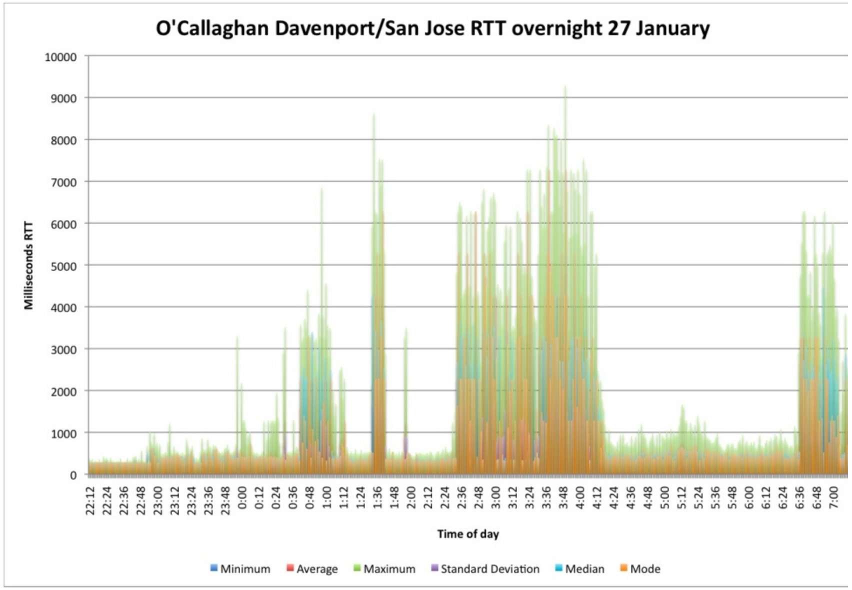
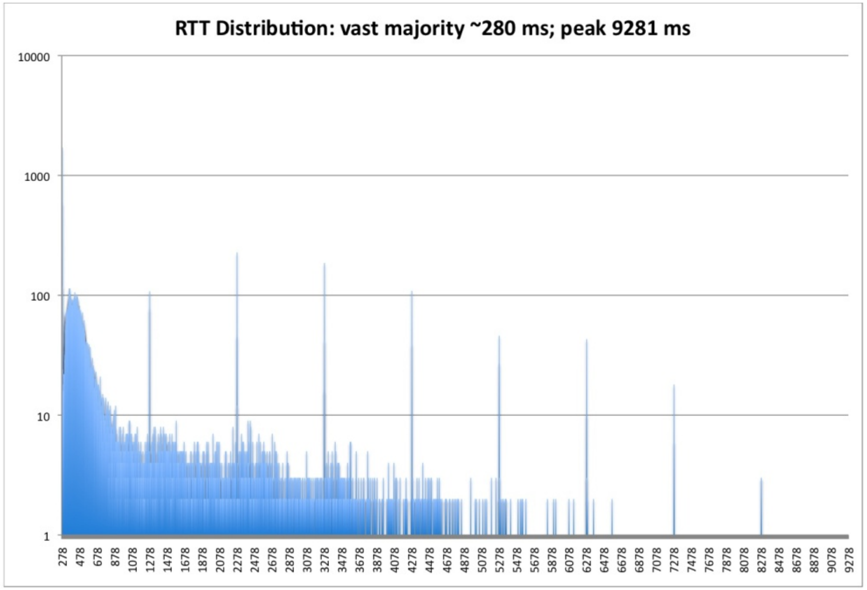
Buffer Bloat!

Obesity: it's not just a human problem…

Fred Baker





Best shown using an example…

Ping RTT from a hotel to Cisco overnight
RTT varying from 278 ms to 9286 ms

Delay distribution with odd spikes about a TCP RTO apart;
Suggests that we actually had more than one copy of the same segment in queue

What is buffer bloat? Why do I care?

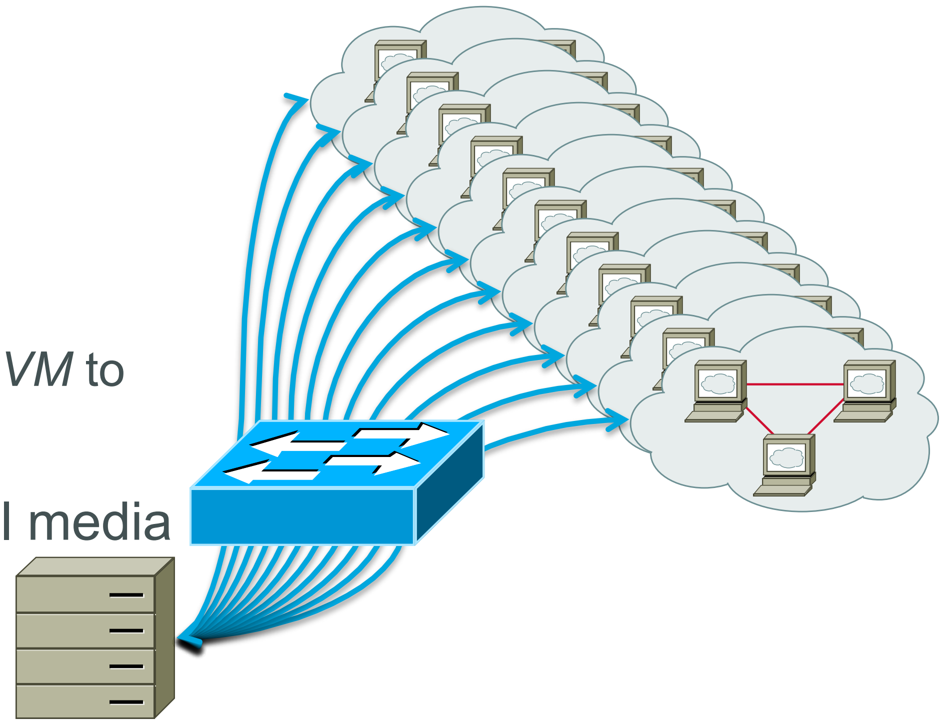
Because few applications actually worked

