This work is partly funded by the German Research Foundation (DFG) through the Center of Excellence (SFB) 627 "Nexus".

Quick-Start, Jump-Start, and Other Fast Startup Approaches

Implementation Issues and Performance

Michael Scharf
michael.scharf@ikr.uni-stuttgart.de
November 17, 2008

Universität Stuttgart
Institute of Communication Networks
and Computer Engineering (IKR)
Prof. Dr.-Ing. Dr. h.c. mult. P. J. Kühn

Outline

- Flow Startup Basics
- Fast Startup Mechanisms
- Implementation Issues
- Performance Experiments
- Conclusions and Future Work

Flow Startup Basics – Introduction

The Flow Startup Challenge

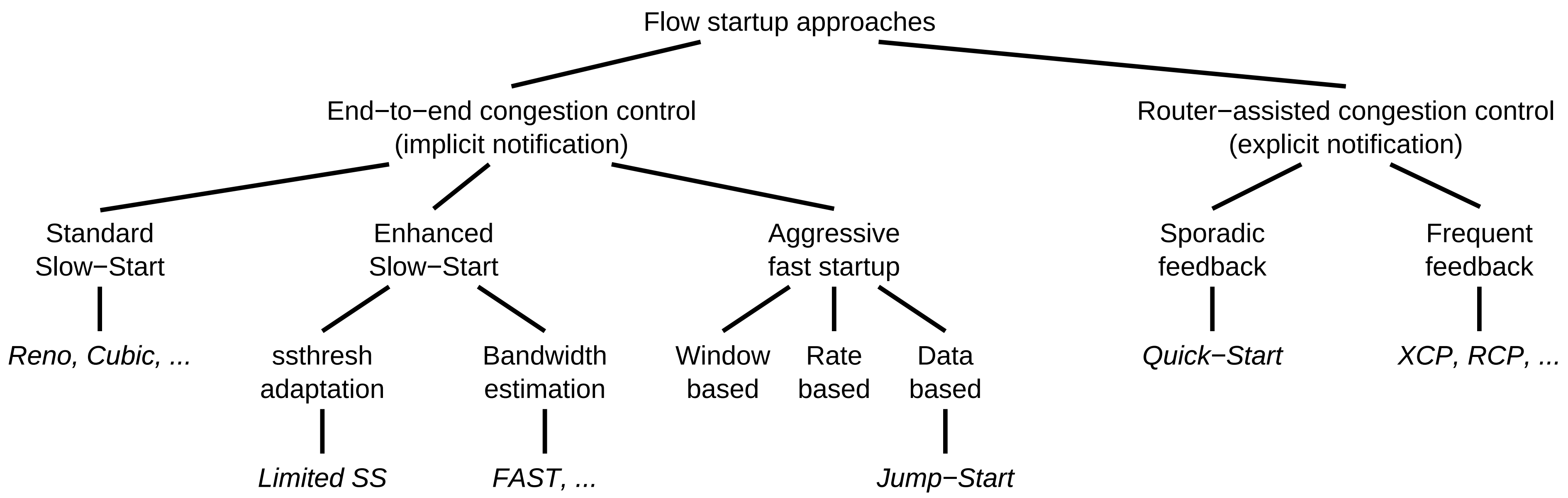
- Default TCP flow startup mechanism since 1988: Slow-Start
- Two important functions
  - Probe the network to find reasonable values for cwnd and ssthresh
  - Initialize the ACK clock
    Doubling cwnd on each arriving ACK is simple and effective
- Slow-Start not perfect for interactive applications

  ... if the bandwidth-delay product is large

  ... for mid-sized data transfers
- Question: Can we do better? Why can't we immediately fully use a path?
  - If it was a trivial problem, it would already have been solved
  - Problem becomes more pressing as bandwidth-delay-products increase
  - Disclaimer: This talk will not answer this question. It only explores the solution space.

