Chirping for Congestion Control - Implementation Feasibility

PFLD*NeT* 2010, Lancaster

Mirja Kühlewind <mirja.kuehlewind@ikr.uni-stuttgart.de>
Bob Briscoe <bob.briscoe@BT.com>

Universität Stuttgart
Institute of Communication Networks
and Computer Engineering (IKR)
Prof. Dr.-Ing. Andreas Kirstädter

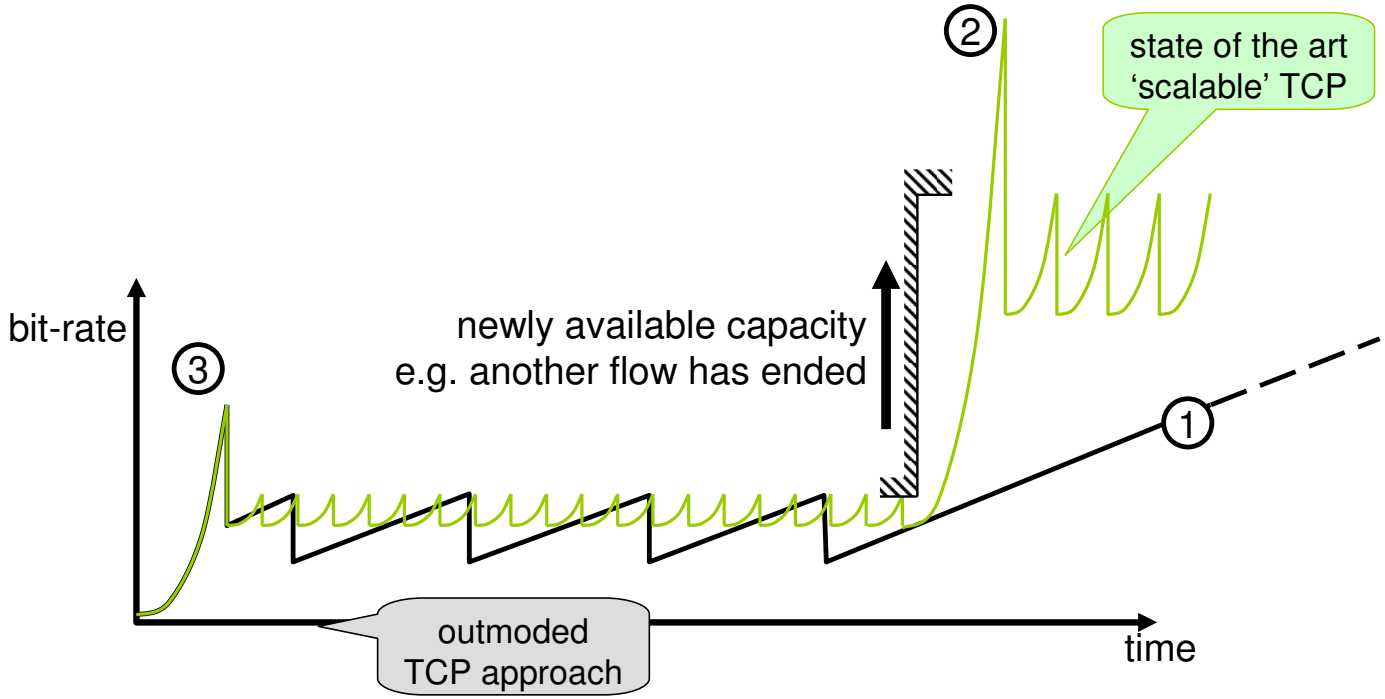
Overview

- Motivation
- Chirping as a Building Block for Congestion Control
- A Chirping Implementation in the Linux Kernel
- Preliminary Results
- Conclusion and Outlook

Motivation

Scaling Problem

1. Original TCP acquires new bandwidth too slowly
2. State-of-the-art approaches overshoot instead
3. Overshoot causes a lot unnecessary congestion



→ Chirping can provide fast feedback information for appropriate congestion control!

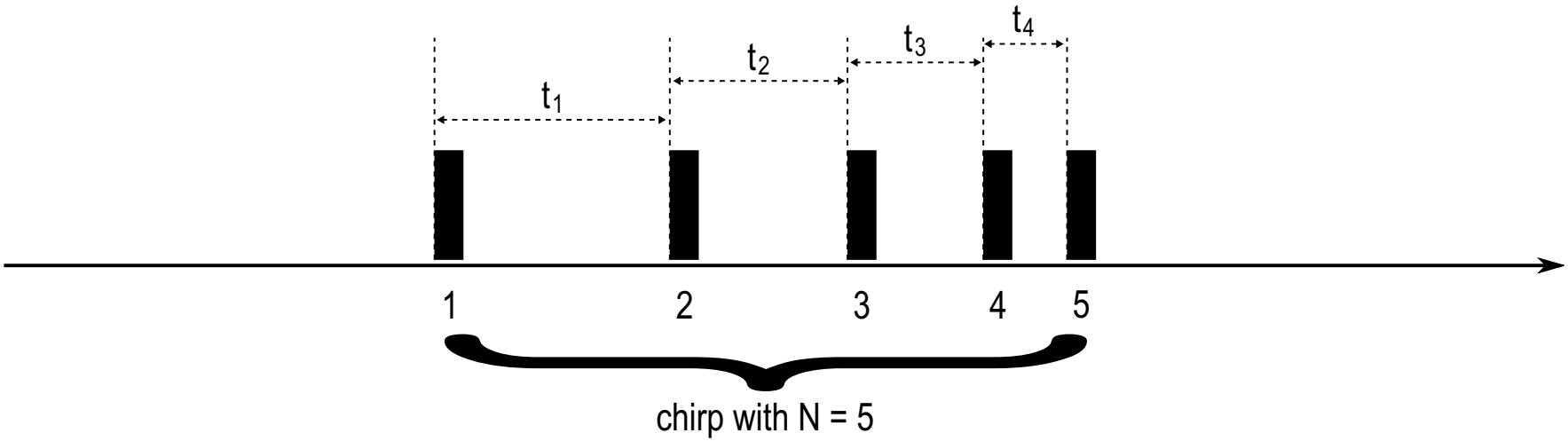
→ But is an implementation of chirping in an real OS feasible?

Chirping

Principle

Chirp: A group of several packets with decreasing inter-packet gaps and increasing rate

– Proposed by pathChirp bandwidth estimation tool [1]



- Bandwidth estimation based on self-induced congestion
- Feedback for monitoring of one-way delay

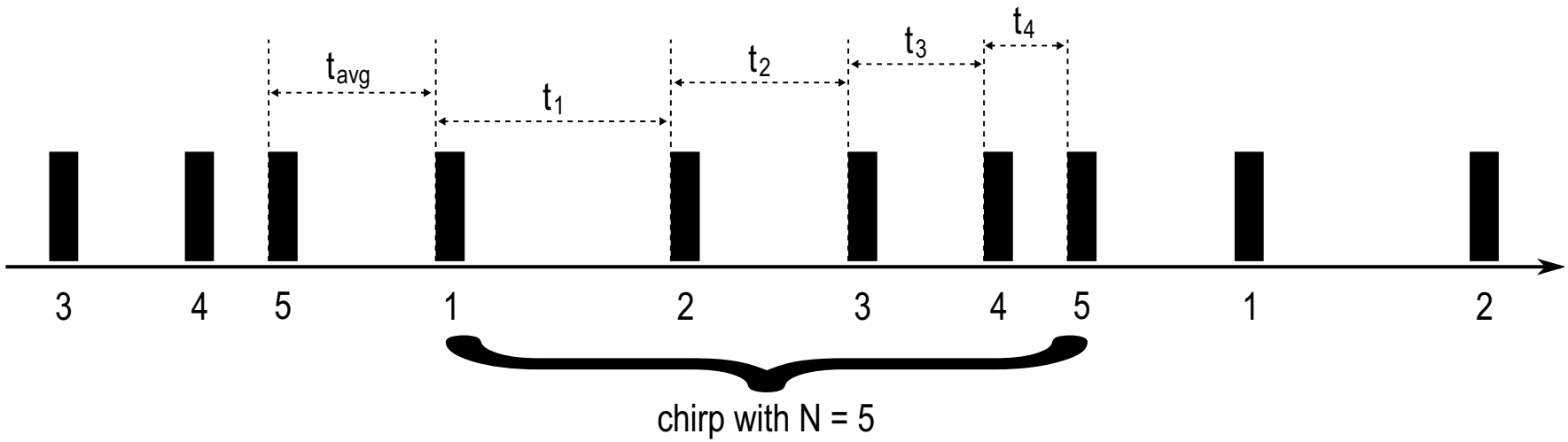
[1] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil and L. Cottrell. "pathChirp: Efficient Available Bandwidth Estimation for Network Paths". Passive and Active Measurement Workshop 2003