Persistent Deep Queues

- In access paths (Cable Modem, DSL, Mobile Internet)
  - Generally results from folks building a deep queue with permissive drop thresholds
  - One DSL Modem vendor provides ten seconds of queue depth
- In multi-layer networks (WiFi, Input-queued Switches)
  - ***Channel Acquisition Delay***
  - Systems not only wait for their own queue, but to access network
  - In WiFi, APs often try to accumulate traffic per neighbor to limit transition time
  - In Input-queued switches, multiple inputs feeding the same output appear as unpredictable delay sources to each other
  - In effect, **managing *delay* through queue**, not queue depth

Data Center Applications

- Names withheld for customer/vendor confidentiality reasons
- Common social networking applications might have
  - $O(10^3)$ racks in a data center
  - 42 1RU hosts per rack
  - A dozen Virtual Machines per host
  - $O(2^{19})$ virtual hosts per data center
  - $O(10^4)$ standing TCP connections *per VM* to other VMs in the data center
- When one opens a <pick your social media application> web page
  - Thread is created for the client
  - $O(10^4)$ requests go out for data
  - $O(10^4)$ 2-3 1460 byte responses come back
  - $O(45 \times 10^6)$ bytes in switch queues **instantaneously**
  - At 10 GBPS, **instant 36 ms queue depth**



Taxonomy of data flows

- We are pretty comfortable with the concepts of mice and elephants
  - "mice": small sessions, a few RTTs total
  - "elephants": long sessions with many RTTs
- In Data Centers with Map/Reduce applications, we also have ***lemmings***
  - $O(10^4)$ mice migrating together
- Solution premises
  - Mice: we don't try to manage these
  - Elephants: if we can manage them, network works
  - Lemmings: Elephant-oriented congestion management results in HOL blocking

