UiO : Faculty of Mathematics and Natural Sciences
University of Oslo

Department of Informatics
Networks and Distributed Systems (ND) group

How to Control a TCP: Minimally-Invasive Congestion Management for Datacenters

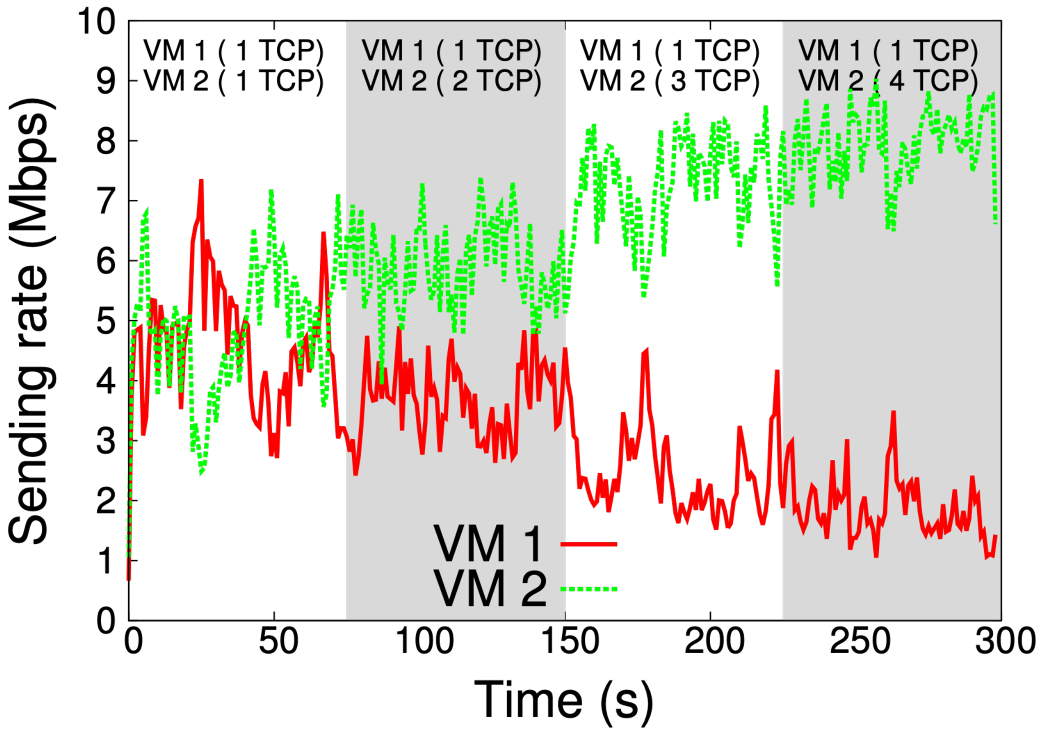
Safiqul Islam, *Michael Welzl, Stein Gjessing*

ICNC 2019, Honolulu
18. 02. 2019

What is this about?

- In multitenant datacenter, the guest OSes of clients may be diverse and utilize an Internet-like mix of old and new TCP congestion control implementations
- This may put some users at a disadvantage, depending on how aggressively their congestion control probes for capacity
  - unfair users may have an incentive to obtain a larger share of the capacity by opening multiple TCP connections →.unsatisfied customers!

What is this about?



Sending rate of two VMs, with 1 flow in VM1 and 1 to 4 flows in VM2

Prior work

- Mechanisms such as Seawall [1], VCC [3] and AC/DC [2] successfully achieve this sender-side control by running dedicated congestion control algorithms as part of the hypervisor infrastructure
  - **But:** how should the new algorithm that is running as part of the hypervisor communicate with the the guest OS?

- Seawall alone takes care of the congestion control
  - CC implementations need to defer all congestion control decisions to the hypervisor (asking for allowance before sending a packet)_
  - the sender and receiver side are altered, and bits from the header are re-purposed to implement the necessary signaling

[1] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proc. of NSDI*, 2011.
[2] K. He, E. Rozner, K. Agarwal, Y. J. Gu, W. Felter, J. Carter, and A. Akella, "AC/DC TCP: Virtual congestion control enforcement for datacenter networks," in *Proc. of SIGCOMM*, 2016
[3] B. Cronkite-Ratcliff, A. Bergman, S. Vargaftik, M. Ravi, N. McKeown, I. Abraham, and I. Keslassy, "Virtualized congestion control," in *Proc. of the ACM SIGCOMM*, 2016.

