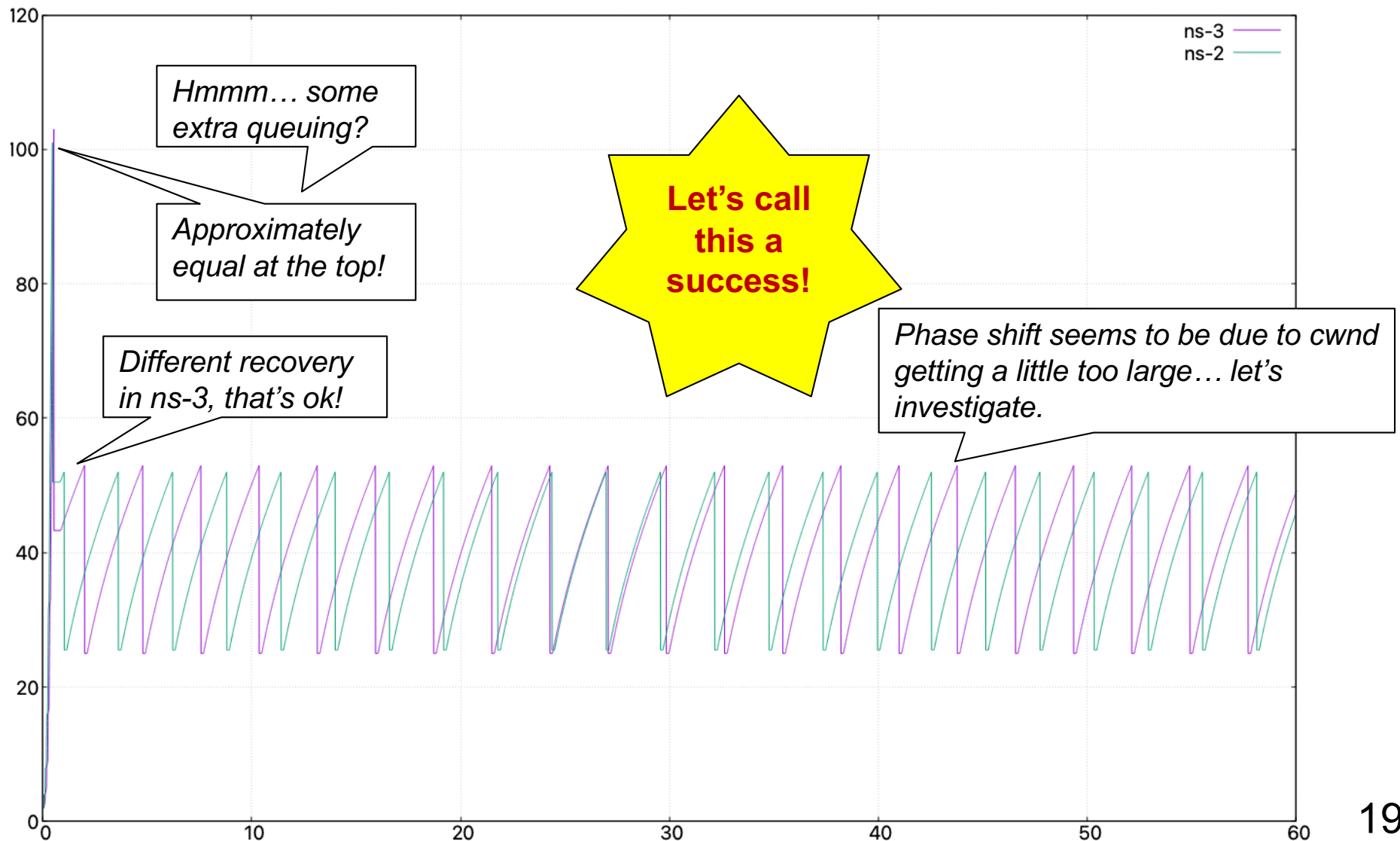
The code block comment says: "Install the default traffic control configuration if (..) "

*My guess:
=> This appears to create a queue that should be empty, but isn't...*

A much better result without it!

Now, without Delayed ACKs

```
Config::SetDefault("ns3::TcpSocket::DelAckCount", UIntegerValue(1));
```

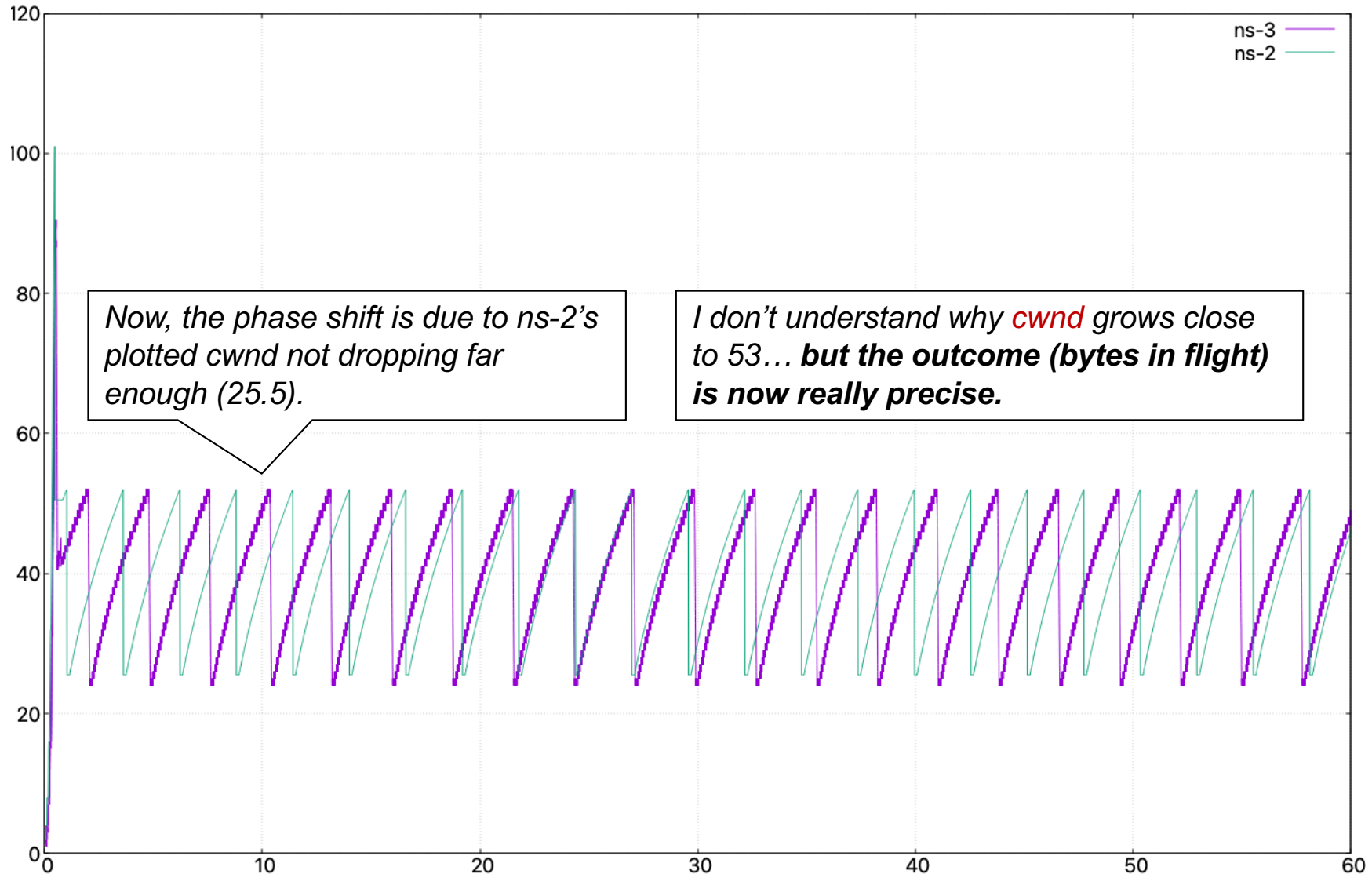


More details on the phase shift

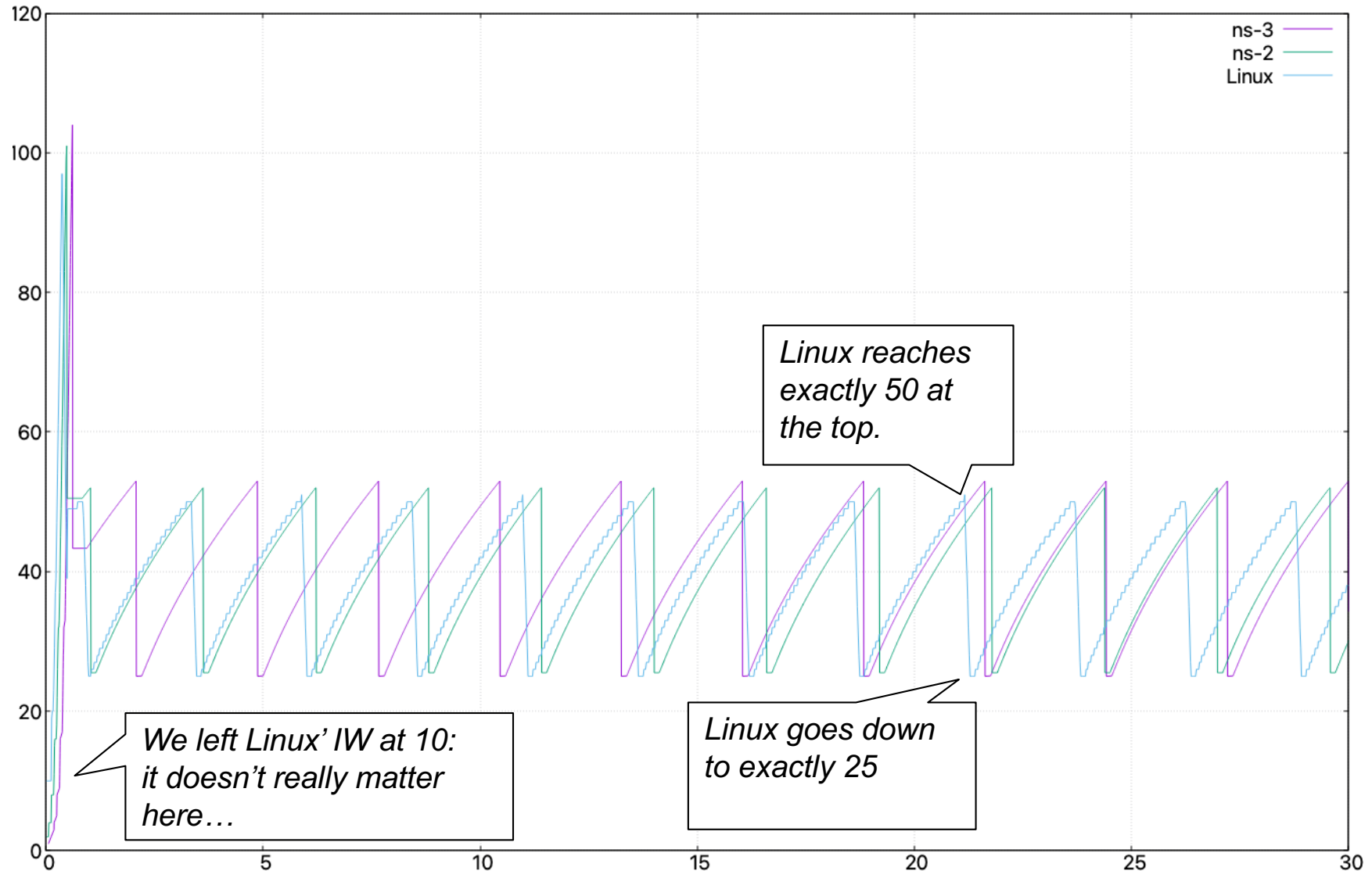
- ns-2's cwnd values at backoff
 - 51.8679, 51.945, 51.9834, 25.5
 - 51 was too much. Then, it took an RTT to learn about it... so we kept going... hence we got close to 52.
 - Either ns-2 rounds down before halving, or it (incorrectly?) uses [FlightSize](#)
- ns-3's cwnd values at backoff
 - 76538, 76565, 76592, 36150
 - Divided by 1446 (segment size): 52.9308, 52.9495, 52.9682, 25
 - Getting close to 52, I can understand, but close to 53?
 - Getting from this to 25 is [100% correct](#), and interesting...
 - In [tcp-congestion-ops.cc](#), [ssthresh](#) is based on [bytesInFlight](#) - Correct!
 - We can trace this variable too; it reaches exactly $73746 / 1446 = 50$!

