Linux Plumbers Conference 2010

TCP-NV
Congestion Avoidance for Data Centers

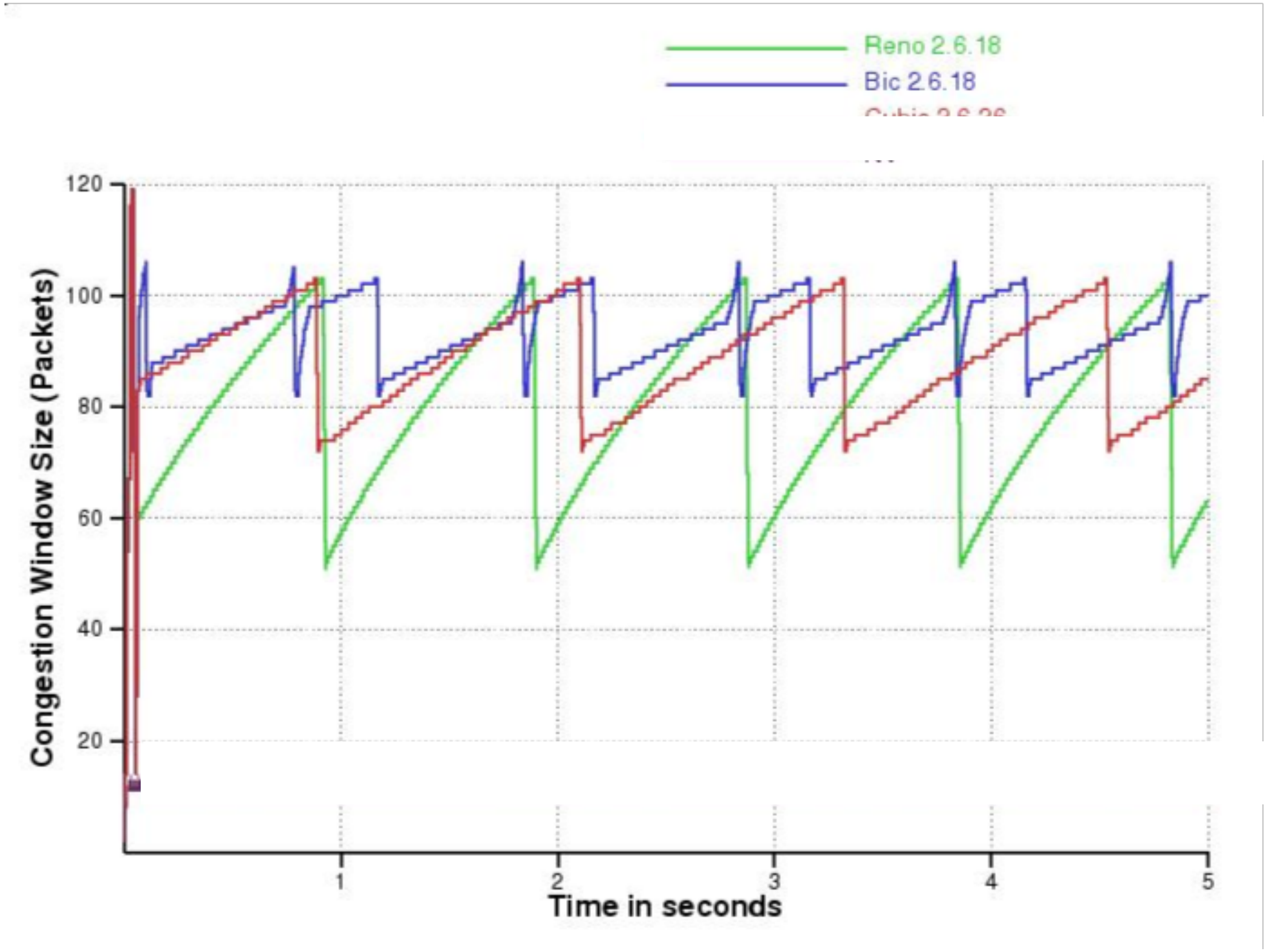
Lawrence Brakmo

Google

TCP Congestion Control

- **Algorithm for utilizing available bandwidth without too many losses**
  - No attempt at eliminating losses
  - Works well at eliminating congestion collapse
- **TCP's congestion control will continually increase its congestion window (# packets in transit) until losses occur**
  - rate = cwnd / RTT
- **I.e. congestion control repeatedly creates congestion and packet losses**
- **Unfairness**
  - Between flows with different RTTs (up to 20-40x)
- **Works well for large transfers and it has proven to work well in many different environments**

