Virtualized Congestion Control

Bryce Cronkite-Ratcliff, et. Al.
VMware, Stanford University, Technion

ACM SIGCOMM 2016

Outline

- Introduction
- Virtualized Congestion Control Design
  - Hypervisor translation techniques
  - Explicit Congestion Notification (ECN) Unfairness
  - Evaluation
- Conclusion

Introduction

- Hyperscale datacenters → Huge growth in network communication
- Large datacenters are deploying new congestion control algorithms
  - DCTCP, TIMELY, etc.
- Problem: How about **multitenant datacenters**?

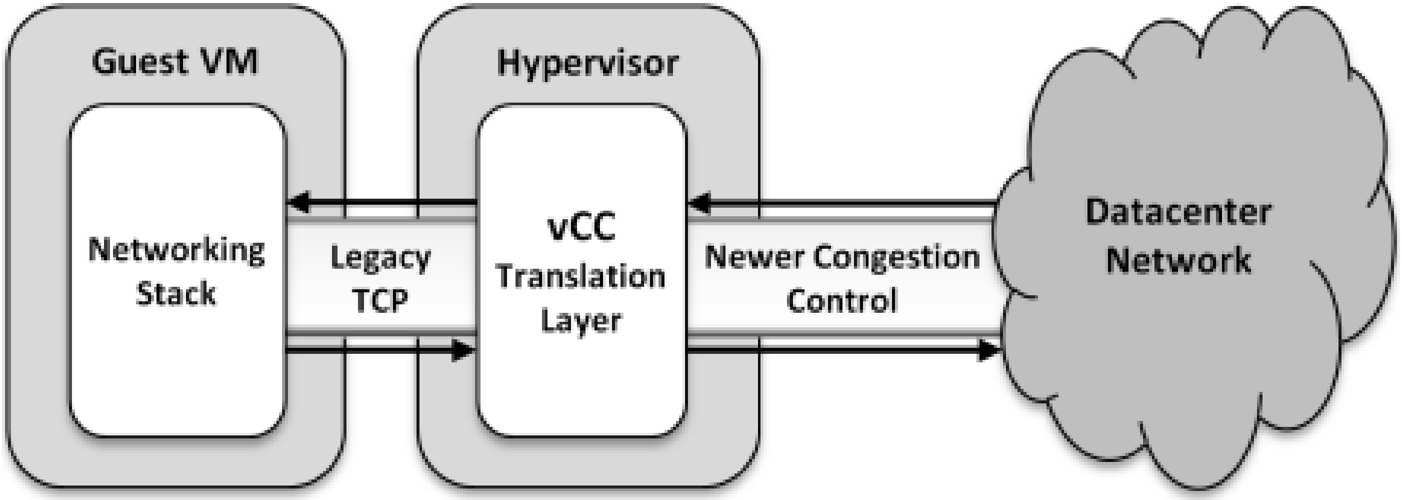
Multitenant datacenter

- Data centers operated by third parties for the benefit of multiple enterprise tenants
  - Many tenants lease and share a common physical infrastructure

- Tenants implement their own congestion control algorithm

- What will happen if
  - Tenants VMs' OSes use different congestion control algorithms?
  - Tenants VMs' OSes use old-fashioned congestion control algorithms?

Problem illustration



- Guest VM uses legacy TCP
- Datacenter hypervisor applies new congestion control

➔ Datacenter must ensure that they **play well together**

How to solve it?

1. Dividing the bandwidth among the tenants (fixed allocation)

How to solve it?

1. Dividing the bandwidth among the tenants (fixed allocation)

prevents statistical sharing of unused bandwidth

How to solve it?

1. Dividing the bandwidth among the tenants (fixed allocation)

prevents statistical sharing of unused bandwidth

2. Modifying datacenter switches and tweak the fairness rules between tenants at each switch

How to solve it?