Underlying theory

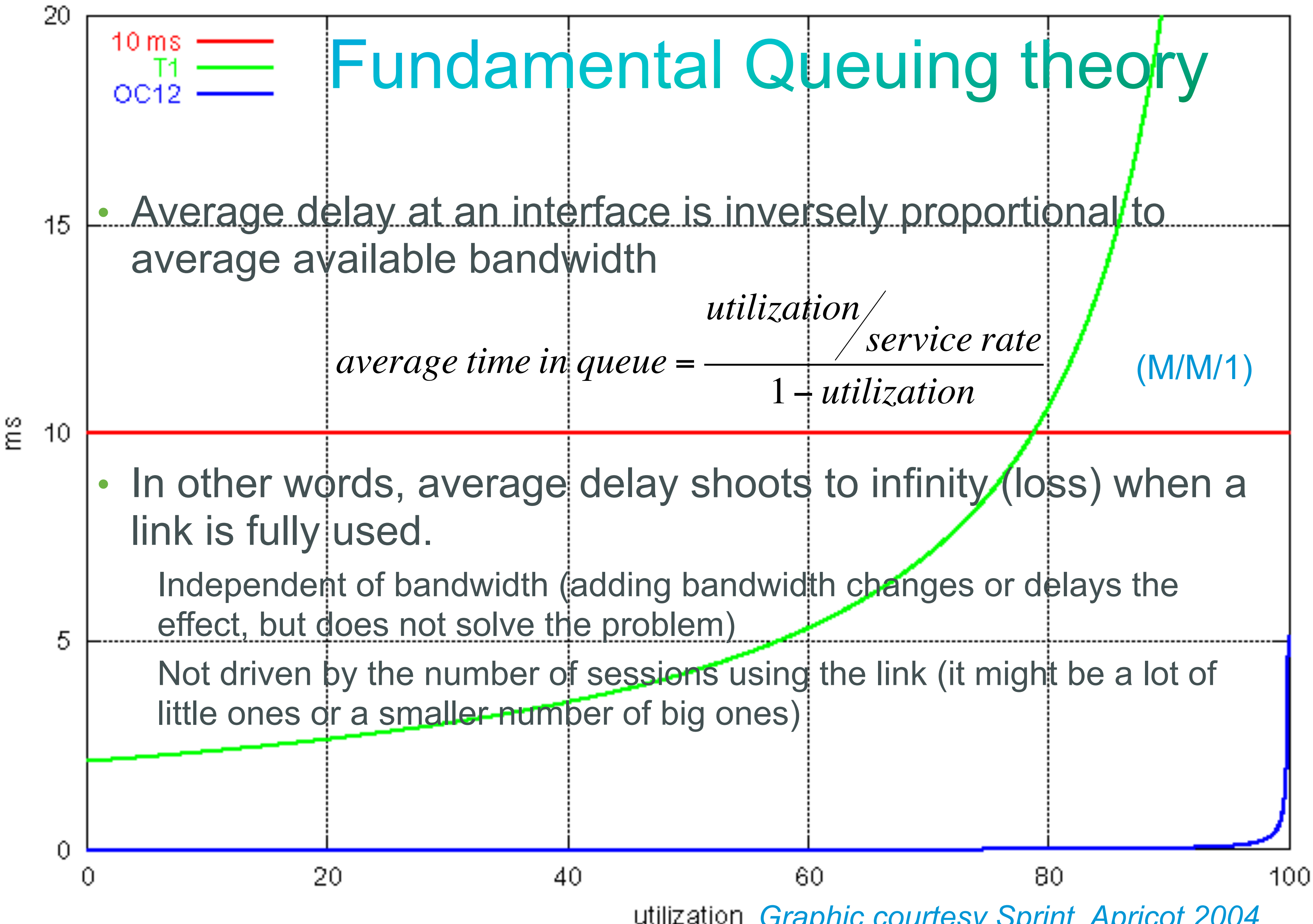
Fundamental Queuing theory

- Average delay at an interface is inversely proportional to average available bandwidth

$$\text{average time in queue} = \frac{\text{utilization} / \text{service rate}}{1 - \text{utilization}}$$

(M/M/1)

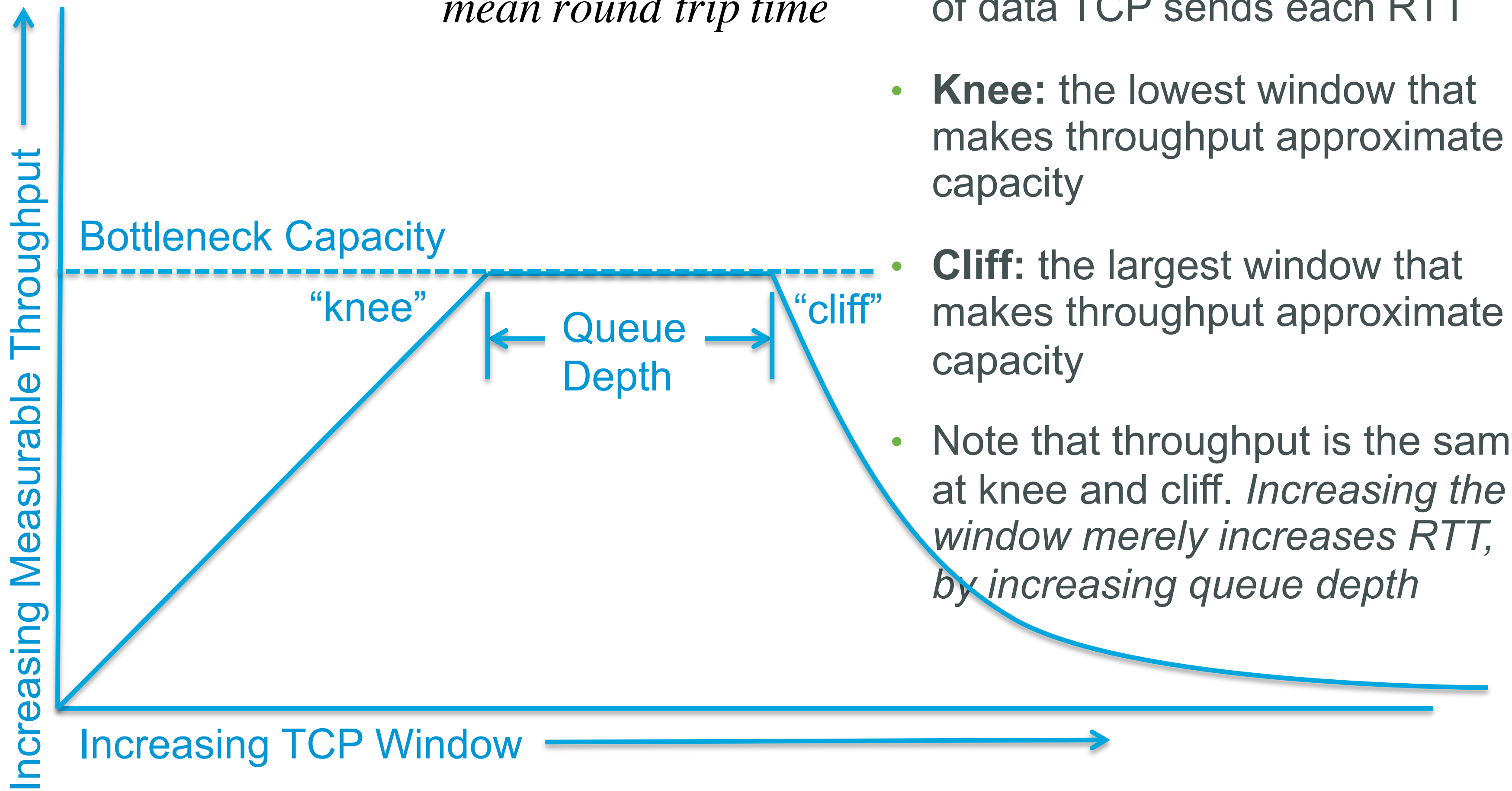
- In other words, average delay shoots to infinity (loss) when a link is fully used.
  - Independent of bandwidth (adding bandwidth changes or delays the effect, but does not solve the problem)
  - Not driven by the number of sessions using the link (it might be a lot of little ones or a smaller number of big ones)