*There should probably be
51-1=50 packets in flight*

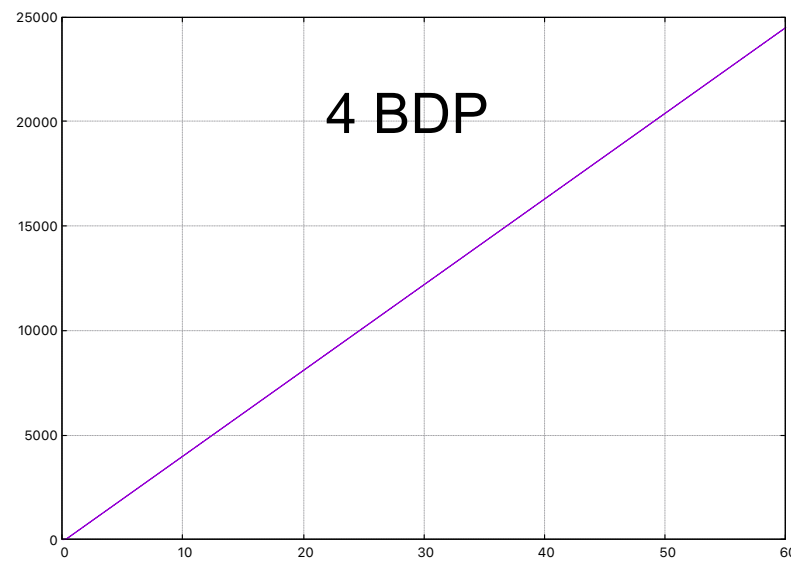
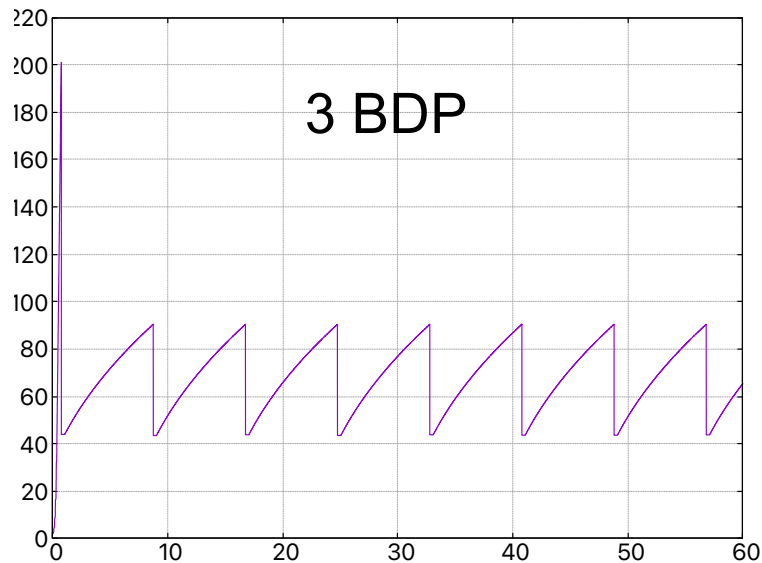
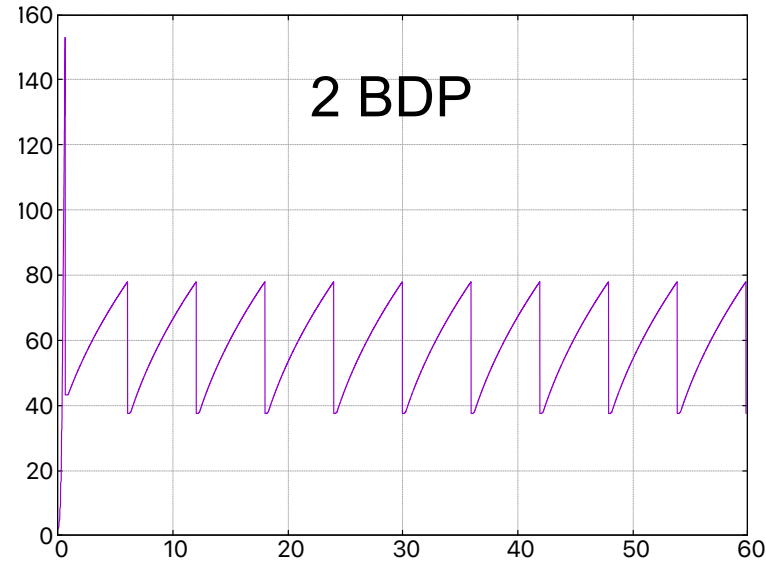
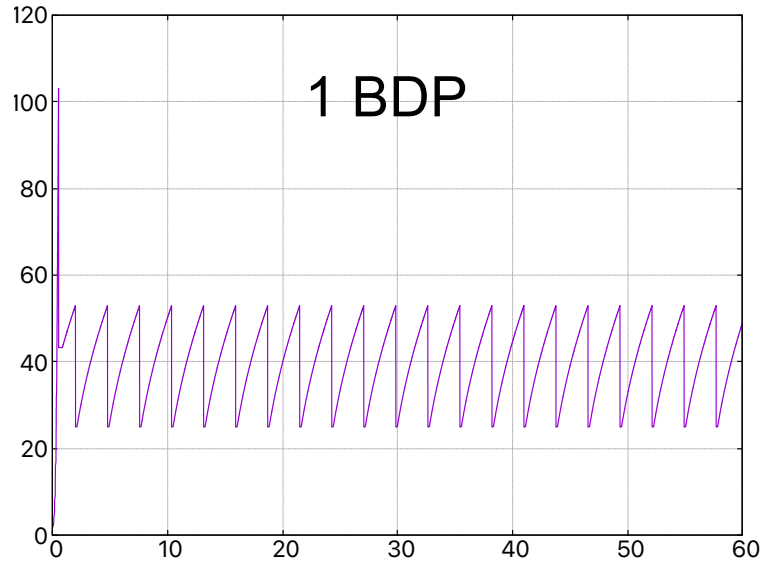
ns-3's BytesInFlight vs. ns-2's cwnd



REALLY precise!



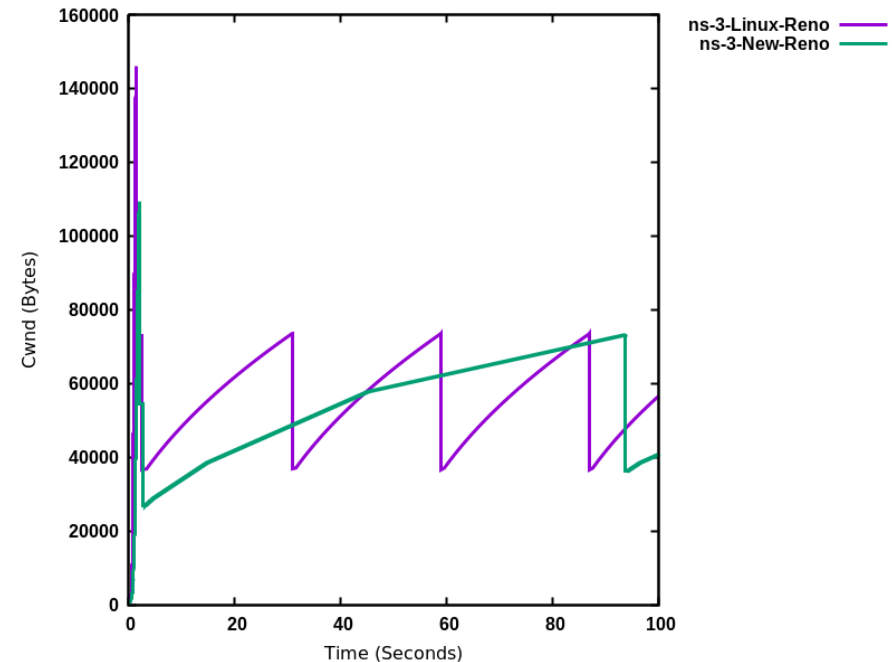
... as long as we keep the queue small.



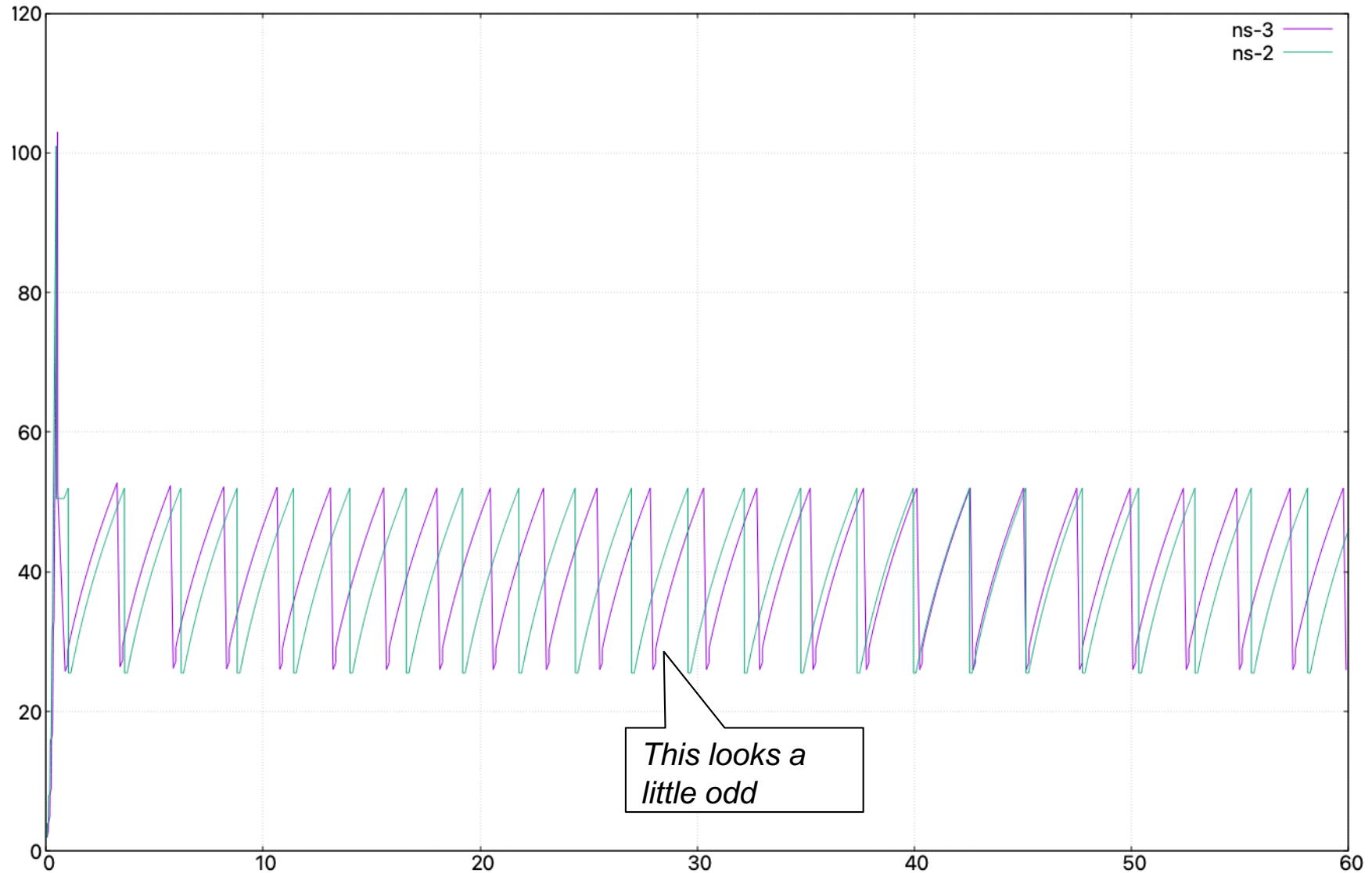
Let's try "TCP Linux Reno"

- About increase behavior, the ns-3 docs say:
<https://www.nsnam.org/docs/models/html/tcp.html>
 - "Linux follows Appropriate Byte Counting while ns-3 NewReno does not."
 - "cwnd is incremented by a full-sized segment (SMSS). In contrast, in ns-3 NewReno, cwnd is increased by $(1/cwnd)$ with a rounding off due to type casting into int."
 - ns-2 seems to have the same increase: the lines were parallel. That's ok.

- But, this plot in the documentation is strange... 3 sawteeth?
That's too slow!
 - ... and it's not what we just saw.



