Congestion Avoidance and Control

Van Jacobson

Anshuman Mohan
CS6410 Fall ’25

Survey

What do you know about the architecture of the Internet?

In groups of two or three, take a minute to discuss!

Usual Suspects

Packet switching

Best-effort delivery

Three-way handshake

OSI model (7 abstraction layers)

History of the Internet

Independently conceived at ARPA, RAND Corp., and National Physical Lab

ARPA funded ARPANET (1969-90), making the USA the leader of the pack

A node was a Honeywell 516 minicomputer

Packet Switching is due to Donald Davies at NPL

The 1822 Protocol (1969), due to Bob Kahn, became ARPANET's communication protocol

History of the Internet

Independently conceived at ARPA, RAND Corp., and National Physical Lab

ARPA funded ARPANET (1969-90), making the USA the leader of the pack

A node was a Honeywell 516 minicomputer

Packet Switching is due to Donald Davies at NPL

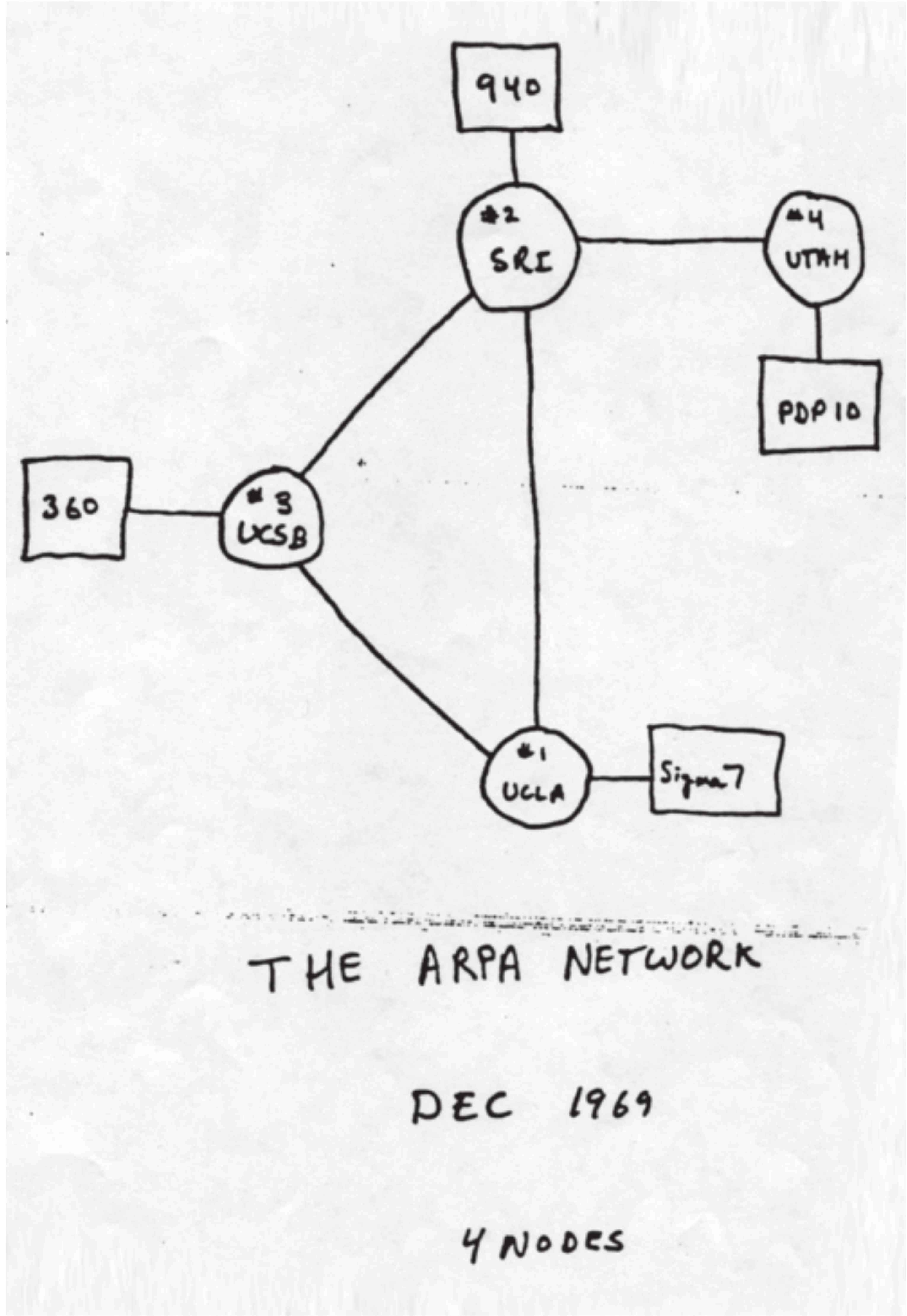
The 1822 Protocol (1969), due to Bob Kahn, became ARPANET's communication protocol

TCP and IP are due to Bob Kahn and Vint Cerf

History of the Internet

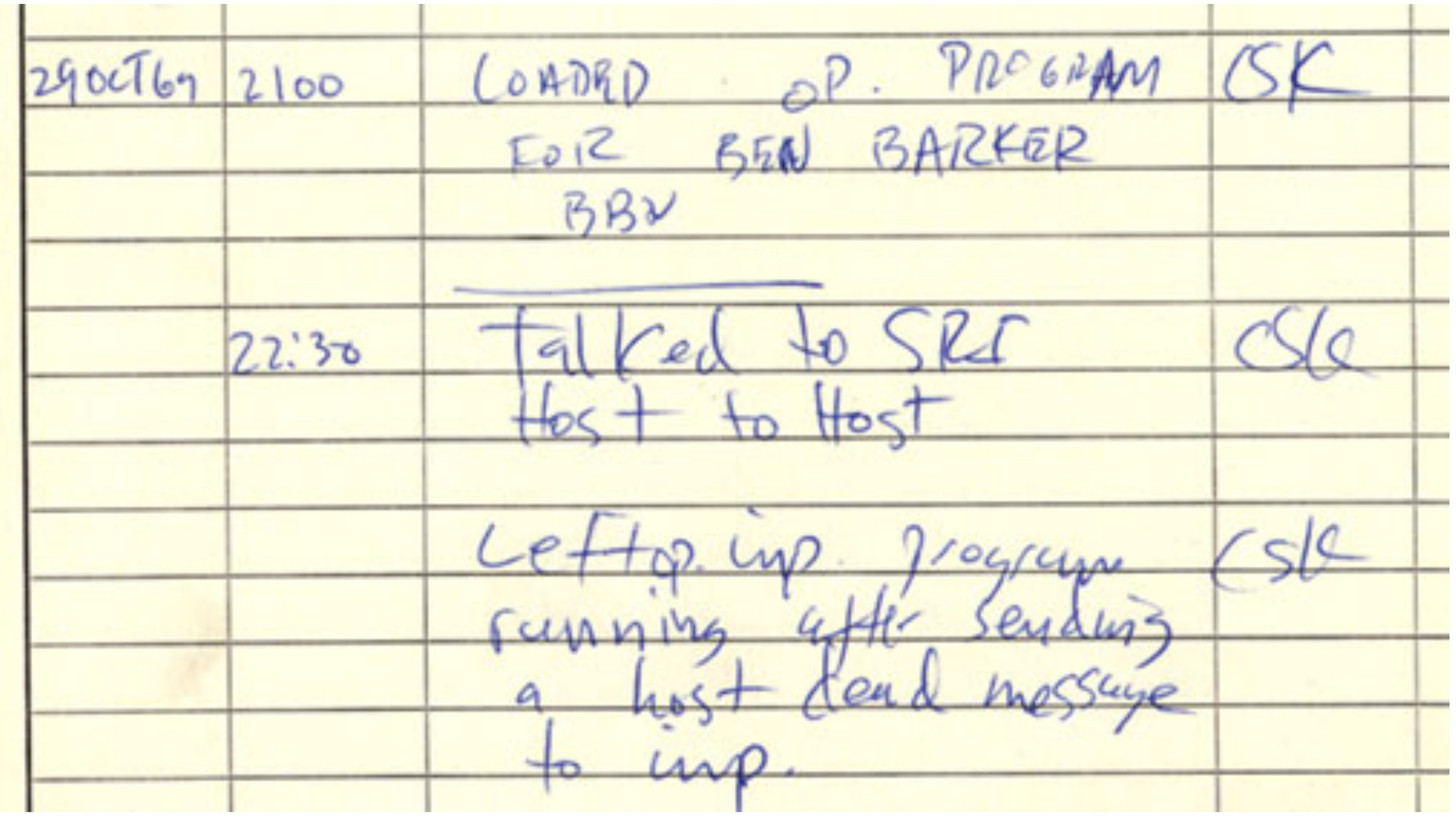
First computers connected electronically in 1969, mostly to debug the 1822 protocol

Sites:
UCLA, Stanford Research Institute,
UCSB, University of Utah



History of the Internet

First computers connected electronically in 1969, mostly to debug the 1822 protocol



Sites:
UCLA, Stanford Research Institute,
UCSB, University of Utah

First message: “lo”, from UCLA to SRI

Network “declared operational” in 1971

By 1973 we had telnet, ftp, and email

History of the Internet

March 1970: ARPANET reaches East Coast

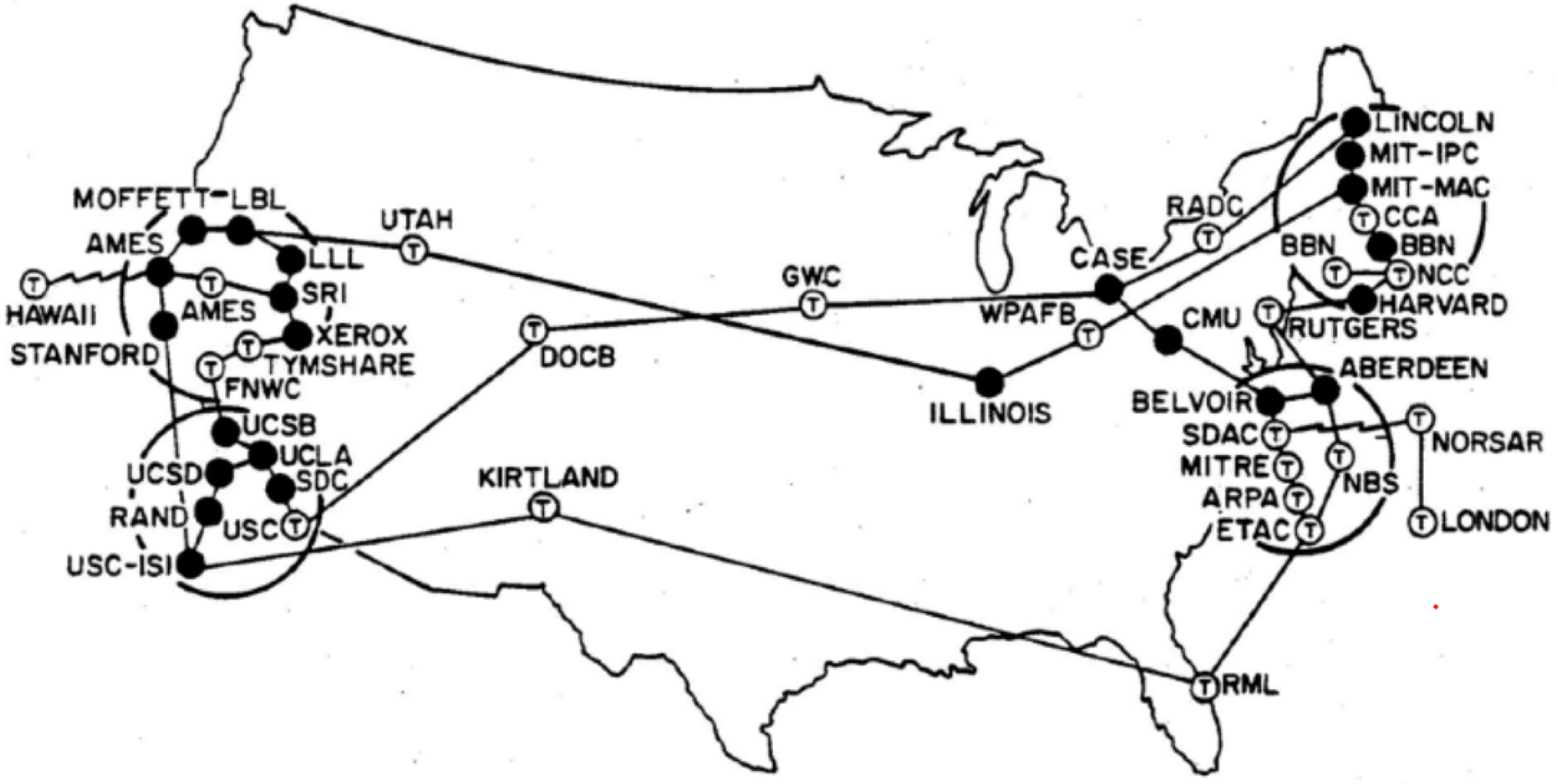
9 IMPs (interface message processors)
in June 1970, 18 IMPs in September 1970

In June 1973, ARPANET and NPL were connected by
satellite link, via NORSAR in Sweden

ARPANET had 213 IMPs in 1981, “with another host
connecting approximately every twenty days”