Chirping

A Building Block for Congestion Control

Chirping for Congestion Control: Continuous transmission of data packets as chirps

– proposed by RAPID congestion control [2]



- Average rate r_{avg} should equal intended sending rate of congestion control
- Actual per-packet rates are lower and higher than r_{avg}

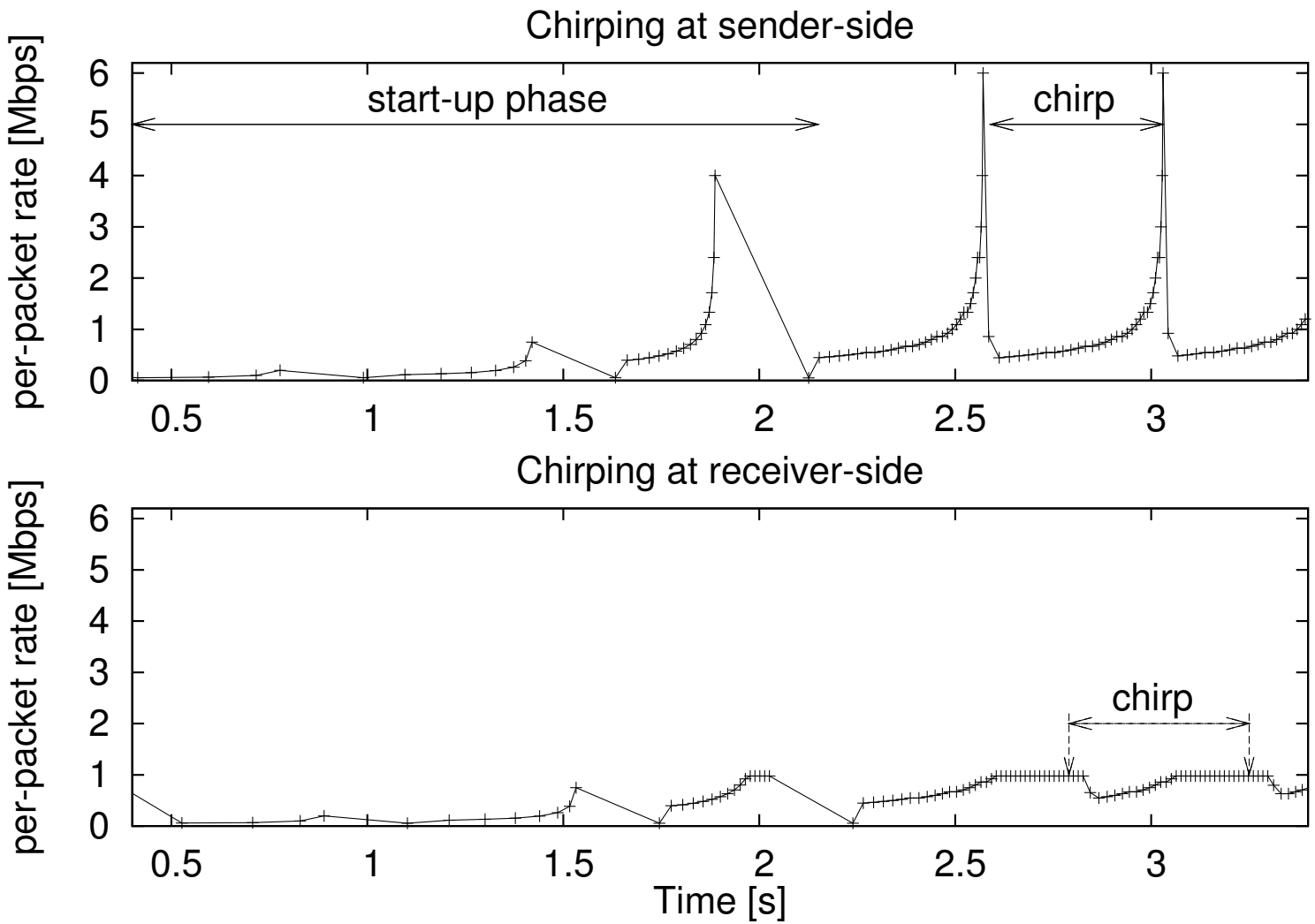
  → Probing for a wide range of possible sending rates but still limited impact of probing on other flows

[2] V. Konda and J. Kaur. "RAPID: Shrinking the Congestion-Control Timescale". In IEEE INFOCOM 2009

Chirping

Preliminary Results

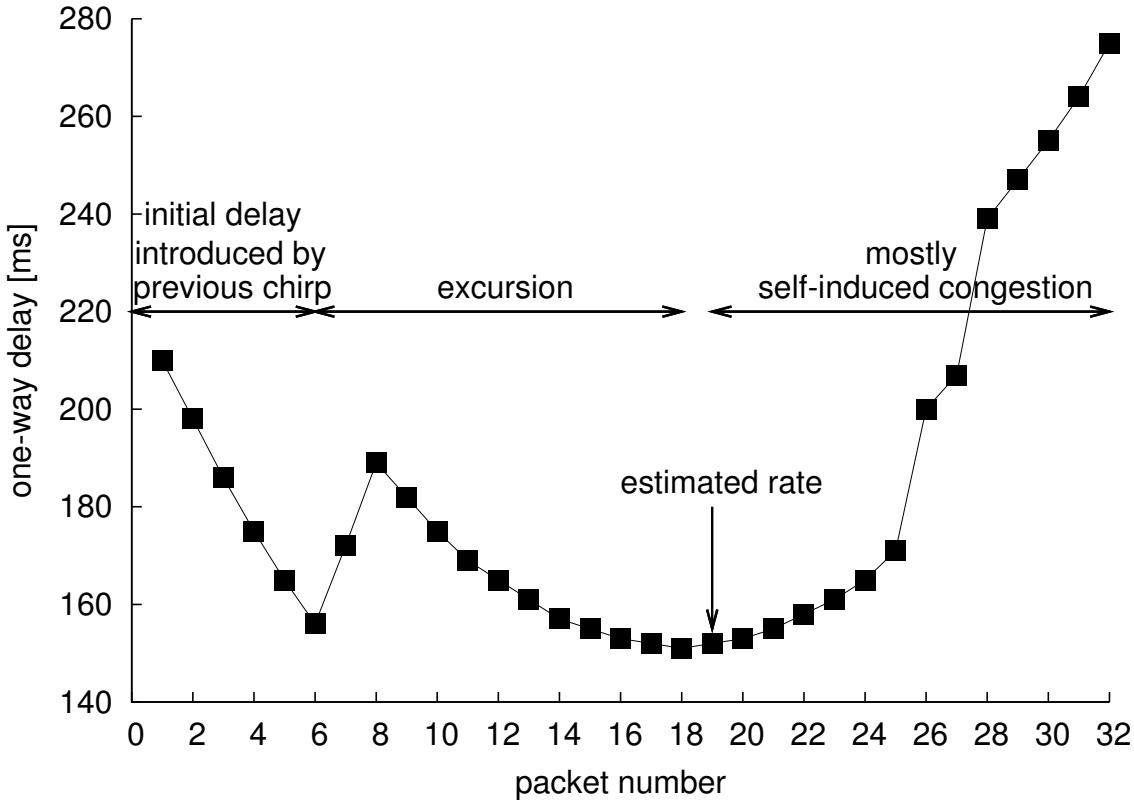
Per-Packet rate of one chirping connection on 1Mbit/s bottleneck link



Chirping

Bandwidth Estimation based on relative One-way Delay

Bandwidth estimation: Monitoring of the relative queuing delays



- Growth in queuing delay between packets: $\triangle q_n = q_n - q_{n-1}$

  $\rightarrow$ Increasing values at the end of reveals available capacity (*self-induced congestion*)

Chirping Implementation in the Linux Kernel

Overview

1. Feedback for one-way delay measurement:
   - Protocol extensions needed
   - Different solutions for deployment proposed
2. Rate estimation:

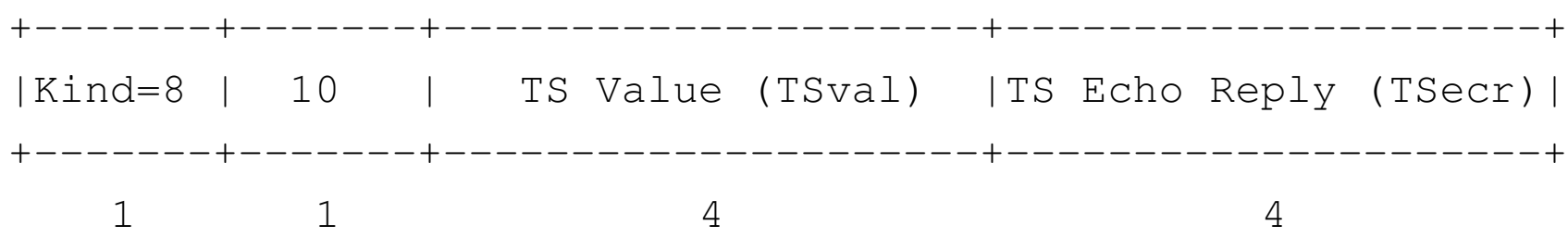
   Algorithm to evaluate the feedback information of one chirp based on pathChirp [1]
3. Rate adaption:
   - Congestion control algorithm evolves the average sending rate of the next chirp (using the available capacity estimated by a previous chirp)
   - CWND should also be updated to a value that allows the intended number of packets to be sent in one RTT
4. Inter-packet gap calculation:
   - Determined by the chosen average sending rate
   - Own algorithm in order to simplify kernel implementation to integer arithmetic

[1] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil and L. Cottrell. "pathChirp: Efficient Available Bandwidth Estimation for Network Paths". Passive and Active Measurement Workshop 2003

Chirping Implementation in the Linux Kernel

Sender-side Delay Measurement based on TCP Timestamps

One-way delay measurement based on TCP Timestamp Option



→ Option header includes echoed timestamp of data packet and ACK timestamp

Challenges

- TCP Timestamp Option does not ensure certain resolution
- Feedback needs to be assigned to one specific packet in a chirp (delayed ACKs?)
- Additional processing delay in the network stack

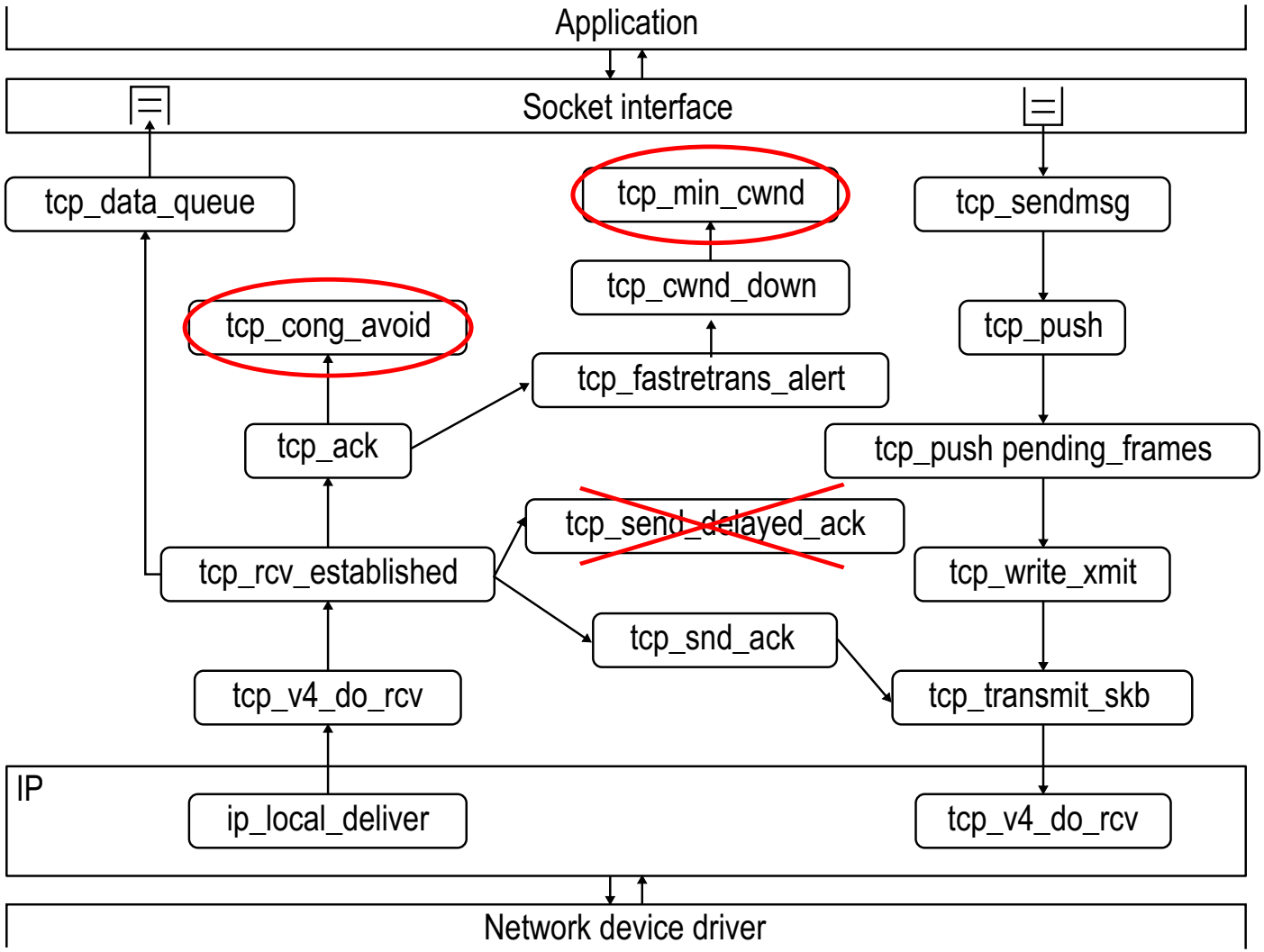
Proposed Solutions

- Negotiation about the TCP receiver behaviour
- Chirp ID attached to packet header instead of state at sender
- Hardware time-stamping at send-out of data packet and ACK to improve accuracy
- Improved accuracy by use of the actual sending time gaps