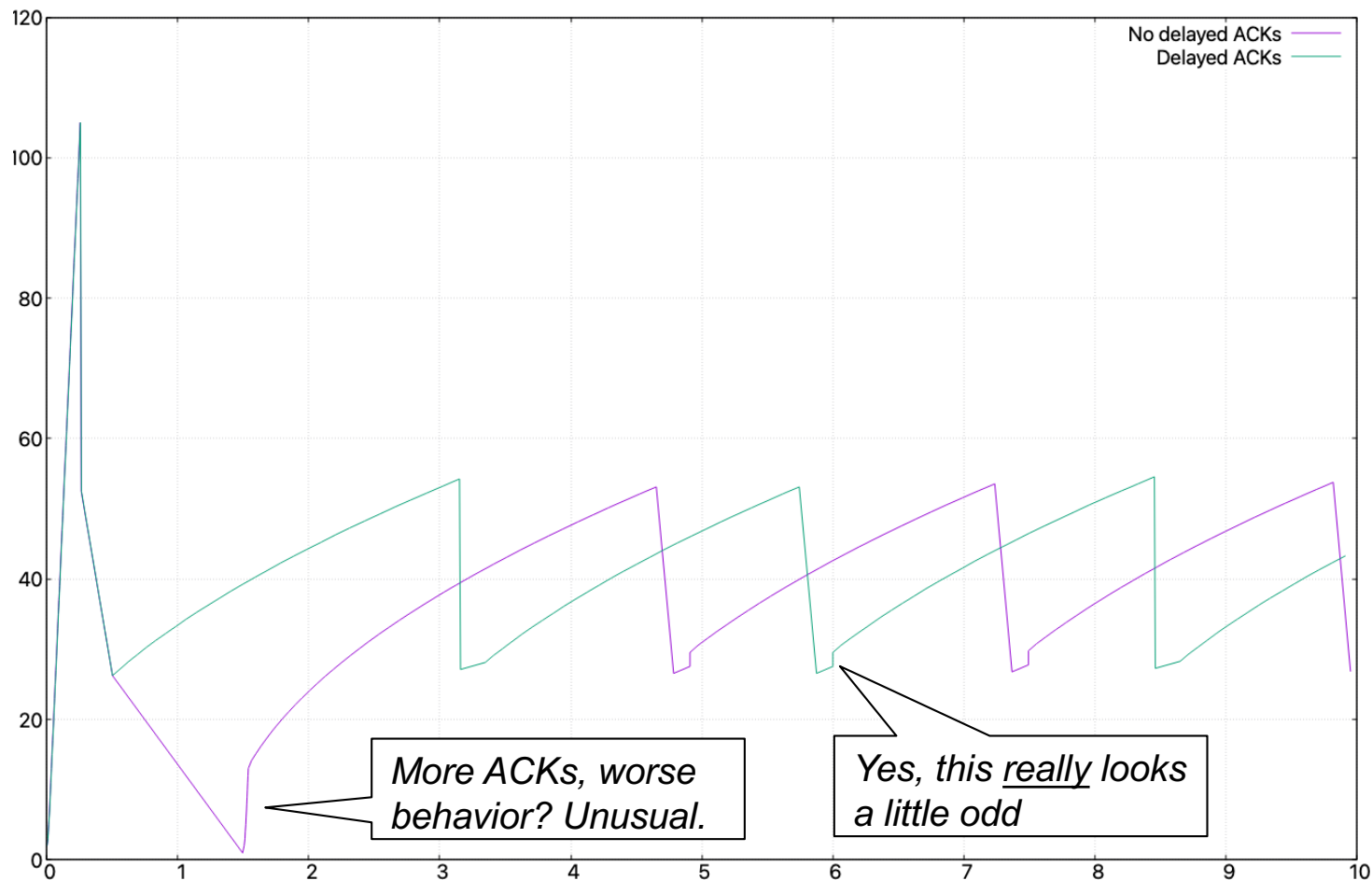
Testing TCP Linux Reno: pretty similar



....but I discovered something weird

...By accident, in loss recovery

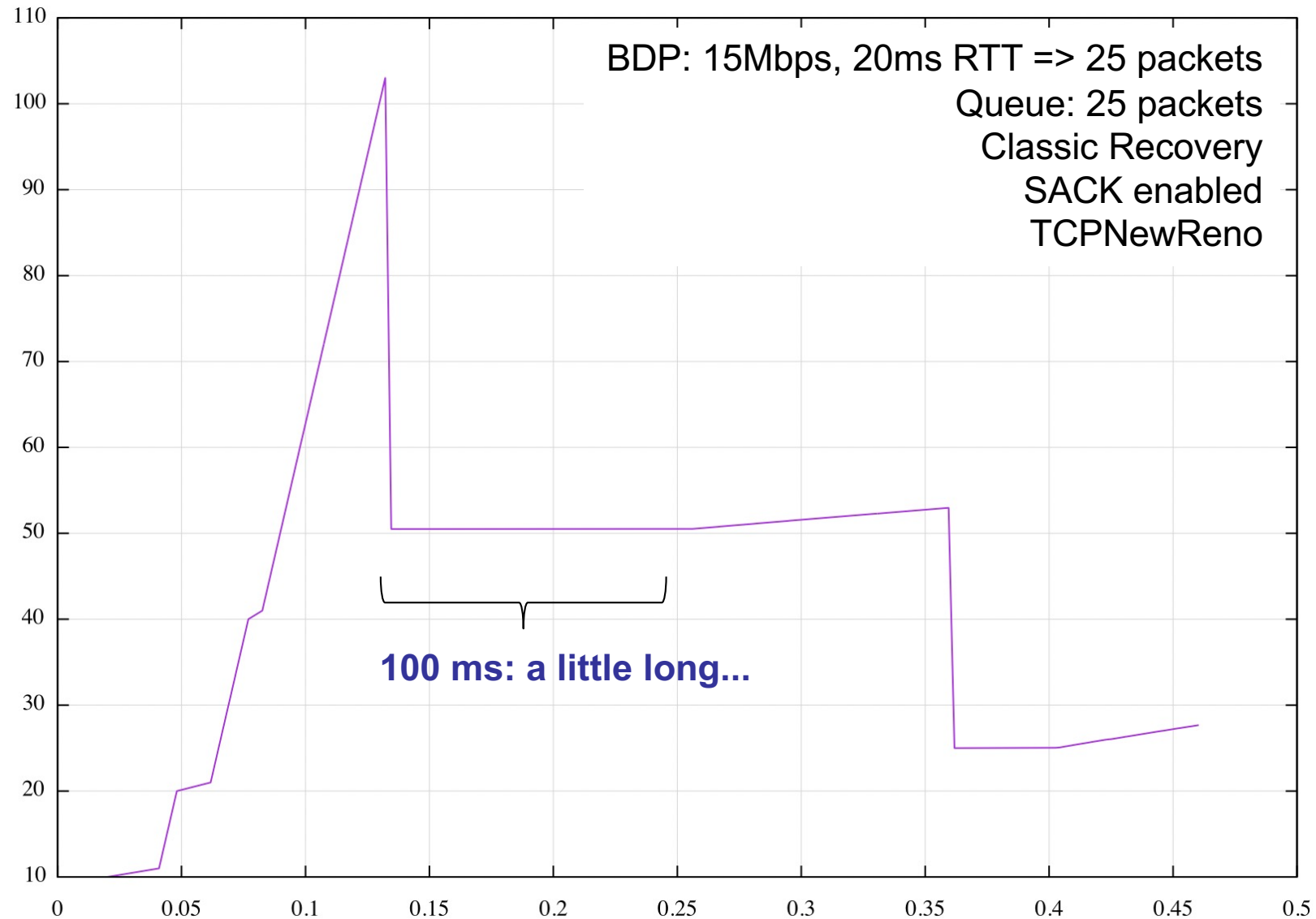
- This is with 5 Mbps, 2ms (i.e. 4ms RTT), 50 packet queue (3 BDPs)



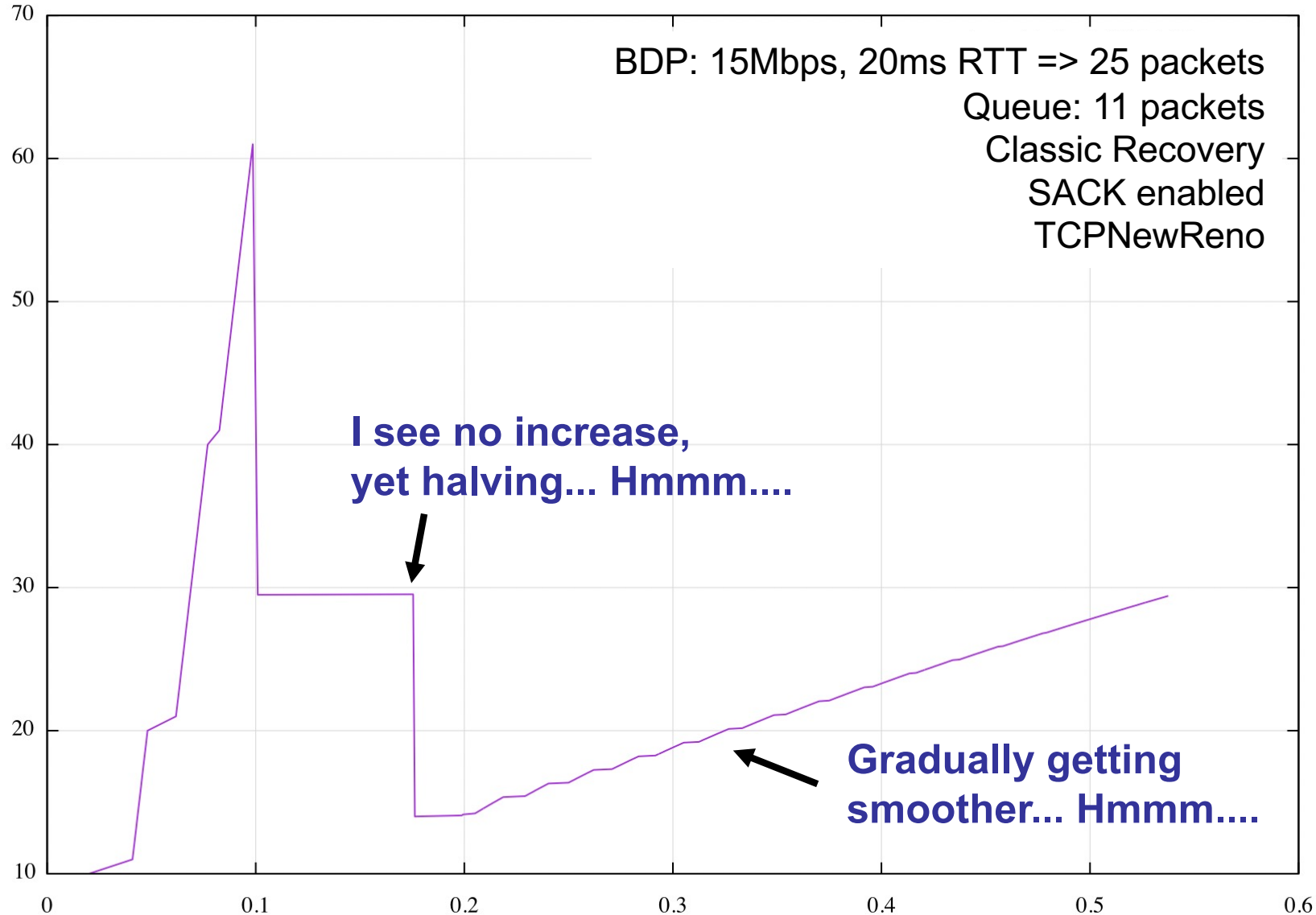
Q: What about 4 BDPs?