Rapid expansion fueled by NSF-funded CSNET (1981)

e Internet

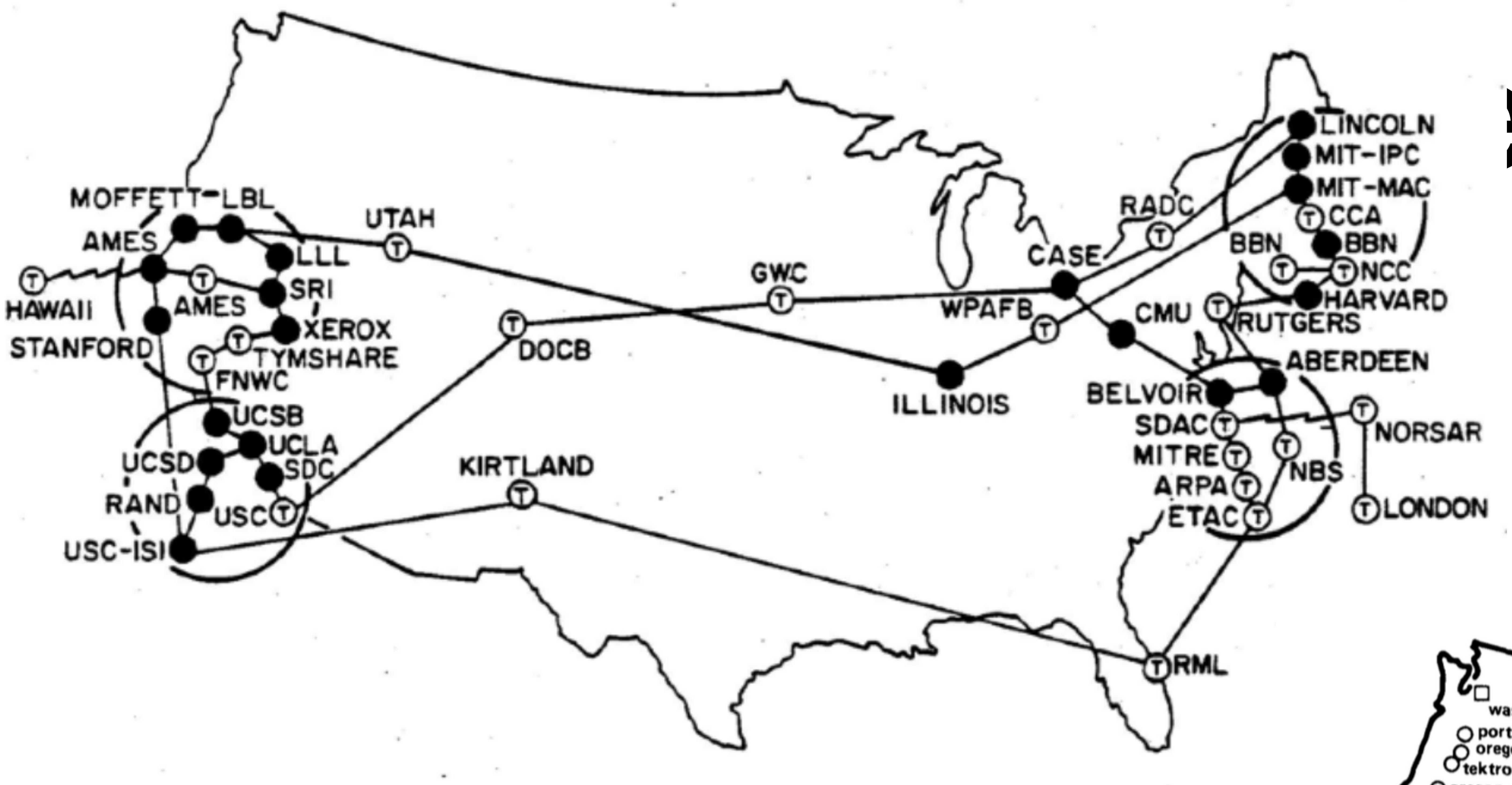


ted by
satellite link, via NORSAR in Sweden

ARPANET had 213 IMPs in 1981, “with another host connecting approximately every twenty days”

Rapid expansion fueled by NSF-funded CSNET (1981)

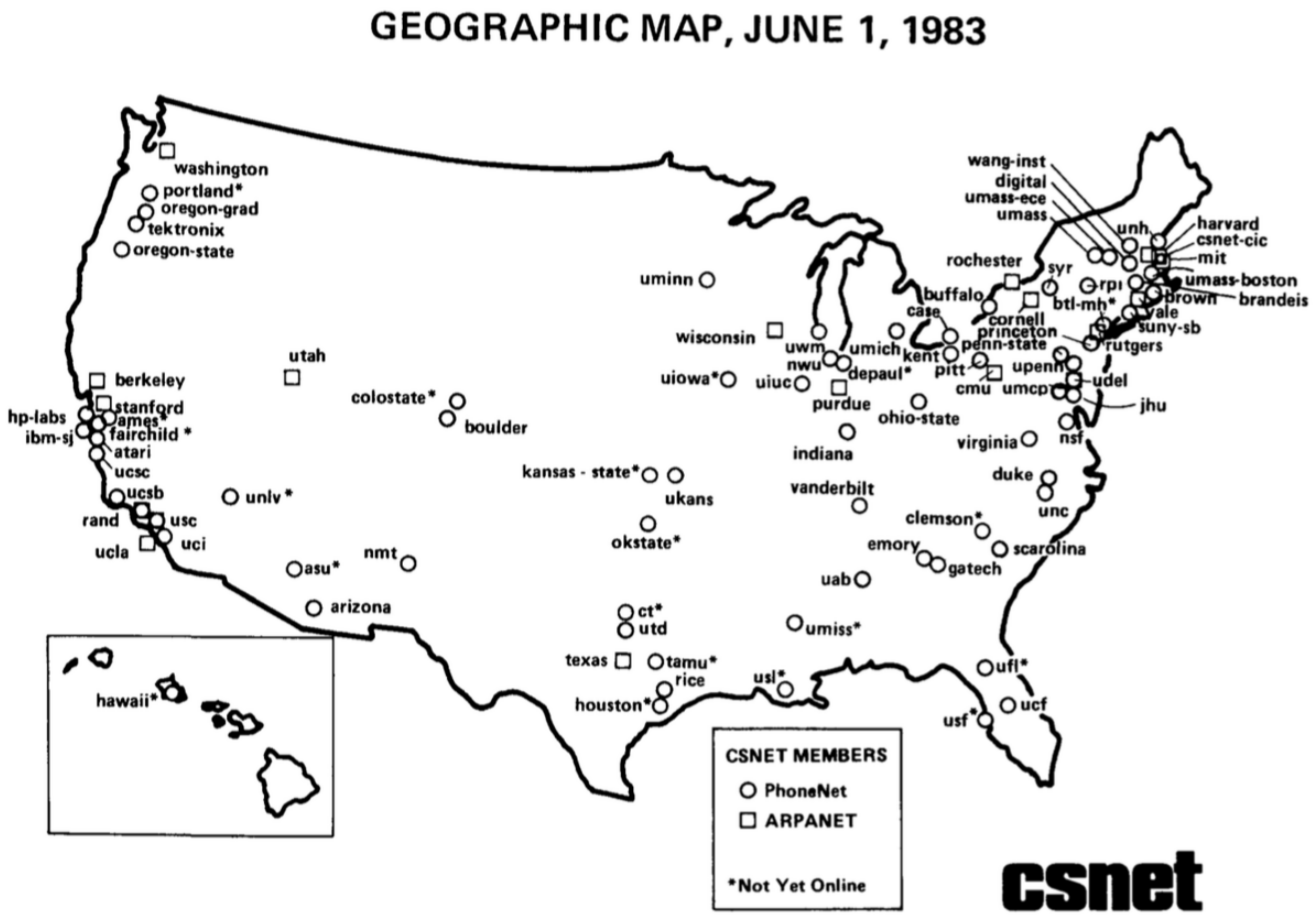
Internet



satellite link, via NORSAR in Sweden

ARPANET had 213 IMPs in 1981, “with
connecting approximately every twenty

Rapid expansion fueled by NSF-funded



Loss of Innocence: October 1986

First instances of congestion collapse

Throughput drops from 32 Kbps to 40 bps

Is it TCP? Yes!

Can it be fixed in general? Also yes!

Congestion Avoidance and Control

Van Jacobson

SIGCOMM ’88
9.6k citations
Not VJ’s most-cited work (more on that later)

Van Jacobson

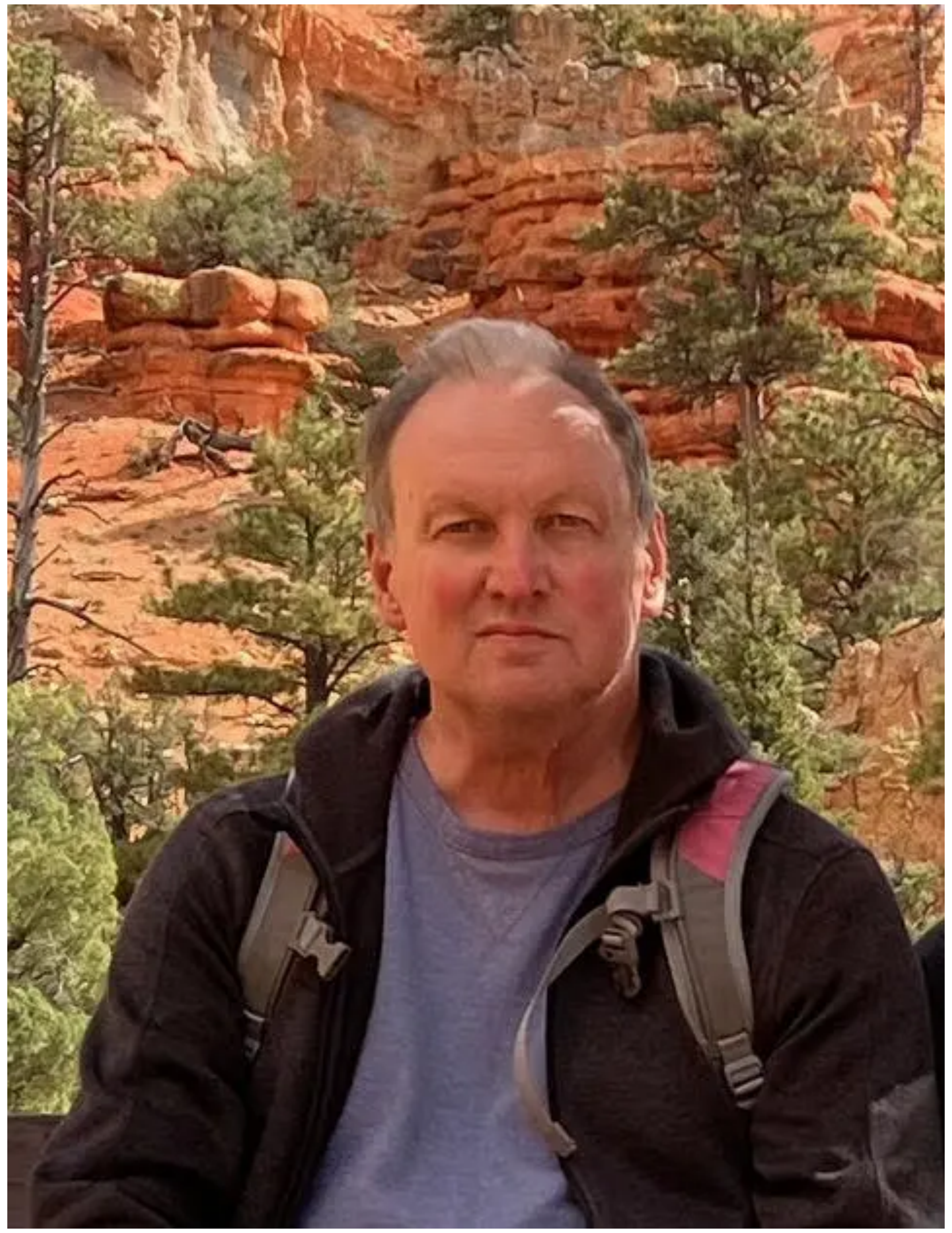
In huge part responsible for the Internet as we know it!
Especially through his work on TCP/IP

UArizona → Lawrence Berkeley National Lab →
Cisco → Packet Design, Inc. → Xerox PARC →
Named Data Networking Consortium (present?) →
Google(present?)

SIGCOMM Lifetime Achievement Award
Koji Kobayashi Award
Internet Hall of Fame

Mike Karels

Notre Dame → UC Berkeley →
Berkeley Software Design, Inc. →
Secure Computing Corporation →… → Retired,
volunteered for FreeBSD Foundation



“Pivotal figure in the history of BSD UNIX, a
respected member of the FreeBSD community”