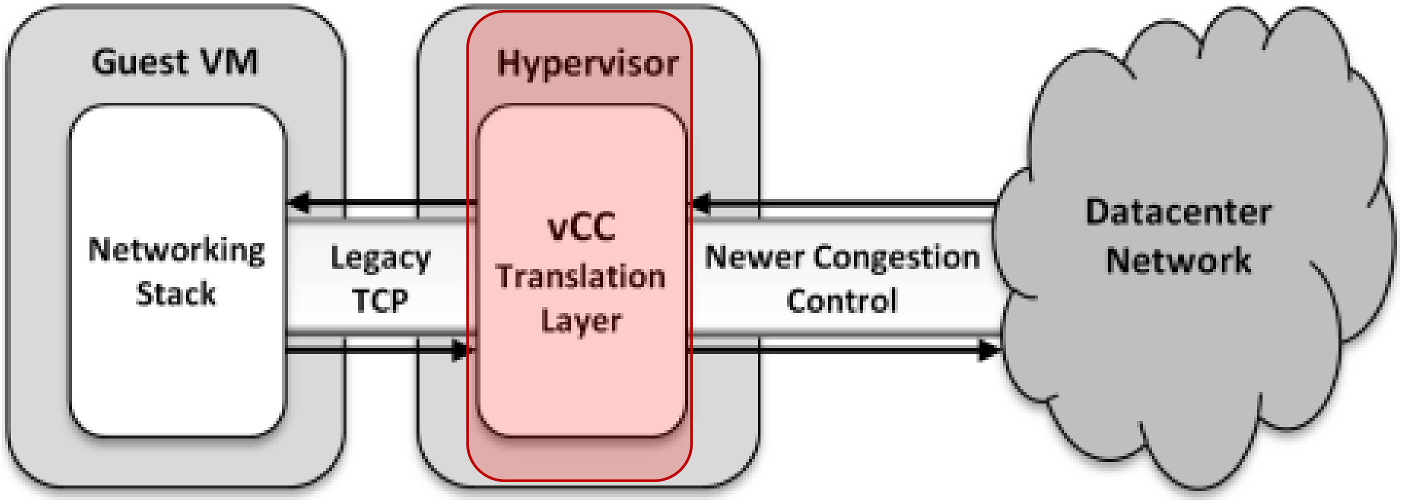
~~1. Dividing the bandwidth among the tenants (fixed allocation)~~

prevents statistical sharing of unused bandwidth

~~2. Modifying datacenter switches and tweak the fairness rules between tenants at each switch~~

as the number of tenant algorithms increases,
this approach becomes harder to deploy

Authors' solution



- Introducing a **translation layer**
- Giving the illusion to each of the VM guests that it keeps using its own congestion control algorithm
- Taking advantage of the fact that all traffic passes through hypervisors

Virtualized Congestion Control (vCC)

$$g(T(x)) = f(x).$$

- x: input sequence
- f(x): output obtained by datacenter congestion control
- T(x): output of vCC translation
- g(x): output obtained by VM congestion control

vCC design

Hypervisor translation techniques

- vCC translates between congestion control in the guest VM and data center

- Candidates for above goal
  - Write into/read from guest memory
  - Split connection
  - Buffer packets/ACKs
  - Duplicate ACKs
  - Throttle the window
  - Modify the 3-way handshake

Write into/read from guest memory

- Modern hypervisors can monitor guest VMs
- Hypervisor can directly write TCP parameters in the guest memory and registers
- Hypervisor also can read TCP parameters in the guest VMs

➔ Tenants may not accept that the hypervisor writes into/read from the VM memory

Split connection

- The split-connection approach breaks a TCP connection into several sub-connections
- Acknowledge packets to the guest VM at some desired rate
- Send them on the datacenter network using the desired target congestion control algorithm

➔ The solution goes against TCP end-to-end semantics

Buffer packets/ACKs

- Buffer in-flight packets and ACKs

- vCC can buffer in-flight packets and retransmit according to its own RTOmin buffer

- Hypervisor can pace ACKs to make TCP less bursty

➔ The hypervisor needs to manage packet/ACK buffer

Duplicate ACKs

- Hypervisor can duplicate and resend the last sent ACK
- Force the guest to halve its congestion window

➔ This technique may violate TCP semantics

Throttle the receive window

- Hypervisor can decrease the receive window (to guest VM)

- Force the guest to have fewer outstanding packets
  - # of packets in flight is upper-bounded by the minimum of the congestion and the receive windows

➔ This technique can make the congestion window meaningless, conflicting with common implementations of the TCP buffer management

Modify the 3-way handshake

- Hypervisor can change the options that are negotiated when setting up the connection
  - Modify the negotiated MSS, or enable timestamps