A: The same ever-growing cwnd as with "NewReno"

Loss recovery: perhaps okay with 2 nodes



Loss recovery: perhaps okay with 2 nodes /2



**Next:
3 nodes**

**WHAT have
I done
wrong?**

**How can
this "ruin"
the cwnd?**

```
NodeContainer leftNodes;
leftNodes.Create(2);

PointToPointHelper leftPointToPoint;
leftPointToPoint.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("25p"));
leftPointToPoint.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
leftPointToPoint.SetChannelAttribute("Delay", StringValue("15ms"));

NetDeviceContainer leftDevices;
leftDevices = leftPointToPoint.Install(leftNodes);

NodeContainer rightNodes;
rightNodes.Add(leftNodes.Get(1));
rightNodes.Create(1);

PointToPointHelper rightPointToPoint;
rightPointToPoint.SetQueue("ns3::DropTailQueue", "MaxSize", StringValue("11p"));
rightPointToPoint.SetDeviceAttribute("DataRate", StringValue("5Mbps"));
rightPointToPoint.SetChannelAttribute("Delay", StringValue("15ms"));

NetDeviceContainer rightDevices;
rightDevices = rightPointToPoint.Install(rightNodes);

InternetStackHelper stack;
stack.Install(leftNodes.Get(0));
stack.Install(rightNodes);

Ipv4AddressHelper address;
address.SetBase("10.1.1.0", "255.255.255.252");
Ipv4InterfaceContainer leftInterfaces;
leftInterfaces = address.Assign(leftDevices);

address.SetBase("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer rightInterfaces;
rightInterfaces = address.Assign(rightDevices);
```

Again: BDP = 25 packets
Now: 5Mbps, 60ms RTT

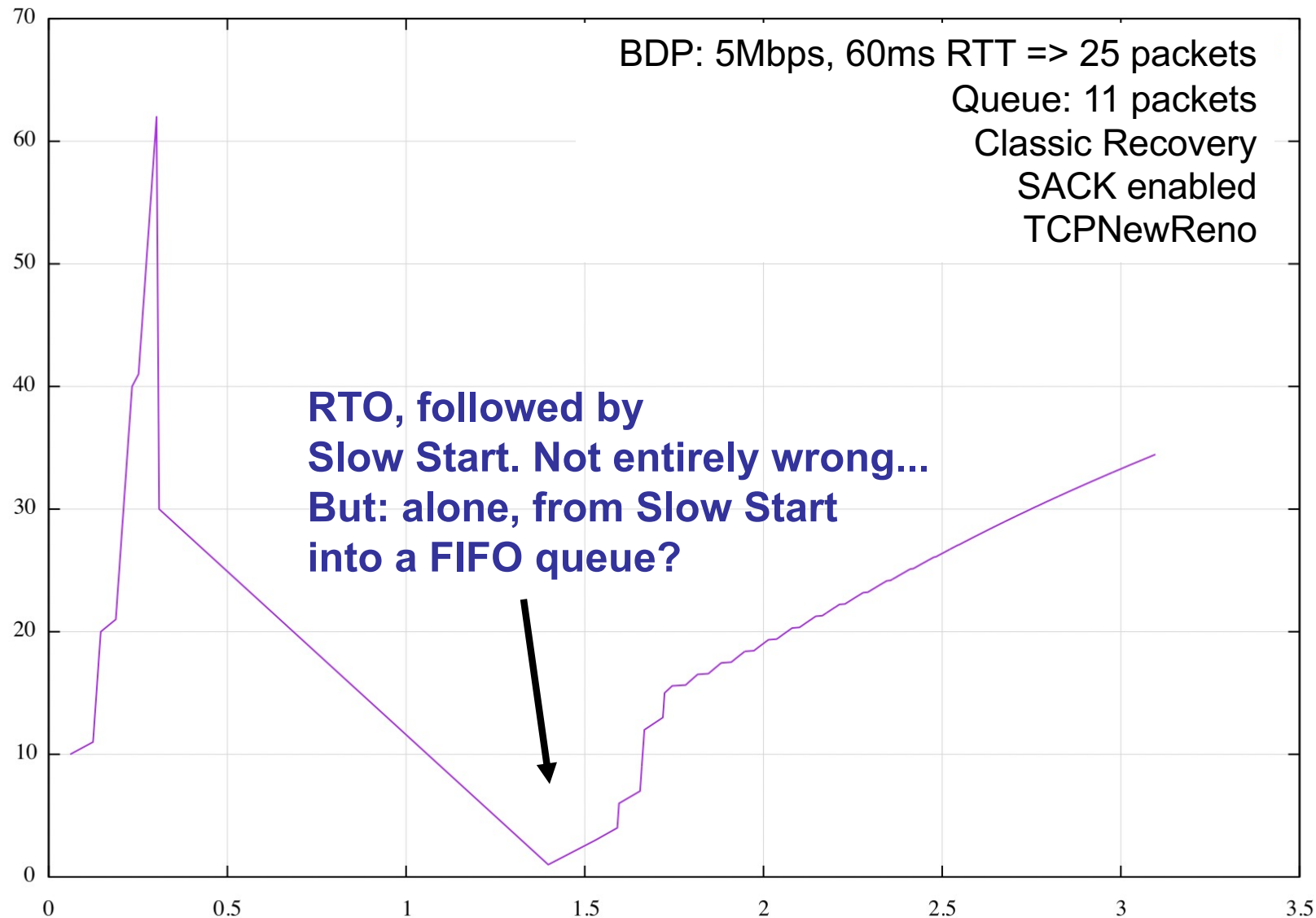
Beware of the following diagrams

- *Not suitable for children*
- *Not suitable for TCP researchers*
- *Not a pretty sight*

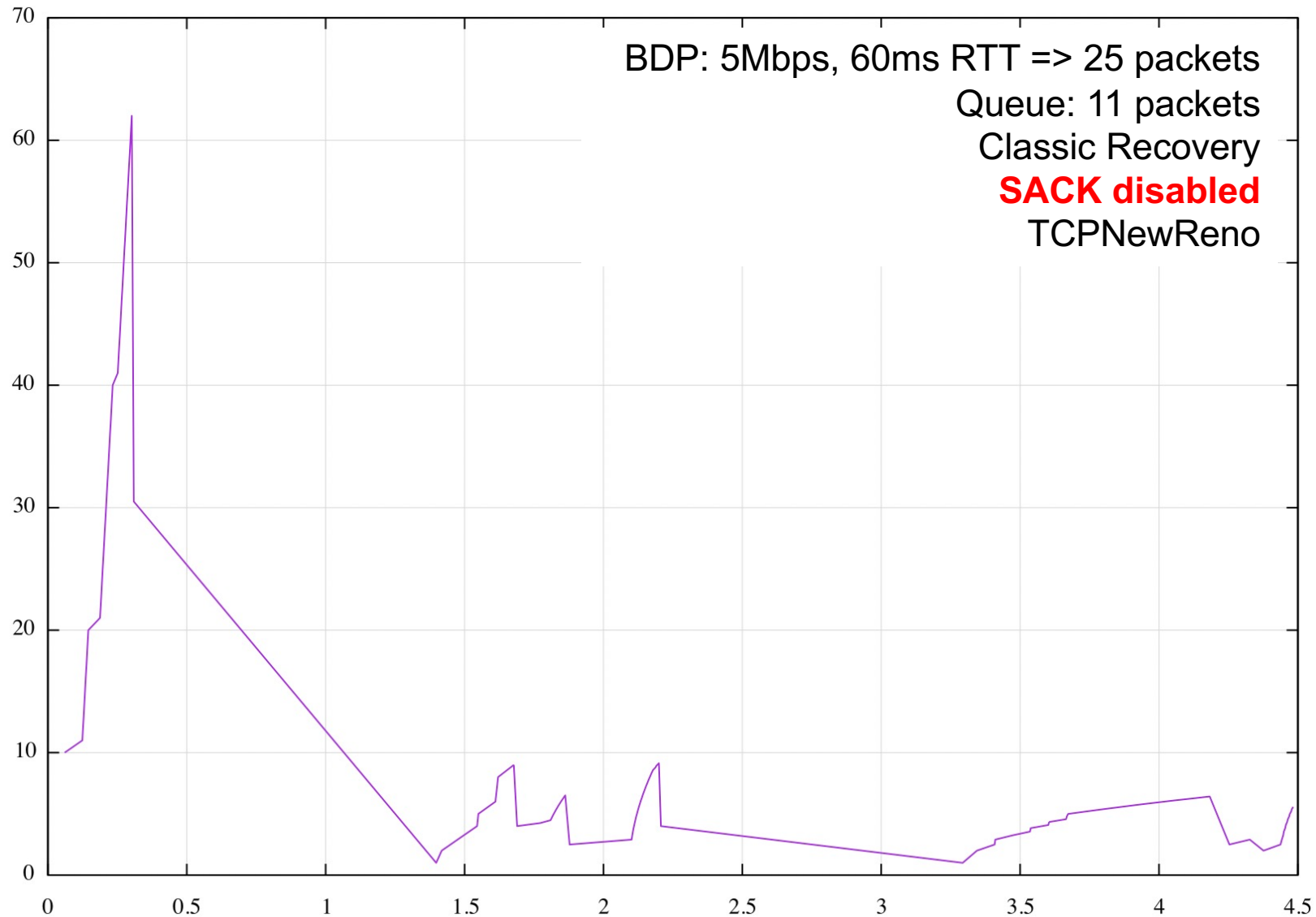
Look away, close your eyes, ...