Congestion Window Graph



Data Center Perspective

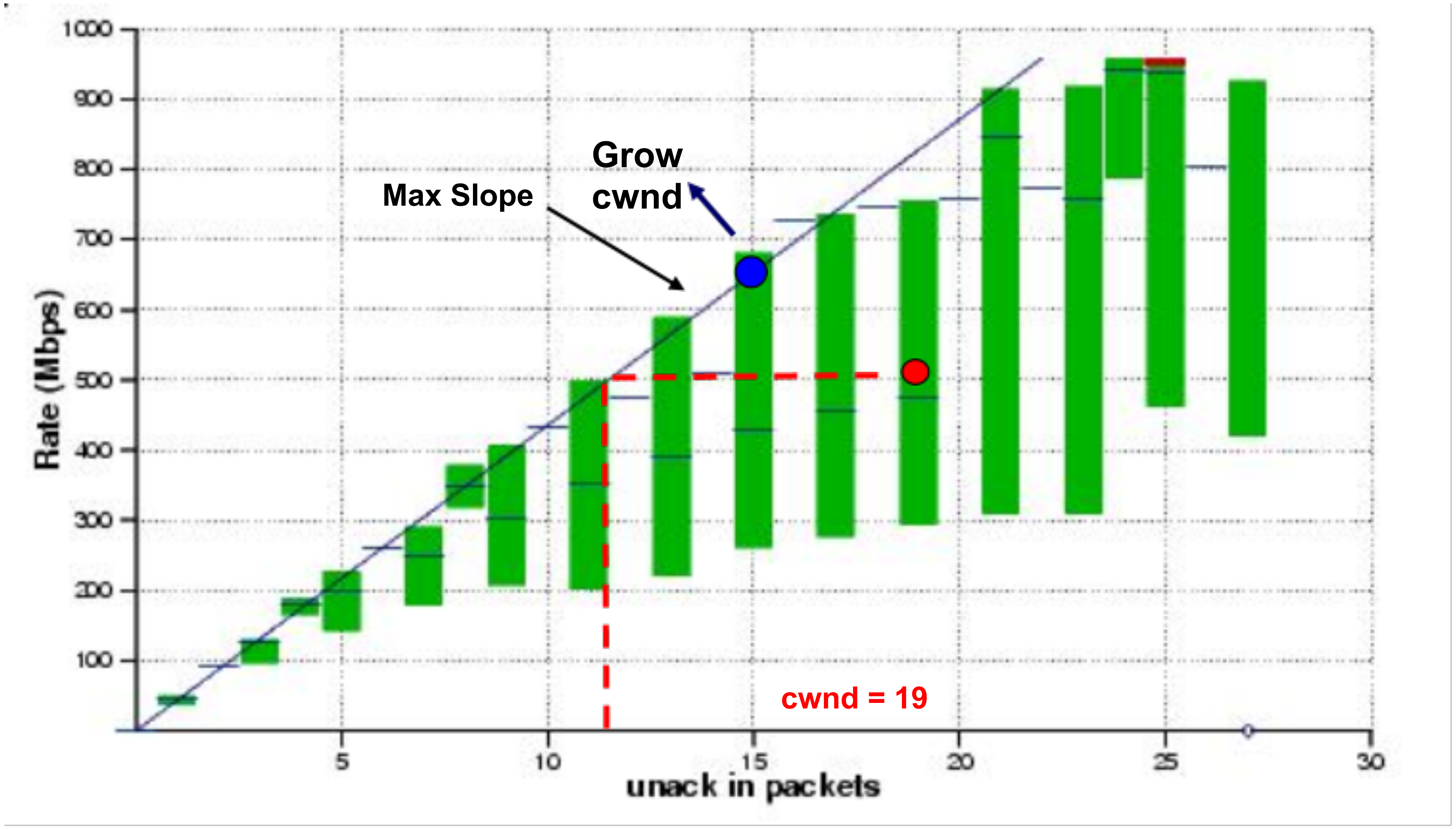
- **RPCs make large percent of traffic**
  - Important to test TCP behavior with RPCs
- **Latency sensitive**
- **Congestion control results in queue build-up at host, switches and routers**
  - Increasing average latency of smaller size RPCs
- **Congestion control results in periodic packet losses**
  - Increasing high percentile latency of smaller size RPCs
- **Need Congestion avoidance**
  - Reduce queue buildup and losses

Congestion Avoidance

- **Goal: Prevent queue build-up and packet losses while making full utilization of available bandwidth**
  - Only grow congestion window when there is bandwidth available
  - Decrease congestion window when congestion (queue build-up) is detected
    - Unlike congestion control which only decreases with losses
- **Typically delay (RTT) based**
  - Increase in RTT implies queue buildup
  - Problem: RTT is VERY noisy