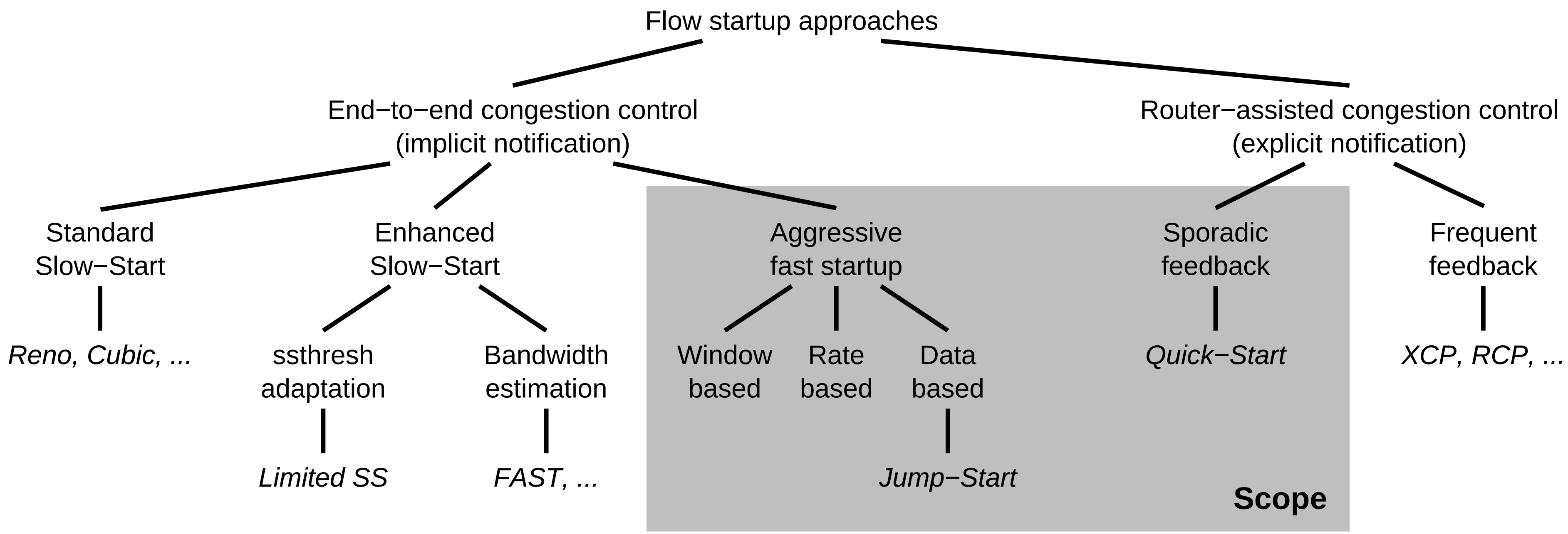
Flow Startup Basics – Some Thoughts

Design Space



Flow Startup Basics – Some Thoughts

Design Space



Further Alternatives

- Middleboxes such as WAN optimizers or transparent HTTP proxies

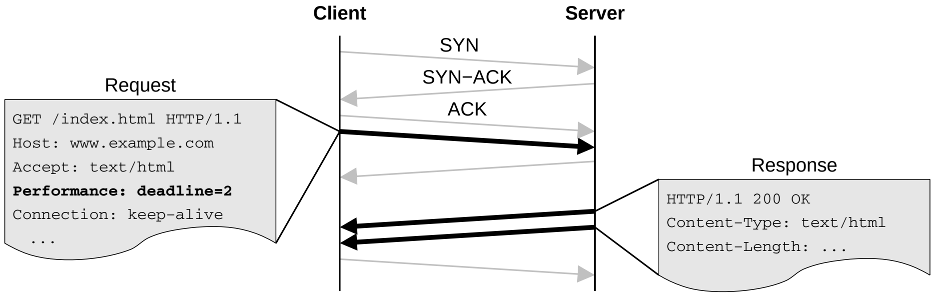
  → Break end-to-end semantics
- Parallel usage of several/many TCP connections

  → Fairness issue and risk of over-aggressiveness

Flow Startup Basics – Some Thoughts

Potential Input Parameters

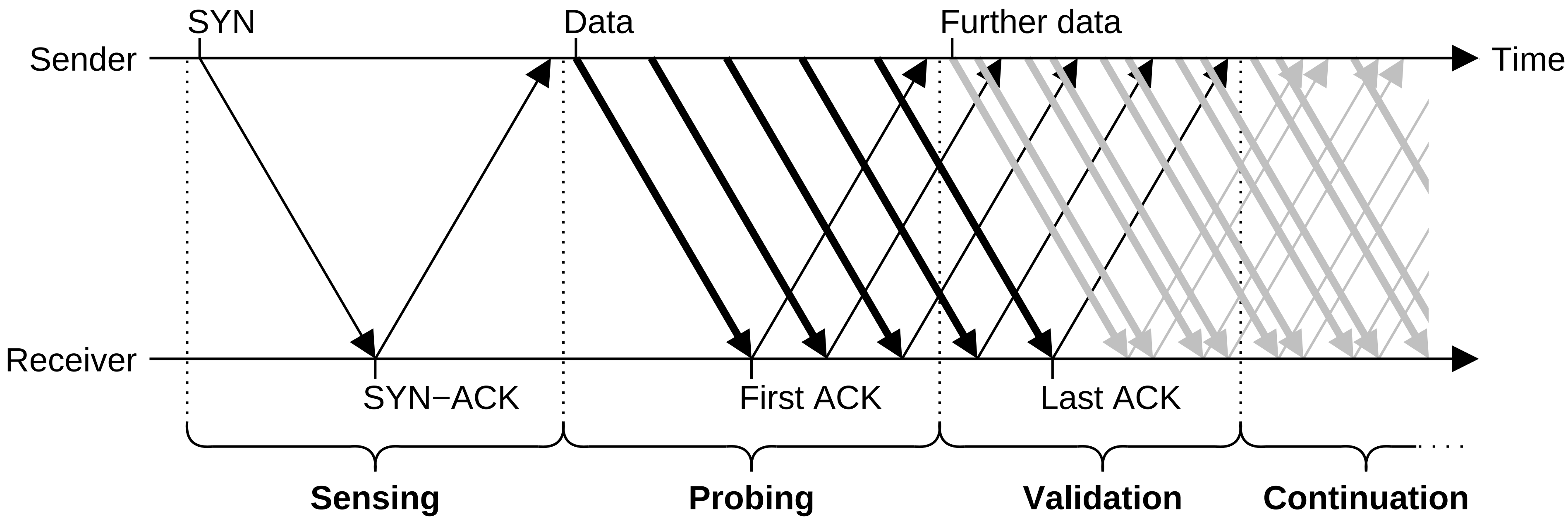
- Round-Trip Time (RTT), known from 3-way handshake
- Cached state variables for this destination
  - Intra-connection ... Slow-Start after idle
  - Inter-connection ... Congestion manager
- Application communication characteristics (e. g., amount of queued data)
- Local interface capacity
  - Automatically detected from network interface
  - Manually configured by administrator/user
- Application requirements (bandwidth, response deadline, ...)
  - Implicitly derived by heuristics inside the network stack (e. g., port 80/http)
  - Explicitly signaled by app interface (e. g., similar to NO_DELAY socket option)



- "Oracle" service that provides rough estimate of end-to-end capacity (ALTO++)
- Explicit router feedback of currently available bandwidth on the path