Common parameters: IW10, no delayed ACKs

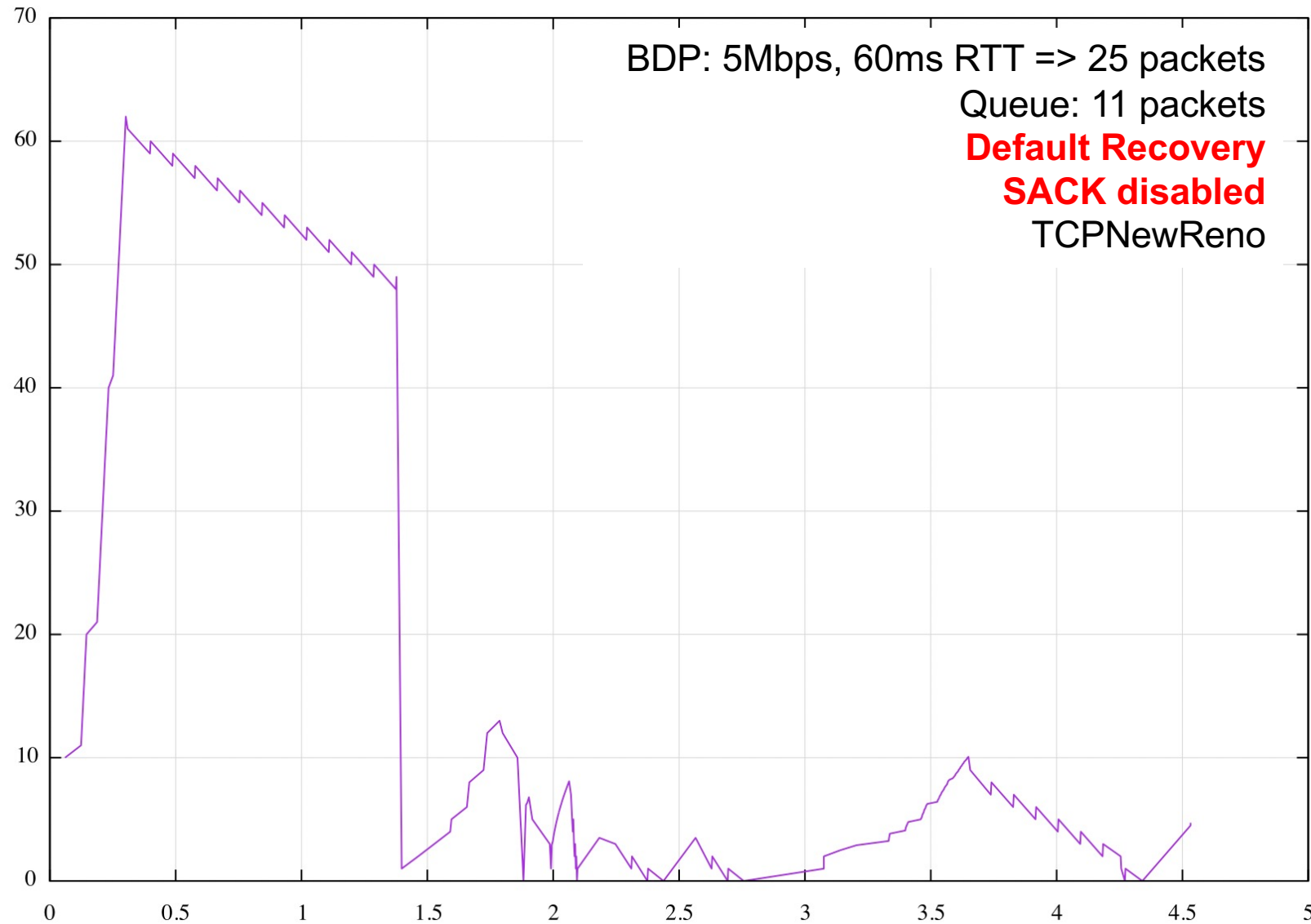
Loss recovery, 3 nodes



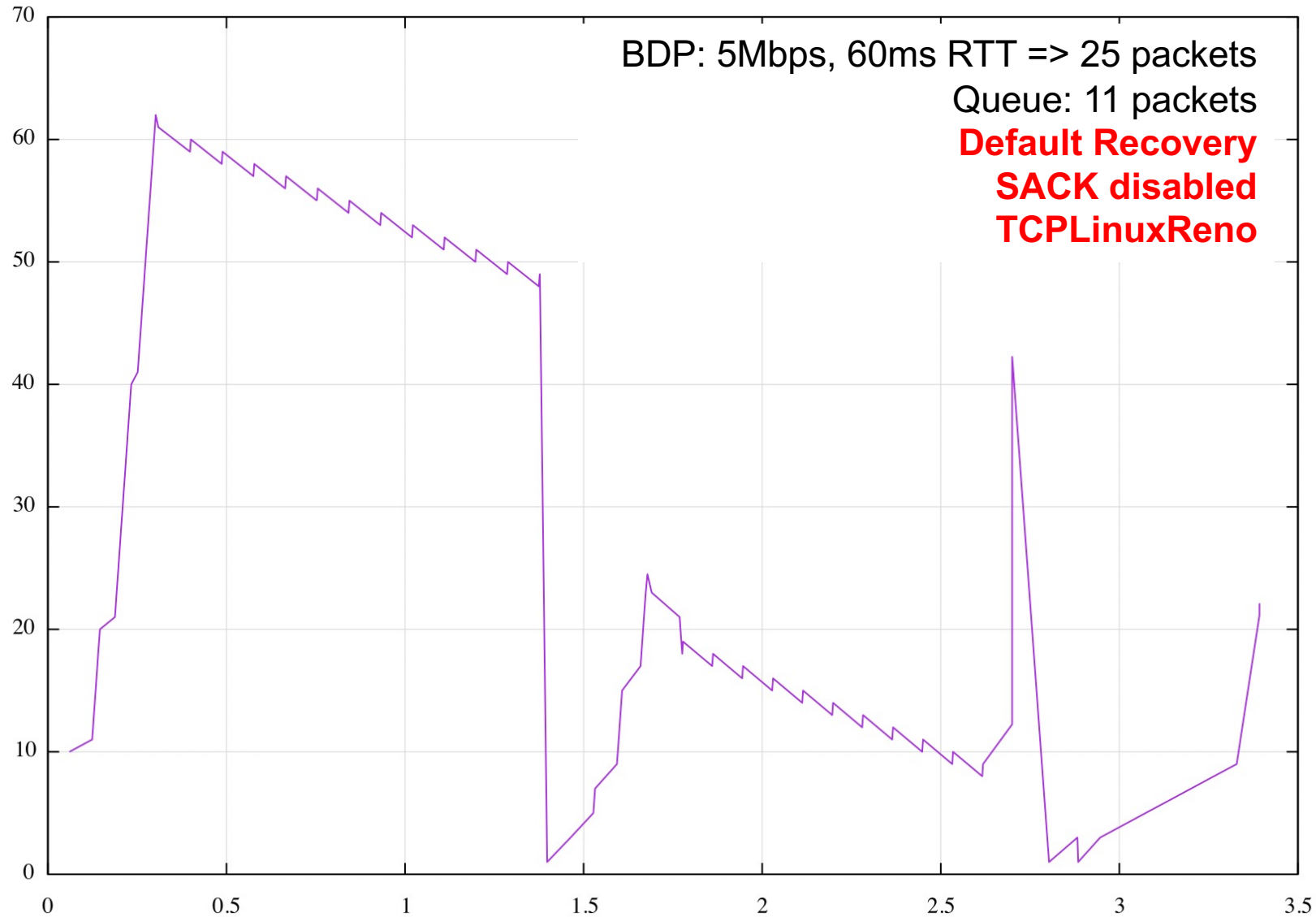
Loss recovery, 3 nodes /2



Loss recovery, 3 nodes /3



Loss recovery, 3 nodes /4

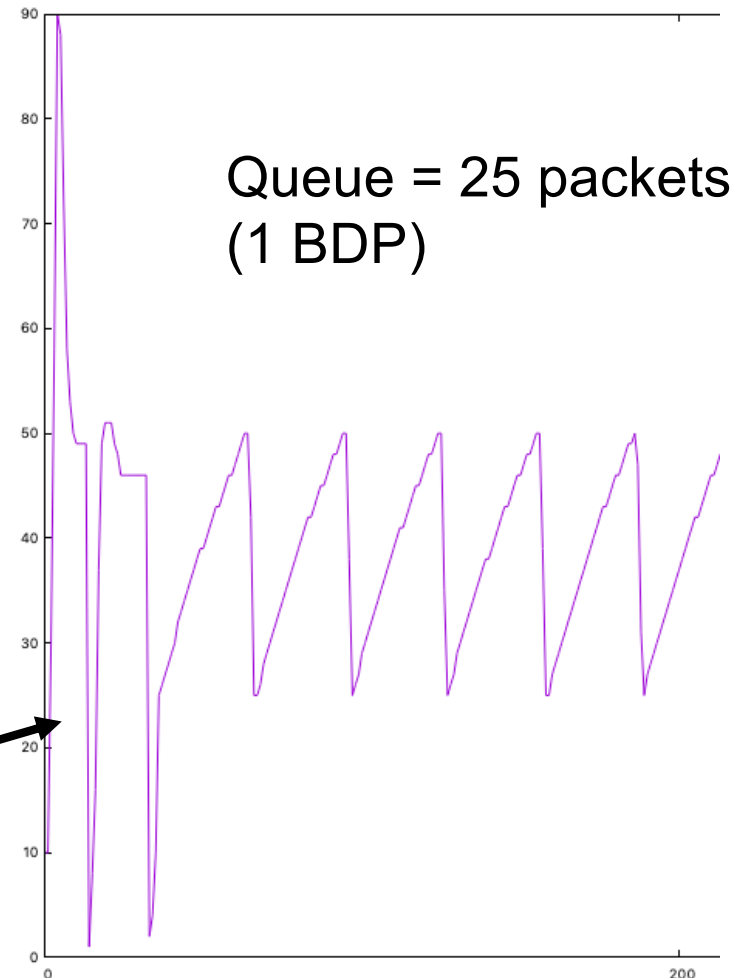


Let me ease the pain a little...

It is possible for heavy queuing to cause quite strange effects

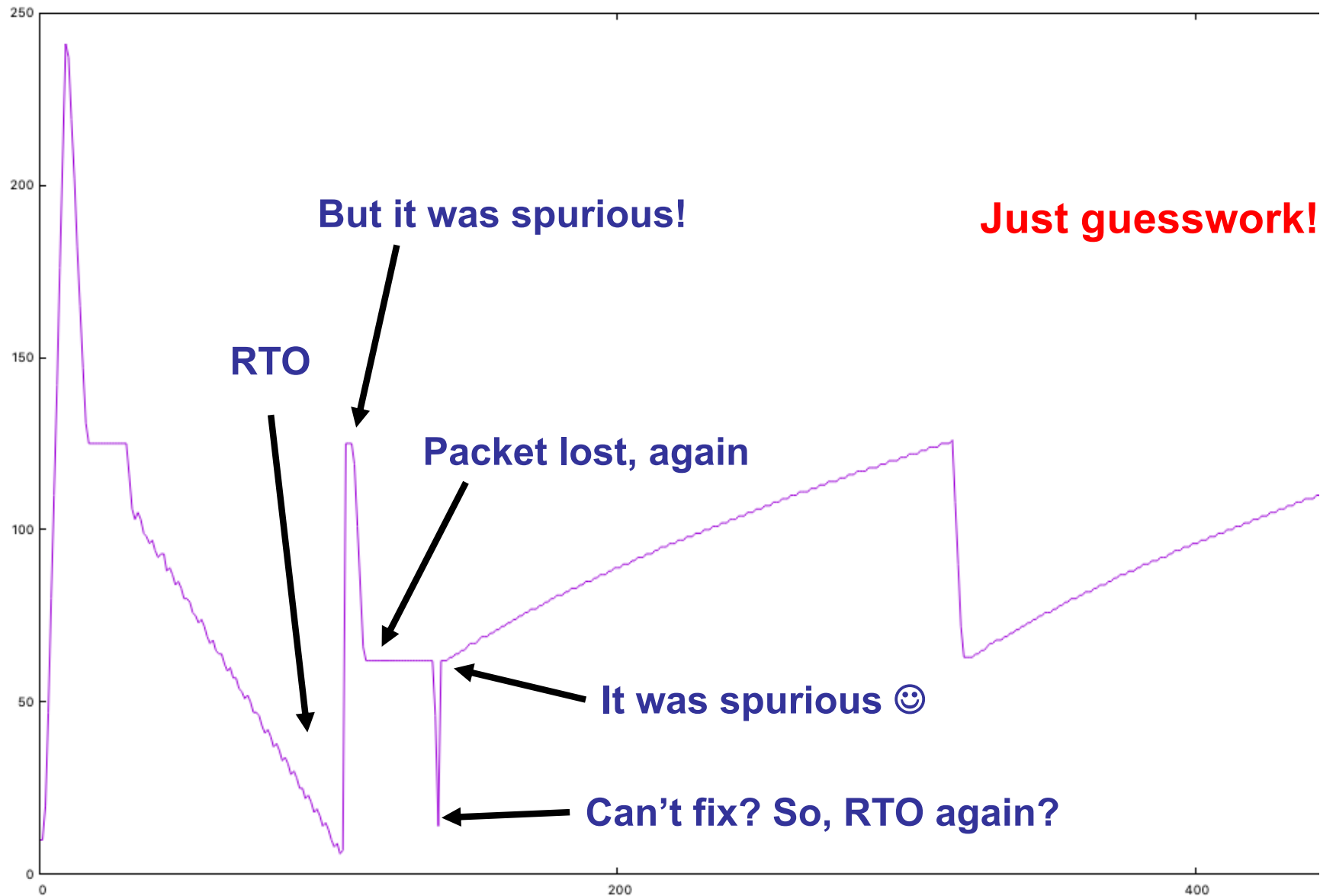
Linux TCP newreno

- *Linux tests with 3 nodes*
- *newreno CC.*
- *“PFIFO” queue*
- *packet size 1500*
- *$15 \text{ Mbps} * 20 \text{ ms} = 25 \text{ pkt BDP}$*

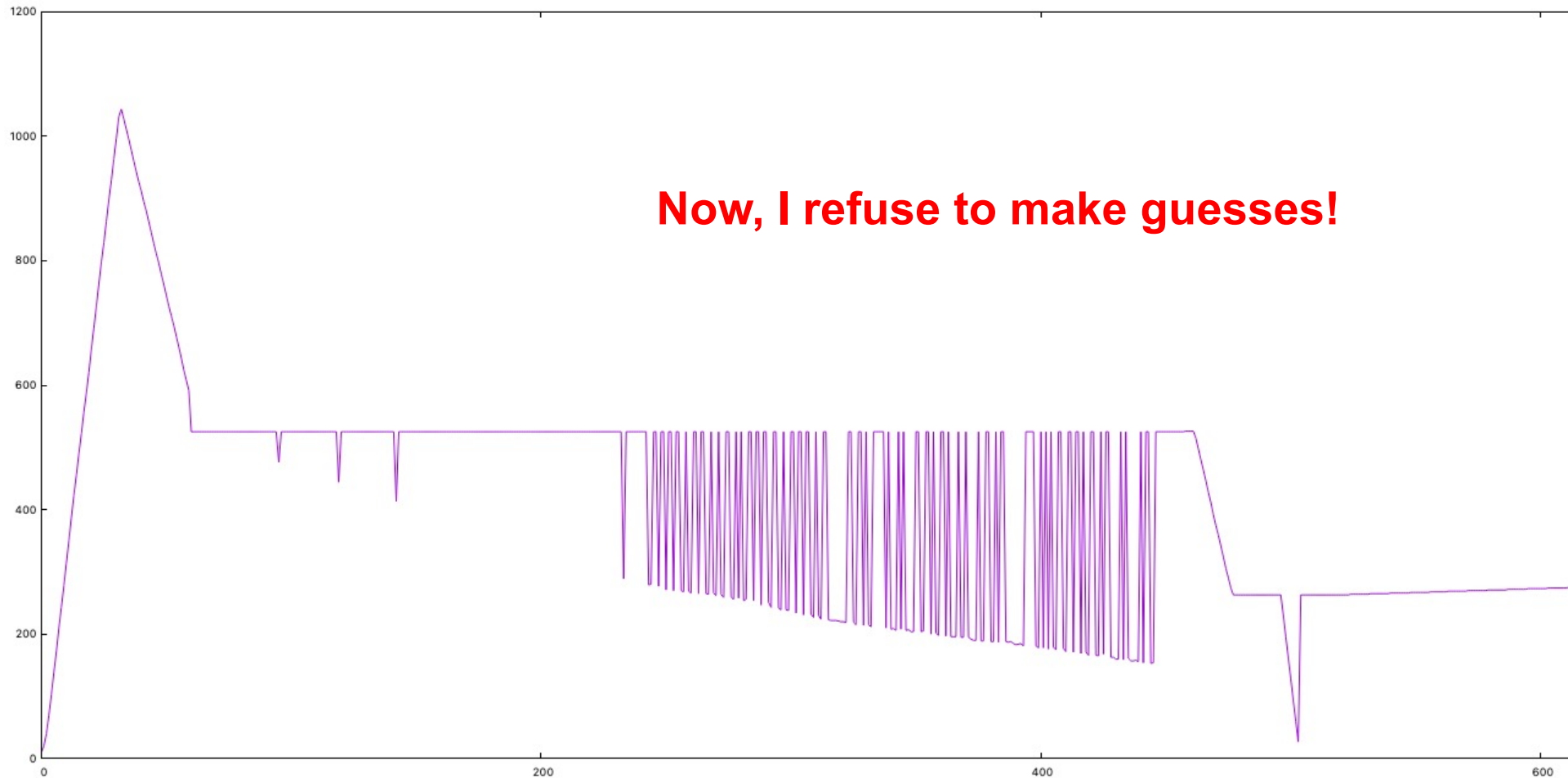


Doesn't look great?!
They're still changing it...