Prior work

- AC/DC do not require updating the guest OS at all,
  - which is a significant advantage: it does not require cooperation of tenants to update the OS (if they do bring their own OS),
    - which reduces burden and allows to *enforce* cooperative behavior

- Changes the receive window (rwnd) as a means to control TCP's behavior
  - A sender can therefore only increase the sending rate as quickly as the TCP implementation inside the guest OS allows

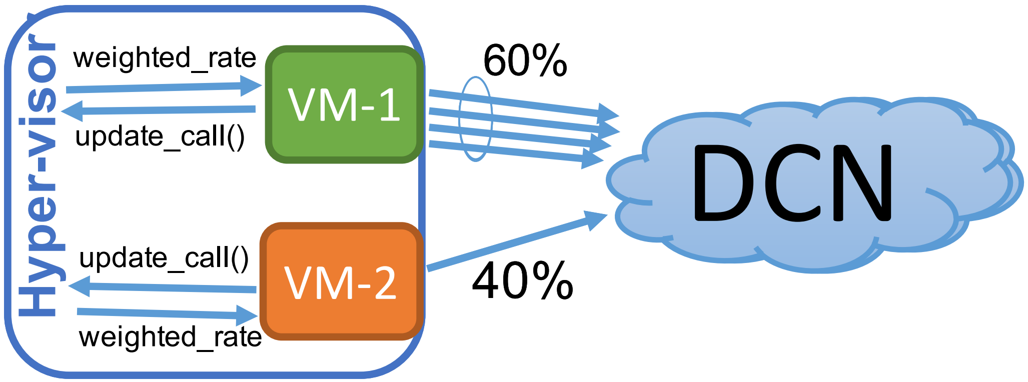
Prior work on congestion management

- Datacenter capacity management
  - Access is controlled at the edges (EyeQ), FairCloud (per flow queues at the switches), Seawall, AC/DC, VCC
- Single-path congestion control coupling
  - By sharing a number of state variables
- Multiplexing
  - By merging application layer datastreams onto a single transport layer connection
- Multi-path congestion control coupling
  - MPTCP's coupling assumes that flows could take a different path, and ideally also traverse different bottlenecks

Our contribution

- A new interface (*ctrlTCP_int*) to communicate between TCP in the guest OS and a hypervisor.
  - A set of TCP connections are controlled via this interface
- Extended our *ctrlTCP* algorithm that emulates the behavior of a single TCP congestion controller
  - Supports prioritization (for practical management of both inter- and intra-VM capacity allocation)
- Show the efficacy of our solution using both ns2 and FreeBSD

CtrlTCP interface



- A middle ground can be found by keeping the guest OS congestion control intact, yet allowing a controlling entity to overrule its decisions