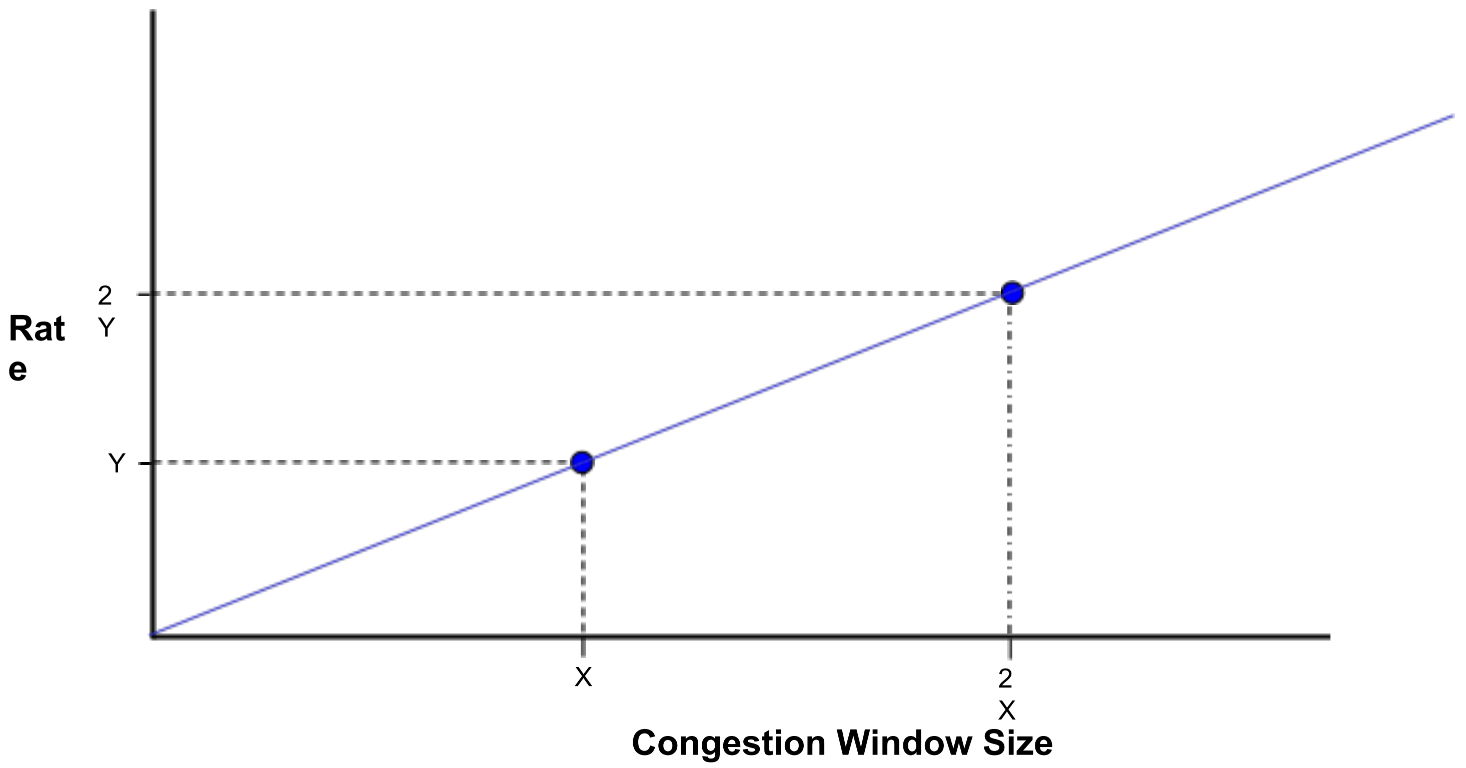
TCP-NV Congestion Avoidance

- We can only detect the level of congestion, not the available bandwidth



Basic Concept of TCP-NV Congestion Avoidance

- **In a perfect world, and while there is no congestion, if a cwnd of X achieves rate Y, then a cwnd of 2X should achieve rate 2Y**