Linux TCP newreno, FIFO queue 4 BDPs



Linux TCP newreno, FIFO queue 20 BDPs



Tracing drops

```
AsciiTraceHelper ascii;  
pointToPoint.EnableAsciiAll(ascii.CreateFileStream("out.tr"));
```

... shows me no drops.

- I also tried this whole exercise with three nodes (sender – p2p – router – p2p receiver), and tried tracing on the two p2p links
 - The exact same result – and also no traced drops anywhere.
- Maybe the drops happened in a “higher” queue... but then, what’s the point of this trace?
 - In ns-2, this ascii trace file showed such drops
 - Maybe it’s my bad. Or maybe we need a simpler “trace-all”.

Summary of issues

- **Code**

- Non-greedy sender: cwnd keeps growing
 - Not very harmful but confusing as output
- Large queue: cwnd keeps growing
- TCP segment size: send the MSS option, make the choice depend on the MTU
- tc related code in [IPv4AddressHelper](#): detrimental behavior – seems like an extra queue of a few packets?
- Maximum cwnd seems to large... though the outcome (bytes in flight) is right
- [TCPLinuxReno](#) shows something odd in cwnd growth, but it may not matter
- More serious: [TCPLinuxReno](#) recovery failure without delayed ACKs. Yes, recovery is complex, but it cannot be worse with more ACKs.
- Generally, recovery rather slow with 2 nodes, and weird with 3 nodes
- Perhaps not very practical that drops are not visible in ascii trace files

- **Tutorial / Documentation**

- Didn't know how to run [fifth.cc](#) (linker problem)
- Dubious [TCPLinuxReno](#) vs. [TCPNewReno](#) diagram

Generally, the tutorial could be easier...

- Why do I need to write a new application just to access cwnd?
- Cool to learn about writing apps, but this should be an extra

The final "hello world".



```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello World!" << endl;
    return 0;
}
```

Part 2:

Only ns-3 could do this for us

Work published in:

- Giuseppe Bianchi, Michael Welzl, Angelo Tulumello, Giacomo Belocchi, Marco Faltelli, Salvatore Pontarelli: "A Fully Portable TCP Implementation Using XFSMs", demo, ACM SIGCOMM 2018, Budapest, Hungary.
3rd place in Undergraduate Student Research Competition (SRC) awarded to Angelo Tulumello
- Giuseppe Bianchi, Michael Welzl, Angelo Tulumello, Francesco Gringoli, Giacomo Belocchi, Marco Faltelli, Salvatore Pontarelli: "XTRA: Towards Portable Transport Layer Functions", in IEEE Transactions on Network and Service Management, On page(s): 1-15, Print ISSN: 1932-4537, Online ISSN: 1932-4537, October 2019.

Code: <https://github.com/angelotulumello/xtra>

Research idea: make TCP portable

- Also to hardware
 - using XFSMs (eXtended Finite State Machines)
- Seemed feasible for Slow Start, Congestion Avoidance... but: Loss Recovery???
- TCP loss recovery is quite complex!

TCP loss recovery in a nutshell

- **Timeout or DupACKs** (Fast Retransmit / Fast Recovery)
 - But also parse “partial ACKs” (NewReno, RFC 6582)
- **We absolutely need to stay in FR**: need enough information from ACKs
 - So, transmit in response to DupACKs #1, #2: Limited Transmit, RFC 3042
 - ... so, retransmit if there’s no new data: Early Retransmit, RFC 5827
- **Let’s do even better, using SACK information**
 - **What to send when, and how much?**
 - “Conservative Loss Recovery Algorithm”, RFC 6675
 - Well known related variant, not standardized: FACK
 - **Get a good (almost “paced”) traffic pattern**
 - Originally, only in Linux: “Rate Halving”
 - Later: Proportional Rate Reduction (PRR), RFC 6937 (now being updated)

Matt Mathis, Jamshid Mahdavi:
“Forward acknowledgement:
refining TCP congestion control”,
SIGCOMM’96.

TCP loss recovery in a nutshell

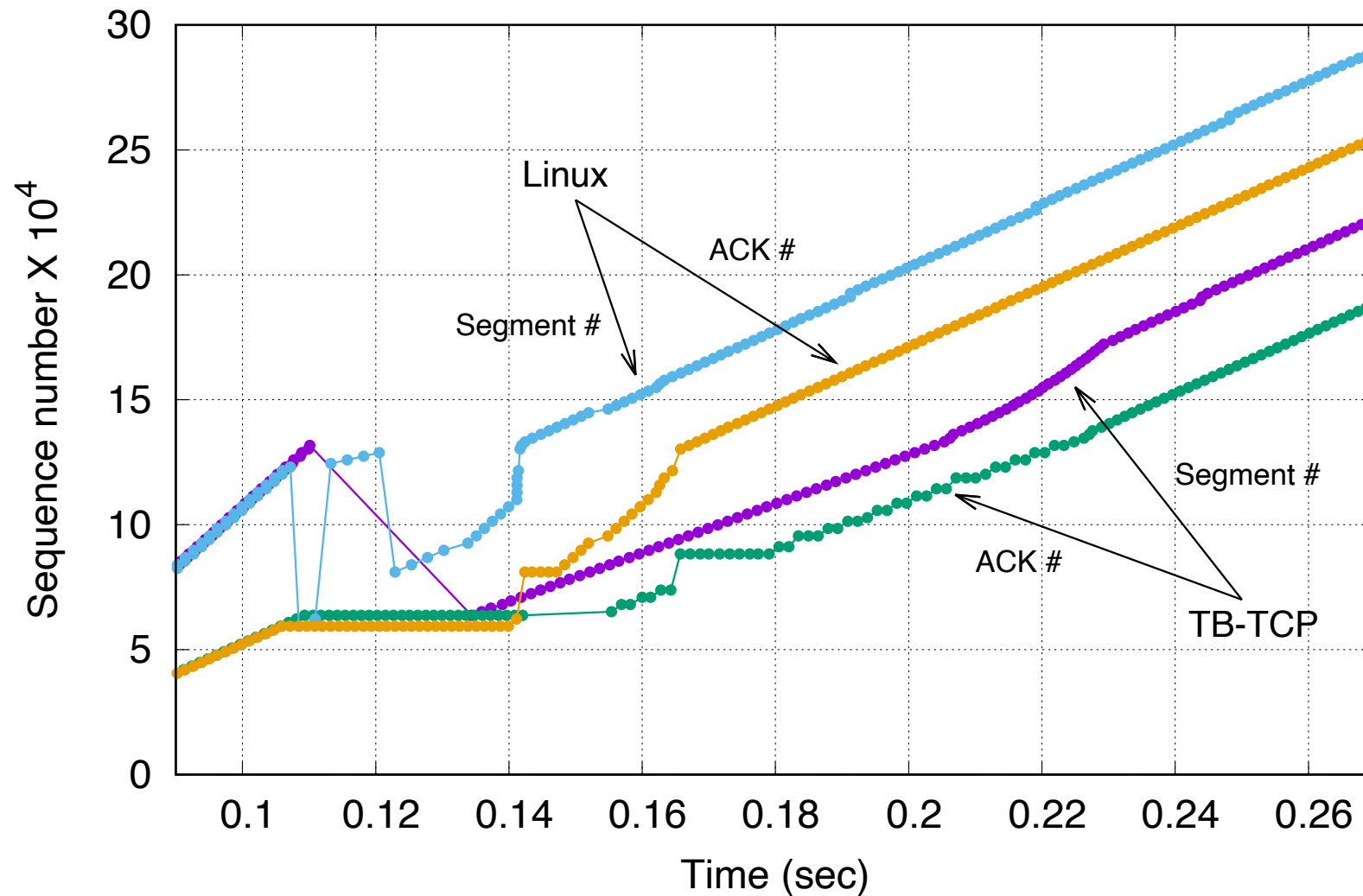
- Tail loss: the end of a short flow
 - Try an extra timer: Tail Loss Probe (TLP)
 - No, better, use timestamps for everything!
 - Combination: Recent Acknowledgement (RACK) + TLP algorithm in RFC 8985
- What if we get it wrong?
 - Spurious Loss Recovery, using Timestamps (Eifel algorithm, RFCs 3522, 4015)
 - ... using Duplicate SACK (DSACK) information: RFC 3708
 - ... using intelligent probing (new data instead of rexmit): F-RTO, RFC 5682
- ... and the RTO calculation can go wrong, too
 - RTO Restart: RFC 7765



Idea: replace all of loss recovery

- It gets much simpler with a timer
- Not a “pluggable congestion control”
 - Not much fun to implement in an OS kernel
- Needed to compare against real Linux and FreeBSD ...
so, real-life tests
 - ns-3’s emulation capabilities to the rescue!
 - ns-3 as a sender, *ncat* as a receiver
 - Starting ns-3 from “*TEACUP*” scripting environment
(TCP Experiment Automation Controlled Using Python
(TEACUP)) - <http://caia.swin.edu.au/tools/teacup/>

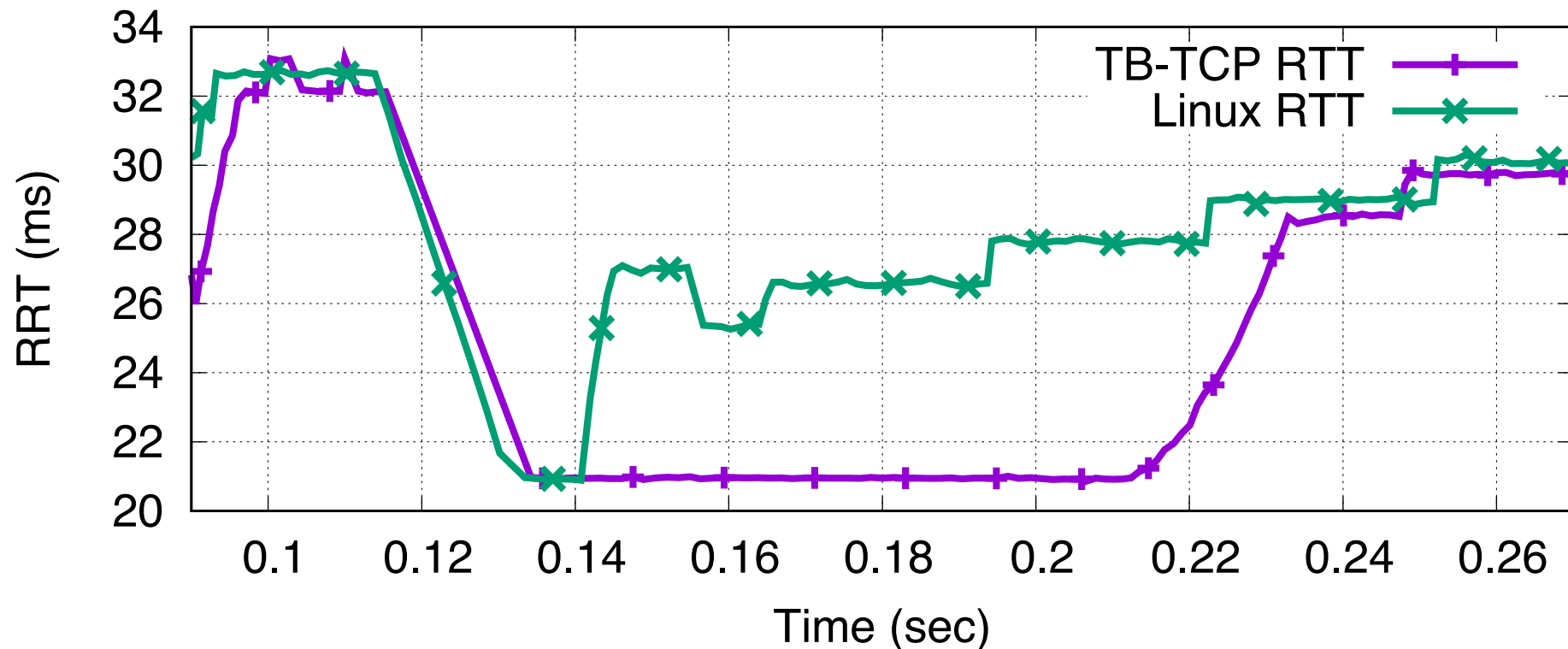
Outcome: suitable indeed!



We could even use SPP !