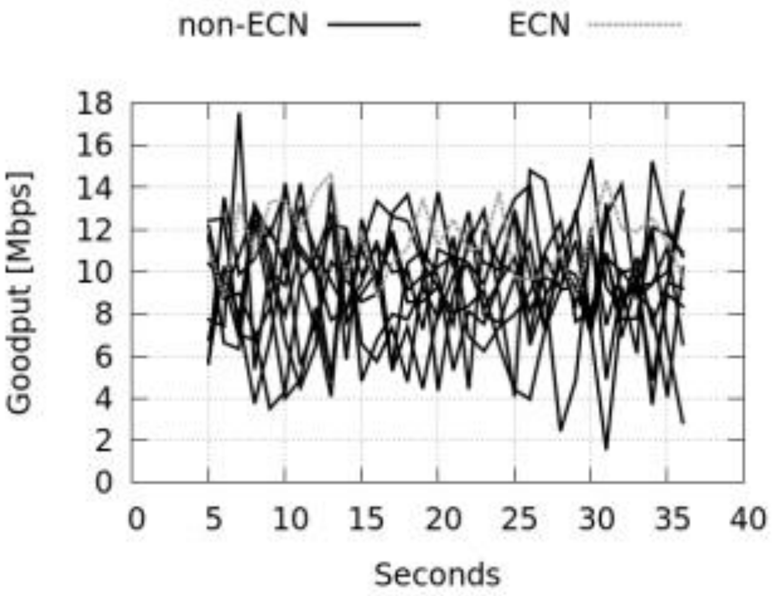
➔ The technique can barely help for most practical benefits without additional techniques

Scenario: ECN unfairness

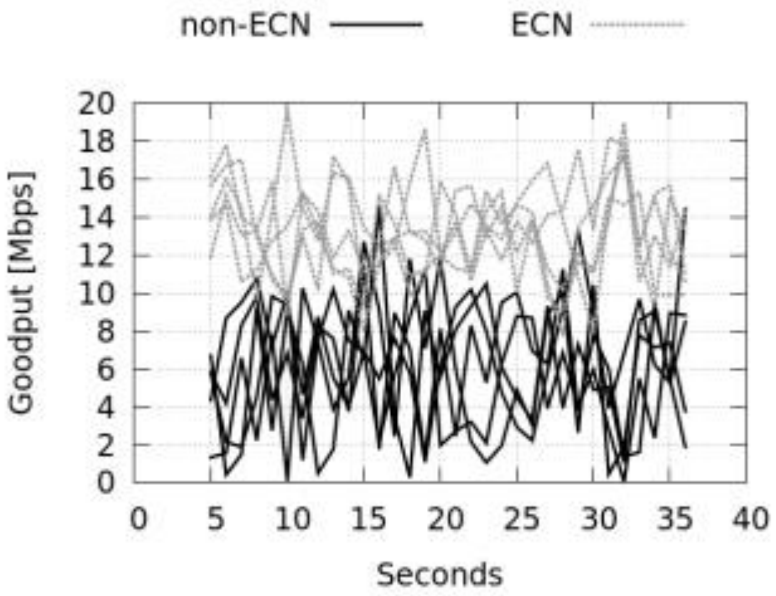
ECN unfairness

- ECN allows flows to react to congestion before any data has been lost
- ECN has not been widely supported in operating systems until recently
- A lack of ECN support can cause such legacy systems to suffer