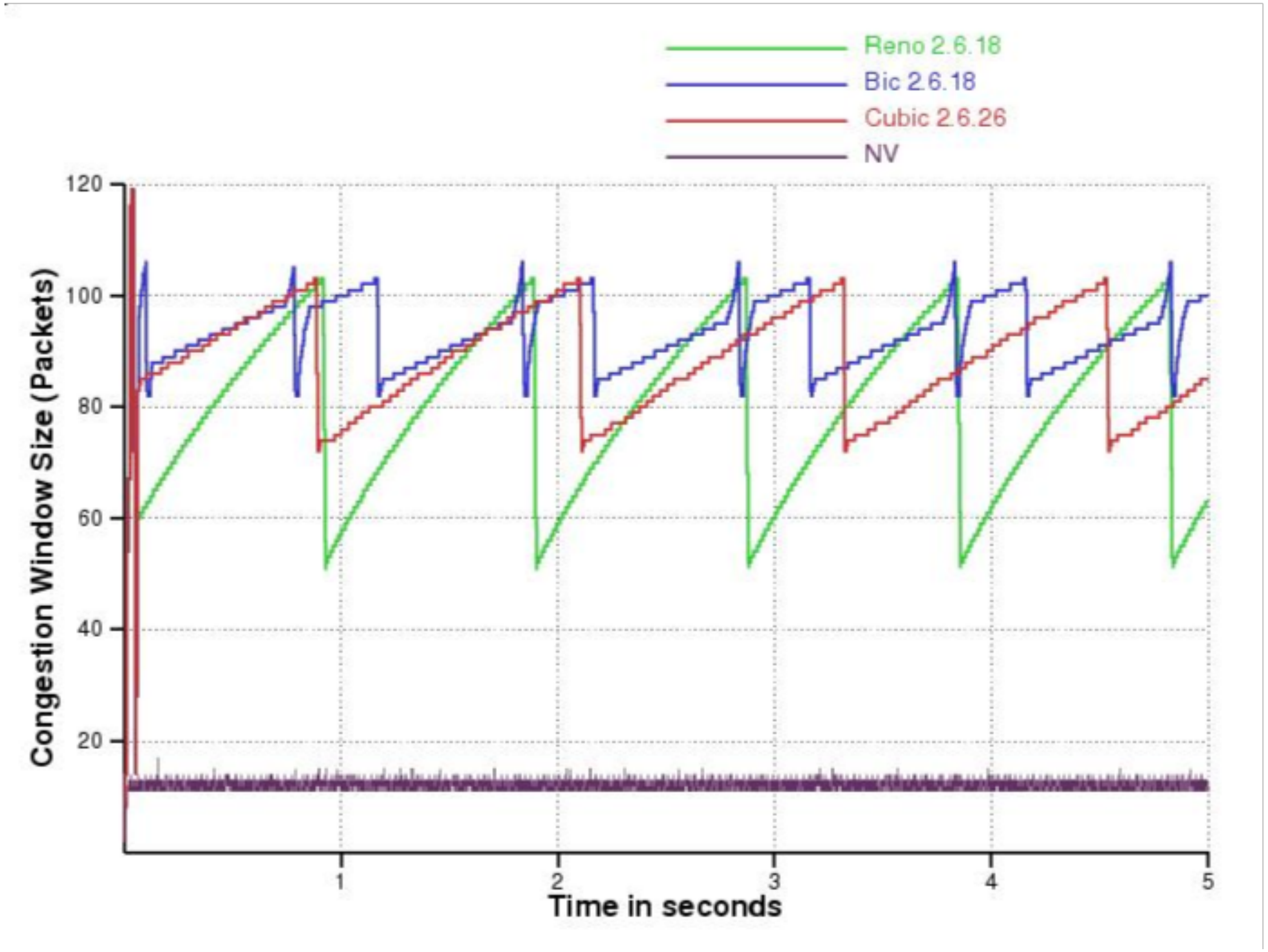
TCP-NV Implementation

- **In Linux 2.6.34, soon in 2.6.36**
- **On top of BIC**
  - BIC behavior when there is no congestion
- **Most of the changes are self contained in one CA function: nvtcp_acked()**
  - Collects appropriate info
  - Every so often makes a CA decision
    - Allow BIC to grow cwnd
    - Don't allow BIC to grow cwnd
    - Decrease cwnd and don't allow BIC to grow cwnd
- **Need to store in-flight every time a packet is sent**
  - Needed by ntcp_acked(), stored in skb
- **Can be disabled through sysctl (in which case it behaves like BIC)**

More Implementation Details

- **Every time a packet is sent**
  - Store time in us, bytes in-flight
- **Every time an ACK is received**
  - Use sent time and in-flight for newest packet ACKed
  - Calculate slope = rate/in-flight
  - Keep largest slope seen so far (recalc every so often)
  - Keep largest rate seen since last CA decision
- **Every so often**
  - pred_cwnd = max_rate/max_slope
  - if pred_cwnd+pad > cwnd
    - Reduce cwnd by percent of congestion
    - Stop BIC from growing cwnd
  - else
    - Allow BIC to grow cwnd

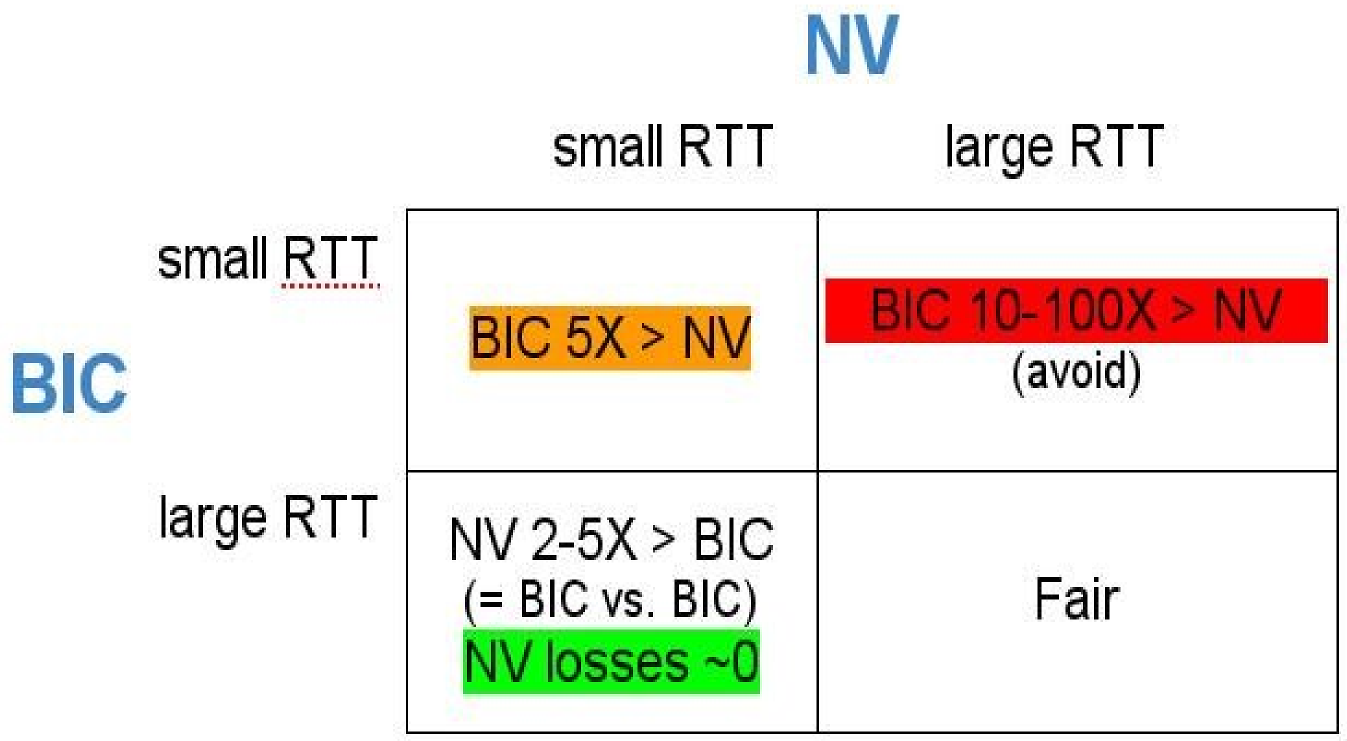
Congestion Window Graph



Issues

- **TSO, LRO and interrupt coalescence**
  - Reduces number of data points
    - May only get one ACK per RTT
  - Need to inflate window for one flow to use full bandwidth
- **Coexistence with congestion control**
  - As CC increases cwnd, CA reduces its own
  - Usually cannot do both
  - Unless RTT of CA is << RTT of CC
    - Traffic within DC CA, traffic outside DC CC
- **TCP-NV needs pacing when** RTT > 1 to 5ms
  - due to big bursts when starting a new RPC
  - not an issue for traffic within a data center or cluster
- **Cannot do CA with small RPCs**
  - Needs a couple of RTTs of data

CC vs CA Fairness



		NV	
		small RTT	large RTT
BIC	small RTT	BIC 5X > NV	BIC 10-100X > NV (avoid)
	large RTT	NV 2-5X > BIC (= BIC vs. BIC) NV losses ~0	Fair

Issues ...