S. Zander, G. Armitage, "Minimally-Intrusive Frequent Round Trip Time Measurements Using Synthetic Packet-Pairs", (short paper) 38th IEEE Conference on Local Computer Networks (LCN), Sydney, Australia, October 2013.

<http://caia.swin.edu.au/tools/spp/>



However, multiple flows...

- Up to 3 flows send SYN at the same time
 - The next flows only start after the first SYN-ACK arrives

Caused by this code in "src/internet/model/arp-cache.cc":

```
.AddAttribute("PendingQueueSize",  
    "The size of the queue for packets pending an arp reply.",  
    UIntegerValue(3),  
    MakeUIntegerAccessor(&ArpCache::m_pendingQueueSize),  
    MakeUIntegerChecker<uint32_t>())
```

Thank you, Mohit P. Tahiliani
& Tommaso Pecorella !

ArpCache... how
could anyone guess?

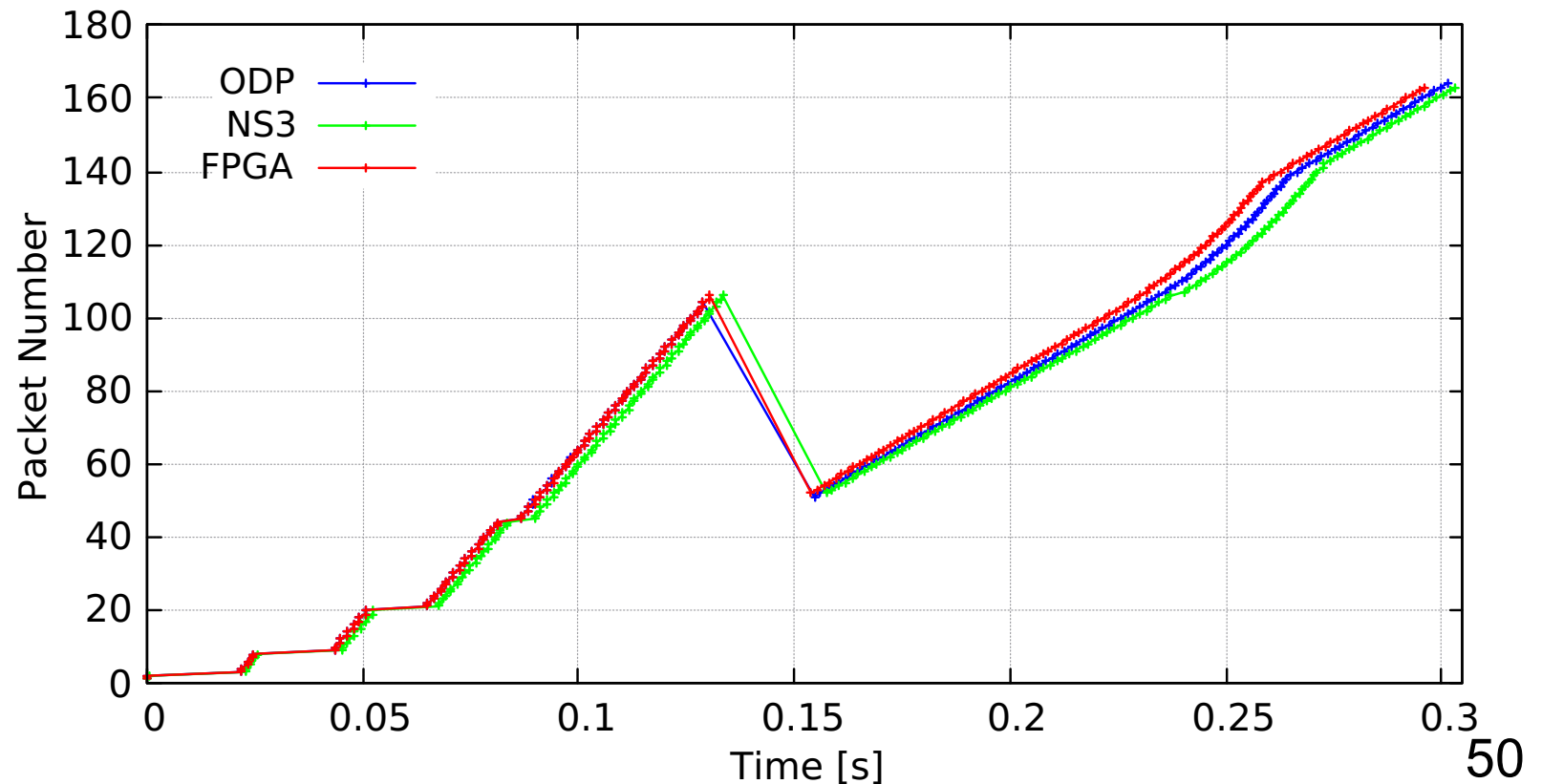
This default setting can be changed by using the following line (to be used before installing the Internet stack on the nodes):

```
Config::SetDefault("ns3::ArpCache::PendingQueueSize", UIntegerValue(10));
```

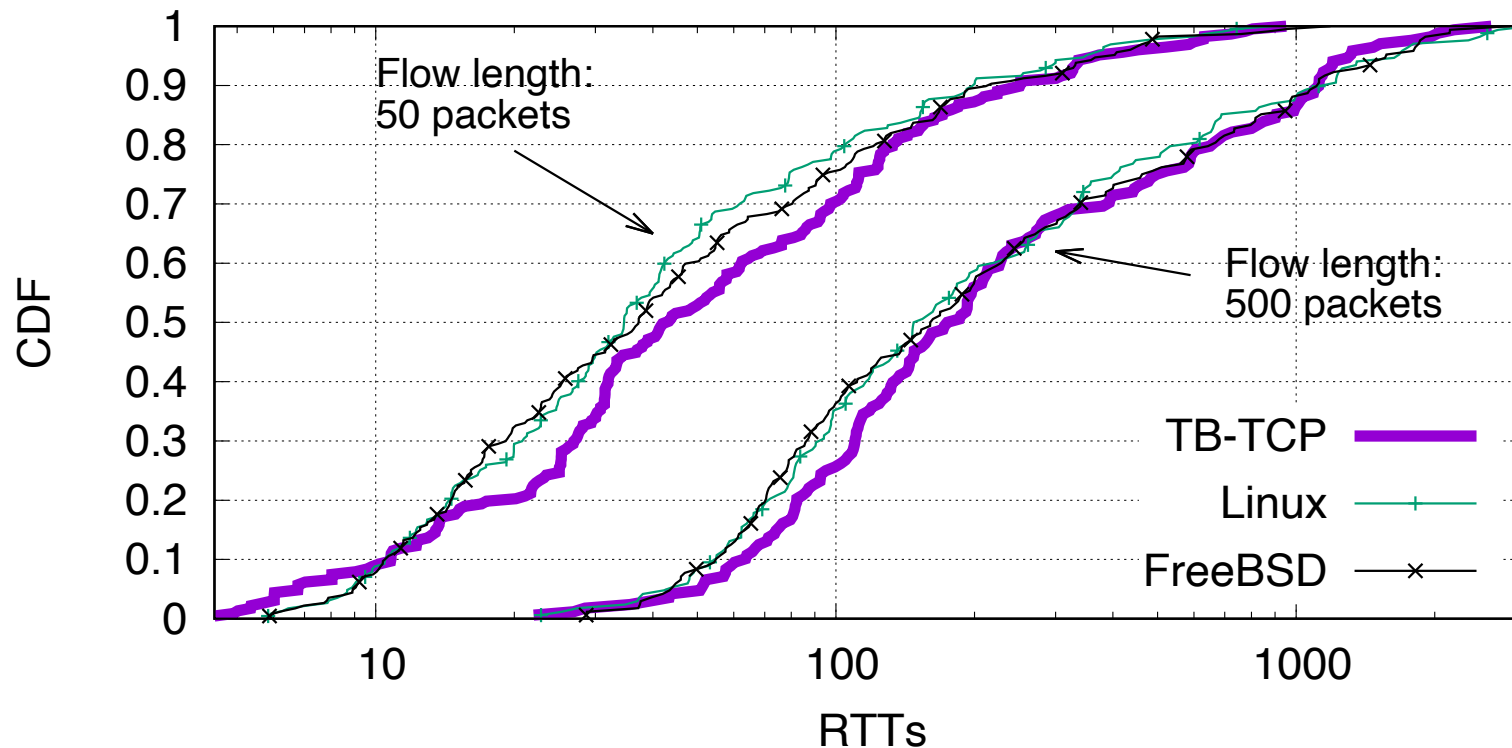
What we were able to pull off

Conversion of TCP to XFSM: exact same code runs on..

1. XFSM execution engine in ns-3
2. OpenDataPlane + DPDK
3. FPGA



... and yes, also multiple flows!



*Flow
Completion
Time (FCT) of
the slowest of
6 flows*

Varied parameters:

Bottleneck capacity (1-5, 10 and 15 Mbit/s), RTT (10, 20, 40, 80, 100, 150, 200 ms), length of the bottleneck queue (5, 10, 50, 100, 200 packets). Filtered out cases where flows did not reach the considered length during the measurement period => 168 (50 packets) and 227 (500 packets) experiments

Part 3: The role of ns-3 in TCP (and related) research

ns and TCP

- Let's face it: people hardly ever use ns-3 for TCP
 - (Real!) conversation with a colleague:
 - *Me: "I'm preparing these keynote slides... and I find many strange things related to TCP."*
 - *Colleague: "That's not surprising: ns-3 isn't really meant for TCP, right?"*
 - Definitely not the first time I hear this !
- ns-2 was the de facto standard for TCP development
 - Significant overlap between people writing...
 1. ns-2 code
 2. FreeBSD code
 3. RFCs

Q1: Why has this changed? (What can we do?)

Q2: Do we even want to go back to this? (If so, why?)













Q1: Why has this changed?

(What can we do about it?)

6 reasons to use a simulator for TCP (and related) research

1. Trustworthy reference code
2. Easier than changing an OS kernel
3. Faster (simulation time \neq real time)
4. Able to simulate richer scenarios
5. Very consistent behavior: without randomness, the same input always yields the same output
6. Availability of others' code

ns-2 (once upon a time, not now!) vs. ns-3

	ns-2	ns-3
1. Trustworthy reference code		
2. Easier than changing an OS kernel		
3. Faster (simulation time != real time)		
4. Able to simulate richer scenarios		
5. Consistency (without randomness)		
6. Availability of others' code		

My own example was [case 2](#)... but focusing on recovery is a bit special.

[Case 4](#) may well be the primary reason for some people to use ns-3+TCP!

[Case 5](#) is not a big deal: testbed behavior can be quite consistent.

What's needed for "thumbs up"?

- Trustworthy reference code
 - That could be written... also needs some validation results, some community convincing... perhaps a paper showing it?
- Easier than changing an OS kernel
 - Fix errors, make the tutorial easier!
 - Also, ease of use: e.g., accessing cwnd
 - Add a TCP chapter? E.g., on “hello world” without artificial loss!
- Faster (simulation time != real time)
 - I don't know... maybe an unfounded complaint??
- Availability of others' code
 - This should happen as a consequence.

**Q2: Do we even want to go
back to this?**

(If so, why?)

TCP is a mess today...

Some guesswork – discrepancy examples (really only my belief):

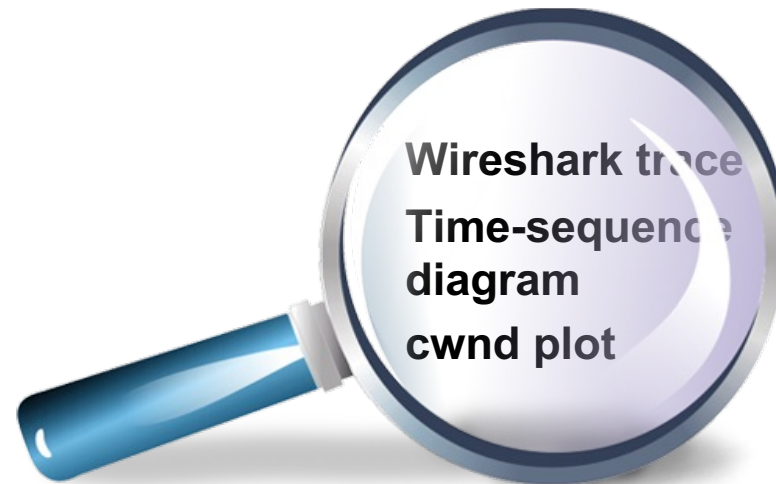
- Proposed Standard RFCs that are not implemented in open code
 - **HyStart++** (RFC 9406), **Congestion Manager** (RFC 3124), **User Timeout Option (UTO)** (RFC 5482)
- Experimental / Informational RFCs that are widely used
 - **IW10** (RFC 6928), **RACK-TLP** (RFC 8985), **Cubic** (RFC 8312), **PRR** (RFC 6937), **Early Retransmit** (RFC 5827), **DSACK** (RFC 3708), **parts of TCB sharing** (RFC 9040)
- Things that are widely used but no RFCs
 - **HyStart** (not HyStart++)
 - **Pacing**
 - **BBR**

Some of these are on their way towards Proposed Standard.