Flow Startup Basics – Some Thoughts

Fundamental Fast Startup Phases

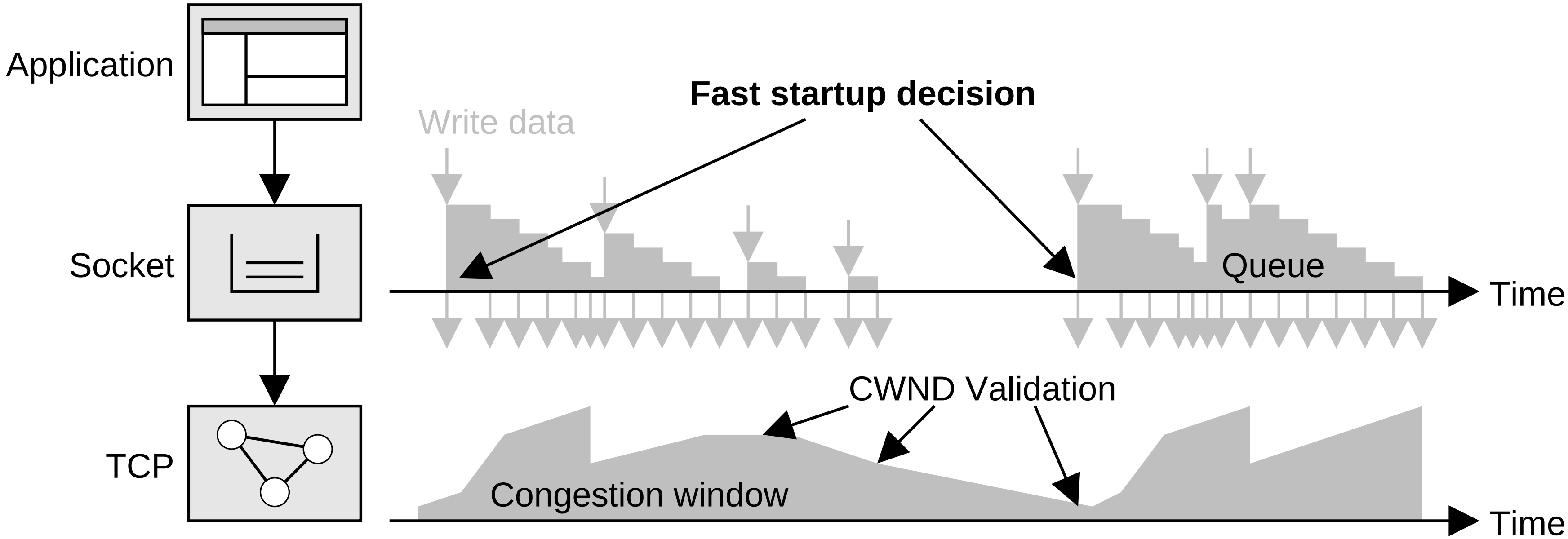


- **Sensing**: Derive basic path characteristics
- **Probing**: Start to send data
- **Validation**: Test whether the initial choice was reasonable
- **Continuation**: Switch to continuous congestion control

→ Several different options how to realize these phases

Flow Startup Basics – Some Thoughts

Slow-Start is Not Only Occurring After Connection Setup...



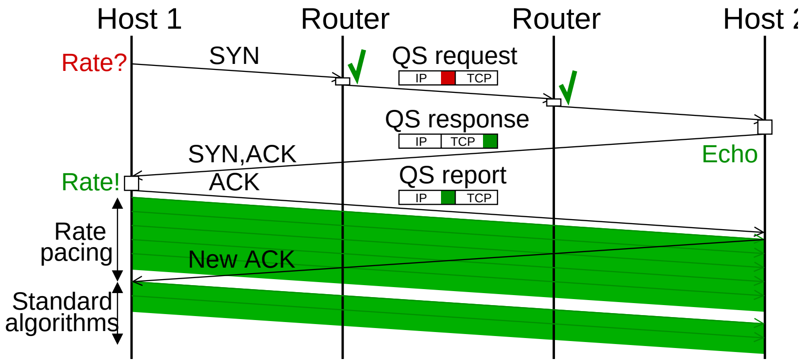
- Congestion window validation (RFC 2861) reduces cwnd during application-limited periods or after longer idle times
- Fast startup also after idle times possible
- Typically, more information about path characteristics available ... But is it still valid?

→ Several further degrees of freedom whether to repeat a fast startup

Fast Startup Mechanisms – Quick-Start

Quick-Start TCP Extension Overview

- Specification in RFC 4782 (experimental)
- **Sensing**: Explicit router feedback to determine the available bandwidth on the path
  - Request for a data rate in a new IP option
  - All routers have to approve the request
  - Resulting rate returned in a new TCP option



- **Probing**: Rate paced transfer with approved rate
- **Verification**: Undo of cwnd increase in case of packet loss of paced packets
- **Continuation**: Default TCP congestion control
- Open questions: Adapt ssthesth after verification? Router admission control strategy?

→ "Ask before you shoot"