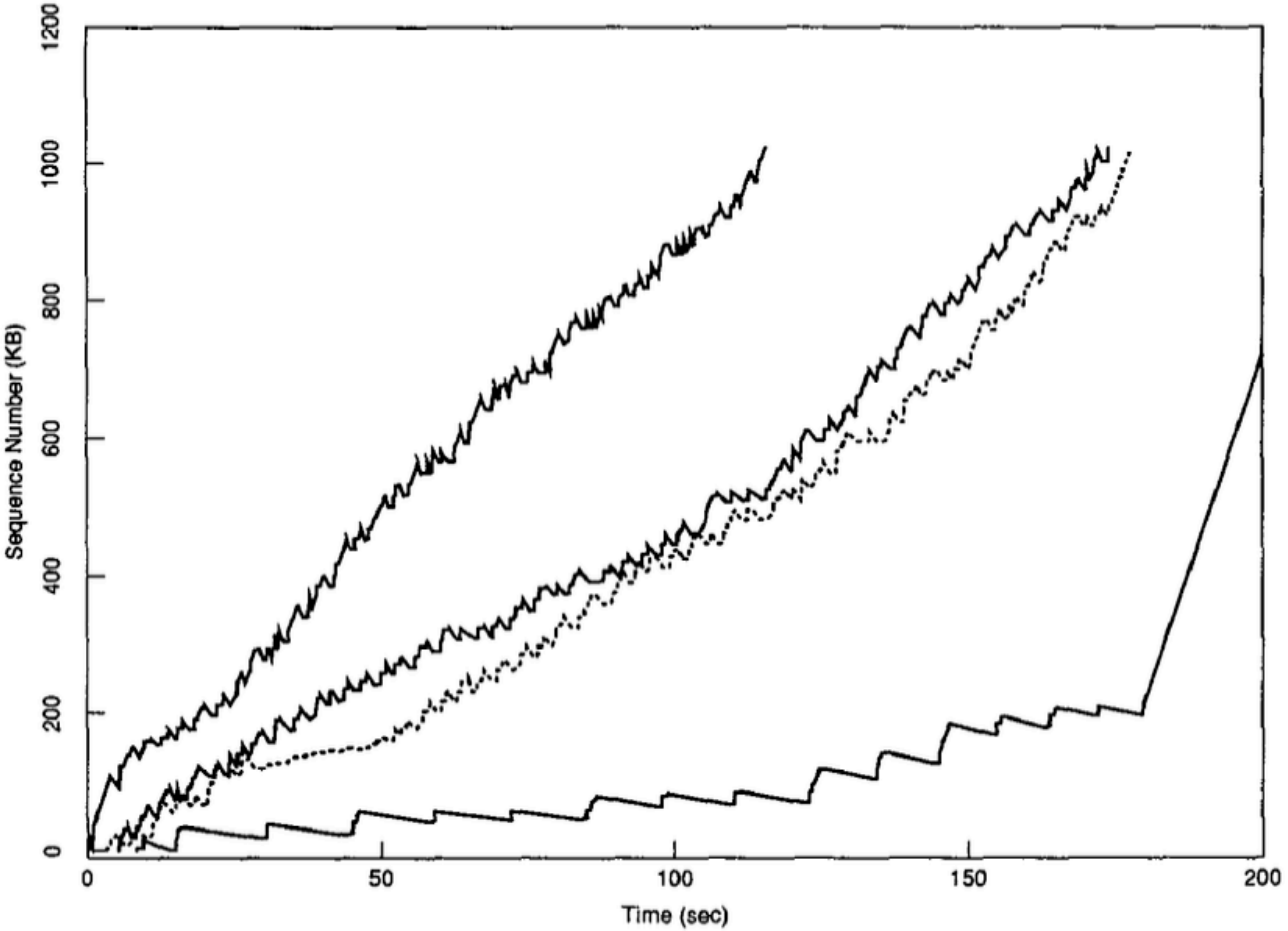
Issue

There is high traffic, and some links are slower than others

Packets are dropped or delayed, especially over those slower links

Additional communications are triggered by those drops and delays

But that communication *is also traffic*, and this only makes things worse



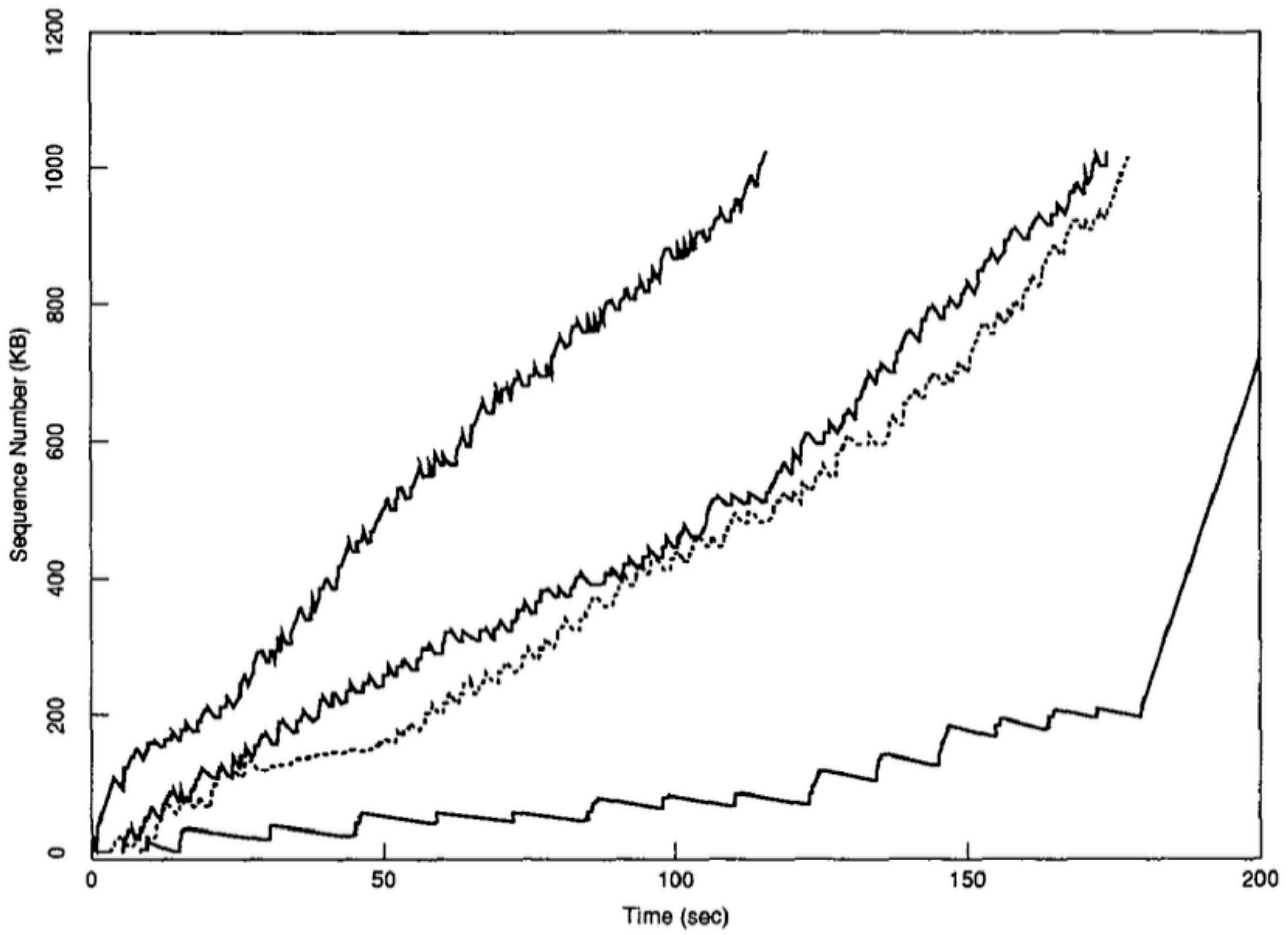
Sequence Number (KB)
0
200
400
600
800
1000
1200
0
50
100
150
200
Time (sec)

Issue

Four simultaneous TCP conversations, w/o congestion avoidance

4k out of 11k are retransmissions

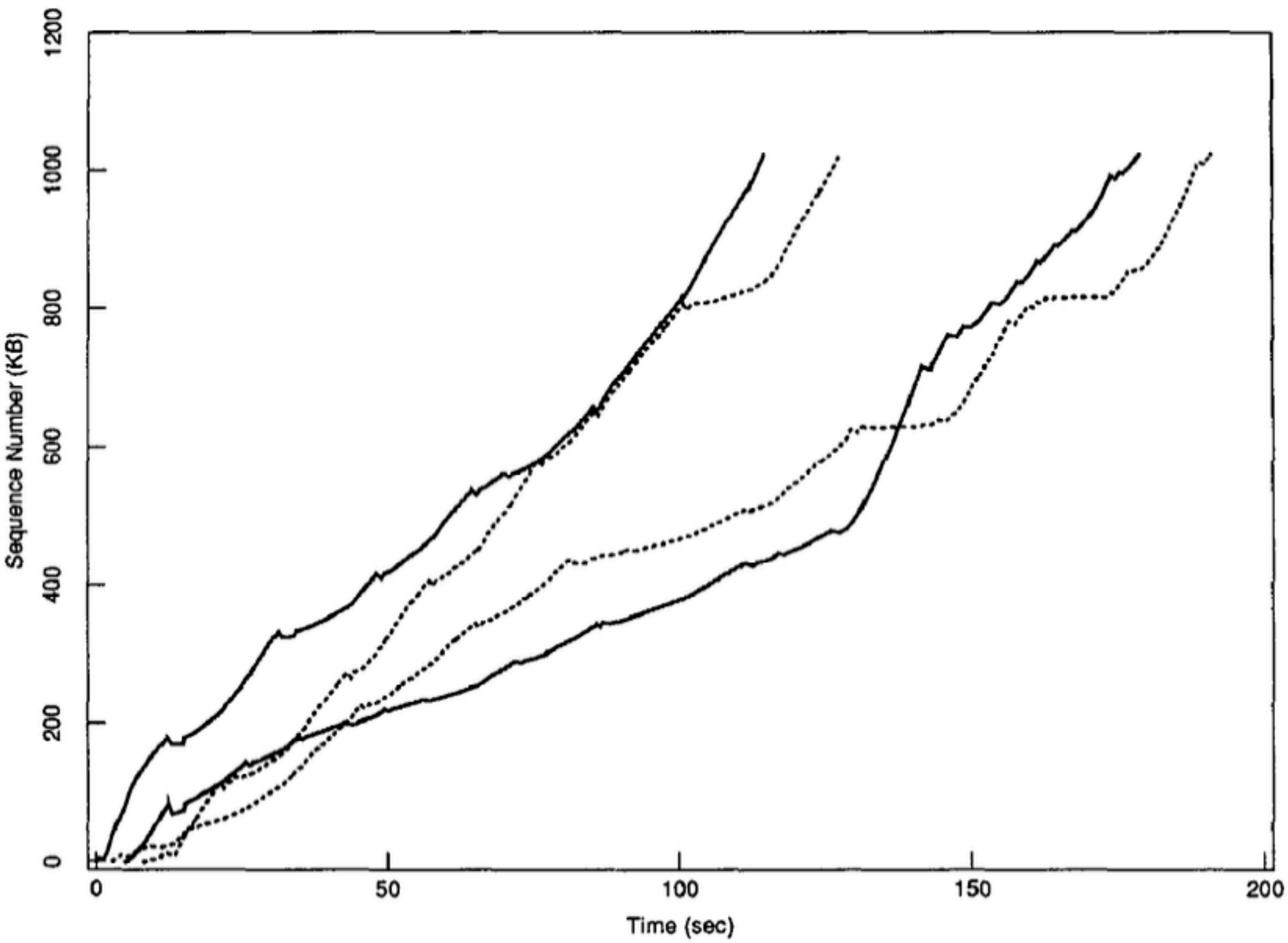
25KBps link is shared as:
8, 5, 5, 0.5
and the rest (6KBps) has vanished



Solution

Small changes at sender/receiver have powerful emergent effects!

Concretely: seven new algorithms, all mostly small interventions



Sequence Number (KB)
1200
1000
800
600
400
200
0
0
50
100
150
200
Time (sec)

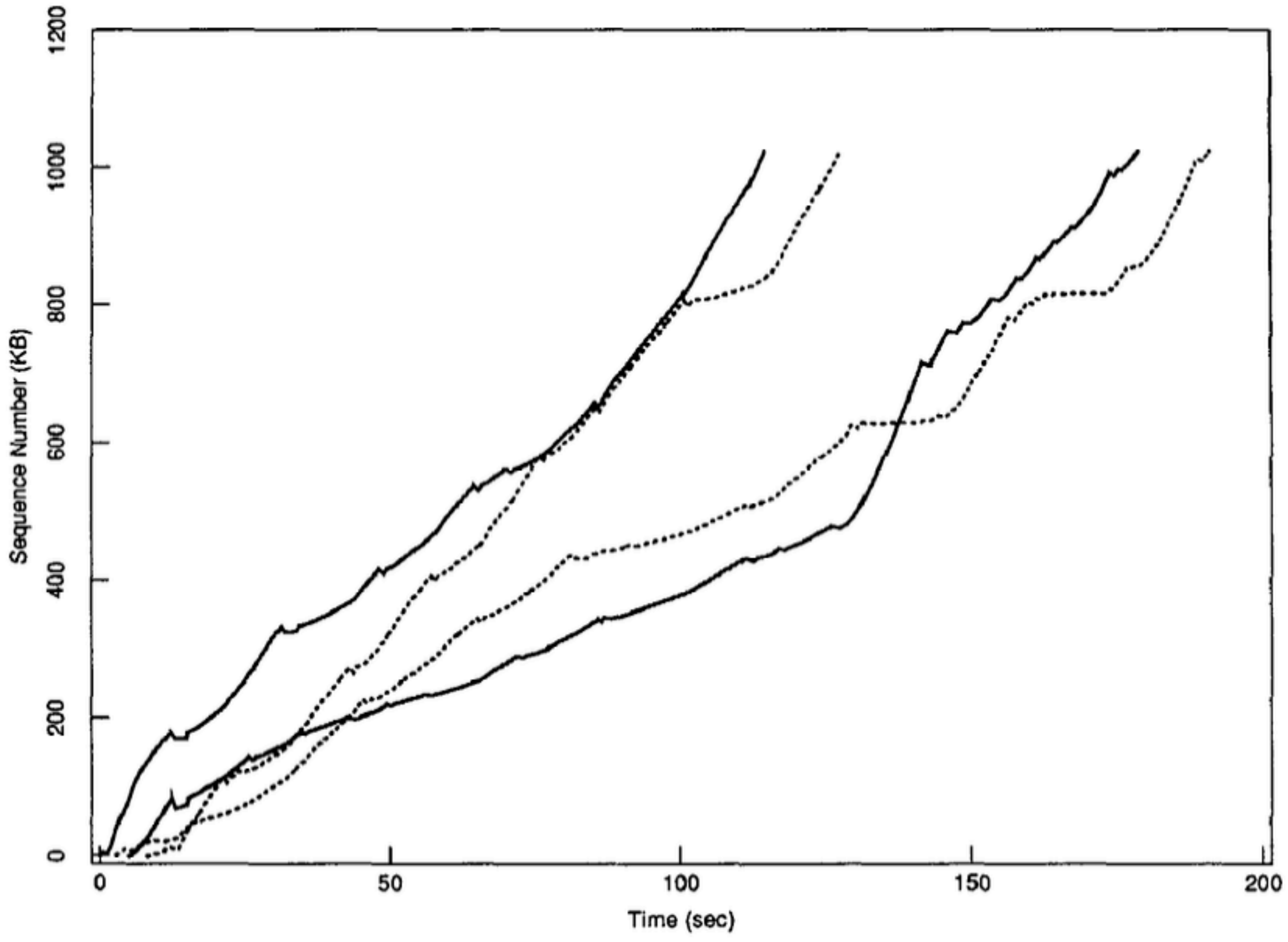
Solution

Four simultaneous TCP conversations, with congestion avoidance

89 out of 8281 are retransmissions

25KBps link is shared as:
8, 8, 4.5, 4.5
and there was no wastage

Difference due to receivers



Conservation of Packets

A simple idea:

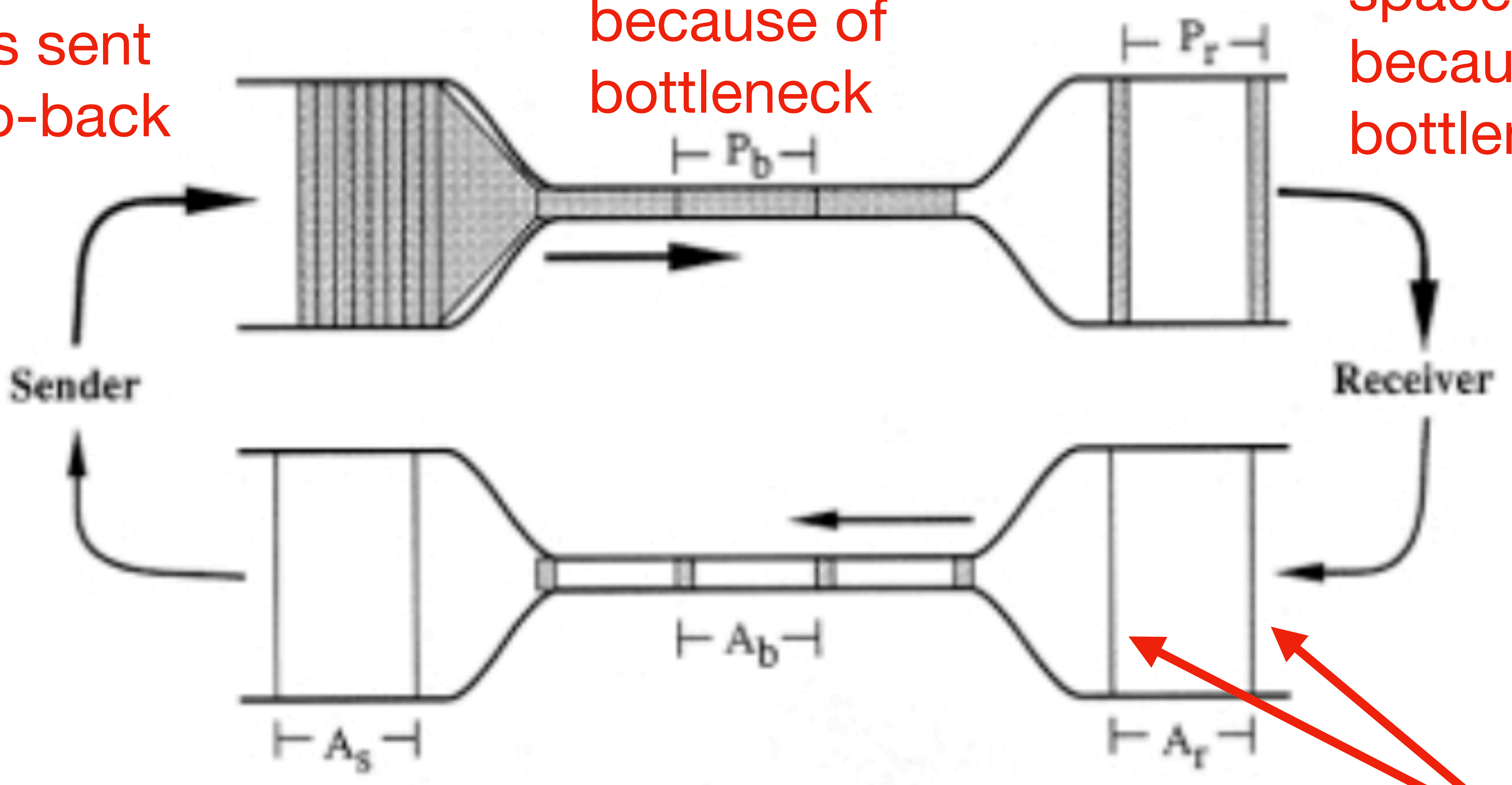
If everything is stable and the “window” of data is full…
… no new packet should be inserted until a old packet leaves.

Violations of Conservation

packets sent back-to-back

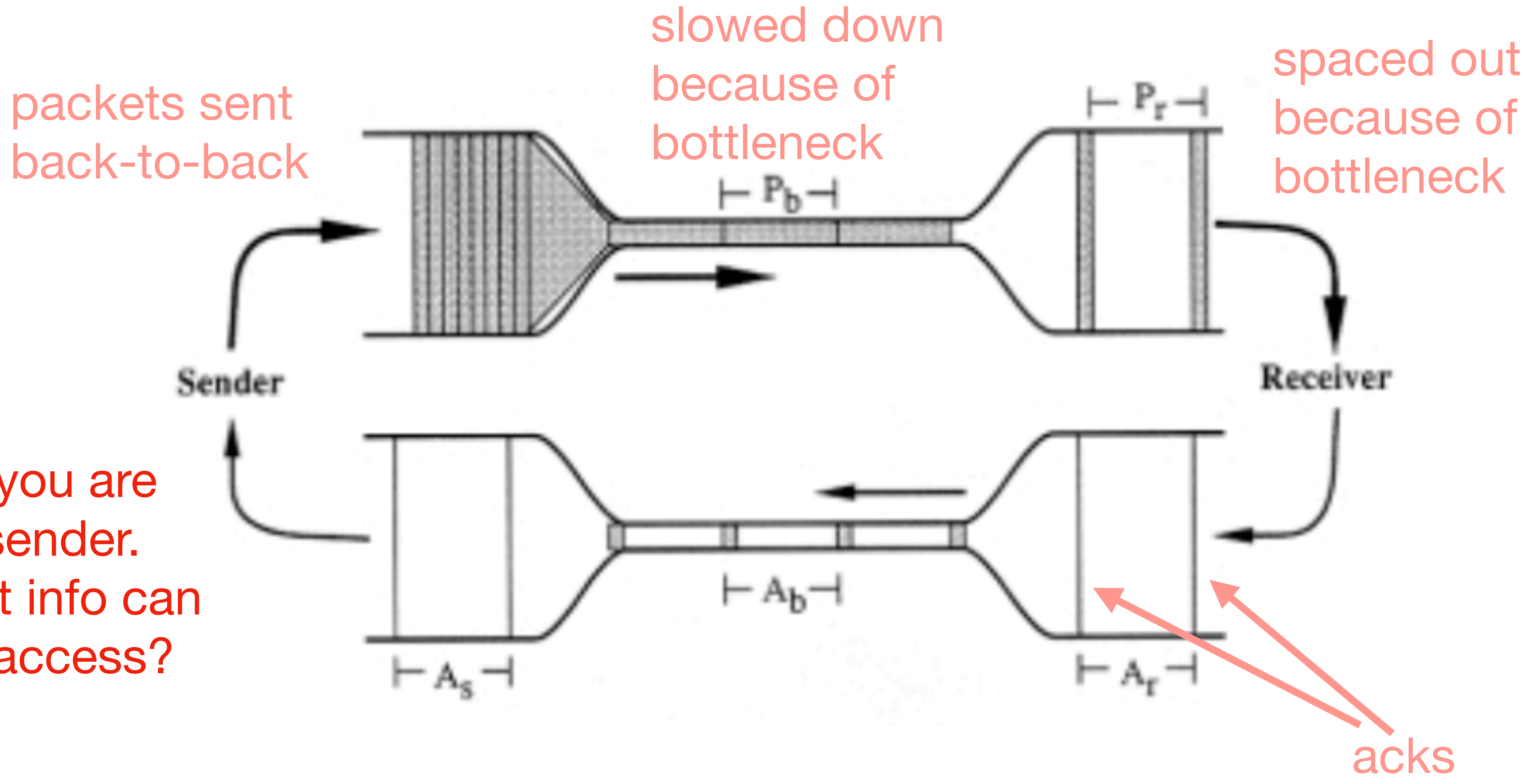
slowed down because of bottleneck

spaced out because of bottleneck



acks

Discussion



Say you are the sender. What info can you access?

Violations of Conservation

The connection doesn't get to equilibrium

A sender injects a new packet before an old packet has exited

The equilibrium can't be reached because of resource limits along the path

Violations of Conservation

The connection doesn't get to equilibrium because of the sender

Equilibrium is reached, but the sender violates it

~~A sender injects a new packet before an old packet has exited~~

The connection doesn't get to equilibrium because of resource limits

~~The equilibrium can't be reached because of resource limits along the path~~

Van Jacobson’s Interventions

The connection doesn’t get to equilibrium because of the sender

Equilibrium is reached, but the sender violates it

The connection doesn’t get to equilibrium because of resource limits

Van Jacobson’s Interventions

The connection doesn’t get to equilibrium because of the sender

- Slow start

Equilibrium is reached, but the sender violates it

- RTT variance estimation
- Exponential retransmit timer backoff

The connection doesn’t get to equilibrium because of resource limits

- More aggressive ACK policy
- Dynamic window resizing

Van Jacobson's Interventions

The connection doesn't get to equilibrium because of the sender

- Slow start

Equilibrium is reached, but the sender violates it

- RTT variance estimation
- Exponential retransmit timer backoff

The connection doesn't get to equilibrium because of resource limits

- More aggressive ACK policy
- Dynamic window resizing

Slow Start

The sender relies on ACKs to figure out its rate of sending

Think of ACKs as a low-res “view” into the murky network

Once it’s running, this simple system is nice and stable

The problem is getting it running

Slow Start

Sending too much traffic at startup only clogs things up

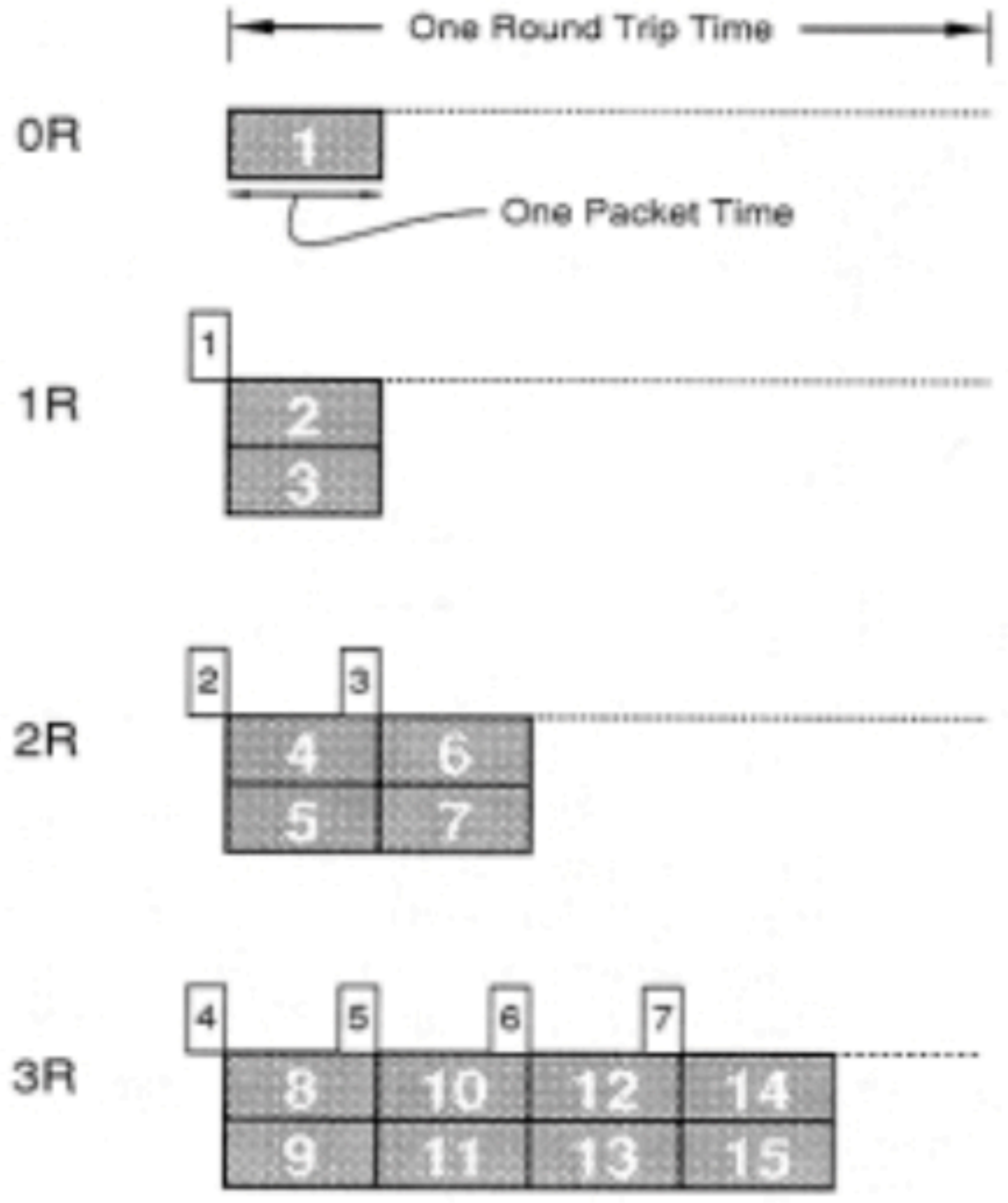
Solution: gradually ramp up sender-side traffic

Specifically, a remarkably simple strategy proves to be exactly correct:
- Maintain a congestion window *cwnd* per connection
- Set *cwnd* to 1 at start/restart
- After each ACK for new data, increment *cwnd*
- Send **`min`**(*cwnd*, $\text{window}_{\text{recv}}$)

Hey, this is exponential! So our gradual ramp up isn't even that gradual

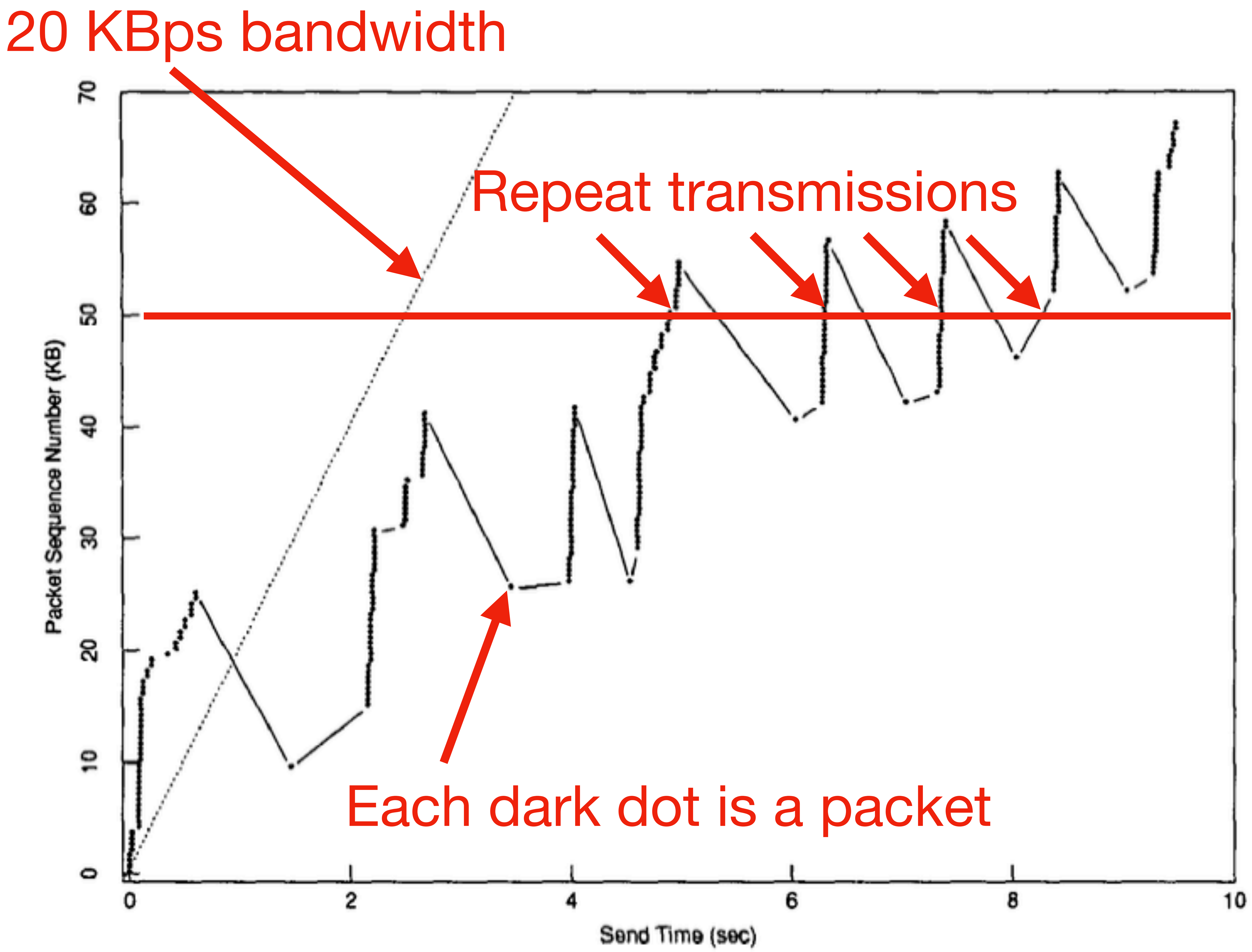
Slow Start

- Maintain a congestion window *cwnd* per connection
- Set *cwnd* to 1 at start/restart
- After each ACK for new data, increment *cwnd*
- Send **`min`**(*cwnd*, $\text{window}_{\text{recv}}$)