TCP is a mess today... /2

- Can the IETF even keep track?
 - Open code: Linux (= Google) vs. FreeBSD
 - Closed code: Microsoft vs. Apple
 - ... vs. IETF standards!
 - Most researchers work with the Linux code
- ML-based congestion control: how could we even standardize?
- BBR moves ahead, in various ways... will they update the draft?
When?
 - draft describes BBRv2, but Linux code is BBRv1
 - Still the old BBRv1? Probably not? Rather, BBRv1 ½?
- Perhaps we need a new "authority" code base !

TCP evaluation



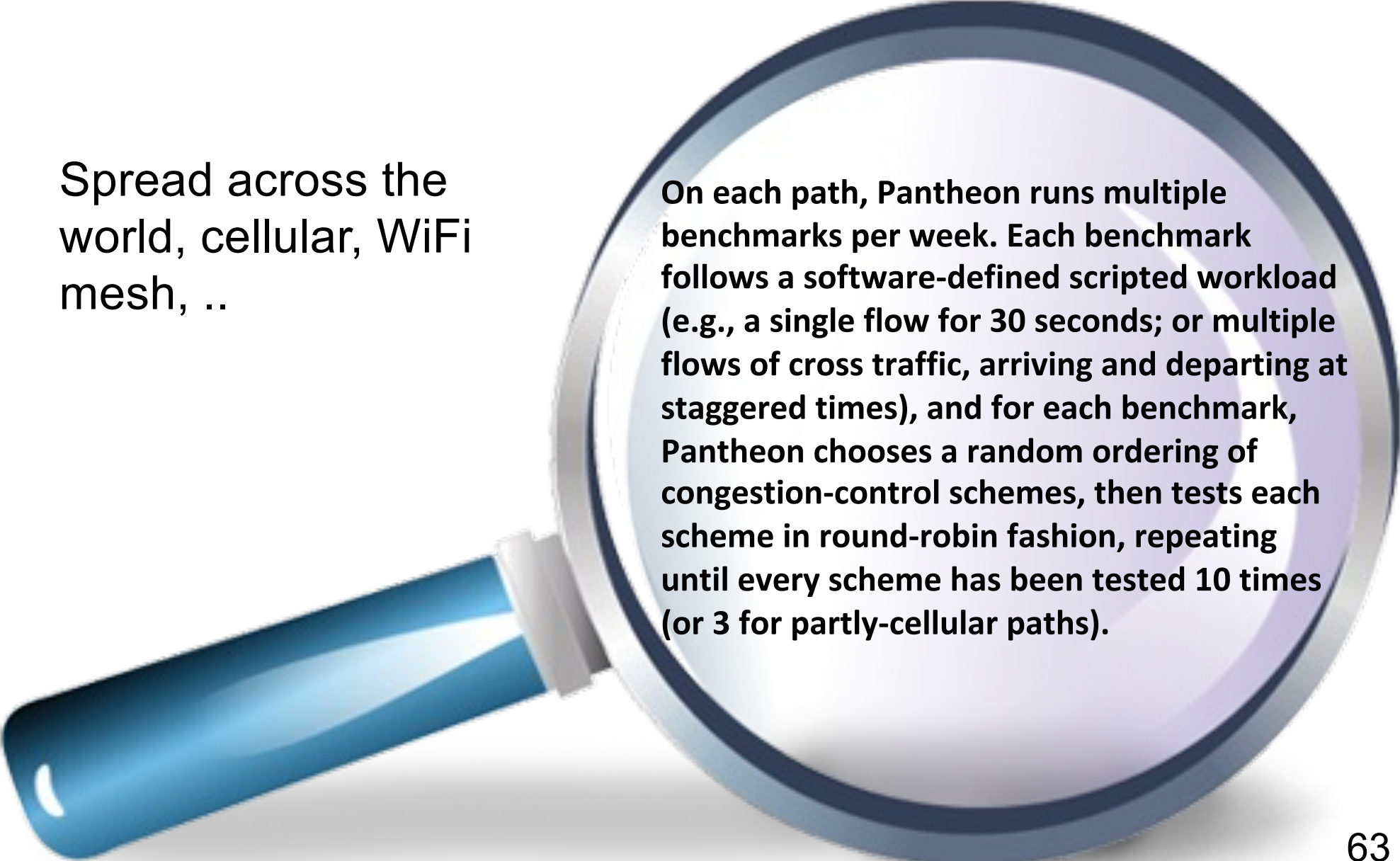
TCP evaluation



- TCP Evaluation Suite: a long set of tests, many outputs
 - ns-2 code exists, Internet-draft and paper exists
 - Never standardized, and will never be...

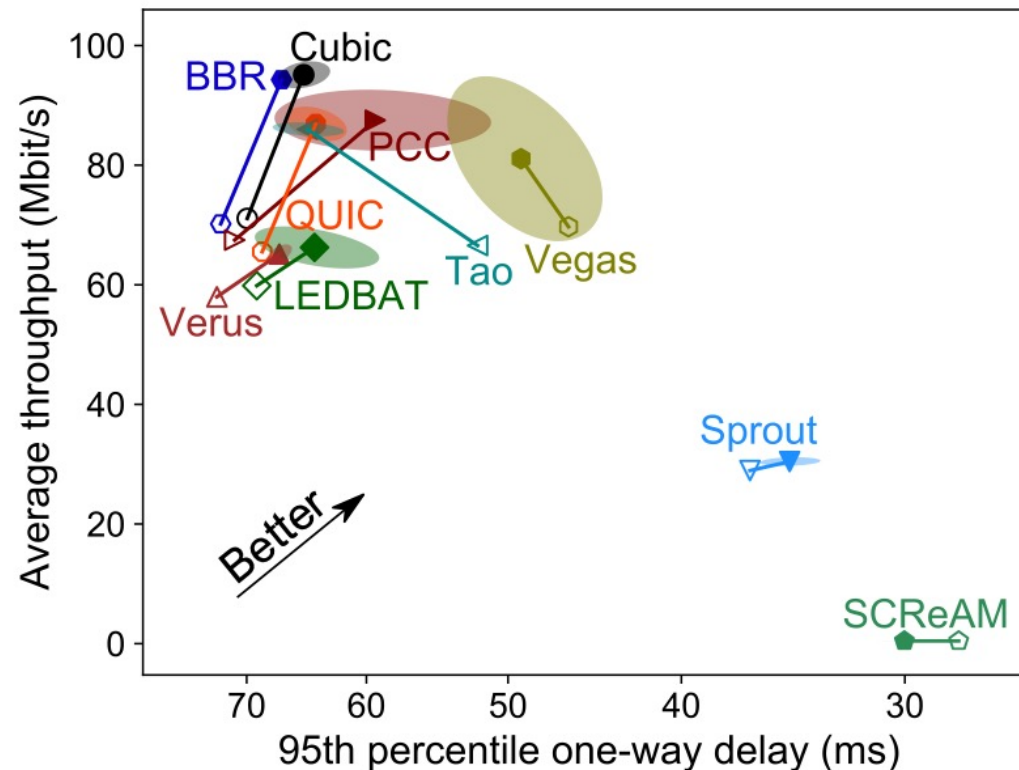
Pantheon <https://pantheon.stanford.edu>

Spread across the world, cellular, WiFi mesh, ..



On each path, Pantheon runs multiple benchmarks per week. Each benchmark follows a software-defined scripted workload (e.g., a single flow for 30 seconds; or multiple flows of cross traffic, arriving and departing at staggered times), and for each benchmark, Pantheon chooses a random ordering of congestion-control schemes, then tests each scheme in round-robin fashion, repeating until every scheme has been tested 10 times (or 3 for partly-cellular paths).

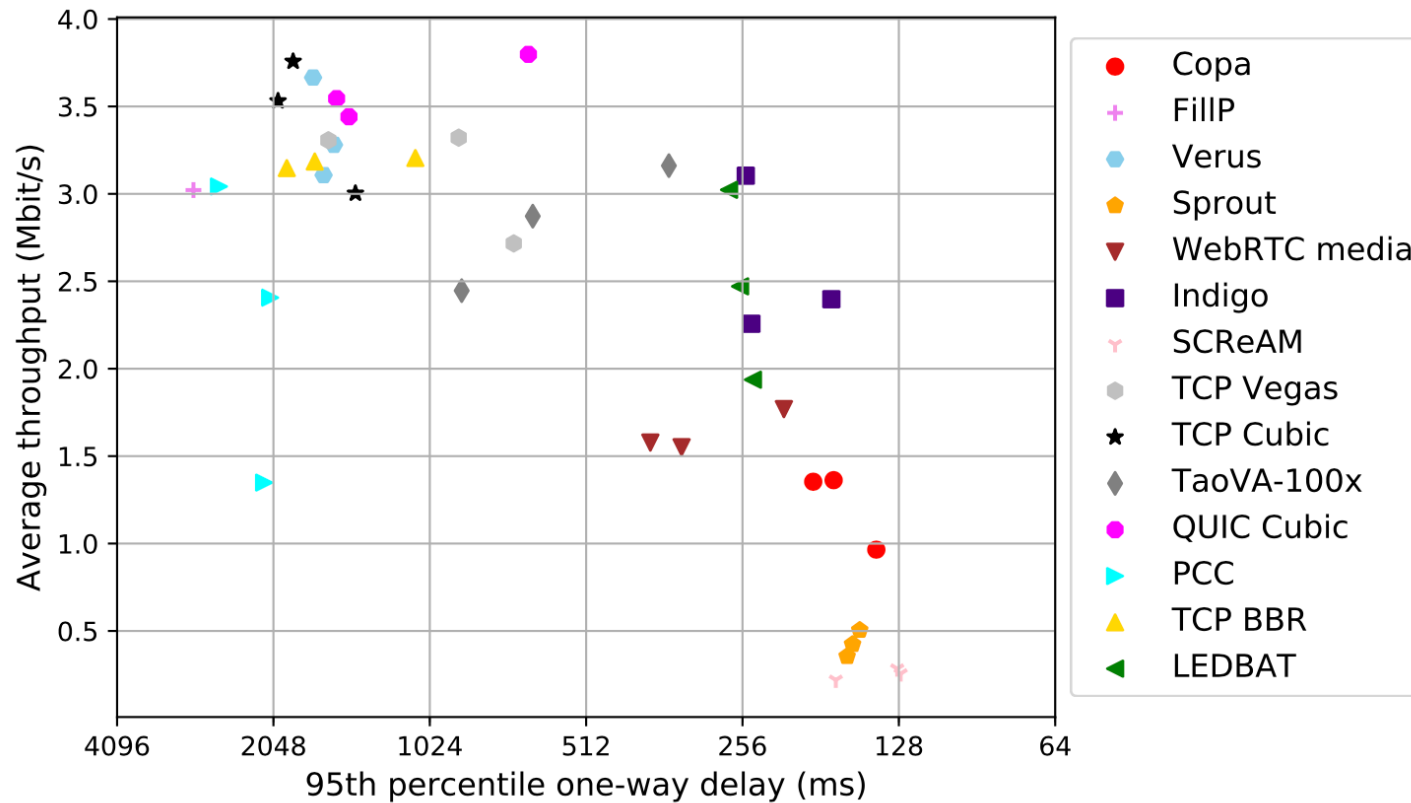
Typical Pantheon output (per scenario)



Francis Y. Yan, Jestin Ma, Greg D. Hill, Deepti Raghavan, Riad S. Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for internet congestion-control research. In Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '18). USENIX Association, USA, 731–743.

Many mechanisms available

test from AWS Brazil 2 Ethernet to Colombia ppp0, 3 runs of 30s each per scheme
3 flows with 10s interval between flows



From the Pantheon webpage

However...

⚠ **End of support:** After enabling our own algorithm ([Indigo](#)) and assisting four algorithms from other research groups in publishing at NSDI '18 ([Copa](#) and [Vivace](#)), ICML '19 ([Aurora](#)), and SIGCOMM '20 ([TCP-TACK](#)), we regret to announce that we will no longer maintain Pantheon's codebase, accept new schemes, or run new experiments. Please check out our new video streaming platform [Puffer](#), which also welcomes novel congestion-control algorithms to test with video traffic.

- Wouldn't it be cool if I could run:
`ns3 testmytcp protocol_list scenario_choice`
... and get such outputs?
With more cc's, and the newest?

Thank you!

Questions?

Thank you: Mohit P. Tahiliani, Tommaso Pecorella, Safiqul Islam