Chirping Implementation in the Linux Kernel

Implementation Structure

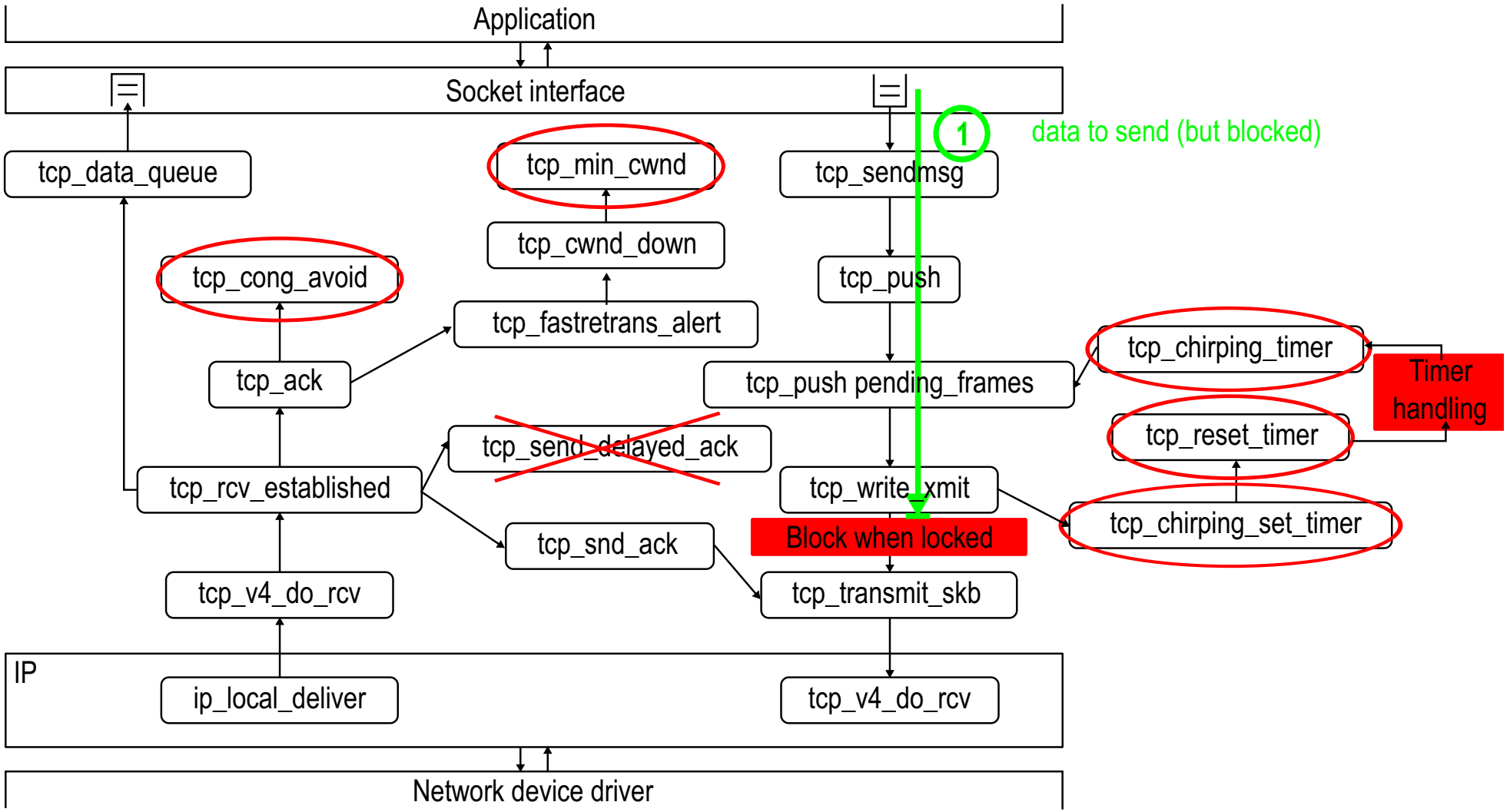
→ Extended congestion control kernel module interface and TCP timer for send-out timing



Chirping Implementation in the Linux Kernel

Implementation Details

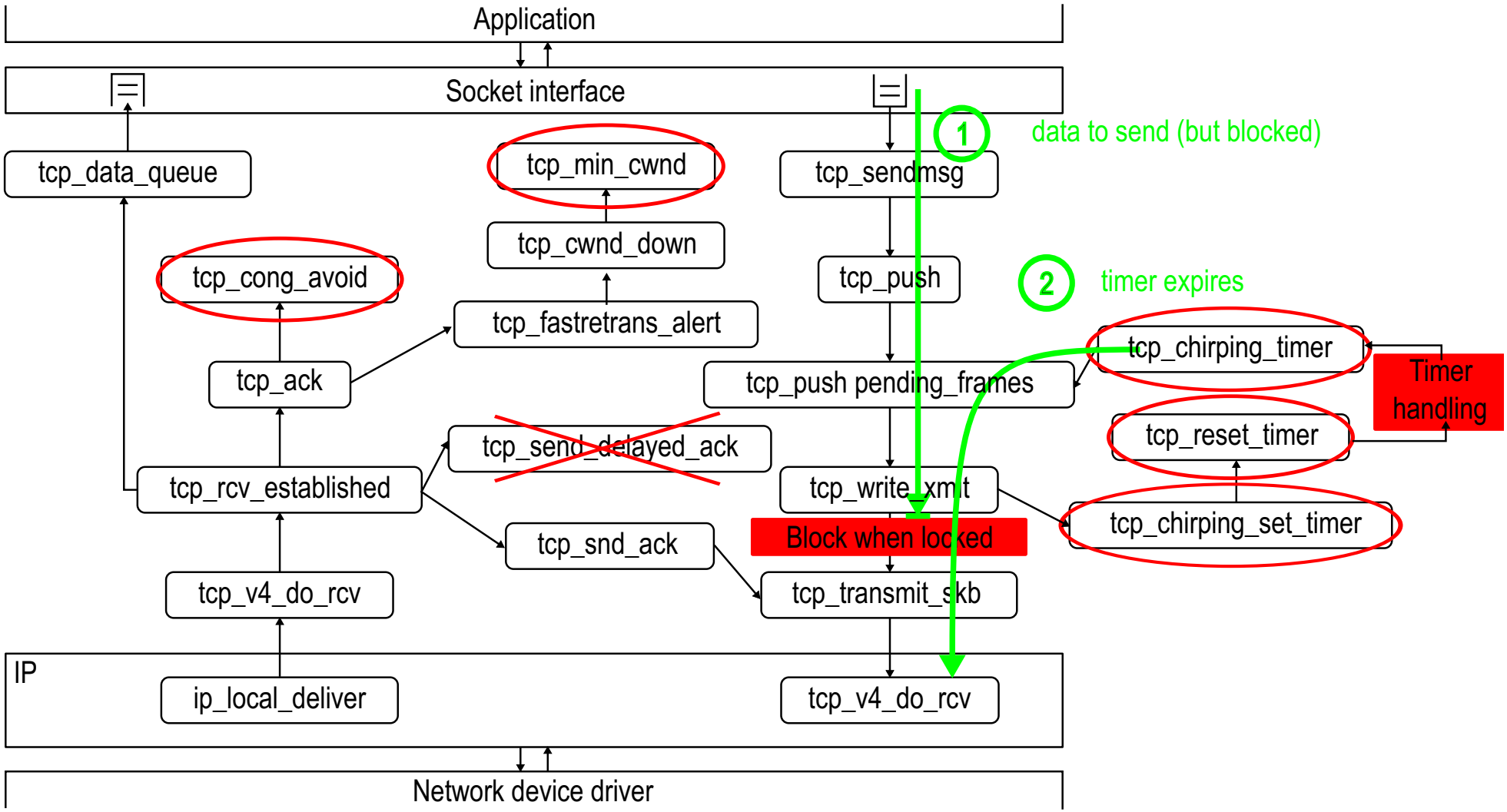
→ Extended congestion control kernel module interface and TCP timer for send-out timing



Chirping Implementation in the Linux Kernel

Implementation Details

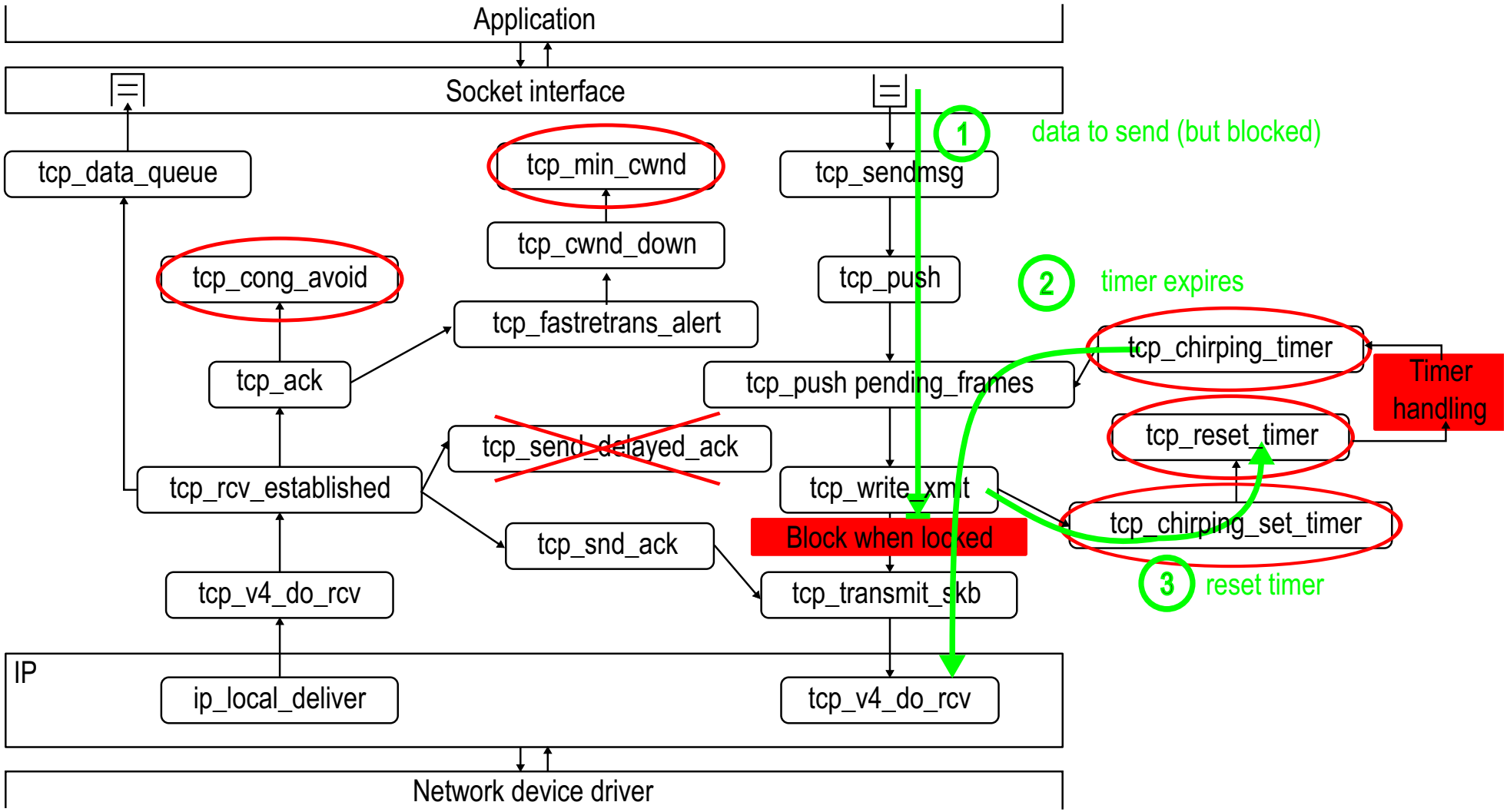
→ Extended congestion control kernel module interface and TCP timer for send-out timing