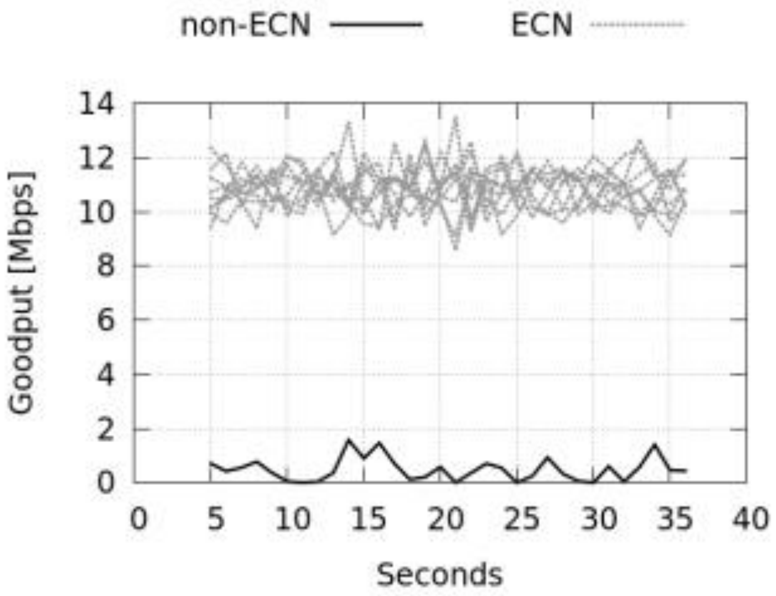
ECN unfairness



(a) 9 non-ECN vs. 1 ECN flows



(b) 5 non-ECN vs. 5 ECN flows



(c) 1 non-ECN vs. 9 ECN flows

- Non-ECN flows show lower goodput than ECN flows
- The greater # of ECN flows, the greater the unfairness

Analysis on ECN unfairness

- The average queue length measured by the switch grows beyond RED (Random Early Drop) threshold eventually
- **ECN** flow's packets are just marked → halve the window → Congestion avoidance phase
- **Non-ECN** flow's packets are dropped → timeout occurs → Slow start phase ➔ **Low throughput**