Fast Startup Mechanisms – Jump-Start

Jump-Start Overview

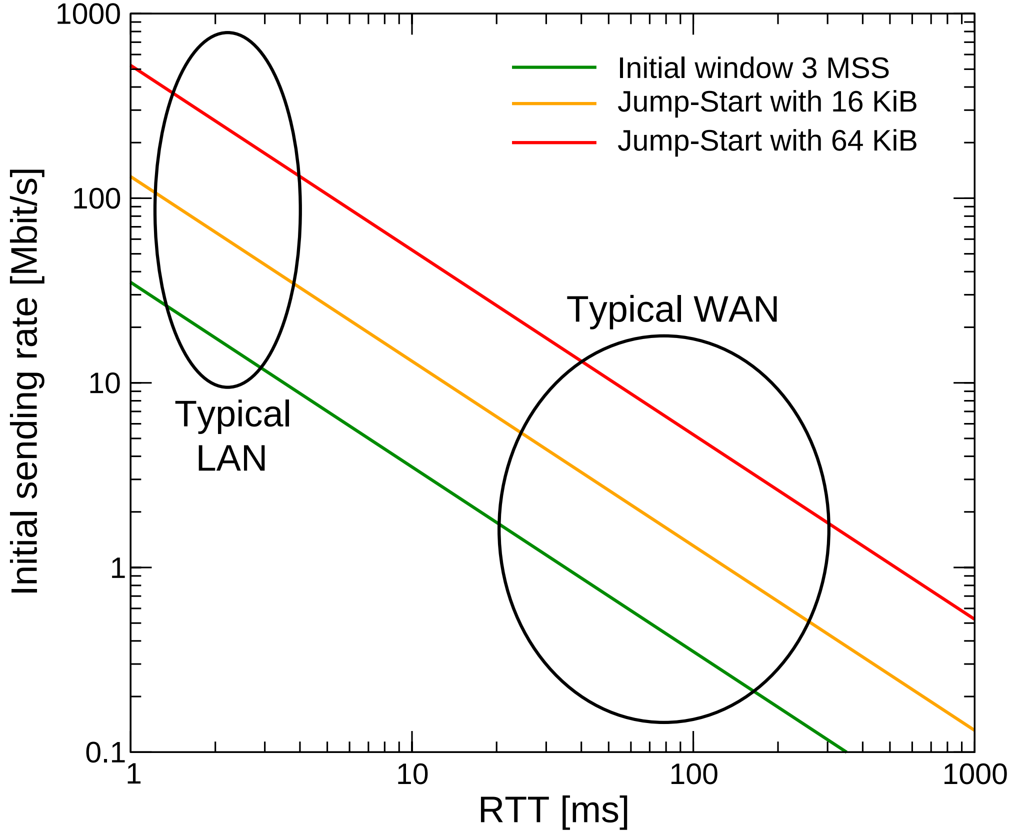
- Proposed by D. Liu, M. Allman, S. Jin and L. Wang at PFLDNET 2007
  - Basic idea: Play out the data queued in the socket paced over the first RTT
  - Risk over-shooting, but carefully reduce window if packet loss has occurred
- **Sensing**: Default TCP, just estimate RTT
- **Probing**: Send queued data using rate pacing
  - Initial data rate depends on available data, RTT, and receiver window
  - Thresholds possible, i. e., send at most 64 KiB
- **Verification**: Modified TCP loss recovery
  - Normal TCP retransmission mechanism, including cwnd halving
  - Count number R of retransmitted packets
  - If R=0: Continue with default TCP congestion control
  - If R>0: Set cwnd = (D-R)/2 at end of loss recovery, where D is the number packets that have been paced over the first RTT
- **Continuation**: Default TCP congestion control

→ "Shoot before you ask"

Fast Startup Mechanisms – Jump-Start

Open Question: Can This Work?

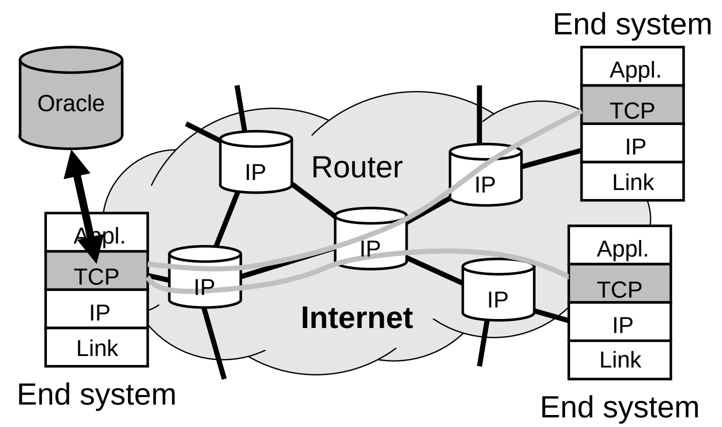
- Heavy-tailed flow sizes: Only few connections use a large initial rate
- The longer the path, the smaller the initial rate (if a maximum threshold is used)



- The existing Slow-Start may also significantly overshoot, without causing too much pain

Fast Startup Mechanisms – Further Alternatives

"More-Start"



- Idea: Quick-Start without explicit router support
- **Sensing**: Choose an initial sending rate
  - e. g. explicitly selected by application

```
u_int rate = 10000000; /* 10 Mbit/s */
setsockopt(fd, SOL_TCP, 15, &rate, sizeof(u_int));
```

  - e. g. from congestion manager, local interface speed
  - e. g. "oracle" service for remote peers (ALTO++)
- **Probing**, **validation**, **continuation**: Similar to Jump-Start

→ "Ask *someone* before you shoot"

"Initial-Start"

- Just increase the initial value of cwnd to a value larger than RFC 3390
- No change in TCP error recovery mechanisms
- Initial cwnd could be statically set, or dynamically be obtained like in the previous approaches

→ "Keep it simple"

Implementation Issues – Quick-Start

Implementation of University of Stuttgart

- Quick-Start TCP and IP functions in Linux 2.6.24 kernel[1]
- Quick-Start IP functions in an IXP 2400 network processor[2]

Lessons Learned

- Quick-Start processing feasible at high link speeds
- Limited implementation complexity
- A couple of challenges
  - Setting of IP options from TCP layer
    - TCP MSS must be reduced to leave space for IP option
    - Interactions with TCP segmentation offload (TSO)
  - Multiple parallel requests, SYN cookies
  - Automatic determination of link capacity

[1] M. Scharf and H. Strotbek, "Performance evaluation of Quick-Start TCP with a Linux kernel implementation," Proc. IFIP Networking 2008, Springer LNCS 4982, pp. 703-714, May 2008
[2] S. Hauger, M. Scharf, J. Kögel, and C. Suriyajan, "Quick-Start and XCP on a network processor: Implementation issues and performance evaluation," Proc. IEEE HPSR, May 2008