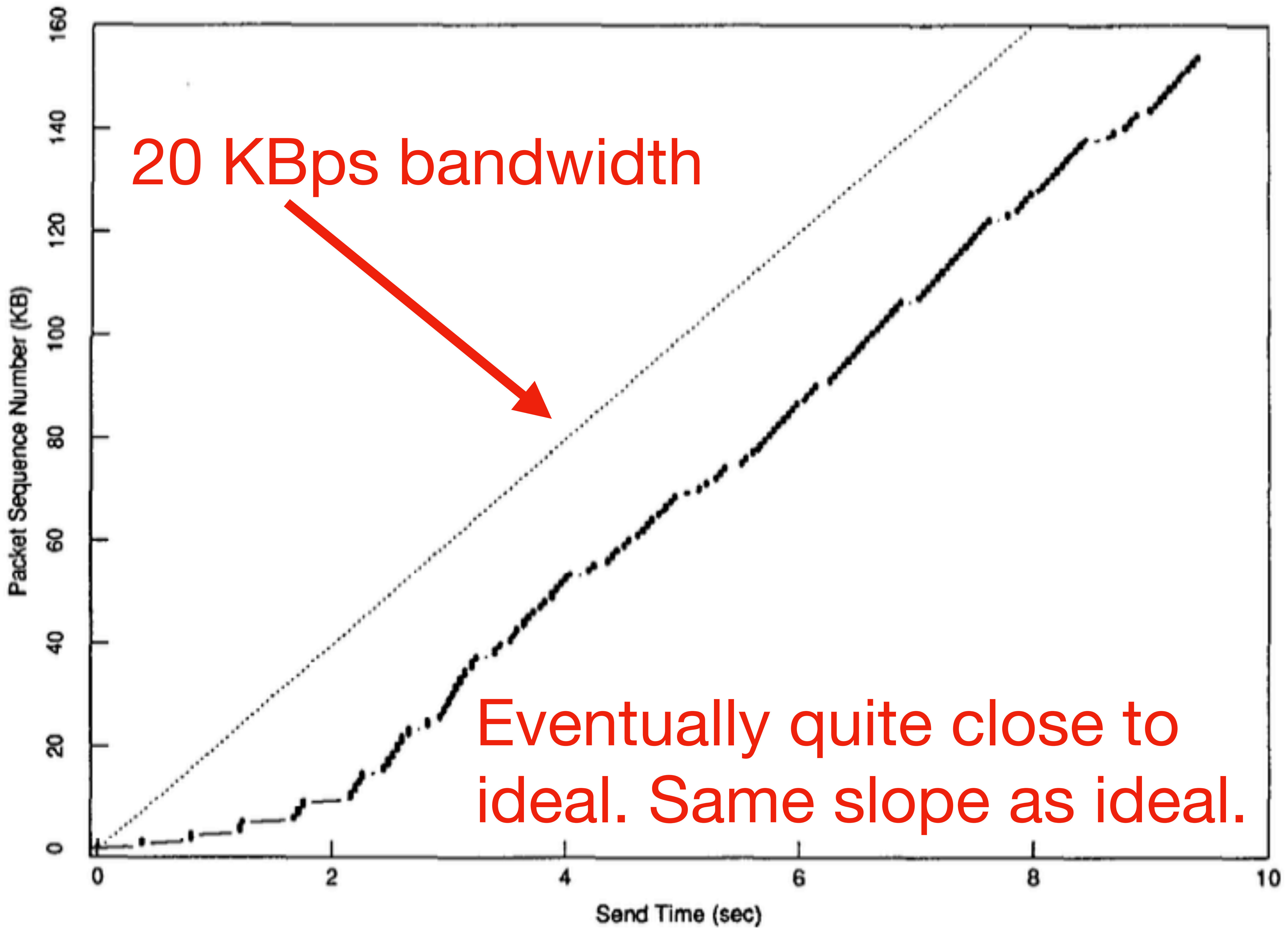


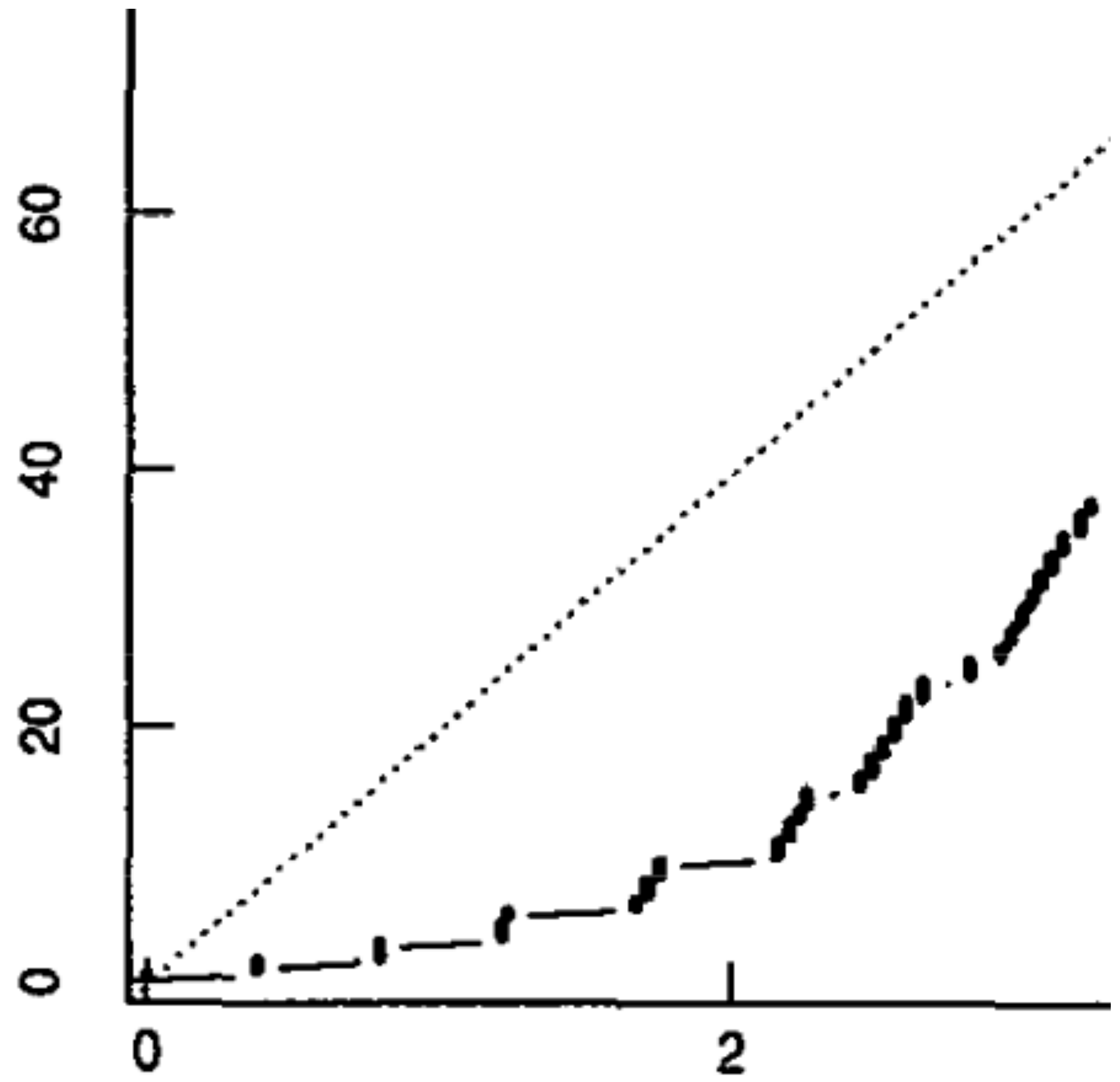
Slow Start

"Without slow-start, when 10 Mbps Ethernet hosts talk over the 56 Kbps Arpanet via IP gateways, the first-hop gateway sees a burst of eight packets delivered at 200 times the path bandwidth. This burst of packets often puts the connection into a persistent failure mode of continuous retransmissions."



Slow Start





Zooming into the slow start: not ideal. But not so bad, and not terribly long.

Van Jacobson’s Interventions

The connection doesn’t get to equilibrium because of the sender

- Slow start

Equilibrium is reached, but the sender violates it

- RTT variance estimation
- Exponential retransmit timer backoff

The connection doesn’t get to equilibrium because of resource limits

- More aggressive ACK policy
- Dynamic window resizing

RTT Variance Estimation

Why would a well-meaning sender ever violate equilibrium?

Because it believes that the packet has been lost, so it retransmits

When the original packet actually *has* been lost, this is okay!

When the original packet is enroute but delayed, this is wasteful:
“This is the network equivalent of pouring gasoline on a fire.”

RTT Variance Estimation

The sender has a *round trip time estimator*

This needs to be aware of *variations* in RTT

And those variations are very counterintuitive!
“If the network is running at 75% of capacity, […]
one should expect RTT to vary by…”
- trick question; it won’t vary because we’re below 100% capacity
- factor of 4
- factor of 16

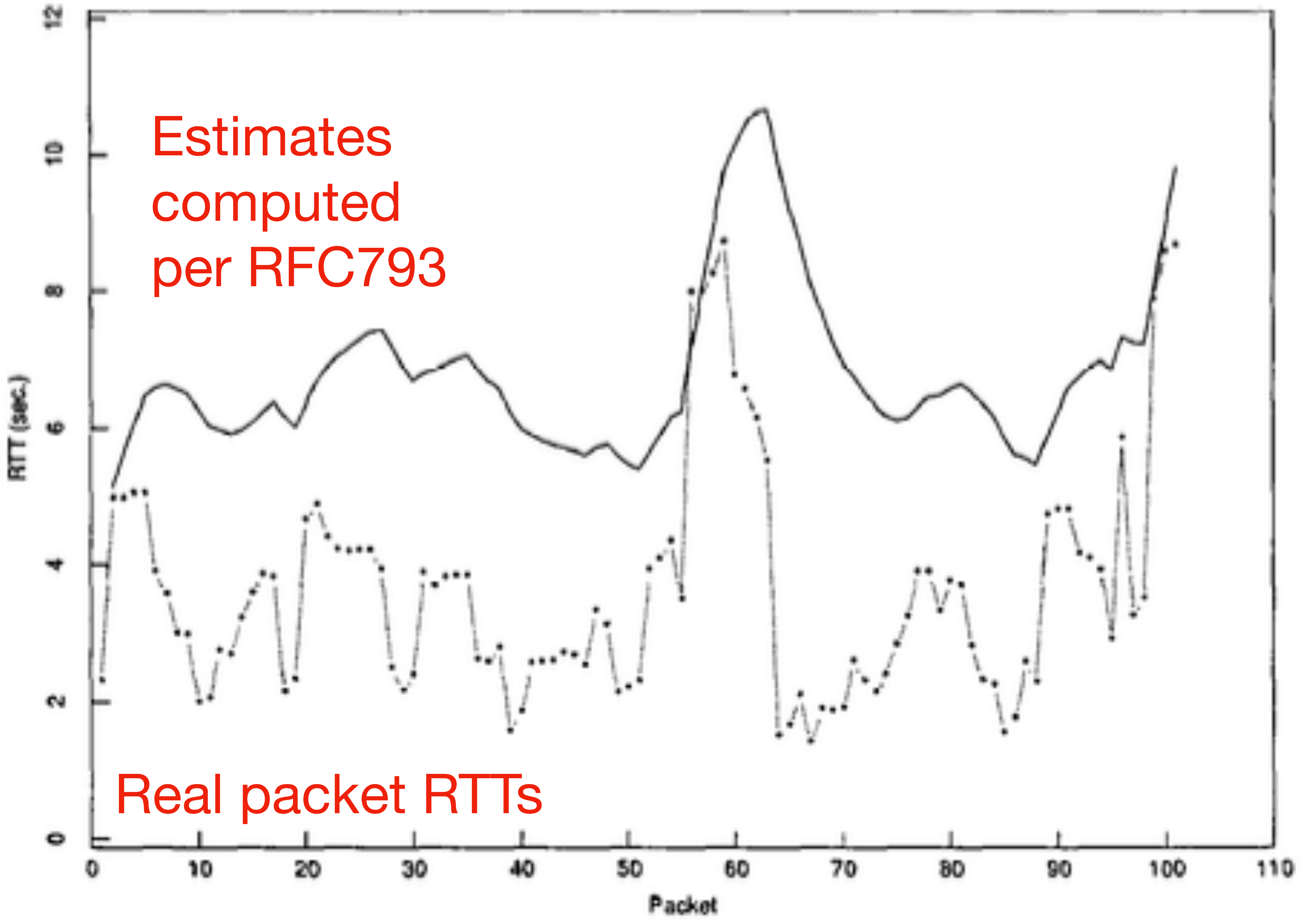
RTT Variance Estimation

“If the network is running at 75% of capacity, […] one should expect RTT to vary by a factor of 16.”