Chirping Implementation in the Linux Kernel

Implementation Details

→ Extended congestion control kernel module interface and TCP timer for send-out timing



Chirping Implementation in the Linux Kernel

Open Issues

- Timer-based sent-up does not use ACK-clocking
  → Addition interupt processing burden
  → Could be solved by the use of hardware timers in future
- Timer resolution has to be high enough to serve high-speed links (hrtimers in the Linux kernel provide nanosecond resolution)

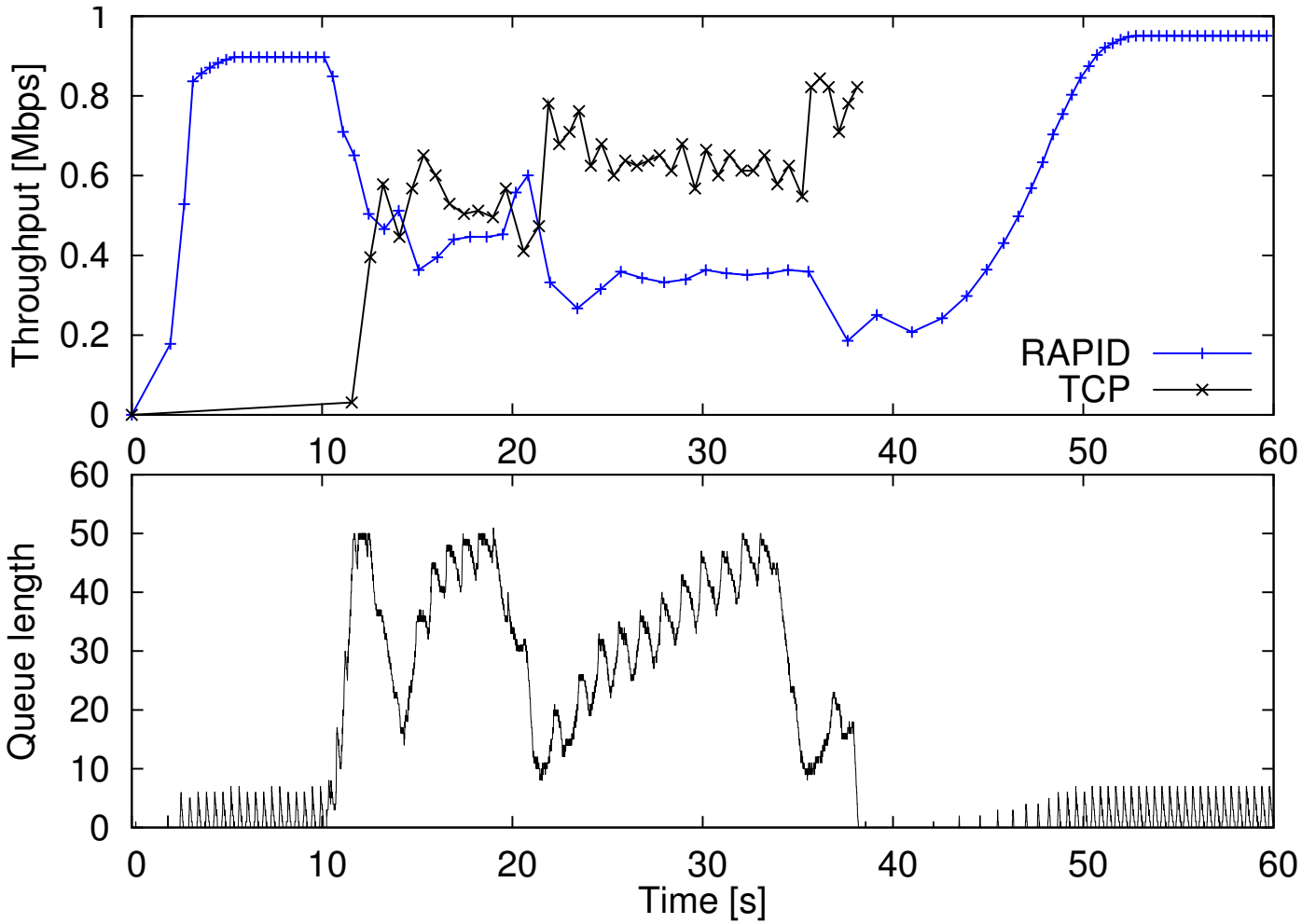
Futher improvements

- Fully based on inter-packet time gaps instead of rate
- N should be an the integer power of 2
  → Initiallly hard-coded to N = 32 ($=2^5$)
- Algorithm for Inter-packet gap Calculation
  – Harmonic progression of rates by linear decrease of inter-packet gaps:
    $gap_i = gap_{i-1} - gap_{step}$ with $gap_{step} = (2 * gap_{avg}) / N$
  → Implementation with integer arithmetic