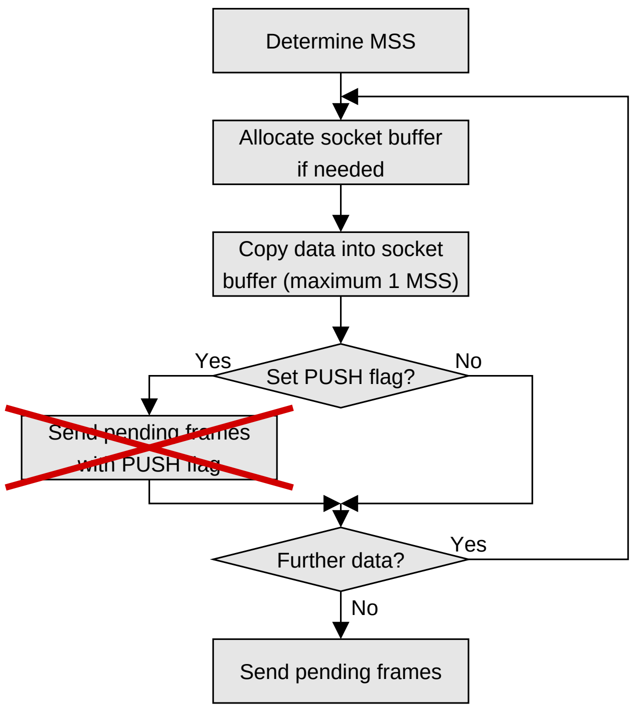
Implementation Issues – Jump-Start

Implementation of University of Stuttgart

- New Jump-Start implementation in Linux 2.6.24 kernel
- Work in progress, not completely validated so far

Lessons Learned

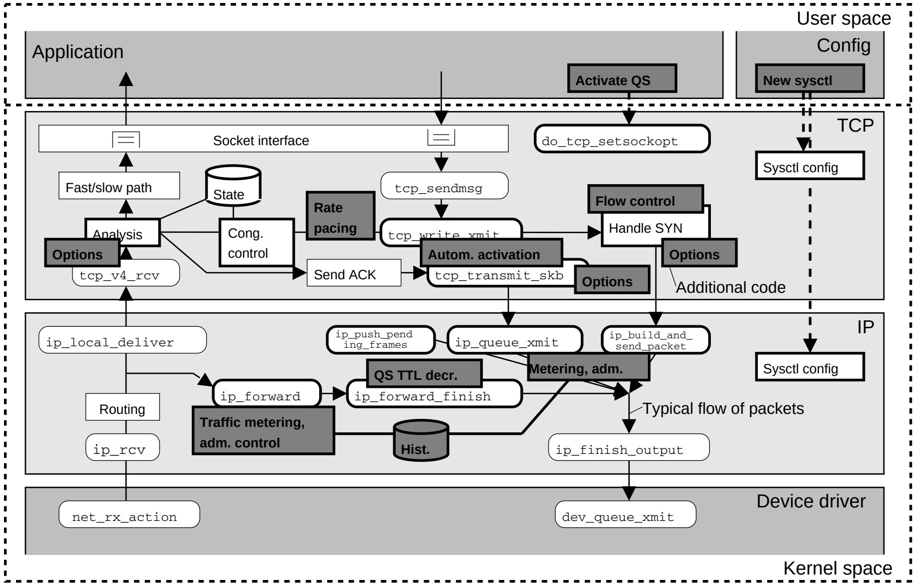
- Of course, Jump-Start is doable
- Needs a state engine and timers for rate-pacing
- How to determine the queued data in socket?
  - Easy: `sk->sk_write_queue.qlen`
  - However, socket processing workflow must be adapted
- Modified TCP error recovery
  - Easy: Additional counter for retransmissions
  - However: `cwnd=max(1,(D-R)/2)`
- Problem with flow control, similar to Quick-Start[1]



[1] Michael Scharf, Sally Floyd, Pasi Sarolathi: "TCP Flow Control for Fast Startup Schemes", July 2008, draft-scharf-tcpm-flow-control-quick-start-00.txt

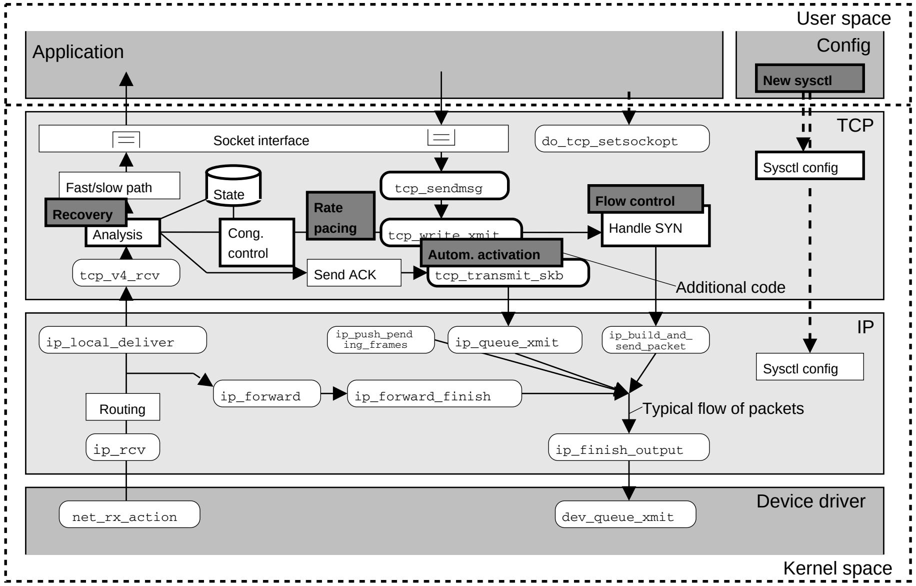
Implementation Issues – Complexity Comparison