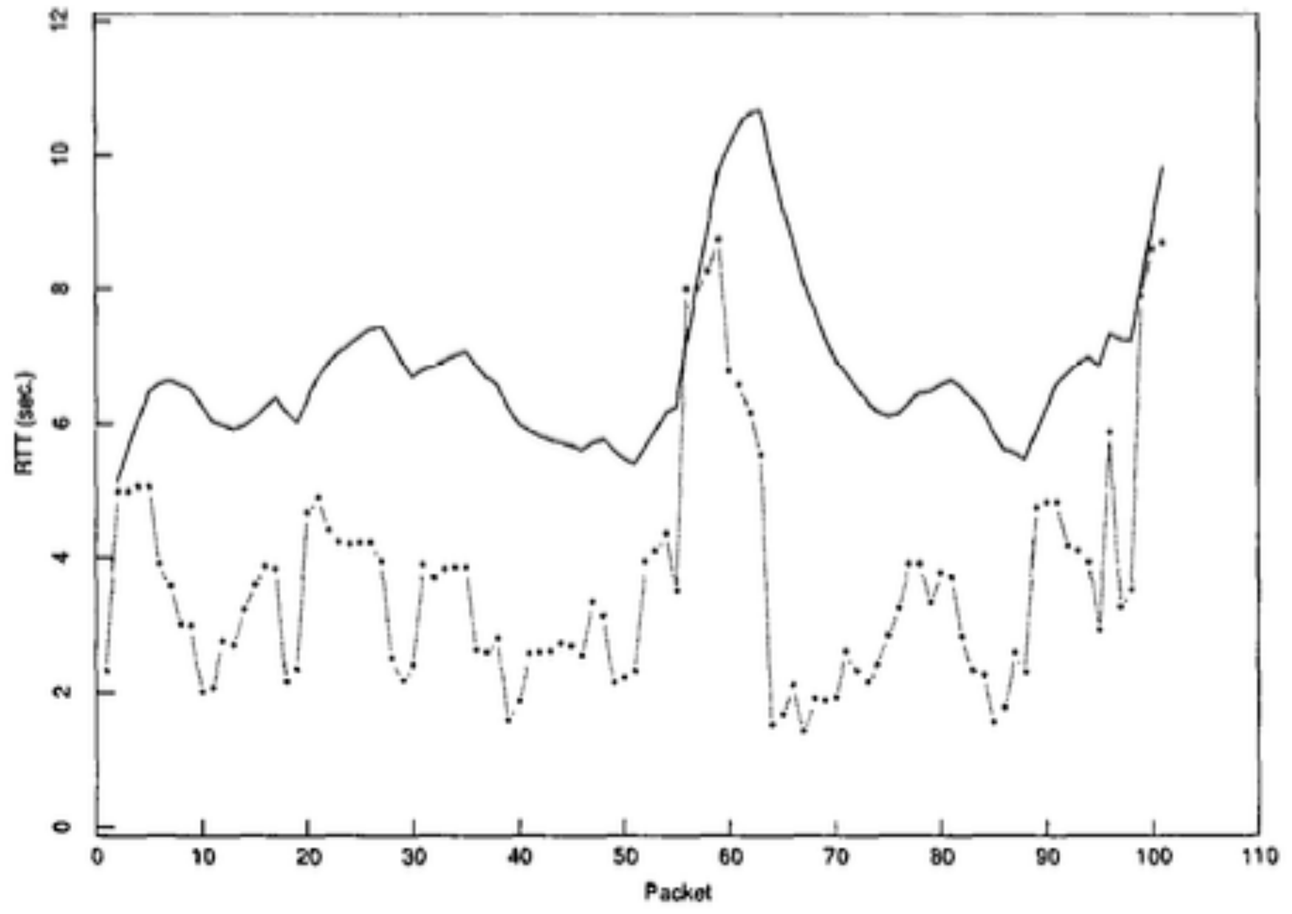
And we don’t need to get anywhere close to 75% to find trouble

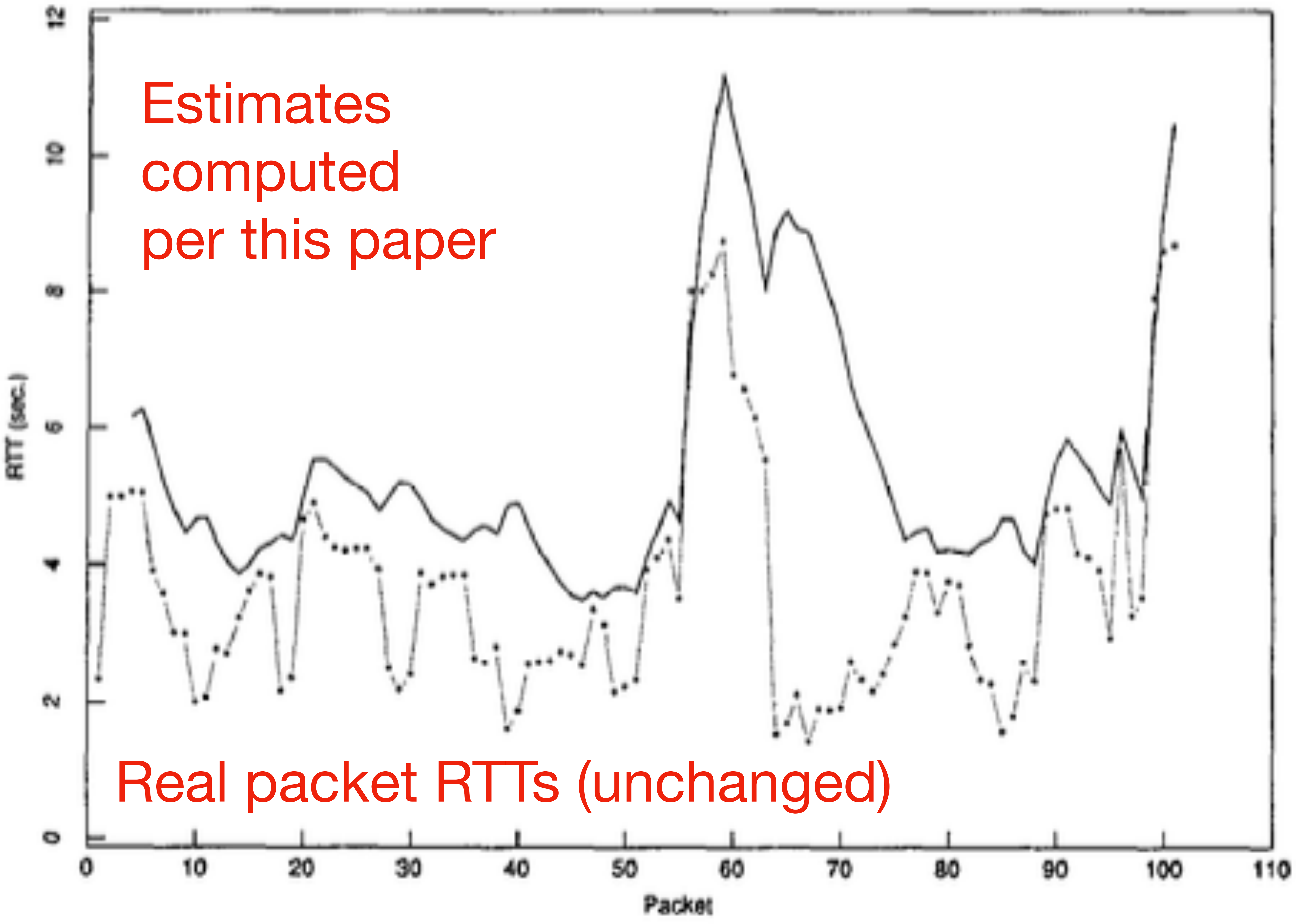
If you followed RFC793, which was in force at the time, you would begin to send unnecessary retransmits at 30% capacity

RTT Variance Estimation



RTT Variance Estimation





RTT Variance Estimation

Details are elided away into Appendix A, and I won't go over them

Takeaway:
Retransmitting is not going away; it's a fact of life
We just have a more accurate timer now
And that lets us be less trigger-happy with retransmits
And this has a large emergent effect

Van Jacobson’s Interventions

The connection doesn’t get to equilibrium because of the sender

- Slow start

Equilibrium is reached, but the sender violates it

- RTT variance estimation
- Exponential retransmit timer backoff

The connection doesn’t get to equilibrium because of resource limits

- More aggressive ACK policy
- Dynamic window resizing

Exponential Backoff

Say a packet has to be retransmitted repeatedly.
How should those retransmits be spaced?

Solution: wait for time t, then nt, the n^2t, etc.

Anyone see a problem?

This grows very fast! But linear systems theory proves that this is the only fix

It can be *capped*, and that is indeed a real deployed technique

Paper is sparse on details, but this is a standard technique that you can look up

Van Jacobson’s Interventions

The connection doesn’t get to equilibrium because of the sender

- Slow start

Equilibrium is reached, but the sender violates it

- RTT variance estimation
- Exponential retransmit timer backoff

The connection doesn’t get to equilibrium because of resource limits

- More aggressive ACK policy
- Dynamic window resizing

More Aggressive ACK Policy

Discussion: let’s get to the bottom of this together

Van Jacobson’s Interventions

The connection doesn’t get to equilibrium because of the sender

- Slow start

Equilibrium is reached, but the sender violates it

- RTT variance estimation
- Exponential retransmit timer backoff

The connection doesn’t get to equilibrium because of resource limits

- More aggressive ACK policy
- Dynamic window resizing

Congestion Avoidance

Sender claims packet is lost

Thanks to the previous interventions, we *do* believe the sender

- ~~Don't believe sender~~
- Believe sender

  Very rare; disregard

  - ~~Packet was damaged enroute~~
  - Packet was dropped enroute (buffer overflow)
    - How to recognize this? Packet loss!
    - What do to about this?

Congestion Avoidance

Say we model congestion by observing average queue lengths

We will average over some interval of time, e.g., RTT

So L_i is the load at interval i

We don't expect L_i to be zero even in an uncongested network

There are intrinsic delays related to queuing/dequeuing packets from the buffer

So we can say: $L_i = N$, where N accounts for those delays

Congestion Avoidance

So we can say: $L_i = N$, where N accounts for inherent queuing delays

But that was for an uncongested network

Assuming some congestion (and therefore some retransmits), a better model is:

$$L_i = N + \gamma L_{i-1} + \delta L_{i-2} + \ldots$$

The Greek-letter terms can be tuned to model more or less retransmission

Observe the emergent behavior! For sufficiently large Greek terms, this is exponential!

Dynamic Window Resizing

Our congestion model is:

$$L_i = N + \gamma L_{i-1} + \delta L_{i-2} + \ldots$$

We can only really respond by changing the size of the sender's window, *W:*

$$W_i = dW_{i-1}$$ Where $d < 1$

This matches the congestion model and allows to *also* be exponential

W is the same as *cwnd* from earlier; I am just tracking VJ's notation change

Dynamic Window Resizing

Upon observing congestion:

$W_i = dW_{i-1}$ Where $d < 1$

But what if we don't observe congestion?

$W_i = W_{i-1}$ Too conservative

$W_i = bW_{i-1}$ Too bold

$W_i = W_{i-1} + u$ Just right

Slow Start + Dynamic Window Resizing

We need a little bookkeeping to switch between the two algorithms

We maintain variables:
`cwnd`, **`ssthresh`**

Always send **`min`**(**`cwnd`**, window_{recv})

Upon timeout:
Multiplicative decrease
- **`ssthresh = cwnd ÷ 2`**
- **`cwnd = 1`**