- **Reverse congestion**
  - Congestion in the ACK direction will increase RTT
  - CA mechanism will react to this
- **Solutions**
  - Prioritize pure ACKs in host and switches/routers
    - so they are not affected by congestion
  - Measure reverse delay and adjust appropriately
    - New TCP option
    - Use TCP us Timestamp option
  - Limit how much NV can decrease the cwnd

Issues ...

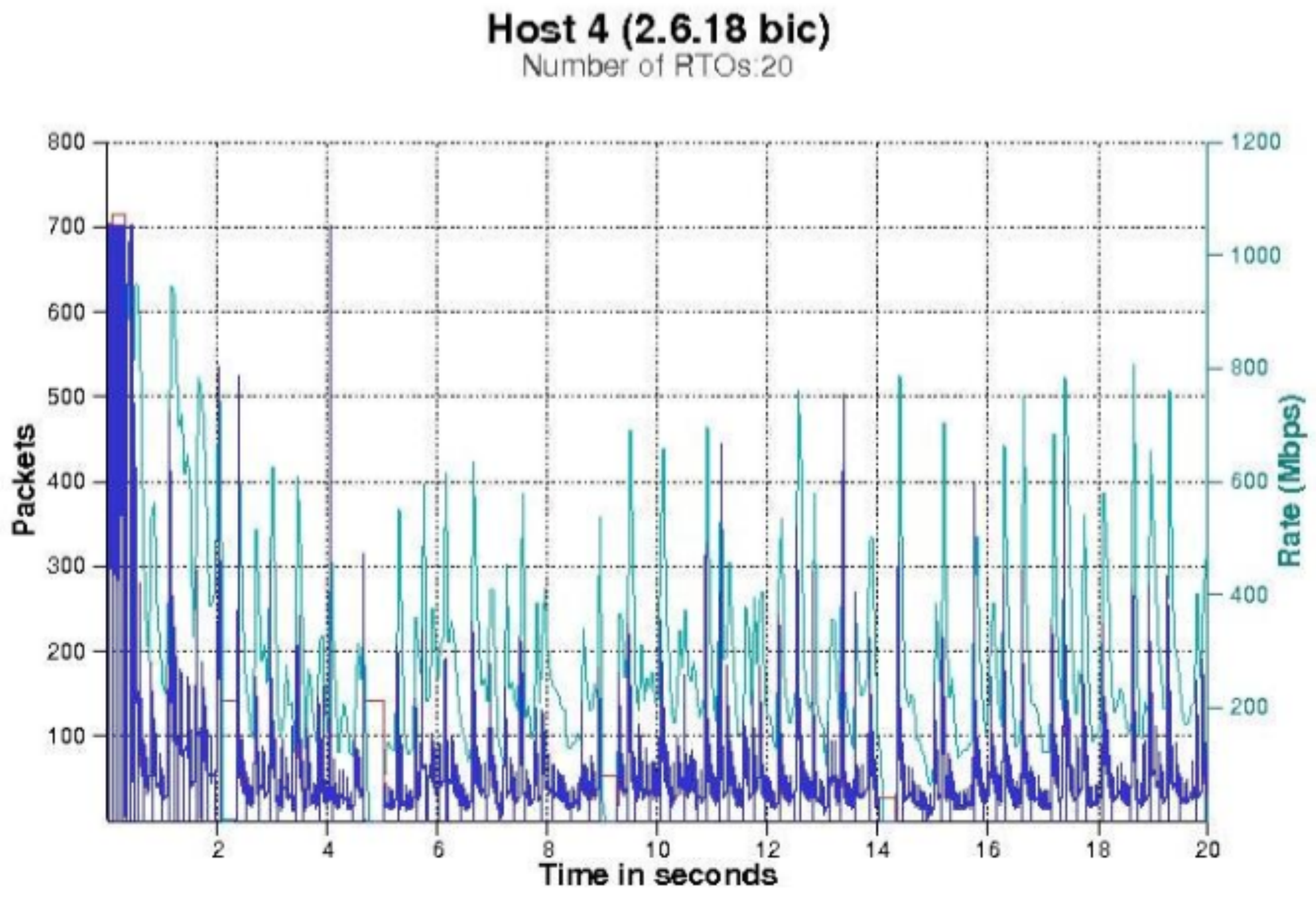
- **Too many flows decrease effectiveness of Congestion Avoidance**
  - # of flows w/o losses is a function of router buffer size
  - With 1.2MB buffer size, can handle up to 128 bulk flows with no losses
    - This has improved and we are working on further improvemens