Graphic courtesy Sprint, Apricot 2004

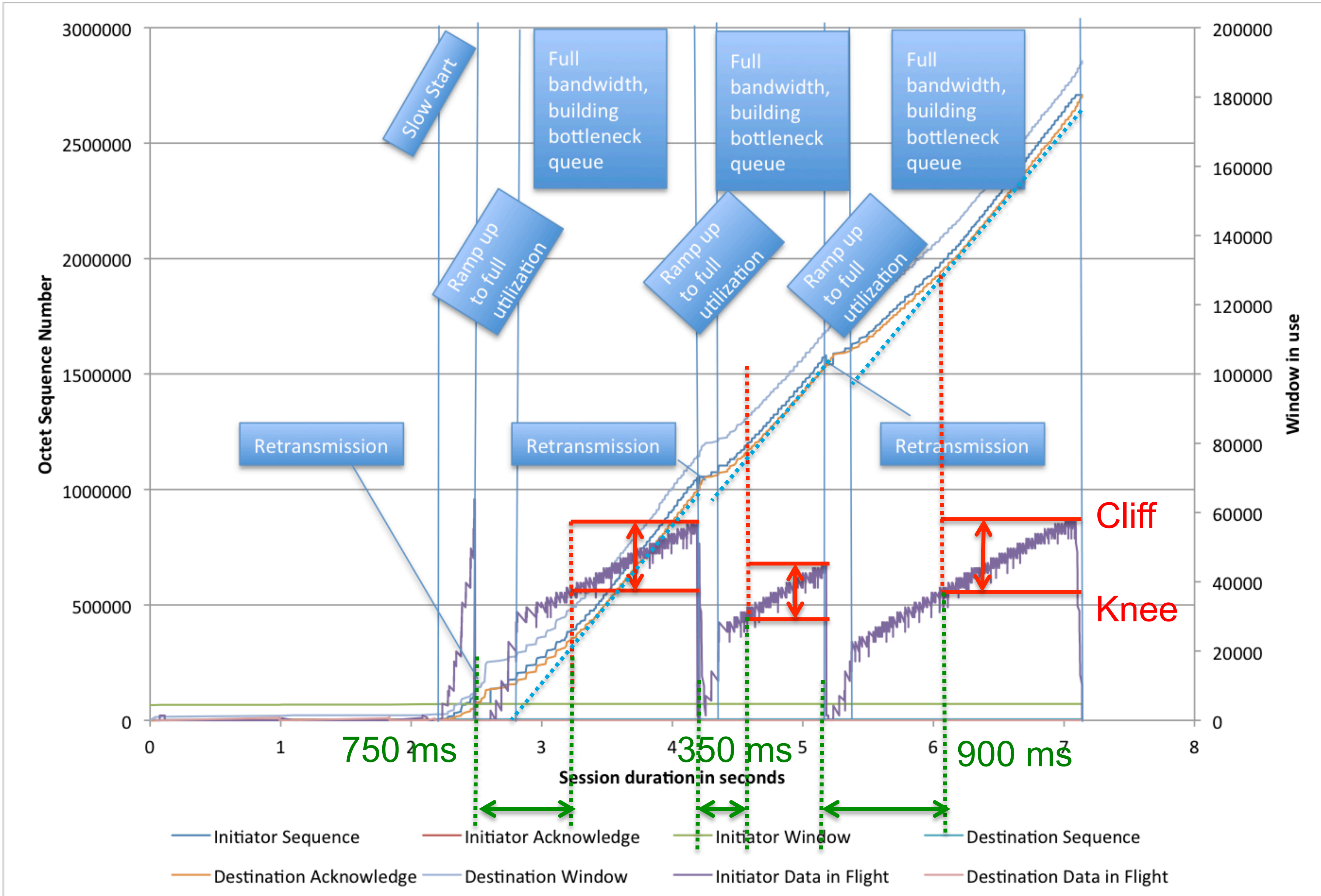
Simple model of TCP throughput dynamics

$$\text{mean throughput} = \frac{\text{effective window in bytes}}{\text{mean round trip time}}$$



- **Effective Window:** the amount of data TCP sends each RTT
- **Knee:** the lowest window that makes throughput approximate capacity
- **Cliff:** the largest window that makes throughput approximate capacity
- Note that throughput is the same at knee and cliff. *Increasing the window merely increases RTT, by increasing queue depth*

Yes, there is a more complex equation that takes into account loss. It estimates throughput above the cliff.