Quick-Start in Linux Kernel



→ ca. 2000 LOC

Jump-Start in Linux Kernel

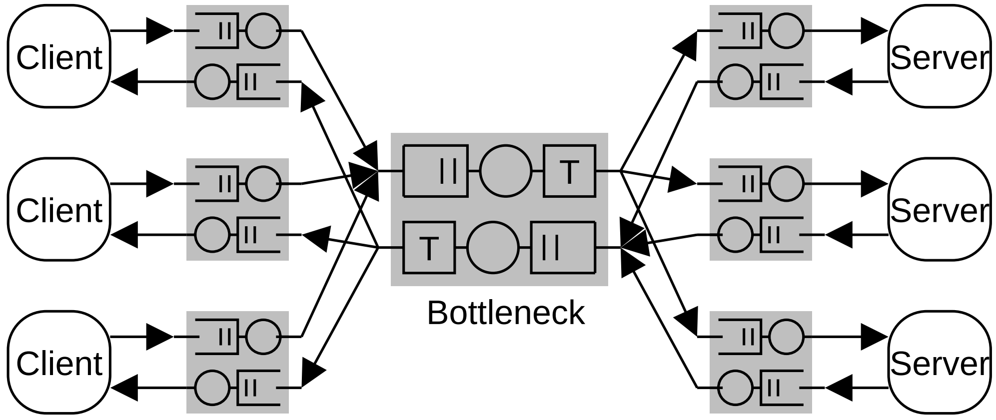


→ ca. 600 LOC

Performance Experiments

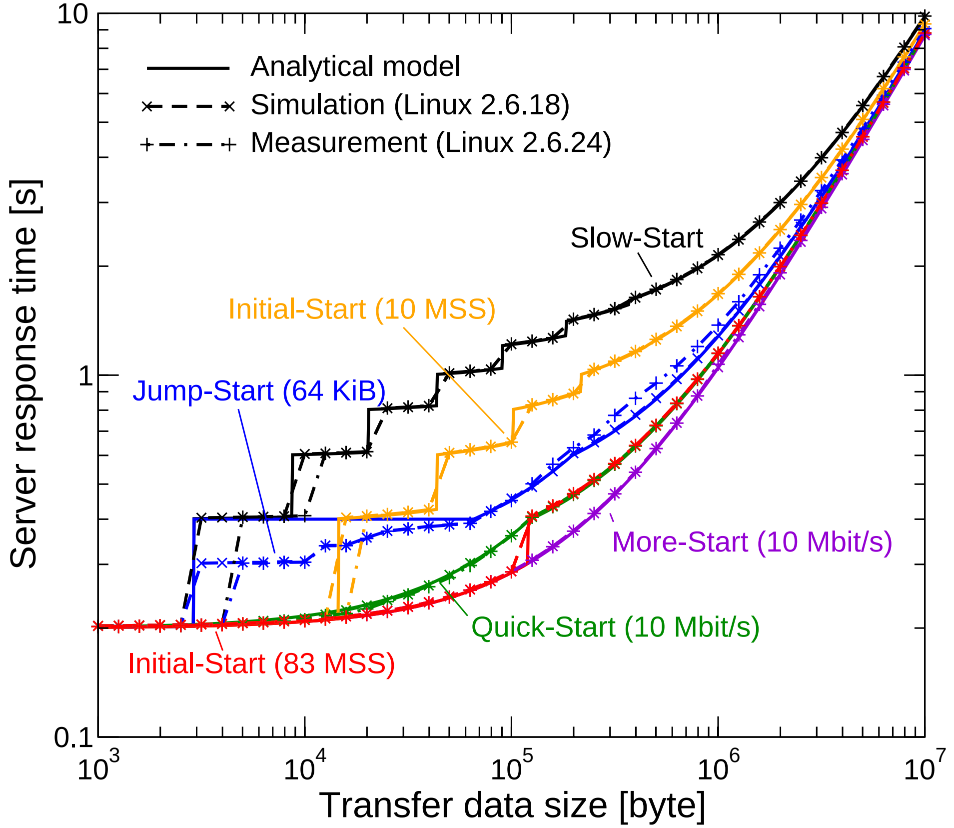
Methodology

- Lab measurements
  - Patched Linux 2.6.24 kernels
  - Two or more PCs, directly connected by Ethernet segments, interface speed manually set
  - Delay emulation by "netem"
- Simulations
  - Simulation of patched Linux 2.6.18 kernel network stacks using the "Network Simulation Cradle" (NSC) version 0.3.0 (Sam Jansen, University of Waikato)
  - Different client-server application workloads
  - Classical dumbbell topology with finite buffer in front of central bottleneck link



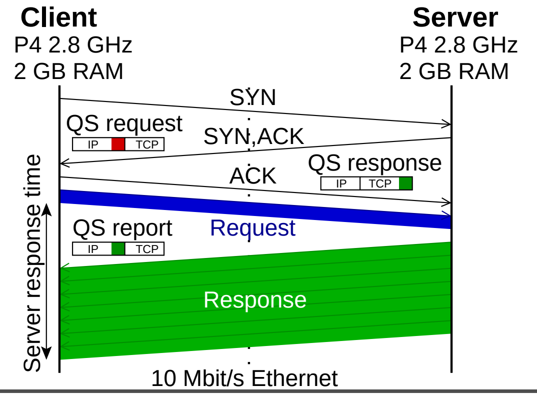
Performance Experiments – Speedup

Illustration of Different Fast Startups (10 Mbit/s, 200 ms RTT)