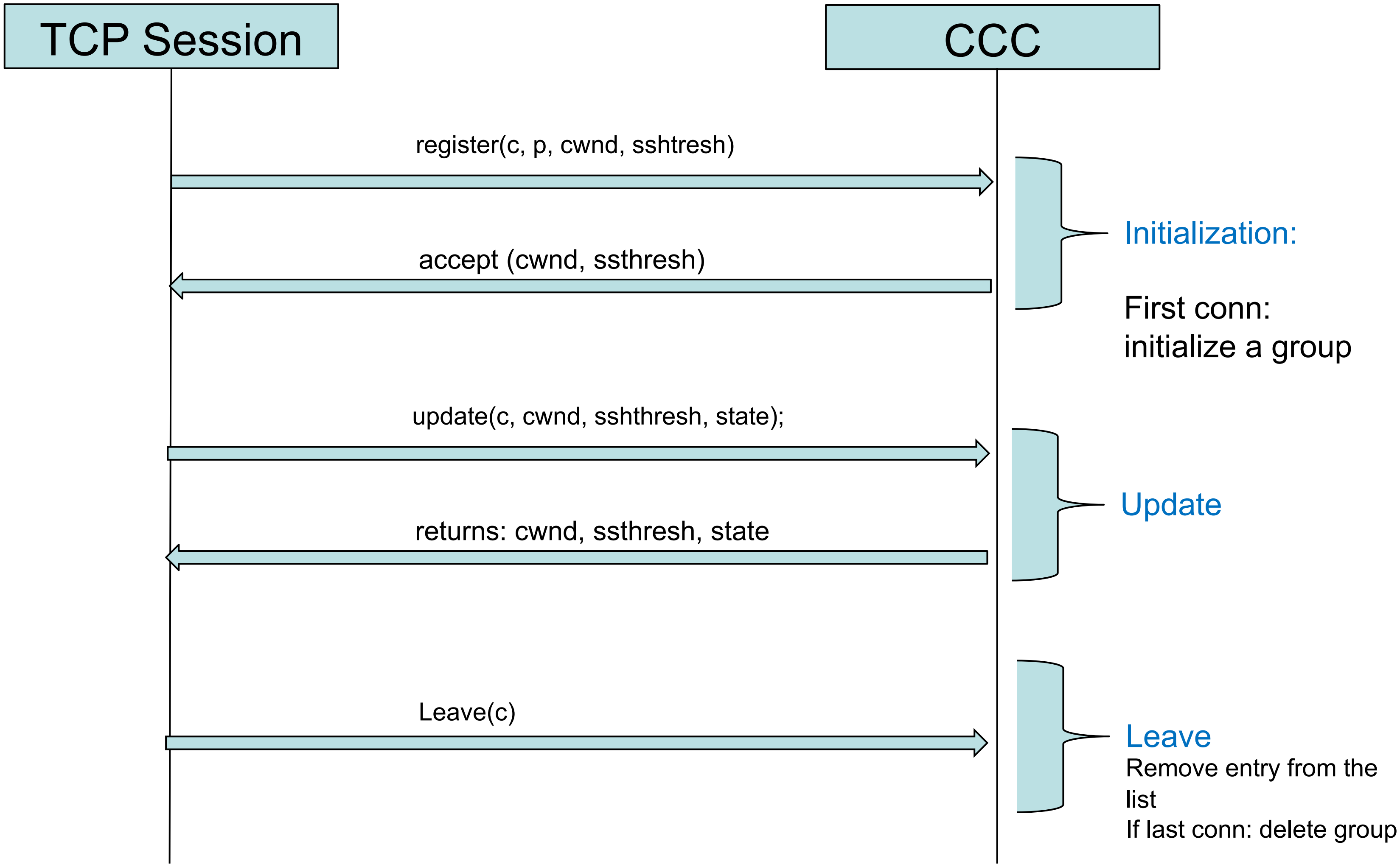
- *ctrlTCP* operates strictly on the *control path*:
  - communicates signals cc in the guest OS and the hypervisor
    - not needed to even examine or count the outgoing or incoming packets

CtrlTCP algorithm

- Each TCP session communicates with an entity that we call a Coupled Congestion Controller (CCC)
  - typically makes decisions that combine the collected knowledge that it receives from all TCP instances that talk to it - thereby “coupling” them in some way
  - A CCC can operate in a hypervisor or in an OS

S. Islam, M. Welzl, H. Kristian, D. Hayes, G. Armitage, and S. Gjessing, “ctrlTCP, reducing latency through coupled heterogeneous multi-flow TCP congestion control,” IEEE GI 2018

ctrlTCP


TCP Session
CCC
register(c, p, cwnd, sshtresh)
accept (cwnd, ssthresh)
Initialization:
First conn:
initialize a group
update(c, cwnd, sshthresh, state);
returns: cwnd, ssthresh, state
Update
Leave(c)
Leave
Remove entry from the
list
If last conn: delete group

Changes in the TCP code

register(c, p, cwnd, sshtresh);
returns: cwnd, ssthresh, state

← **Initialization**: executed at the beginning of the session!