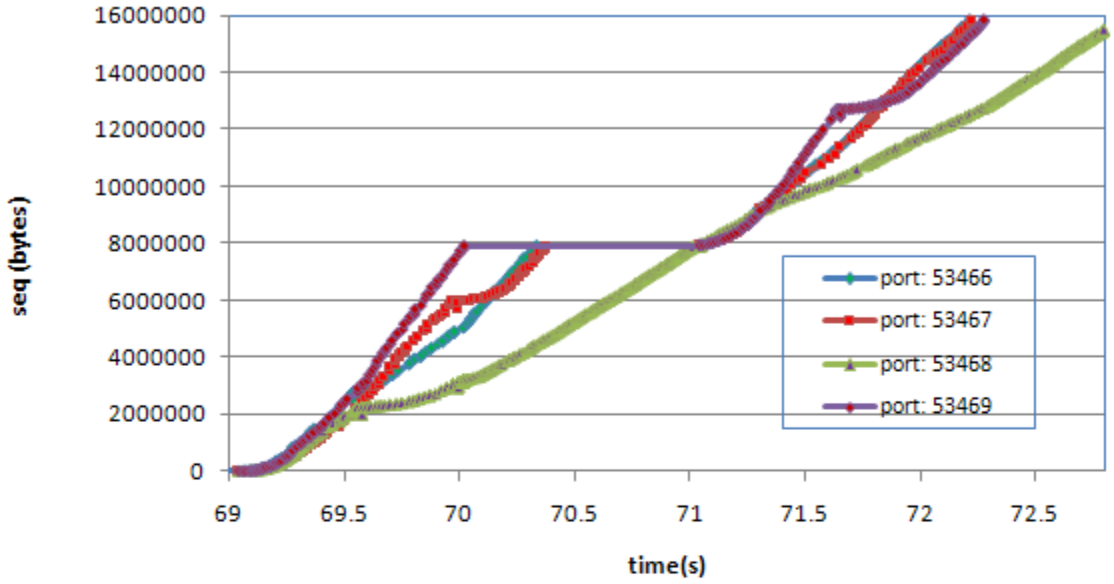
A typical TCP transfer

The effect of multiple TCPs in parallel

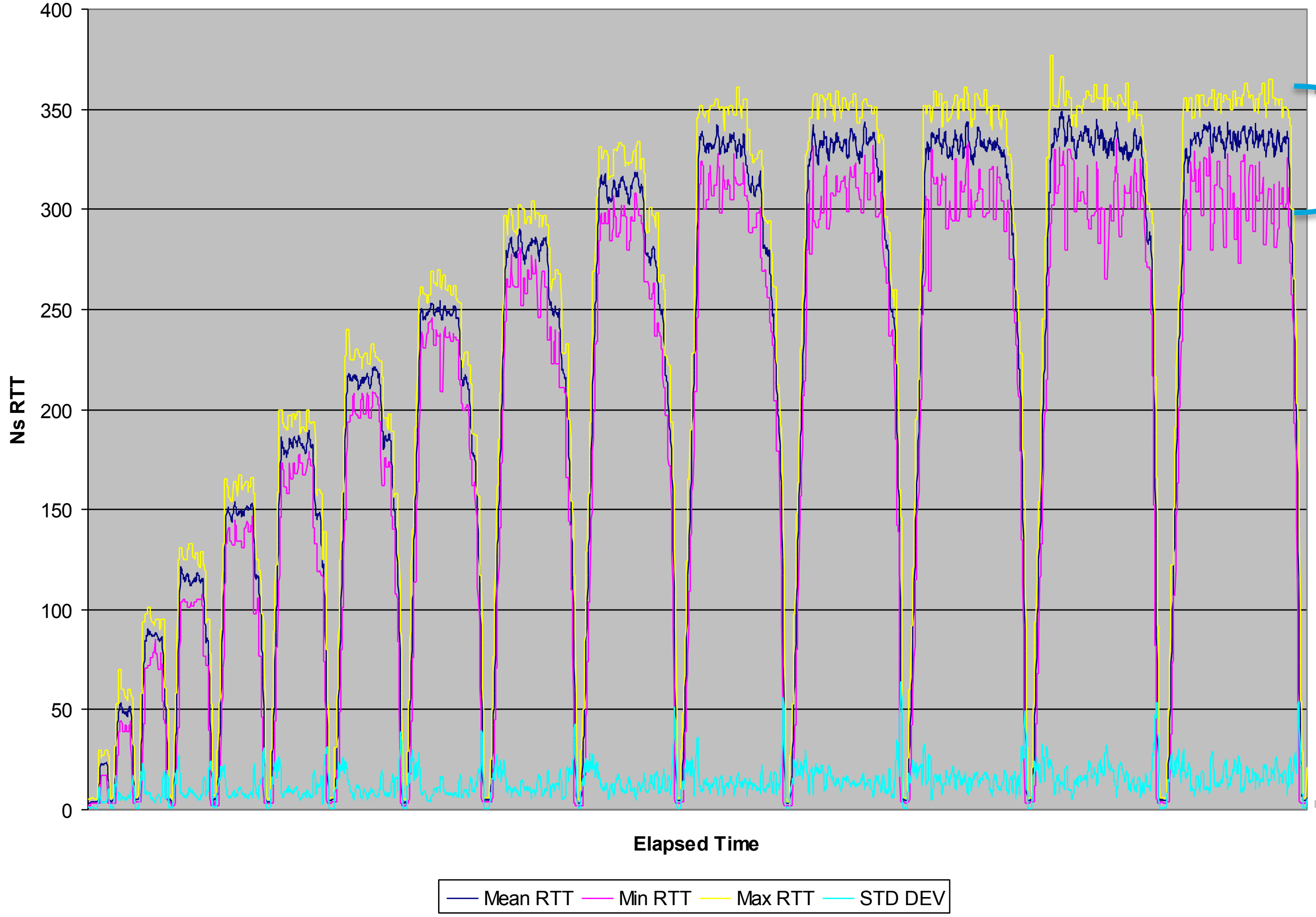
- Case: Multiple TCPs on startup, with one sustained transfer
- Note that

  The **capacity** is the sum of the throughput rates of the several initial TCPs

  The **throughput rate** of the sustained session is limited by TCP's low cwnd value and the rate it can increase (one segment/RTT)