Setup

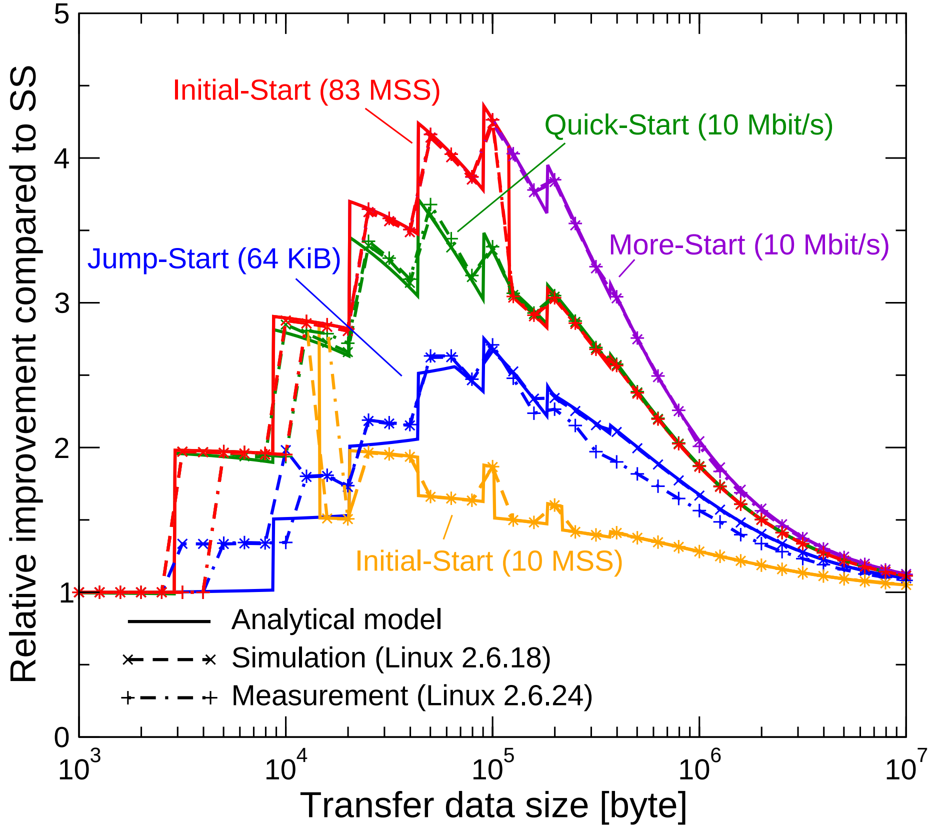
- 2 PCs with a Ethernet link
- Simple C programs for client and server
- Kernel 2.6.24, Ubuntu 7.04, default config.
  - "Cubic" congestion control
  - SACKs enabled
- Additional delay by "netem"
- Quick-Start request by server in SYN,ACK



- As to be expected, all new schemes are faster than Slow-Start
- Somewhat different behavior

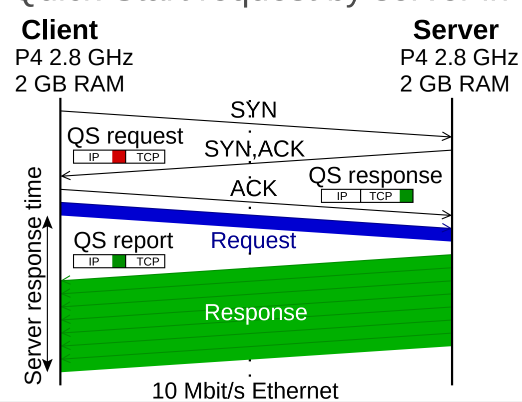
Performance Experiments – Speedup

Speedup Compared to Slow-Start (10 Mbit/s, 200 ms RTT)



Setup

- 2 PCs with a Ethernet link
- Simple C programs for client and server
- Kernel 2.6.24, Ubuntu 7.04, default config.
  - "Cubic" congestion control
  - SACKs enabled
- Additional delay by "netem"
- Quick-Start request by server in SYN,ACK

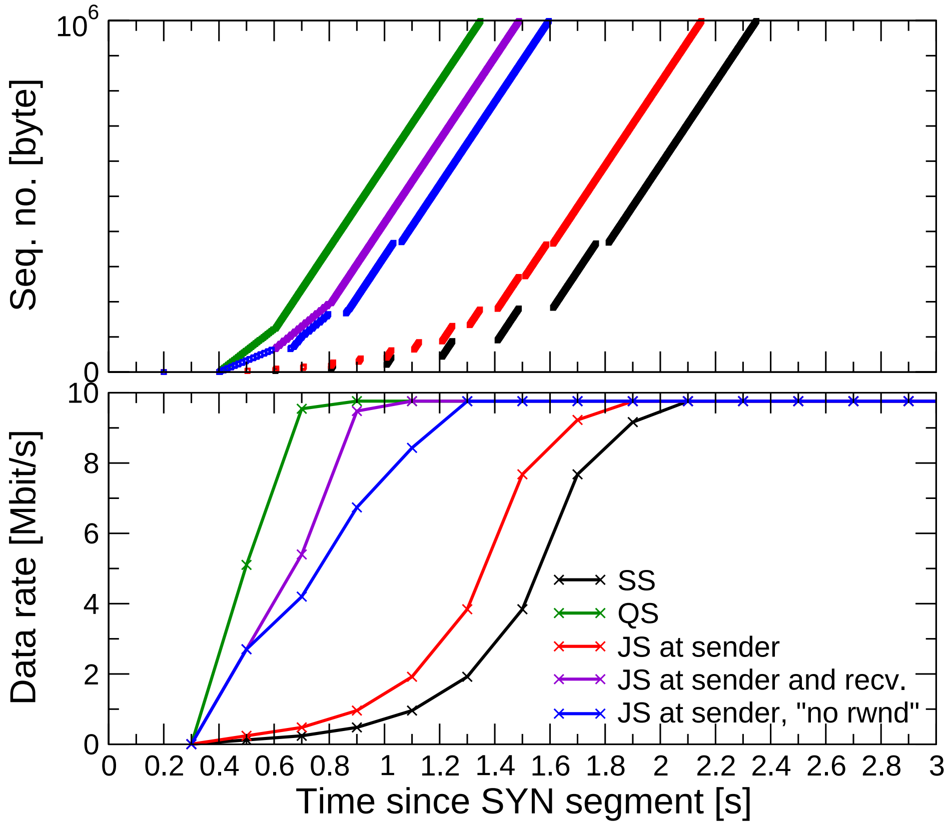


- Significant benefit for mid-sized transfers from 10KB to 1 MB
- Measurement and simulations match analytical models[1]