Preliminary Results

RAPID Congestion Control

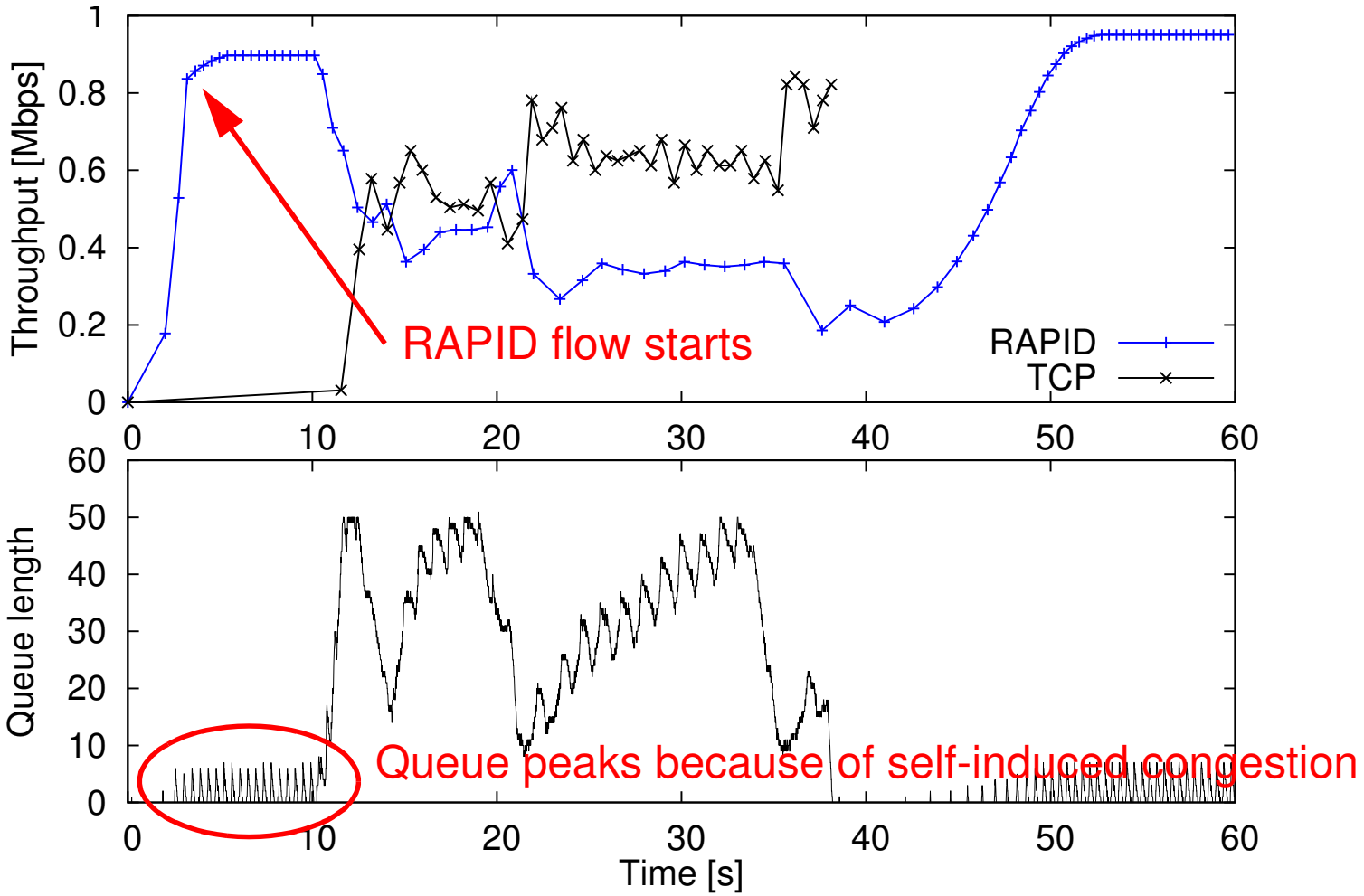
For convergence to equalized rate: $r_{avg} = r_{avg} + \frac{l}{\tau}(ABest - r_{avg})$ $l = \frac{N*P}{r_{avg}}$



Preliminary Results

RAPID Congestion Control

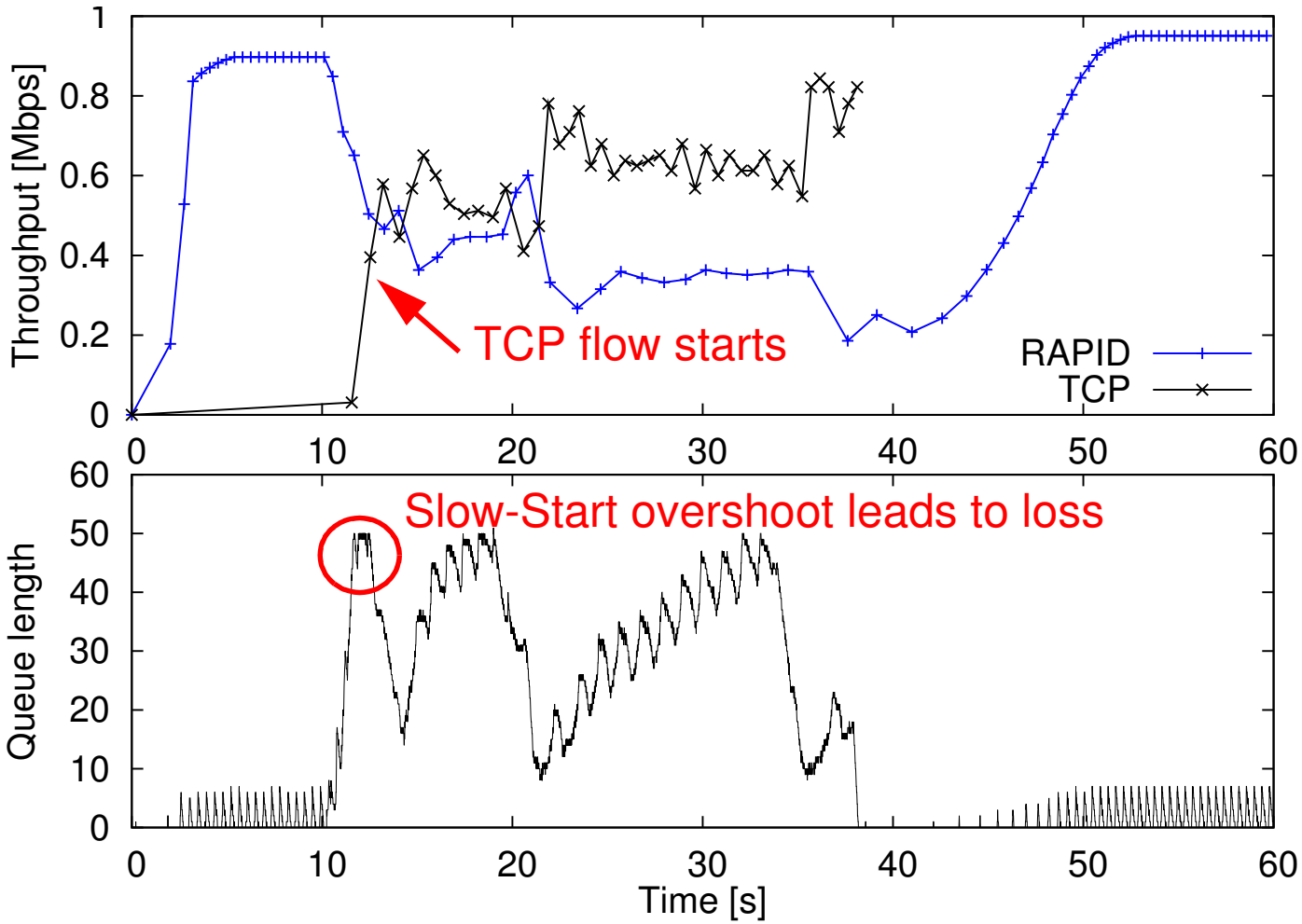
For convergence to equalized rate: $r_{avg} = r_{avg} + \frac{l}{\tau}(ABest - r_{avg})$ $l = \frac{N*P}{r_{avg}}$



Preliminary Results

RAPID Congestion Control

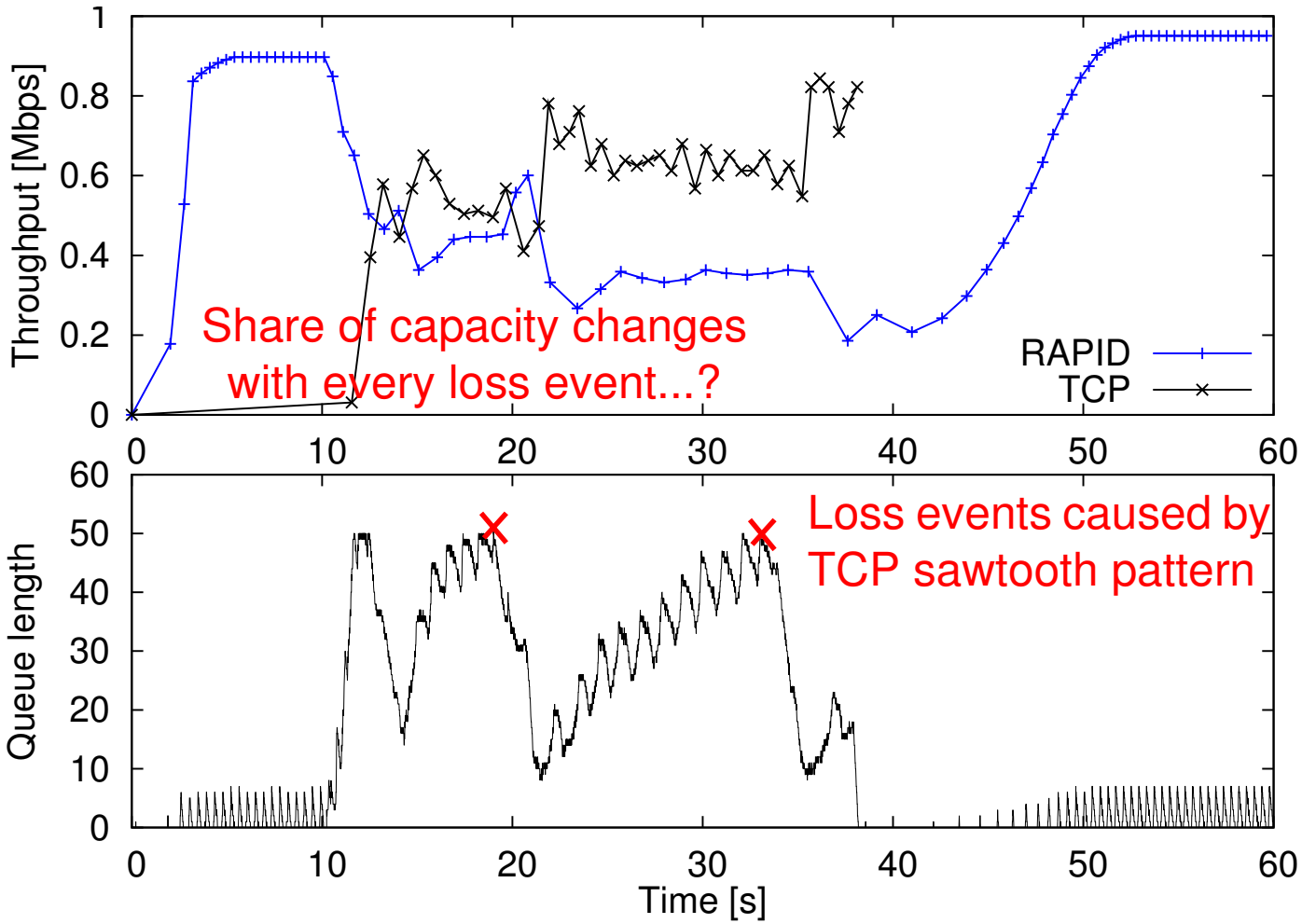
For convergence to equalized rate: $r_{avg} = r_{avg} + \frac{l}{\tau}(ABest - r_{avg})$ $l = \frac{N*P}{r_{avg}}$