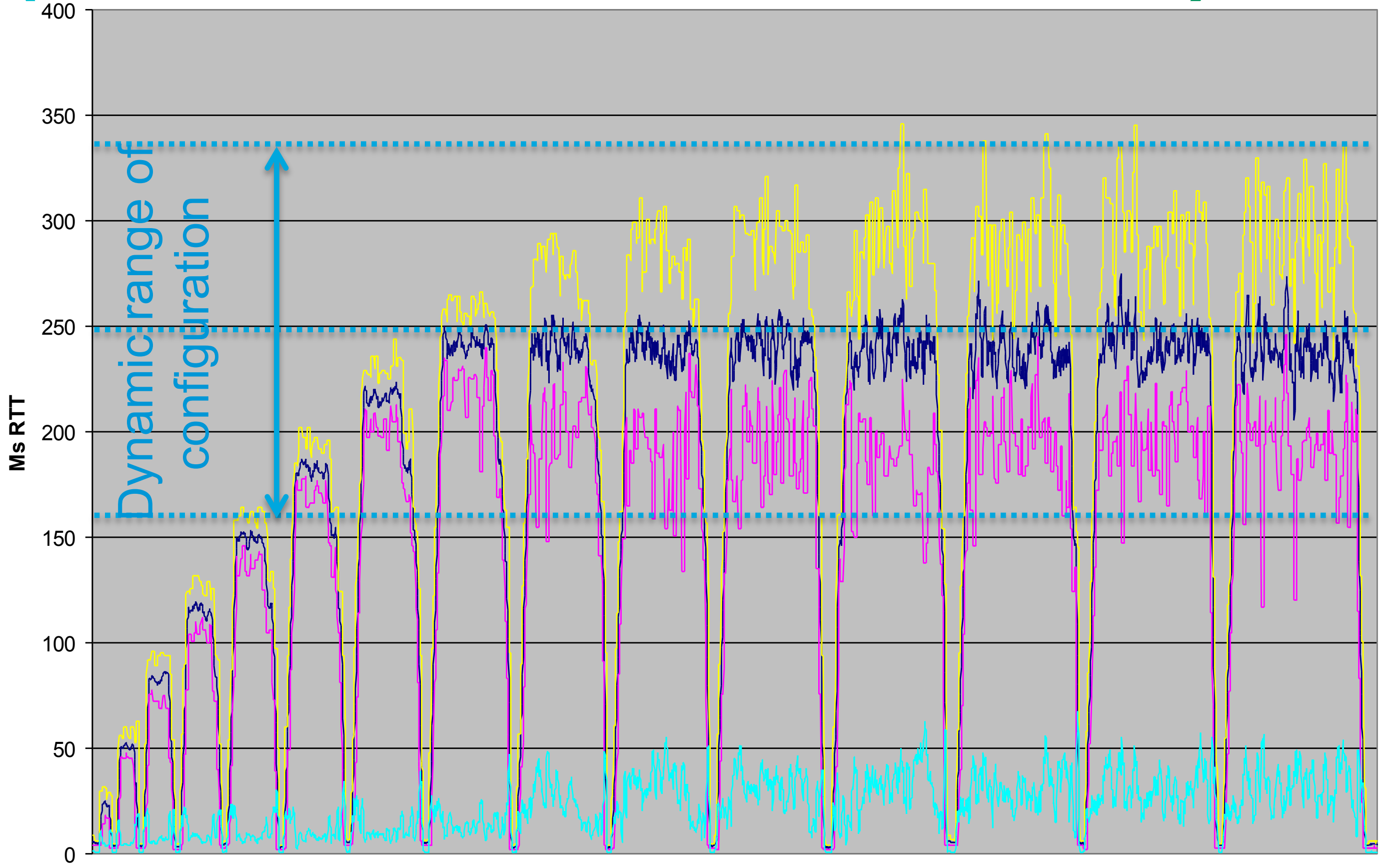
Tail Drop Traffic Timings



Typical variation in delay only at top of the queue

Mean Latency Correlates with Maximum Queue Depth

The objective: ***generate signals early (RED, Blue, AVQ, AFD, etc)***



Additional Capacity to Absorb Bursts

Mean Latency Correlates with target queue depth, min-threshold

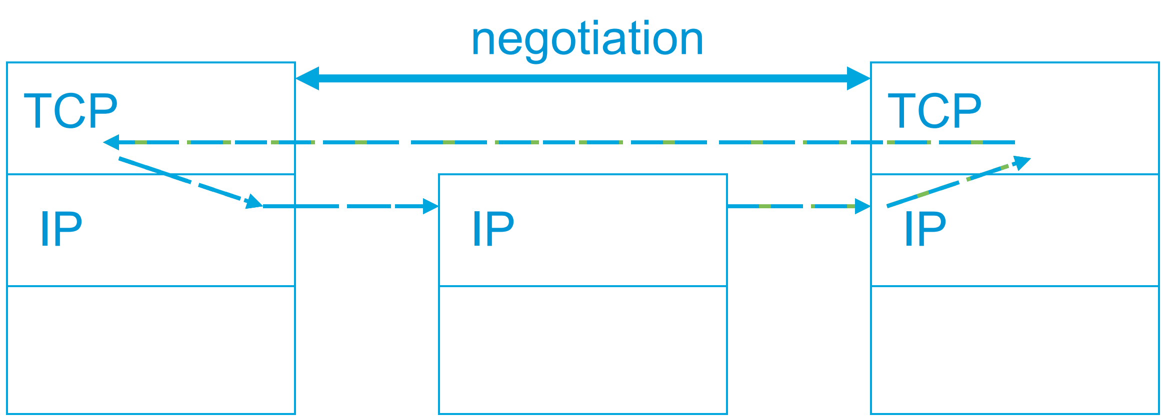
Solution approaches

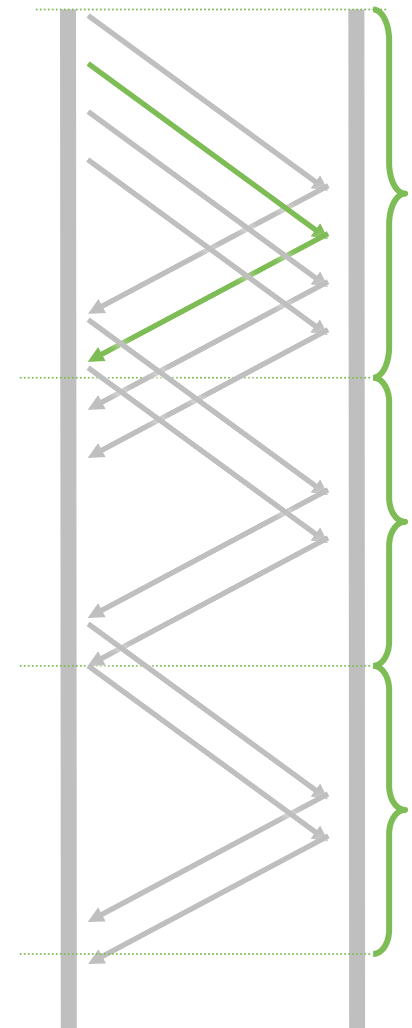
Three parts to the solution

- Bandwidth, provisioning, and session control
  - **If you don't have enough bandwidth for your applications, no amount of QoS technology is going to help.** QoS technology manages the differing requirements of applications; it's not magic.
  - For inelastic applications – UDP and RTP-based sensors, voice, and video, this means some combination of provisioning, session counting, and signaling such as RSVP
- Cooperation between network and host mechanisms for elastic traffic
  - TCP Congestion Control responds to **signals from** the network or **measurements of the network**
  - **Objective:** find an effective window such that **knee <= cwnd < cliff**
- Choices in network signaling
  - Loss – TCP responds to loss
  - Explicit Congestion Notification – lossless signaling from the network