vCC's solution

- vCC transforms non-ECN flows to virtual-ECN flows

Modifying 3-way handshakes

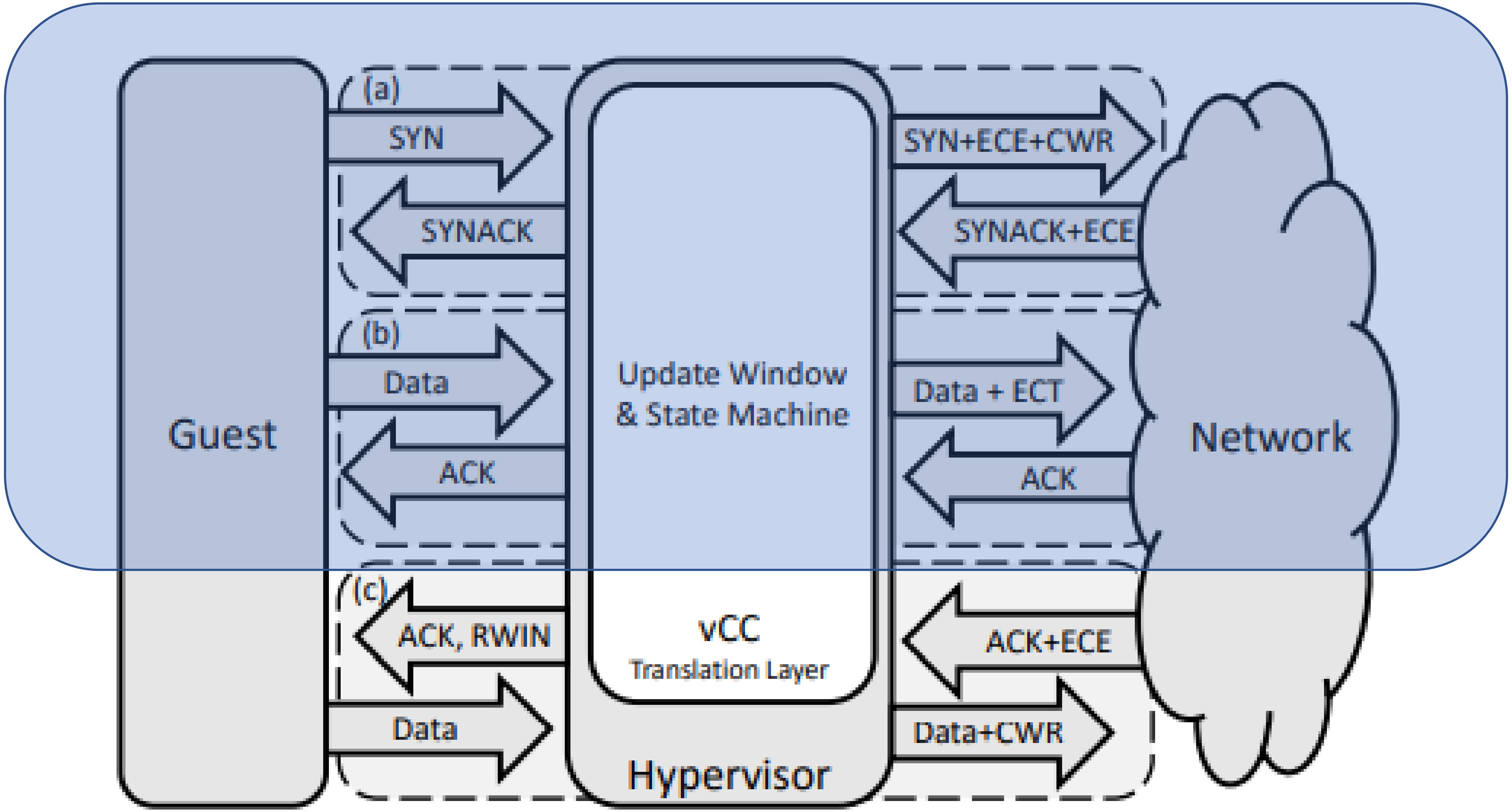
+

Throttling receive window

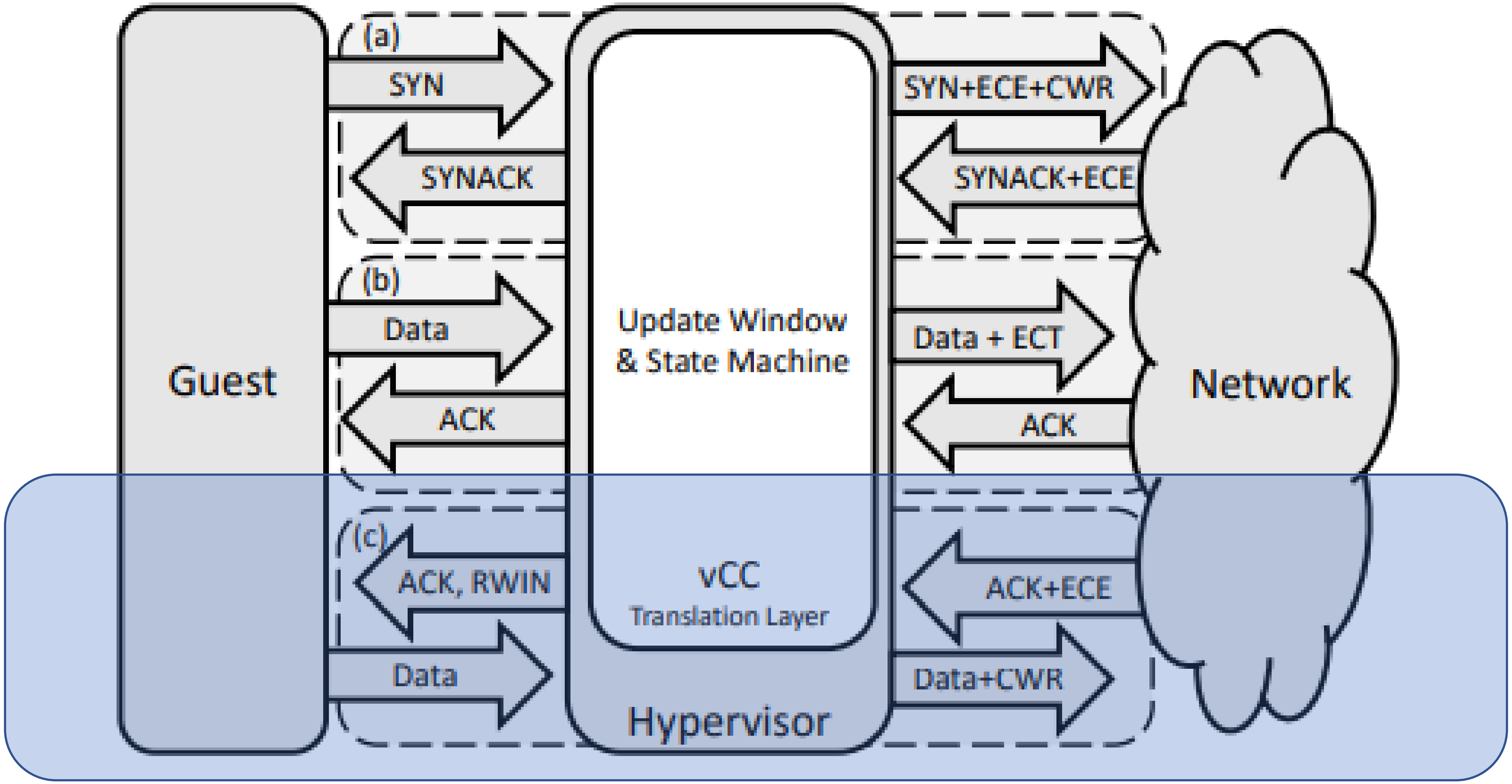
→

Virtualized ECN flows

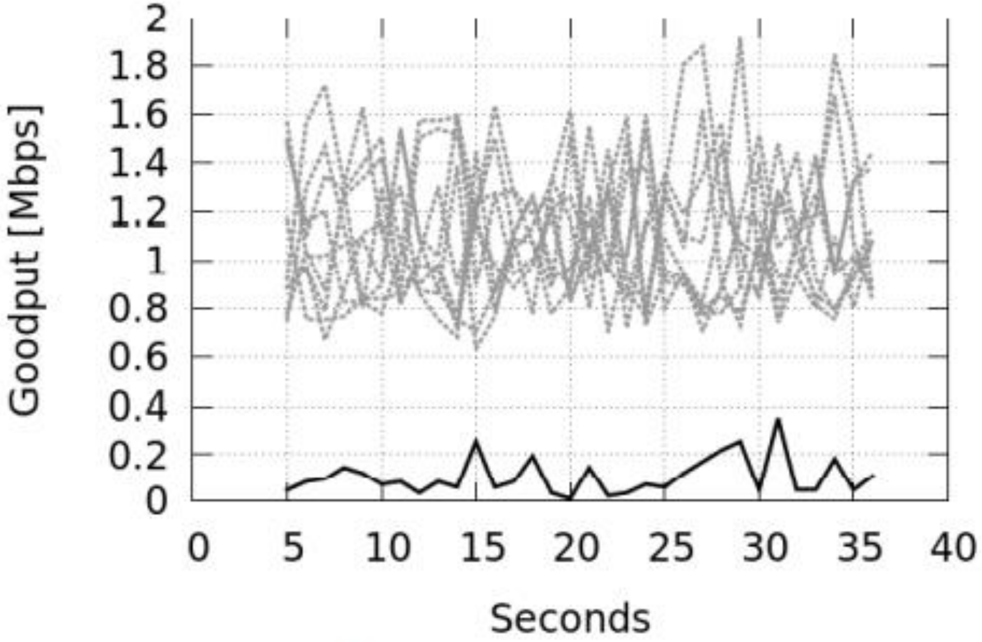
Modifying 3-way handshake



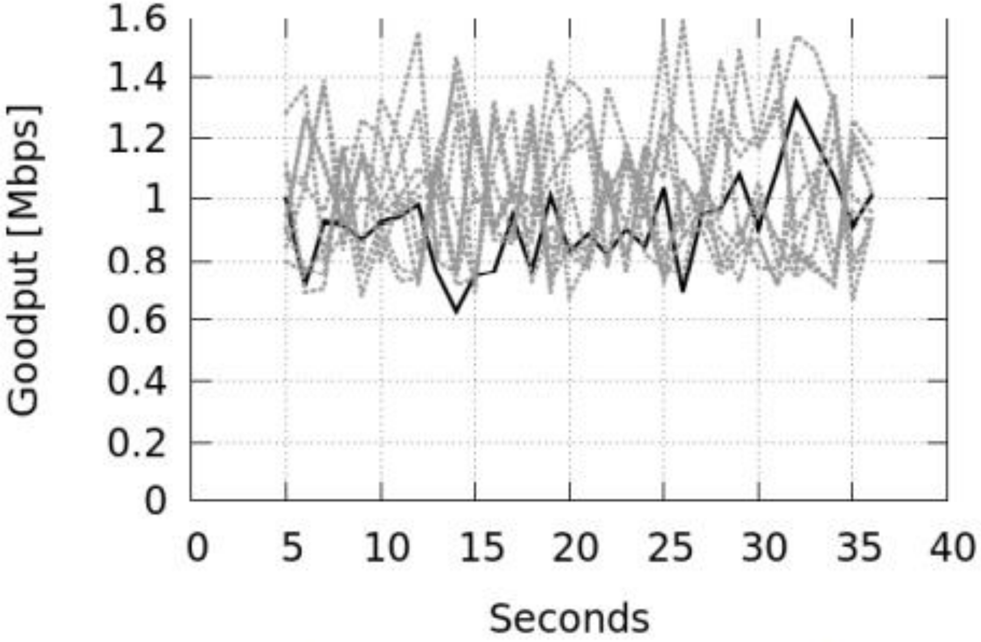
Receive window throttling



Evaluation



(a) 9 ECN flows and one non-ECN flow



(b) 9 ECN flows with one virtual-ECN flow

- Non-ECN flow is starved by multiple ECN flows