Preliminary Results

RAPID Congestion Control

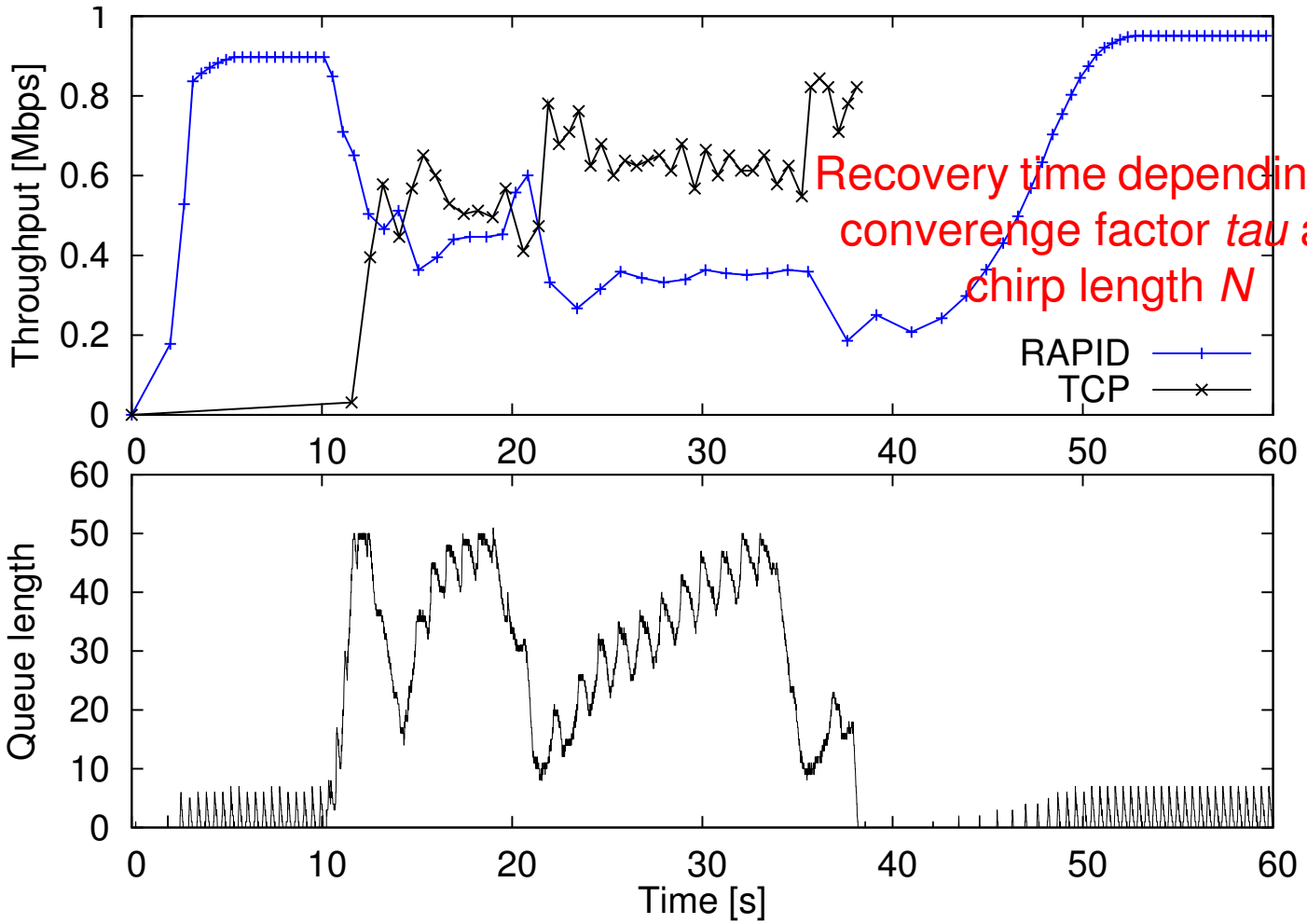
For convergence to equalized rate: $r_{avg} = r_{avg} + \frac{l}{\tau}(ABest - r_{avg})$ $l = \frac{N*P}{r_{avg}}$



Preliminary Results

RAPID Congestion Control

For convergence to equalized rate: $r_{avg} = r_{avg} + \frac{l}{\tau}(ABest - r_{avg})$ $l = \frac{N*P}{r_{avg}}$



Preliminary Results

Next Steps

Design of a robust congestion control based on chirping

- Adaption of chirping parameters to prevailing conditions
- Converenge in capacity sharing also when competing with other protocols
  - RAPID is scavenger protocol: Not designed to take capacity share from loss-based protocols
- Fast feedback chirping information only in addition to other network state information

Future Work

- Impact of short term probing delays on the queue burstiness
- Influence of a large aggregation of probing chirps on the base queue length

  → Reduced overshoot and respectively reduced maximum queue length
- Accuracy of measurement with a large aggregation of probing chirps
- Adaptation of the chirping parameters e.g. selection of chirp size *N*

  → Higher accuracy of chirping information

Conclusion and Outlook

- **Implementation** of chirping as a building block for congestion control in Linux
  - Nanosecond resolution provided by kernel hrtimers is sufficient for today's speed
  - Protocol design for feedback needed for one-way delay measurements
- **Structured the problem space** in three independent sub-problems

  rate estimation, rate adaption and adapting chirp parameters
- **Identified challenges** in implementation and deployability
  - Timer-based implementation does not use ACK-clocking and CWND
  - Negotiation about timestamp resolution and receiver behavior (delayed ACKs)
- **Invented solutions** to reduce implementation complexity and improve accuracy
  - Inter-packet gap calculation with linear decrease of inter-packets gaps
  - Hardware timestamping
  - Use of actual sent-out timestamps (instead of pre-set inter-packet gaps)

→ Show feasibilty of chirping within a contineous data stream!

→ Use faster feedback to enable **more scalable rate adaption with minimal overshoot**!

→ Encourge others to build research on rate estimation, rate adaption and adaption of chirping parameters!