  - Or not: delay-based congestion control doesn't depend on network signals

Avoiding loss: RFC 3168 Explicit Congestion Notification

- Data centers today testing ECN as a solution in order to avoid loss
- Problem: different handling of ECN by different OS's means it has inconsistent effects
- Problem: loss-based and ECN-based TCPs interact in "interesting" ways





Delay-based Congestion Control

- Arguably the most stable approach
- Several algorithms:
  - Vegas
  - CalTech FAST
  - Hamilton/Swinburne CAIA Delay-based Congestion Control
- Applicable to TCP, DCCP, or SCTP

- Vegas
  - Didn't work very well,
  - Tunes to knee plus alpha
- CalTech FAST
  - Simple,
  - IPR issues
  - Yields systemically to loss-based models,
  - Tunes to knee plus alpha

$$cwnd' := cwnd \times \frac{base\ RTT}{mean\ RTT} + \alpha$$

- Hamilton/Swinburne Delay Gradient
  - Implemented in FreeBSD 9.0 and later
  - Tunes to minimize *variation in delay* when it can, *loss* if it determines it is competing with a loss-based competitor

Delay-based Congestion Control

- Implementation of the algorithm proposed by Budzisz et al. [1] (we call it HD)
  - Probabilistic backoff based on inferred path queueing delay