update(c, cwnd, sshthresh, state);
returns: cwnd, ssthresh, state

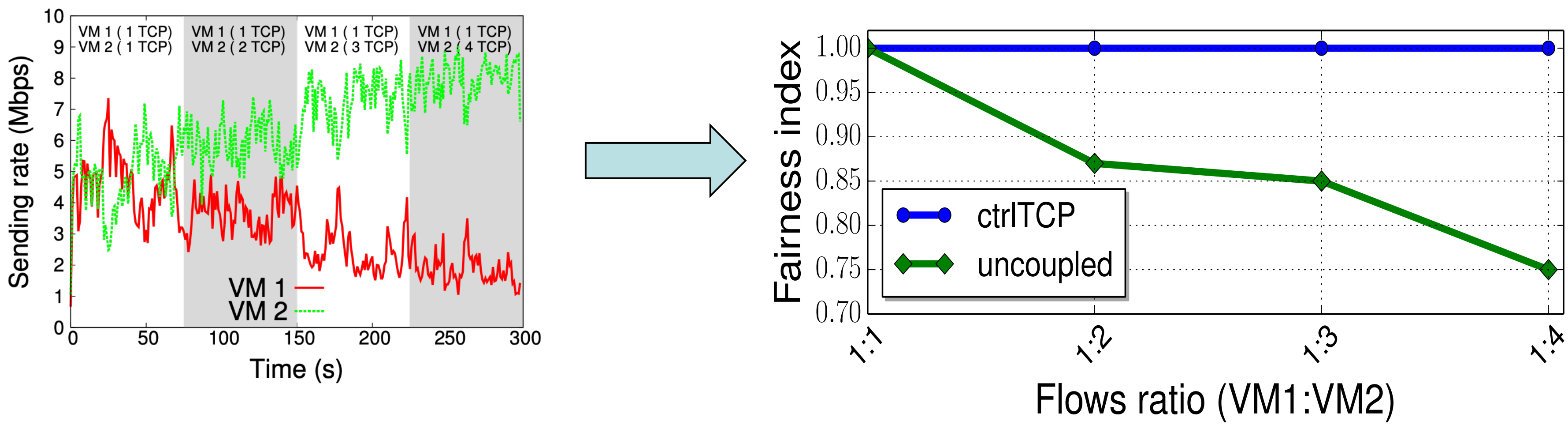
← **Update**: executed whenever a TCP session newly calculates its cwnd

leave(c)

← **Leave**: executed whenever a TCP session is terminated

Evaluation

- Implementation
    - FreeBSD 11 kernel with state shared across the freely available VirtualBox hypervisor
    - ns-2 simulator



Fairness between two VMs, with 1 flow in VM1 and 1 to 4 flows in VM2 across a 10Mbit/s→100ms bottleneck