Results

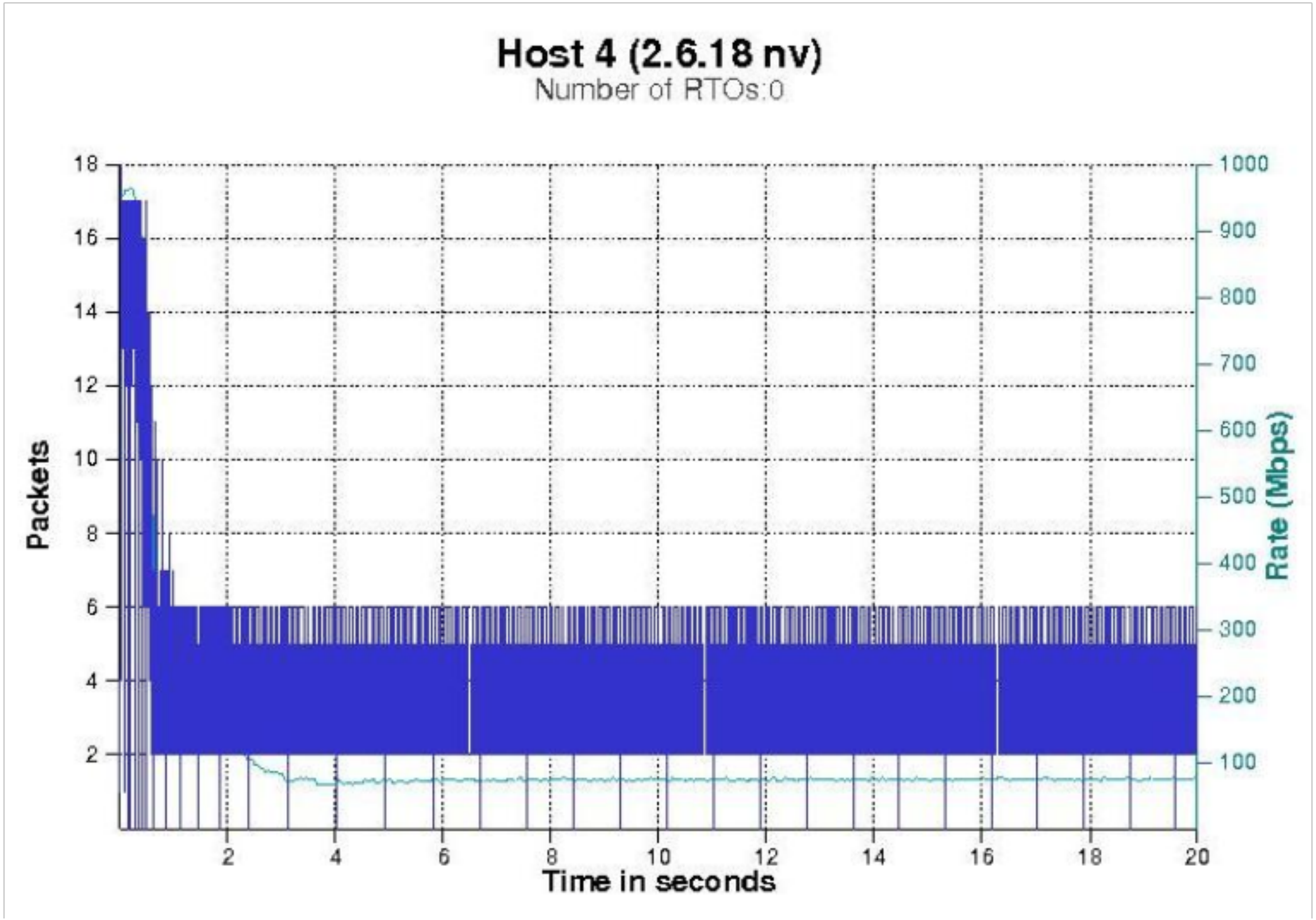
- **Simulations using actual Linux network stack**
  - Using Sam Jansen's Network Simulation Cradle
  - Up to 1000 hosts, 50us to 100ms RTTs
  - Can examine detail behavior of protocol, routers, etc.
  - Can easily move the code to Linux
- **Rack tests with 1G and 10G NICs**
  - Actual hardware
    - Effects of TSO, LRO, interrupt coalescence
- **Starting tests with production workloads**
  - More machines (currently 38)
  - Actual workloads

TCP Burstiness with Small RTT Large RPCs

- Experiment: 128 flows, 100us RTT
- Graph: In-flight (dark blue) and rate (aqua) of 1 flow
- Each large peak is the result of starting a new RPC.