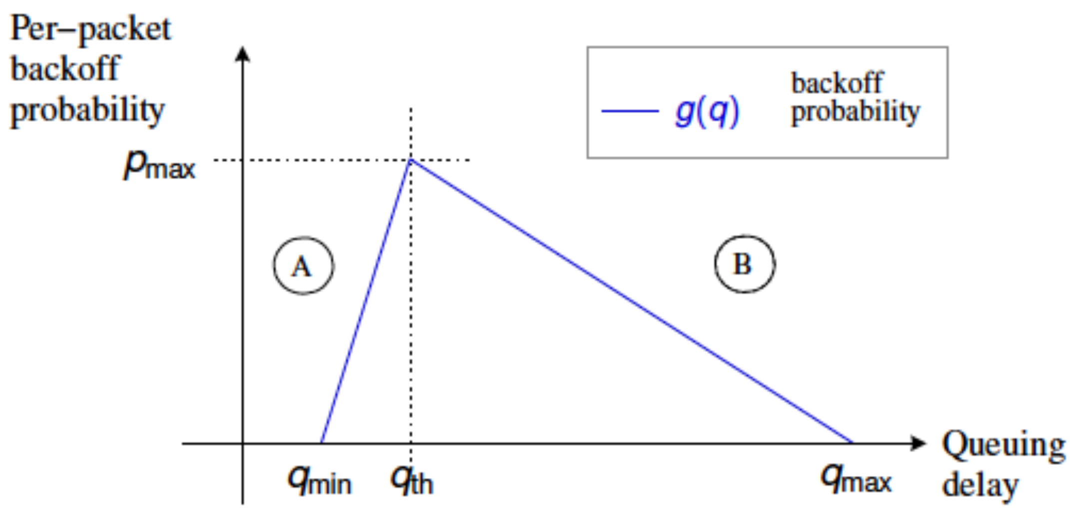
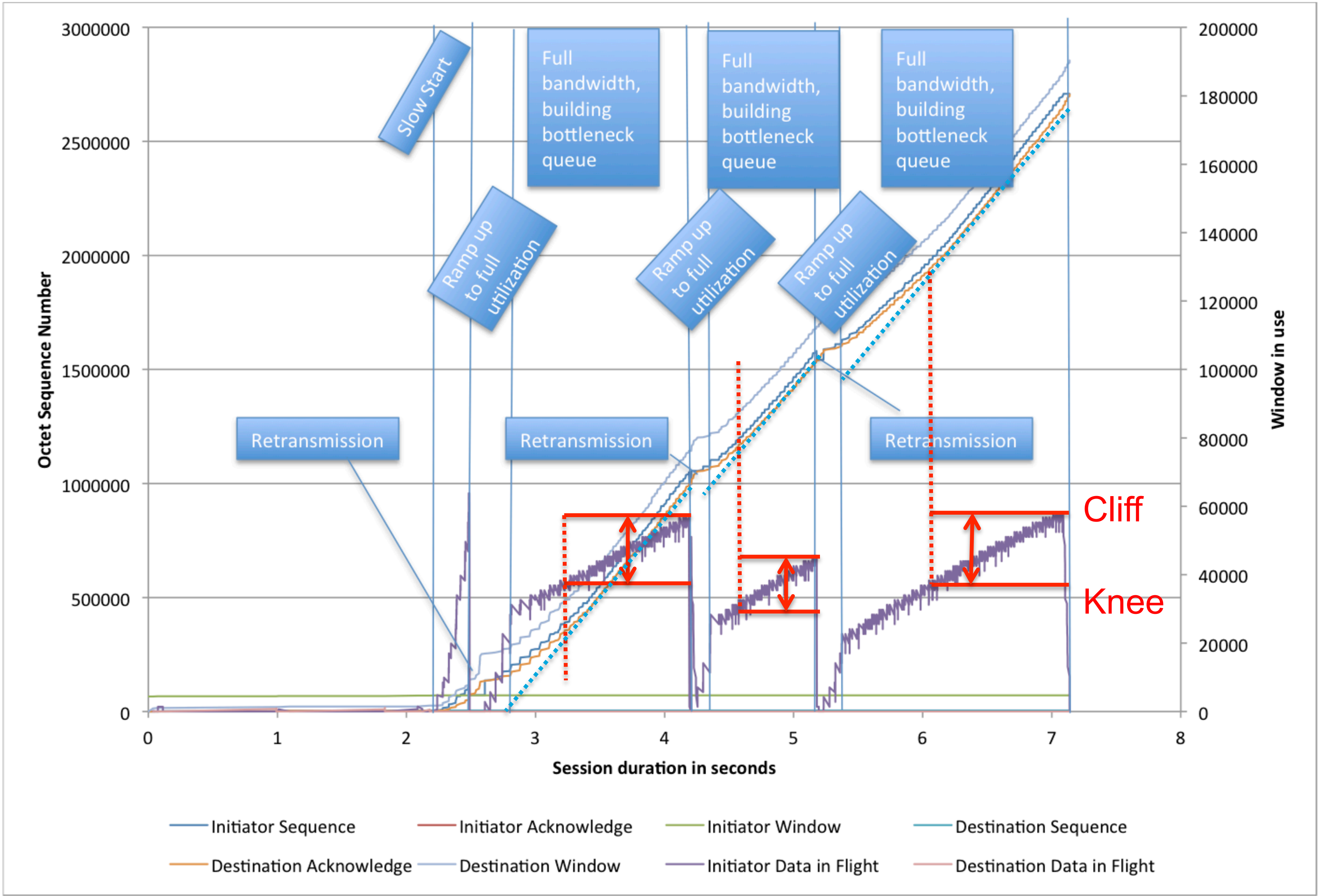


Figure: Per-packet backoff probability as a function of estimated queueing delay[1]



Concerns: noise in the measurement

Map/Reduce: Why TCP?

- I think application designers use TCP because they don't understand what is below the Socket API
- If separate requests use the same channel
  - Loss can happen even in ECN and Delay-based congestion control (although minimized)
  - Head of Line Blocking will still happen
- Could we separate the requests into separate streams?
  - **SCTP** Streams
  - Streams can deliver out of order (No HOL Blocking of subsequent request)
  - Other streams can give fast-retransmit events to blocked stream (reduce impact of HOL Block on a given request)

Looking for published research results on data center lemming migrations

For generic Internet use

- Objective:
  - Traffic is mice and elephants
  - Provide deep physical queue to allow for large bursts
  - Manage queue delay to low average
  - Manage in the direction of the knee
- Cisco looking at at AQM algorithms that are self-tuning
  - Looking at CoDel, etc
  - Not satisfied with what we have found so far
  - Looking at an algorithm based on DPLL design techniques
- Primarily interested in:
  - Mark/drop based on time in queue as opposed to queue depth
  - Relatively shallow marking threshold
  - Relatively deep early drop threshold

In data centers and content networks: Delay-based TCP/SCTP Congestion Control

- Objective:
  - Traffic includes mice and elephants, but contains many lemmings
  - Provide deep physical queue to allow for large bursts
  - Manage queue delay to low average
  - Manage in the direction of the knee
- In my dreams, hosts would implement a delay-based TCP/SCTP
  - SCTP streams used to minimize HOL blocking
  - TCP accepted on incoming sessions only
- CalTech FAST
  - Seeks to keep a quantum in the bottleneck queue
  - Adapts more rapidly than traditional congestion control – both up and down
  - Effective, but doesn't compete well with loss-based algorithms
- Swinburne CAIA Delay-based Algorithm
  - Seeks to keep bottleneck queue statistically empty
  - Competes well with loss-based algorithms

Thank you.