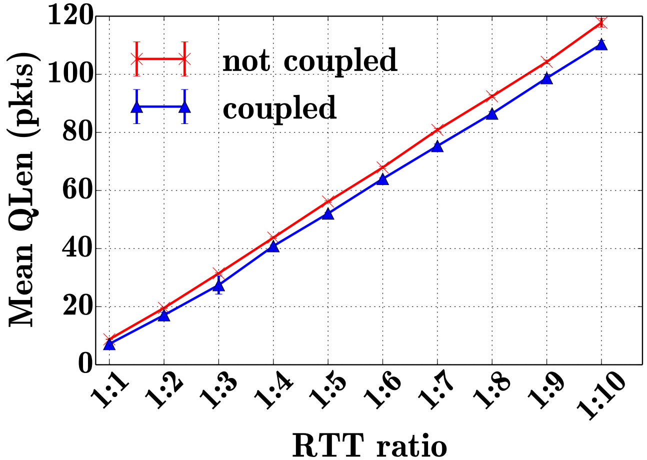
Applicability of simulation results

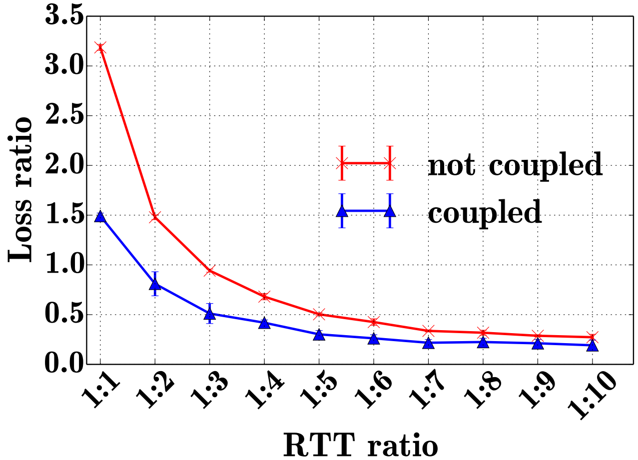
Datacenter	**Internet**	**BDP** (1500 byte packets)
10 Gbit/s, 100 μs [1]	10 Mbit/s, 100 ms	83.3
10 Gbit/s, 10..100 μs: [2]	1..10 Mbit/s, 100 ms	8.3 .. 83.3
1 Gbit/s, 100 μs: [3]	10 Mbit/s, 10 ms	8.3
1 Gbit/s, 250 μs: [4]	25 Mbit/s, 100 ms	208.3

Referenced datacenter conditions are comparable to common Internet bandwidth x delay products

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," *ACM SIGCOMM*, 2010
[2] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," *ACM SIGCOMM*, 2015.
[3] A. M. Abdelmoniem, B. Bensaou, and A. J. Abu, "HyGenICC: Hypervisor-based generic IP congestion control for virtualized data centers," *IEEE ICC*, May 2016
[4] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: Synthesizing existing transport strategies for data center networks," *ACM SIGCOMM*, 2014

Results – mean Q length and loss ratio

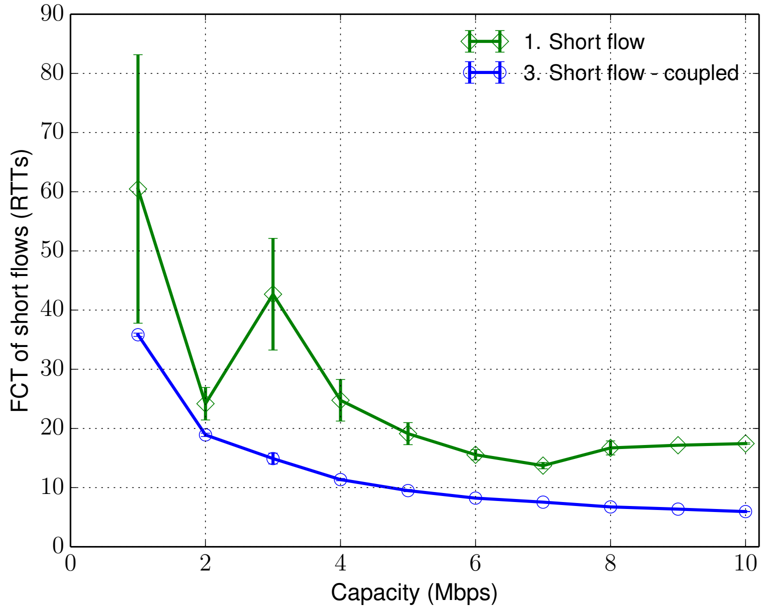




Mean queue length and loss ratio as the RTT ratios between 2 flows is varied

minRTT 20ms, maxRTT 200ms, Bottleneck: 10Mb, preprocessed TMIX background Traffic (taken from 60 minute trace of campus traffic of university of north Carolina – approximate load 50%, RTT of the background traffic 80-100ms)

Results – flow completion time



Flow completion time (FCT) of a short flow, with and without *ctrlTCP*

Long Flow – 25 Mb, short flow - 200KB, capacity varied from 1 to 10 Mb

Conclusion

- Allows datacenter administrators to exert precise control over the relative bandwidth share offered to coupled flows, with only minimal interfacing to the kernel TCP code
- Implementation in the FreeBSD kernel and ns2 simulator
- Works with flows with heterogeneous RTTs
- Eliminates competition and reduces flow completion time
- Future work:
  - by changing the increase/decrease behavior as a function of the number of flows in a coupled group
  - to investigate our solution on 10Gbps links while considering typical practical challenges at high speeds such as CPU delay

Thank you!

Questions?