[1] Michael Scharf, "Performance Analysis of the Quick-Start TCP Extension", Proc. IEEE Broadnets, Raleigh, NC, USA, Sept. 2007

Performance Experiments – Flow Control Issue

The Impact of Linux Buffer Autotuning (10 Mbit/s, 200 ms RTT)



Setup

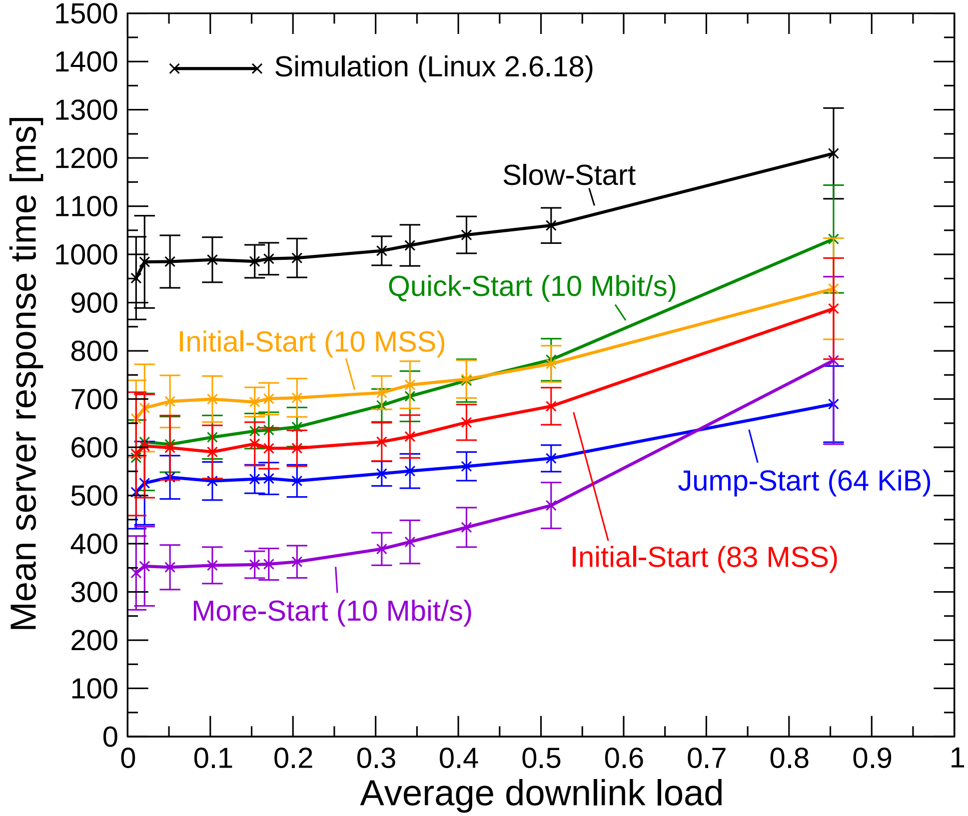
- Simulation with Linux kernel 2.6.18
  - "Cubic" congestion control
  - SACKs enabled
- 1 client, 1 server
- Simple client-server request
- Central buffer: 1000 packets
- Quick-Start request by server in SYN,ACK

- Case 1: Sender modification only: Slow-Start is enforced by flow-control
- Case 2: Receiver that announces large window[1]: Fast startup is possible
- Case 3: Sender selectively ignores rwnd during probing: Works, but is this a good idea?

[1] Michael Scharf, Sally Floyd, Pasi Sarolathi: "TCP Flow Control for Fast Startup Schemes", July 2008, draft-scharf-tcpm-flow-control-quick-start-00.txt

Performance Experiments – Initial Comparison

Shared Bottleneck Scenario (10 Mbit/s, 200 ms RTT)



Setup

- Simulation with Linux kernel 2.6.18
  - "Cubic" congestion control
  - SACKs enabled
- 25 clients, 25 server
- Perstent TCP connections
- Client-server application model
  - Response size Pareto distributed, mean 250kB, shape factor 1.1
  - Neg.-exp. distr. inter-arrival time
- Central buffer: 50 packets
- Quick-Start request by server in SYN,ACK

- Quick-Start: Close to Slow-Start as load increased, because of admission control
- Jump-Start: Reasonable behavior
- More-Start: Effects of over-shooting observable for higher load
- Initial-Start: Setting an initial window of the order of the BDP is critical

Performance Experiments – Further Results

Observations

- Jump-Start is simple and behaves quite well in most experiments so far
  ... but, of course, it can significantly fail as well
- Naively activating Quick-Starts for all small transfers does not improve performance if admission control is used

  → Either explicitly activated by application, or only, if a larger transfer can be expected
- If we had a rough estimate for the available bandwidth, just starting with this rate might not be that harmful
- Benefit of rate pacing vs. just increasing cwnd?
  - Rate pacing seems to be less harmful to competing traffic
  - Not too much difference in case of significant overshoot
- Small total speedup in more complex scenarios (e. g., draft-irtf-tmrg-tests-00.txt)
  - Most RTTs and transfer sizes are small
  - Average improvement for mid-sized transfers less than 1 second
- ...
- But: Not completely backed by data so far

Conclusions and Future Work

Conclusions

- Any fast startup is tricky, and there is no guarantee to be better than Slow-Start
- Router support (Quick-Start TCP) could help
  - But: Significant deployment issues
  - Even with router support an intelligent usage is needed
- End-to-end solution could use further parameters that are locally available
  - Jump-Start is simple and has interesting properties
  - But design space is not completely explored so far
- Ongoing implementation efforts to get fast startup schemes into the Linux kernel
- Early experiments show that speedup is indeed possible

Future Work

- Experiments, experiments, experiments
- New cross-layer interfaces, e. g., between applications and network stack?