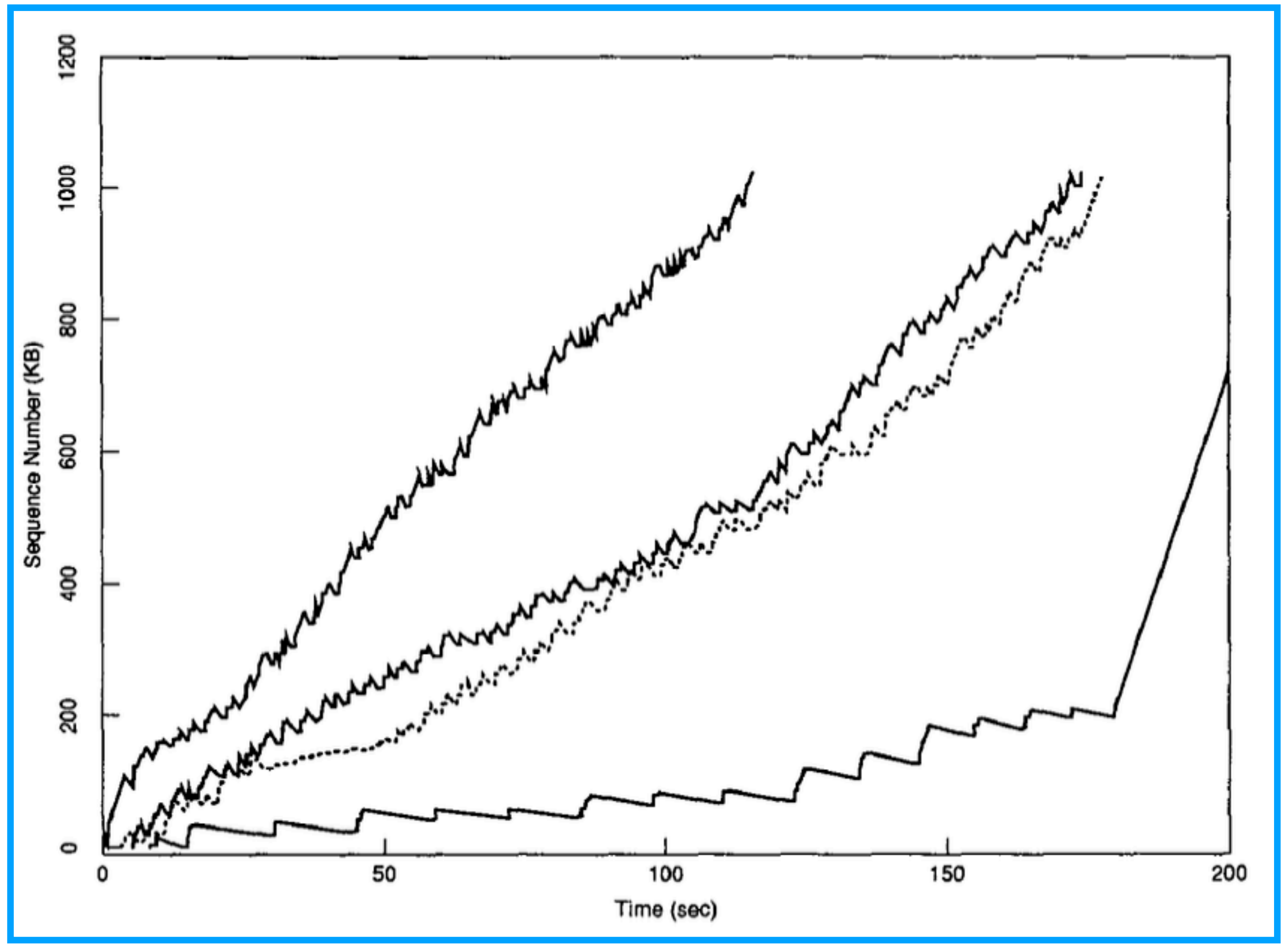
Setup for slow start

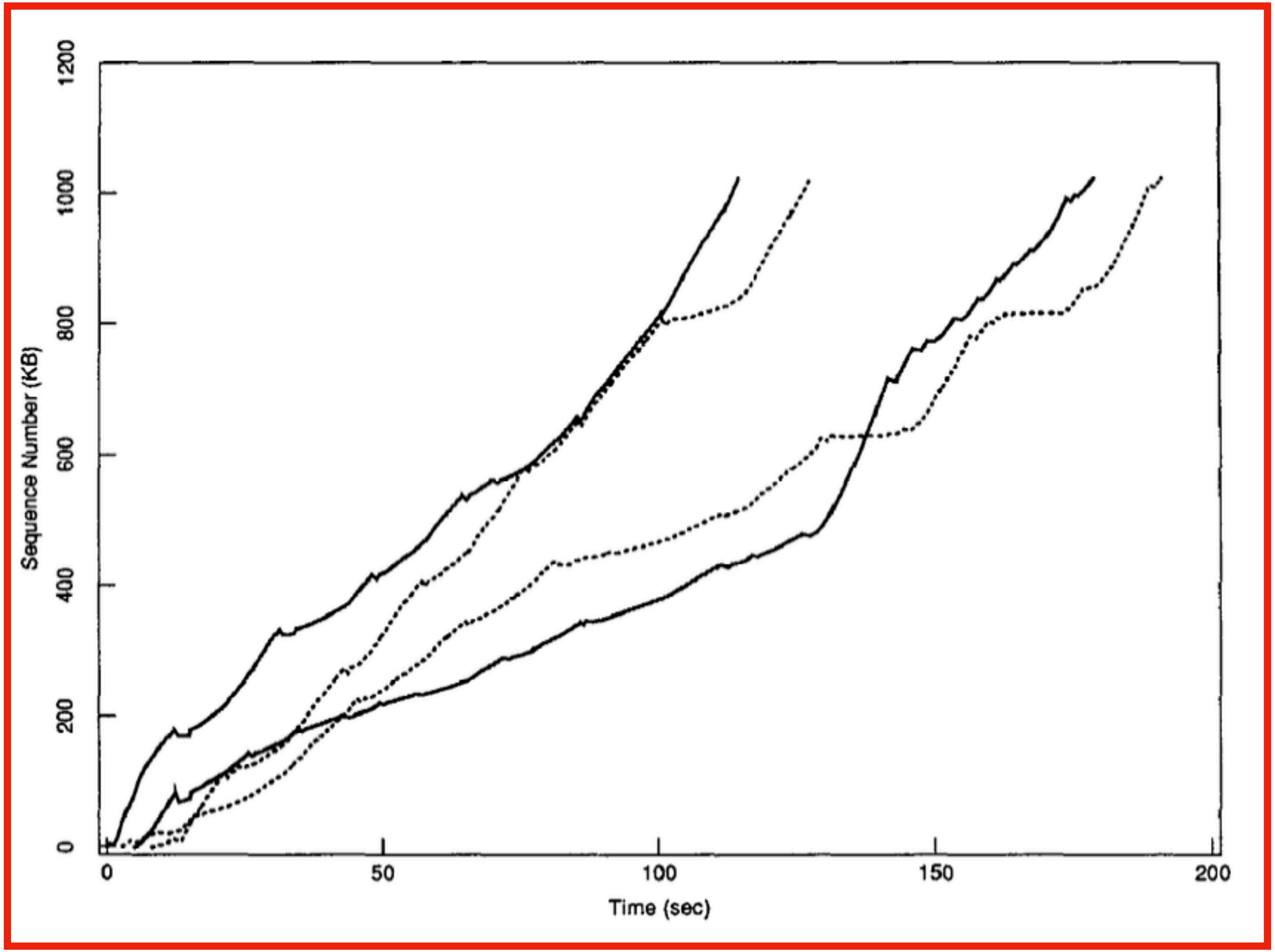
Upon ACK:

```
if (cwnd < ssthresh)
     /* if we're still doing slow-start
      * open window exponentially */
     cwnd += 1
else
     /* otherwise do Congestion
      * Avoidance increment-by-1 */
     cwnd += 1/cwnd
```

Additive increase

This is their choice for *u*

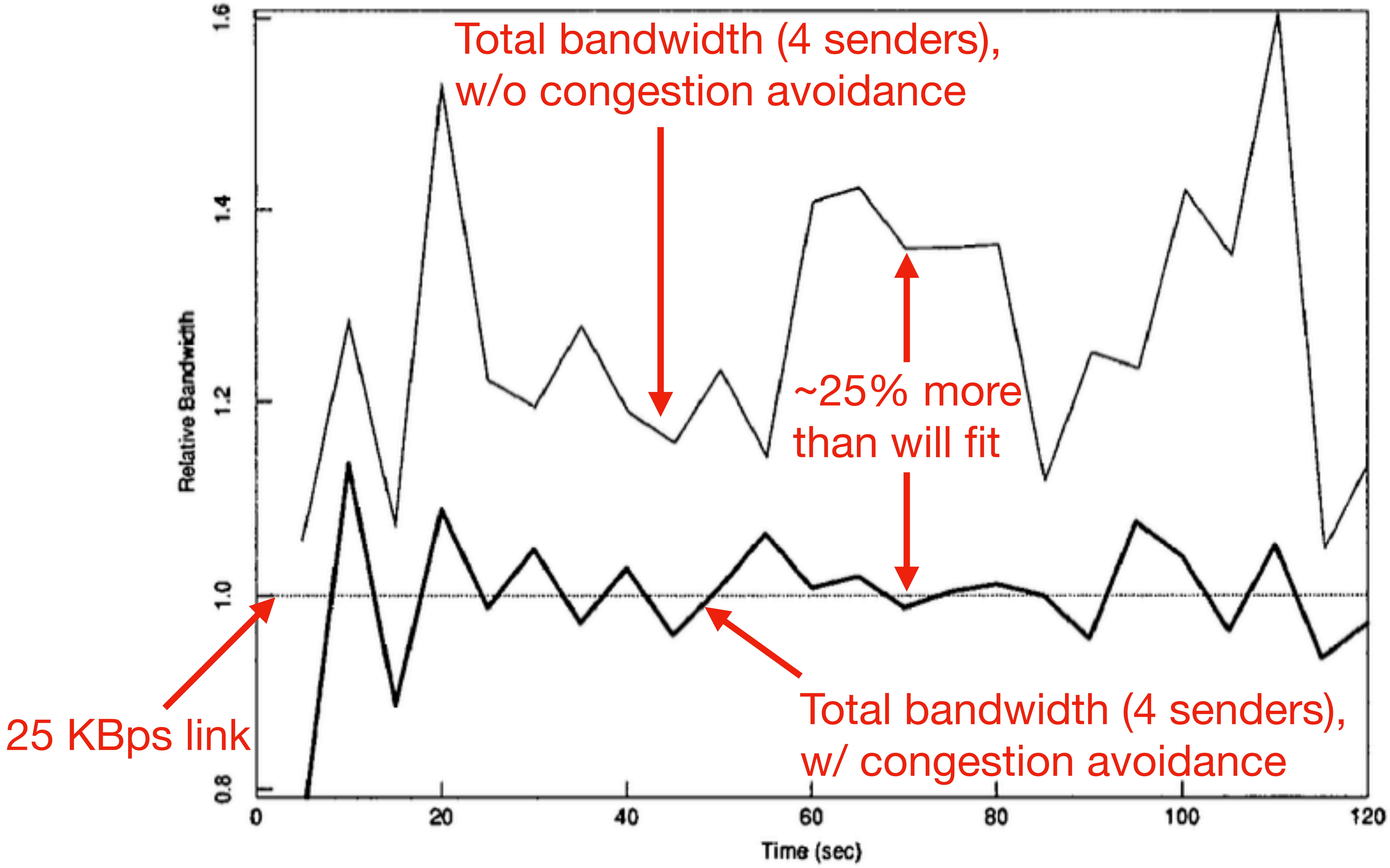


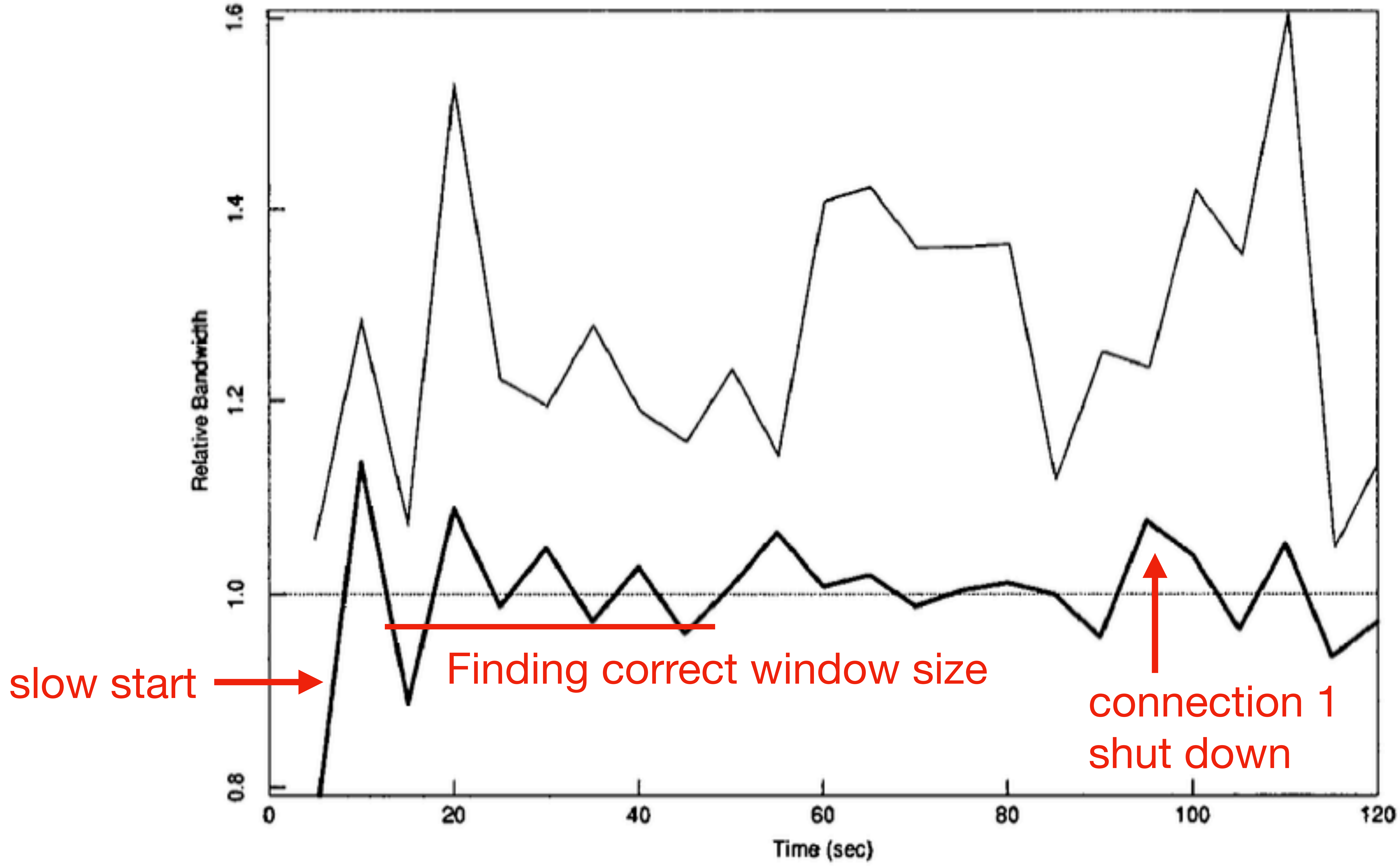


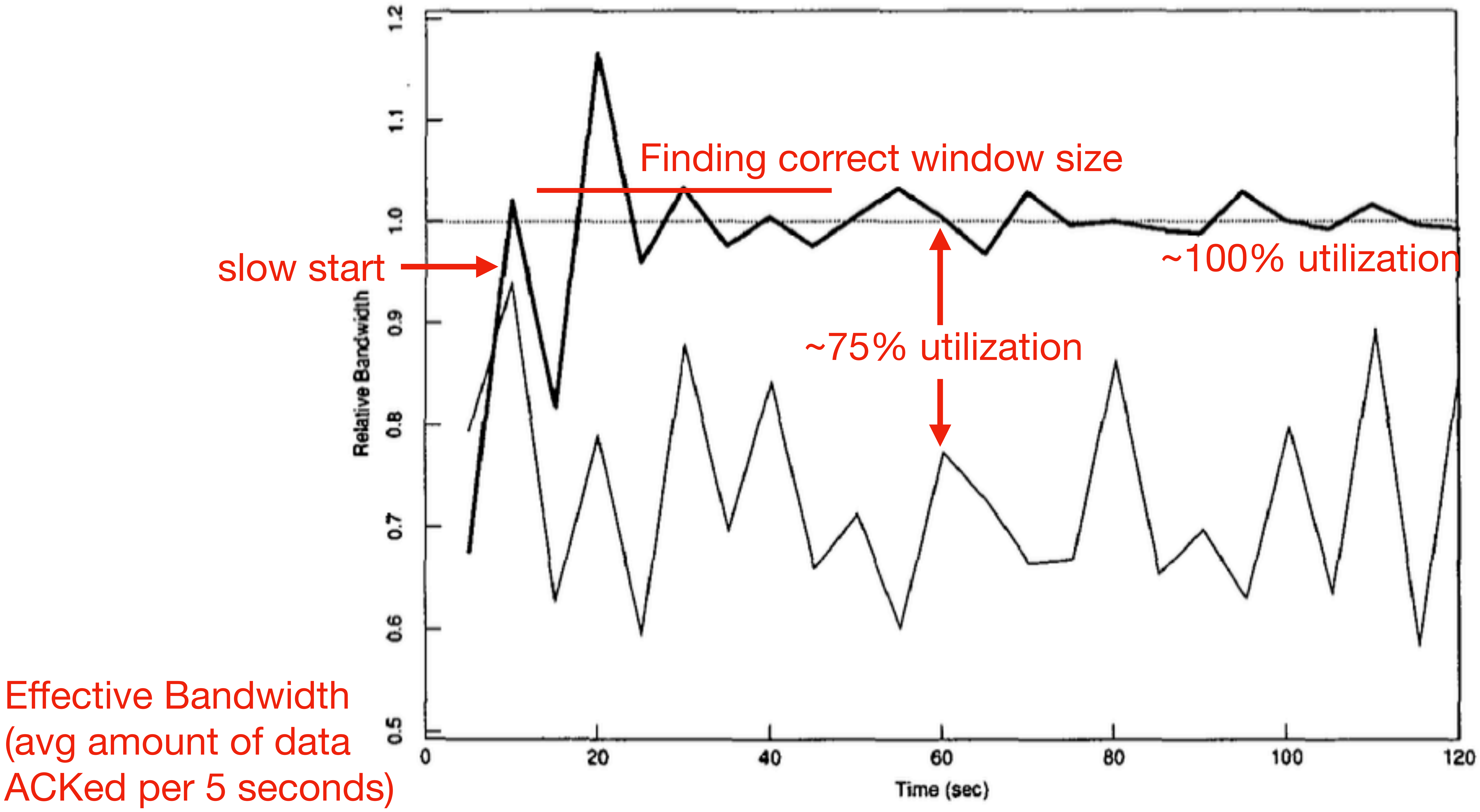
Four simultaneous TCP conversations, without and with congestion avoidance