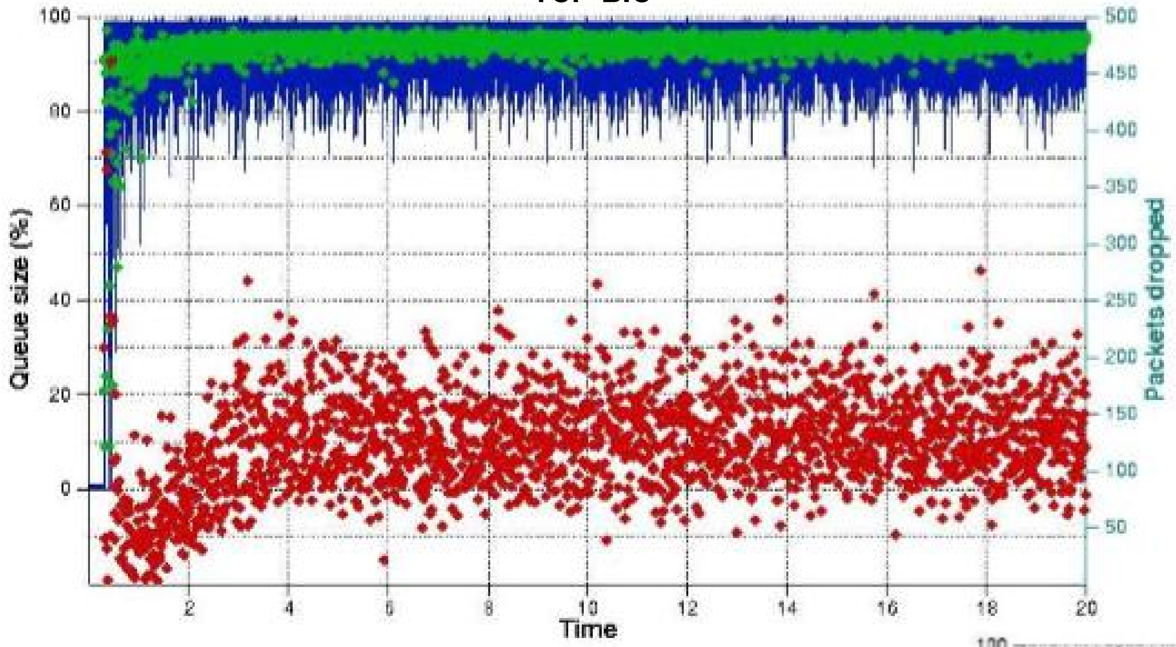


TCP Burstiness with TCP-NV



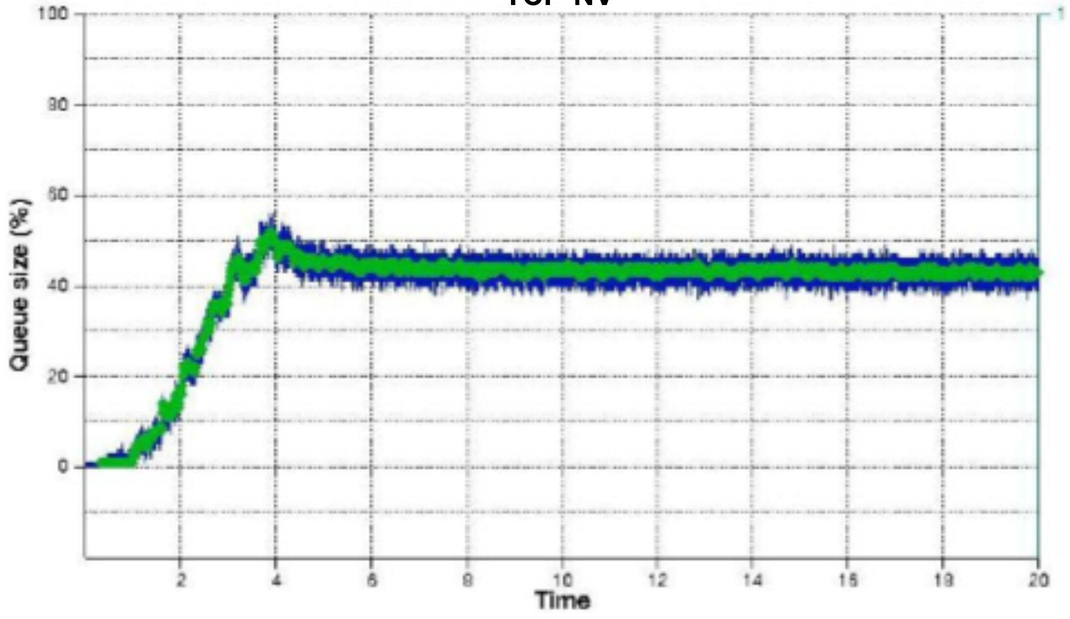
Router Queues with 128 flows
1-8 MB RPCs, 100us RTT, 1.2MB switch buffer