Chirping for Congestion Control

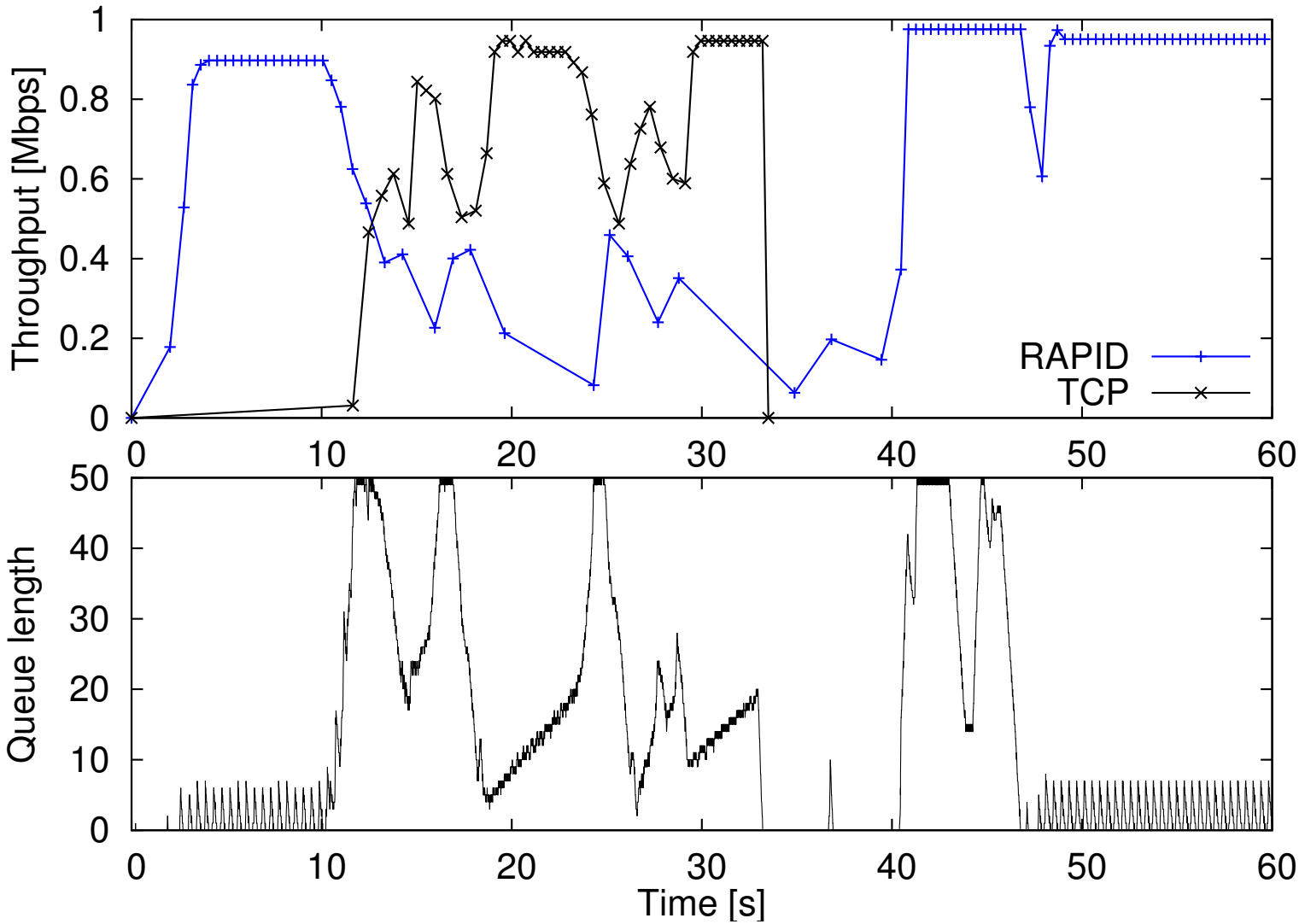
Thank you for your attention!

Questions?

Preliminary Results

RAPID Congestion Control

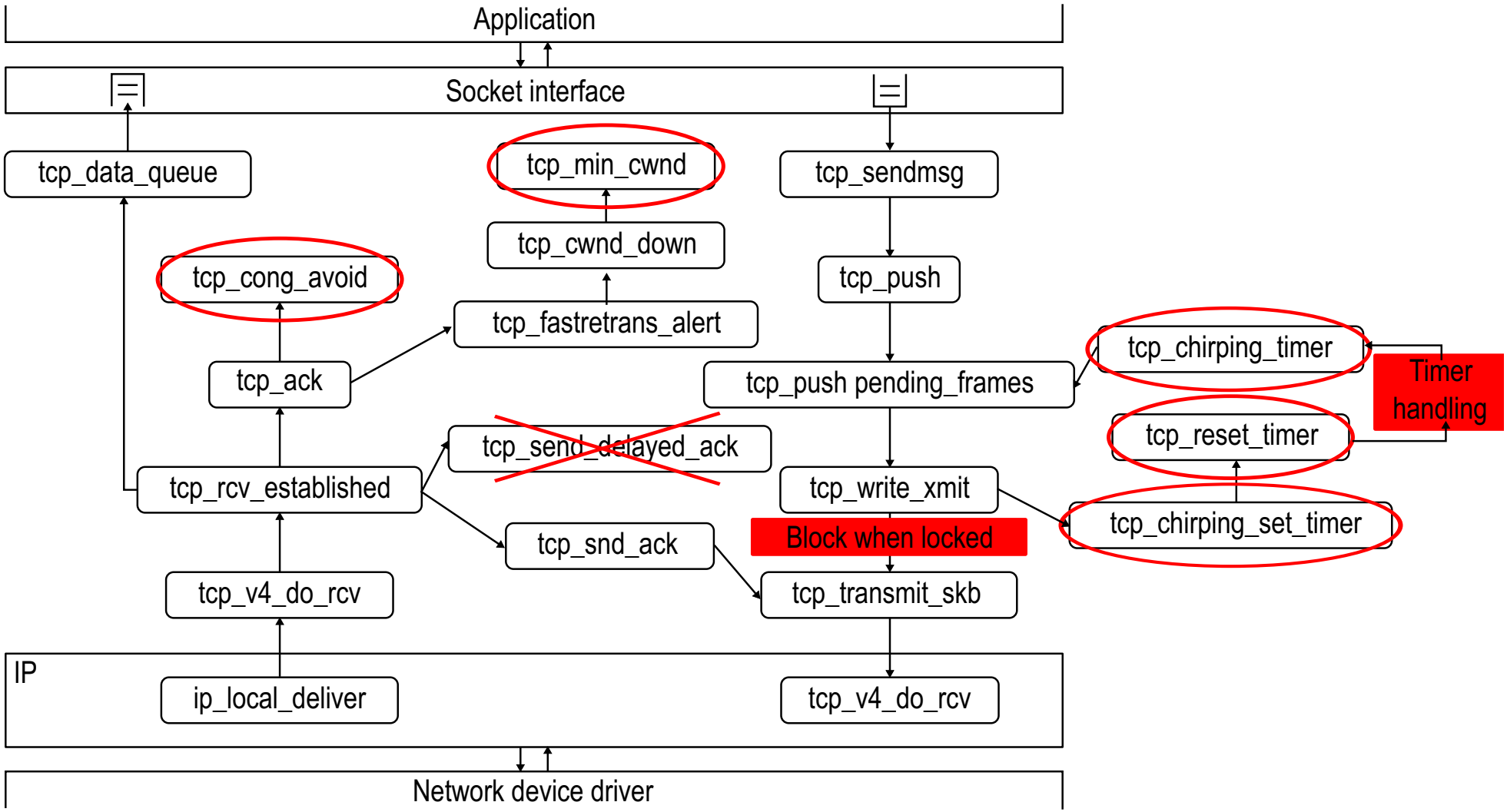
TCP cross traffic starts at 10s (20Mbit) on 1Mbit/s bottleneck link



Chirping Implementation in the Linux Kernel

Implementation Details

→ Extended congestion control kernel module interface and TCP timer for send-out timing



Chirping Implementation in the Linux Kernel

Algorithm for Inter-packet gap Calculation

- Fully based on inter-packet time gaps instead of rate
- N should be an the integer power of 2

  → Initiallly hard-coded to N = 32 (=2^5)
- Harmonic progression of rates

  → Linear decrease of inter-packet gaps: $gap_i = gap_{i-1} - gap_{step}$ with $gap_{step} = (2 * gap_{avg}) / N$
- Implementation with integer arithmetic

$$gap_i = gap_{step} * (N - i + 1) = (2 * gap_{avg}) / N * (N - i + 1) \qquad \text{with } i = 1...N-1;\ gap_0 = gap_{avg}$$

  - Probing range:1/2 r_{avg} to N/4 r_{avg}
  - Maximum rates of harmonic progression not used

→ Slightly lower average rate than the estimated one