- Virtual-ECN flow shows similar performance to ECN flows in terms of goodput

Conclusion

- Multitanent datacenters can suffer from differences in congestion control algorithms between guest VMs and the hypervisor

- vCC enables the datacenter owner to introduce a new congestion control algorithm in the hypervisors

- Hypervisors translate between the new congestion control algorithm and the old legacy congestion control

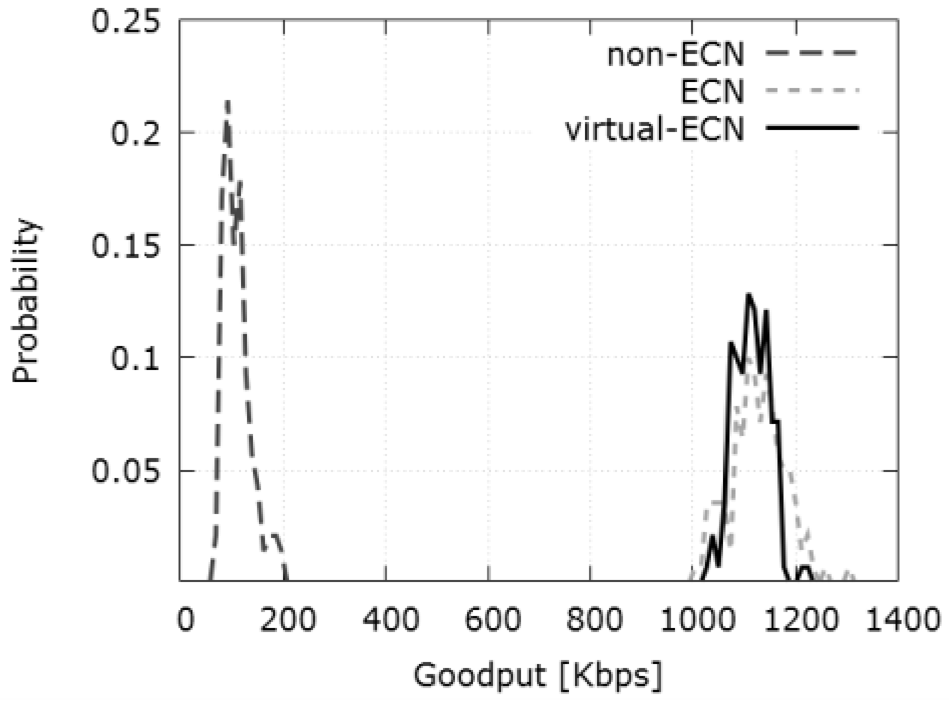


Figure 1: 10 flows share the same bottleneck link: an ECN-unaware flow (*non-ECN*), 8 ECN-enabled flows (*ECN*), and a non-ECN flow augmented by vCC translation (*virtual-ECN*). The figure plots the probability density function, over many runs, of the average goodput of each flow. The non-ECN flow is starved, reaching only 10% of the ECN goodput on average. After translation to virtual-ECN, the average goodput is near identical to that of ECN.

Parameter	Value		
	RED1	RED2	RED3
RED_{min}	90000	30000	30000
RED_{max}	90001	90000	90000
RED_{limit}	1M	400K	400K
RED_{burst}	61	55	55
RED_{prob}	1.0	0.02	1.0

Table 1: RED Parameters used in the experiments.