4k/11k are retransmissions
89/8281 are retransmissions

25KBps link is shared as:
8, 5, 5, 0.5 (6 wasted)
8, 8, 4.5, 4.5 (no wastage)


Total bandwidth (4 senders),
w/o congestion avoidance
~25% more
than will fit
25 KBps link
Total bandwidth (4 senders),
w/ congestion avoidance
Relative Bandwidth
1.6
1.4
1.2
1.0
0.8
0
20
40
60
80
100
120
Time (sec)


1.6
1.4
1.2
1.0
0.8
Relative Bandwidth
0
20
40
60
80
100
120
Time (sec)
slow start
Finding correct window size
connection 1
shut down



Finding correct window size
slow start
~100% utilization
~75% utilization
Effective Bandwidth
(avg amount of data
ACKed per 5 seconds)
Relative Bandwidth
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100
120
Time (sec)

Discussion

The innovations from this paper may well have saved the internet, but (much) more has happened since!

If you know of an additional intervention, shout it out!

Random Early Detection (RED) in 1993
Sally Floyd, Van Jacobson

Explicit Congestion Notification (ECN) in 2001
KK Ramakrishnan, Sally Floyd, David L Black

Much more, sometimes general and sometimes domain-specific

Random Early Detection (RED)

Observation:
Typically we let queues fill up organically, and then we drop further packets

Can anyone think of a problem with that?

First, it is obviously unfair to later flows

Second, and more subtle: it contributes to *TCP global synchronization*
Everyone sends together, everyone suffers drops, so everyone holds back, then everyone uses slow start to ramp up together, and the cycle continues

RED preemptively drops packets before the buffer is full
It can tune what is dropped, and under what conditions

Explicit Congestion Notification (ECN)

Observation:
Why have *packet drops* be the implicit signal of congestion?
Directly challenges a decision from today's paper!

A router that is congested will *still send* a packet, but will set the ECN bits
The receiver will echo this signal to the sender
The sender can respond in various ways, which ECN does not dictate

This has been adopted into the Internet Protocol via RFC 3168, and two bits are set aside in IP headers for ECN

Selective summary of innovations
(may not be compatible)

https://en.wikipedia.org/wiki/TCP_congestion_control

Variant	Feedback	Required changes	Benefits	Fairness
(New) Reno	Loss	—	—	Delay
Vegas	Delay	Sender	Less loss	Proportional
High Speed	Loss	Sender	High bandwidth	
BIC	Loss	Sender	High bandwidth	
CUBIC	Loss	Sender	High bandwidth	
C2TCP[11][12]	Loss/Delay	Sender	Ultra-low latency and high bandwidth	
NATCP[13]	Multi-bit signal	Sender	Near Optimal Performance	
Elastic-TCP	Loss/Delay	Sender	High bandwidth/short & long-distance	
Agile-TCP	Loss	Sender	High bandwidth/short-distance	
H-TCP	Loss	Sender	High bandwidth	
FAST	Delay	Sender	High bandwidth	Proportional
Compound TCP	Loss/Delay	Sender	High bandwidth	Proportional
Westwood	Loss/Delay	Sender	Lossy links	
Jersey	Loss/Delay	Sender	Lossy links	
BBR[14]	Delay	Sender	BLVC, Bufferbloat	
CLAMP	Multi-bit signal	Receiver, Router	Variable-rate links	Max-min
TFRC	Loss	Sender, Receiver	No Retransmission	Minimum delay
XCP	Multi-bit signal	Sender, Receiver, Router	BLFC	Max-min
VCP	2-bit signal	Sender, Receiver, Router	BLF	Proportional
MaxNet	Multi-bit signal	Sender, Receiver, Router	BLFSC	Max-min
JetMax	Multi-bit signal	Sender, Receiver, Router	High bandwidth	Max-min
RED	Loss	Router	Reduced delay	
Prague[15]	Single-bit signal	Sender, Receiver, Router	Low latency, low loss, scalable throughput (L4S[16])	
ECN	Single-bit signal	Sender, Receiver, Router	Reduced loss	

Domain-Specific Innovations

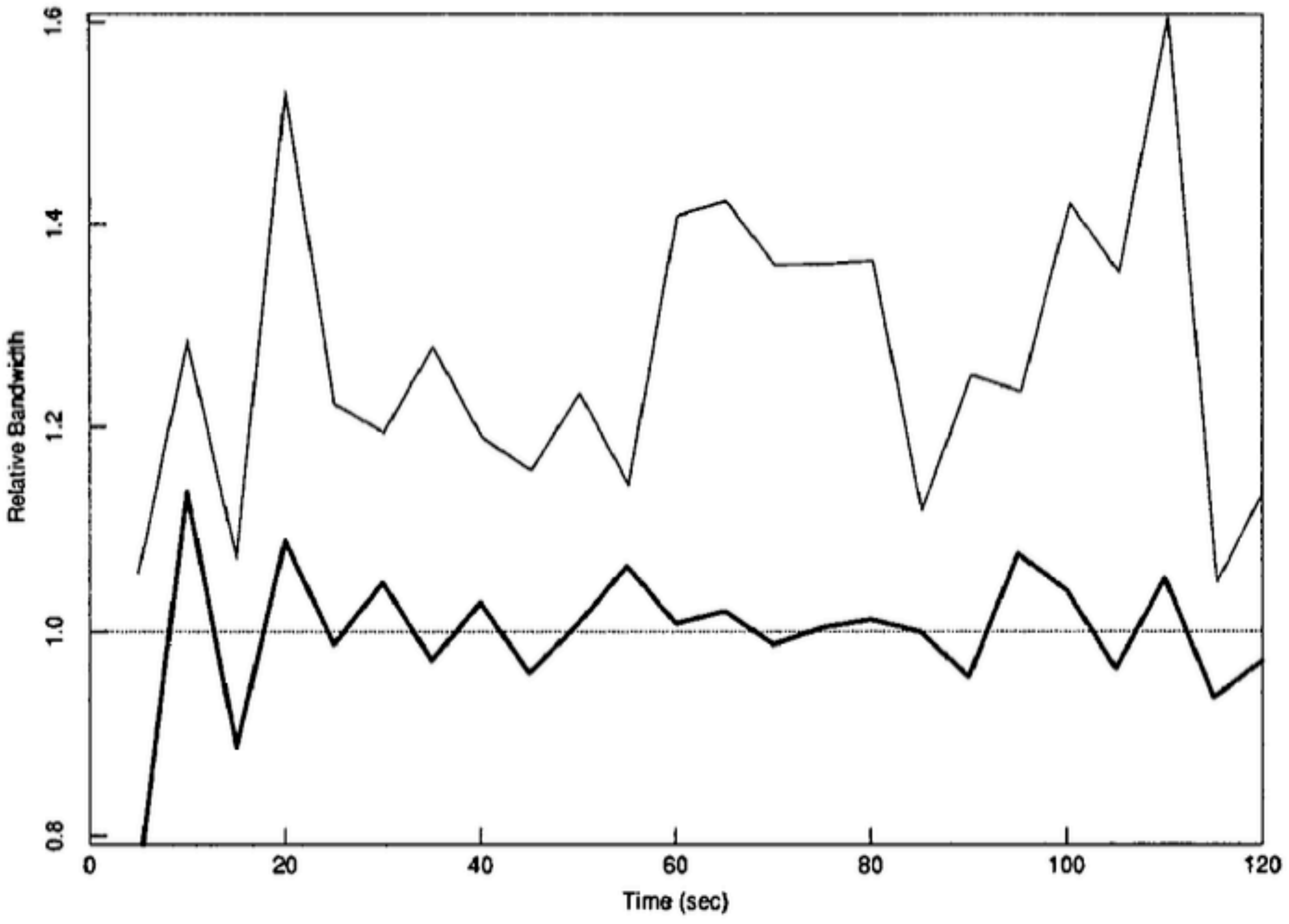
What if you decide to hyper-specialize your network for your job?

You may exploit known patterns and anti-patterns about your traffic,
may totally fail on some metrics and excel at others

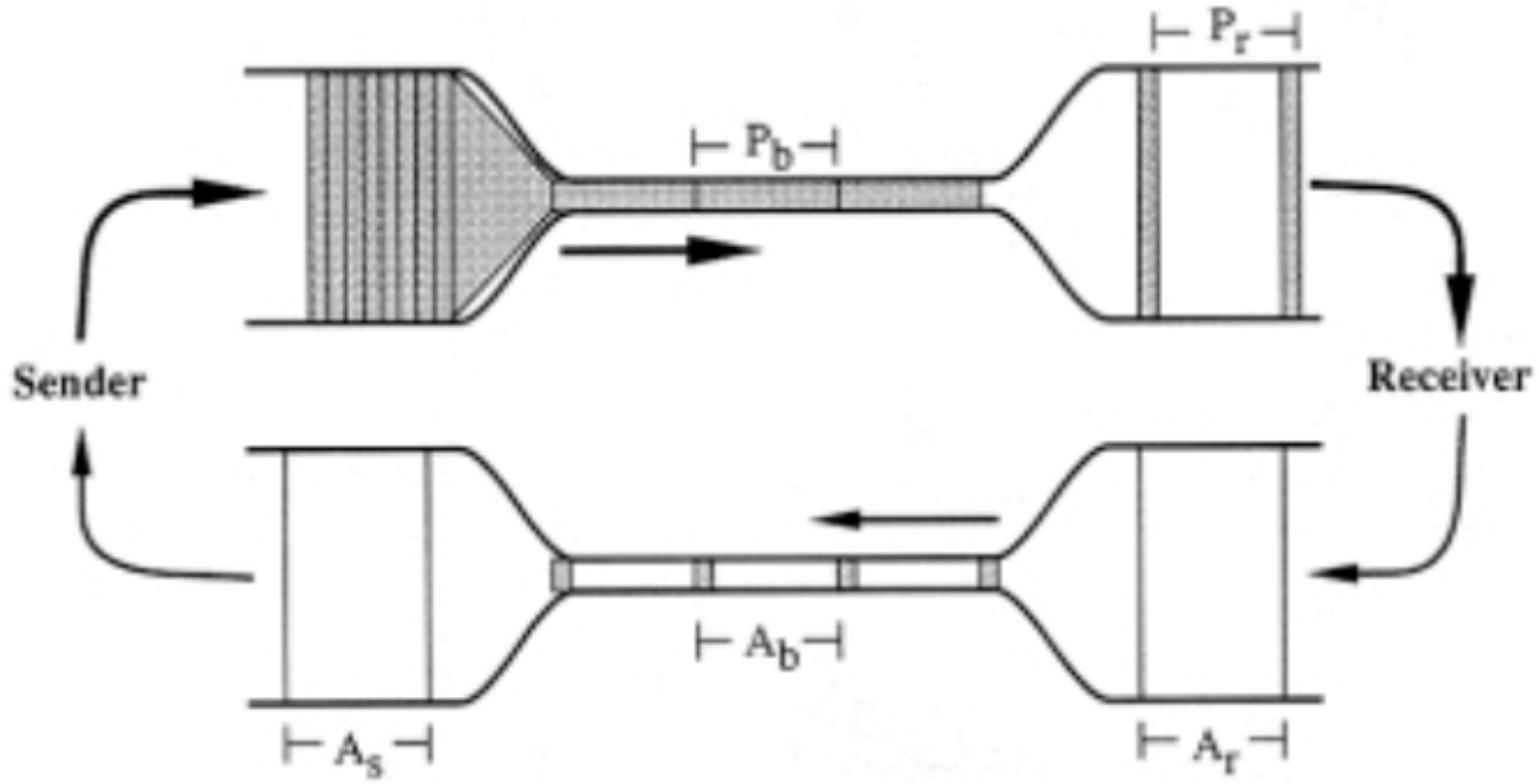
E.g., Data Center TCP (2010):
- Nodes use ECN
- Receiver computes *fraction* of congested packets and reports *that* to sender
- Sender can decide how much to dial back the send rate

E.g., Ultra Ethernet Consortium (2025) is trying to formulate new standards that are specialized for HPC and AI

Summary




```
if (cwnd < ssthresh)
     /* if we're still doing slow-start
      * open window exponentially */
     cwnd += 1
else
     /* otherwise do Congestion
      * Avoidance increment-by-1 */
     cwnd += 1/cwnd
```



Slow start
RTT variance estimation
Exponential retransmit timer backoff
More aggressive ACK policy
Dynamic window resizing

Thanks!