TCP-BIC



13,000 packet losses/s

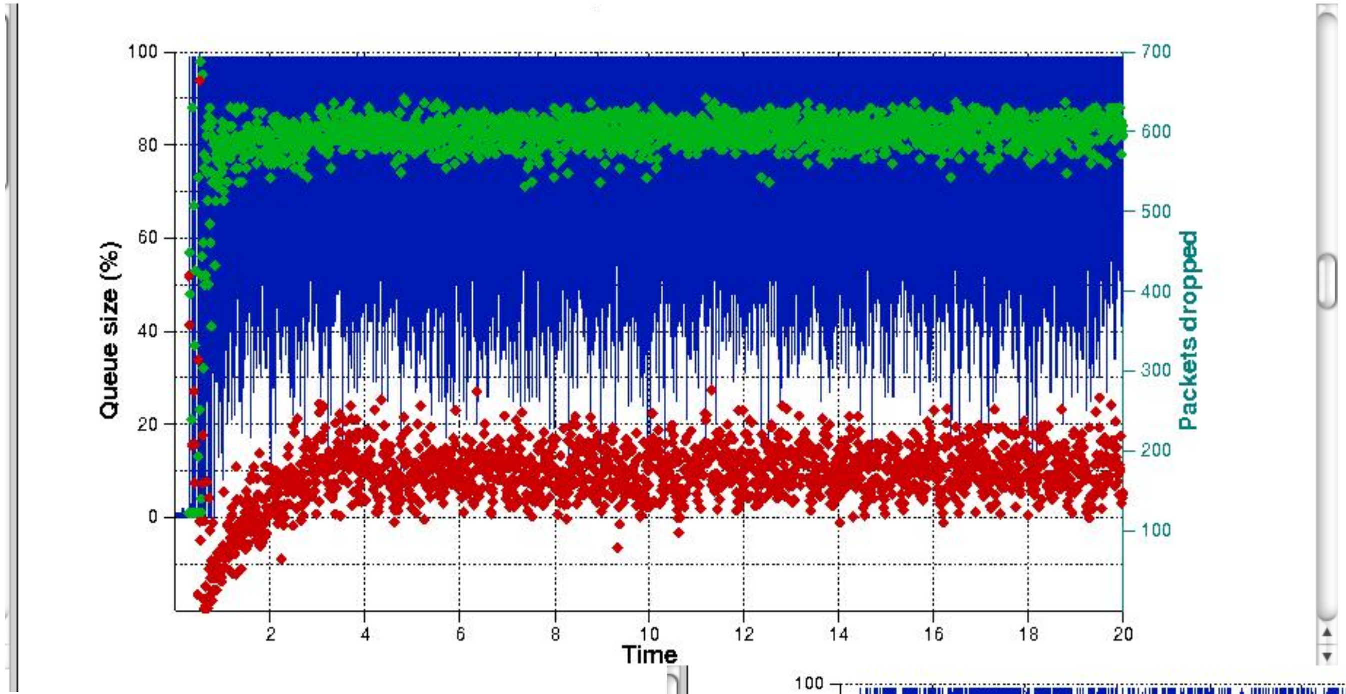
TCP-NV



0 packet drops

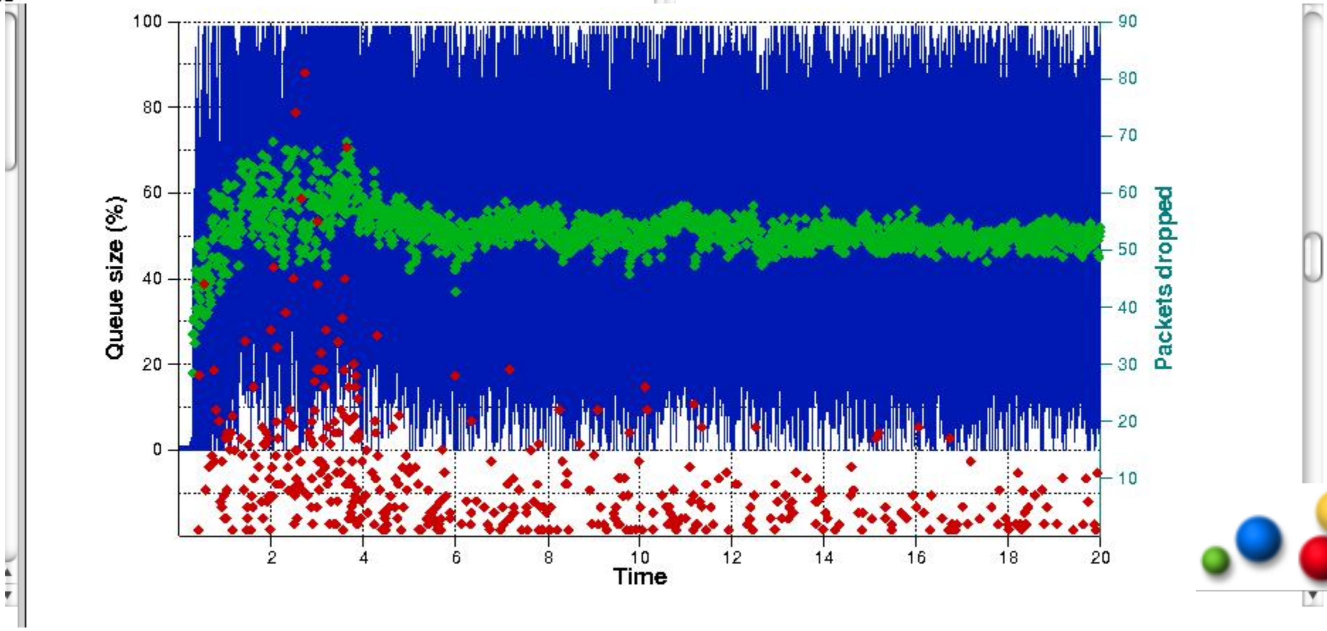
Router Queues with 128 flows
1-8 MB RPCs, 100us RTT, 175KB switch buffer

TCP-BIC



17,800 packet drops/s

TCP-NV



75 packet drops/s

Simulator, 10Gbps, ~1MB RPCs

	TCP-BIC avg of 2,8,32 flows	**TCP-NV avg of 2,8,32 flows**	**TCP-BIC 128 flows**	**TCP-NV 128 flows**
Goodput (Gbps)	8.1	8.8	9.5	9.6
Fairness	0.96	0.94	0.82	0.93
Average latency	642 us	251 us	1.7 ms	527 us
90% latency	448 us	274 us	589 us	559 us
95% latency	494 us	285 us	14.5 ms	584 us
99% latency	11.9 ms	305 us	21.0 ms	616 us
Average queue size	94.6	39.3	152	194
Packets dropped/sec	19000	0	63400	5300

2 1G hosts sending to 1G host (Rack test)

	Flows per host	Busy %	avg cwnd	avg RTT (ms)	Aggregate Rate (Mbps)	Losses %
BIC	1	9.3	119	2.2	918	1.500
NV	1	3.3	14	1.0	925	0.000
BIC	4	11.0	25	1.9	952	2.800
NV	4	6.0	6.4	1.0	944	0.001
BIC	8	9.6	22	1.8	981	3.800
NV	8	6.5	6.5	1.0	974	0.008

2 1G hosts sending to 1G host, Flow Control Enabled

	Flows per host	Busy %	avg cwnd	avg RTT (ms)	Aggregate Rate (Mbps)
BIC	1	7.5	737	15	943
NV	1	2.9	14	1	938
BIC	4	8.7	736	66	944
NV	4	5.9	7	1	944
BIC	8	8.9	737	134	944
NV	8	7.2	8	1.5	944

Distributed Sorting (38 1Gbps machines, real HW)

TCP version	**Execution Time**	**Retransmits**	**RTOs**
Cubic	baseline	baseline	baseline
BIC	-3%	+36%	-25
NV	-5%	-45%	-17%

Conclusions

- **Congestion avoidance is possible in the data center**
    - Can insure everyone is running CA
- **Can reduce losses and queue buildup**
    - But these are workload dependent
- **Can reduce cpu utilization**
    - Due to fewer losses and smaller host queues
- **Small changes to the network stack**
    - Most of the code in a new ca module.
- **Started running larger tests**
    - multi-rack to cluster size
- **Code will be released in December**

The End