PURDUE UNIVERSITY

STyGIANet

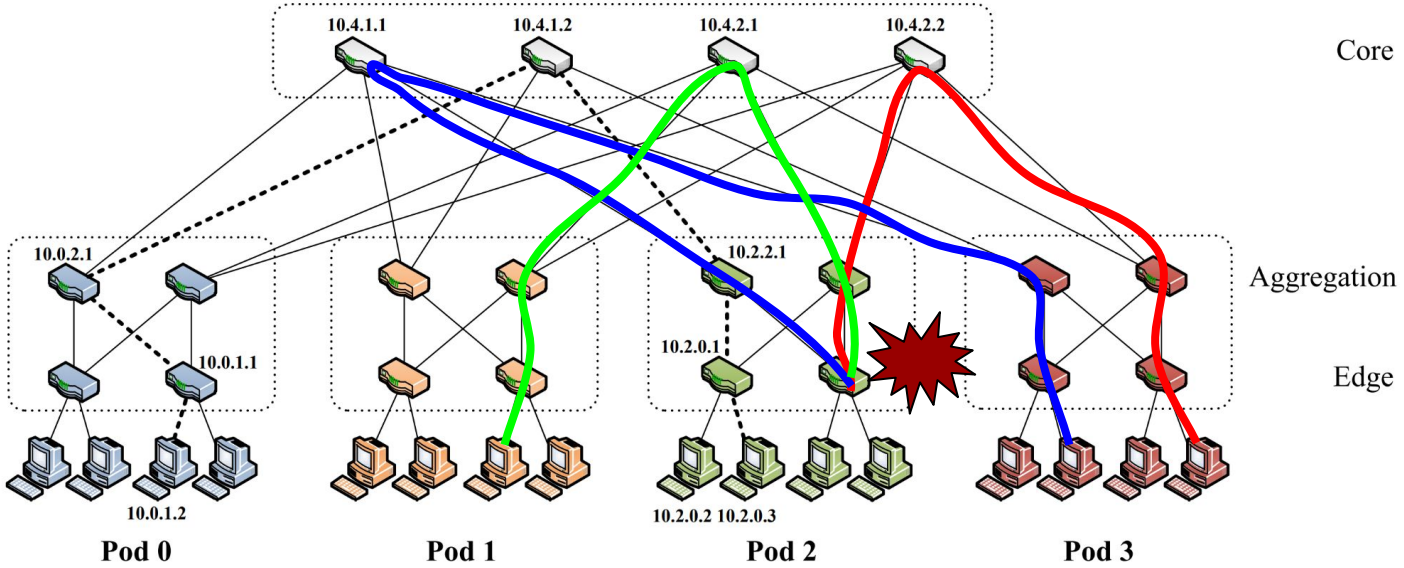
Congestion Control in Datacenters

CS 422 / CS 536

April 9, 2026

Vamsi Addanki
vaddank@purdue.edu

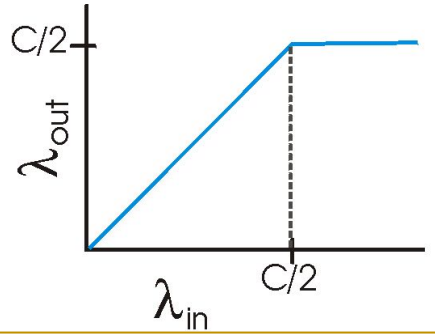
Incast

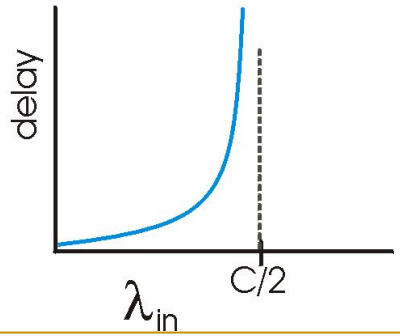

10.4.1.1
10.4.1.2
10.4.2.1
10.4.2.2
Core
10.0.2.1
10.0.1.1
10.0.1.2
10.2.2.1
10.2.0.1
10.2.0.2 10.2.0.3
Aggregation
Edge
Pod 0
Pod 1
Pod 2
Pod 3

Incast

- N flows transmitting towards a single receiver (referred as N:1 incast)
- Sudden congestion caused by simultaneous bursty transmissions
- Increased queueing delays
- Buffer overflow and packet losses
- Severely impacts flow completion times

All the above sound familiar?





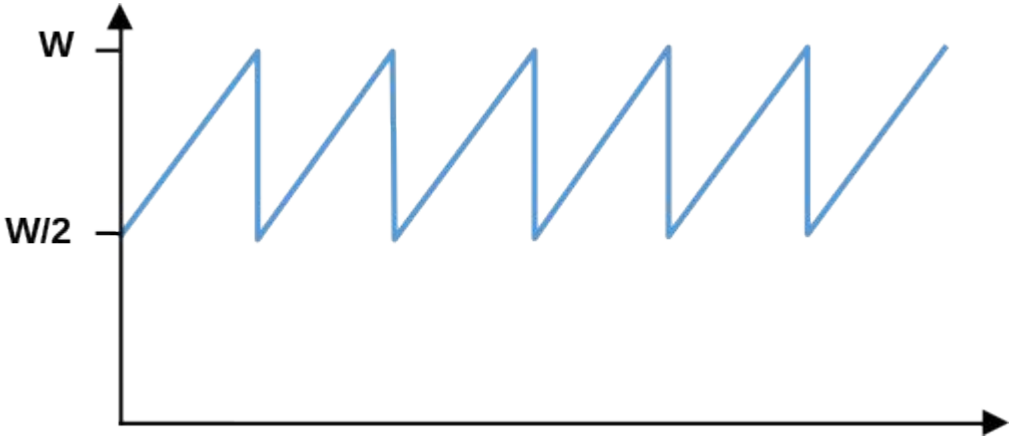
Incast

- Throughput collapse (similar to the internet in 1980's)
  - Large number of concurrent flows cause increased queueing delays, packet loss and eventually drop (collapse) in throughput

Chen Y, Griffith R, Liu J, Katz RH, Joseph AD. Understanding TCP incast throughput collapse in datacenter networks. InProceedings of the 1st ACM workshop on Research on enterprise networking 2009 Aug 21 (pp. 73-82).

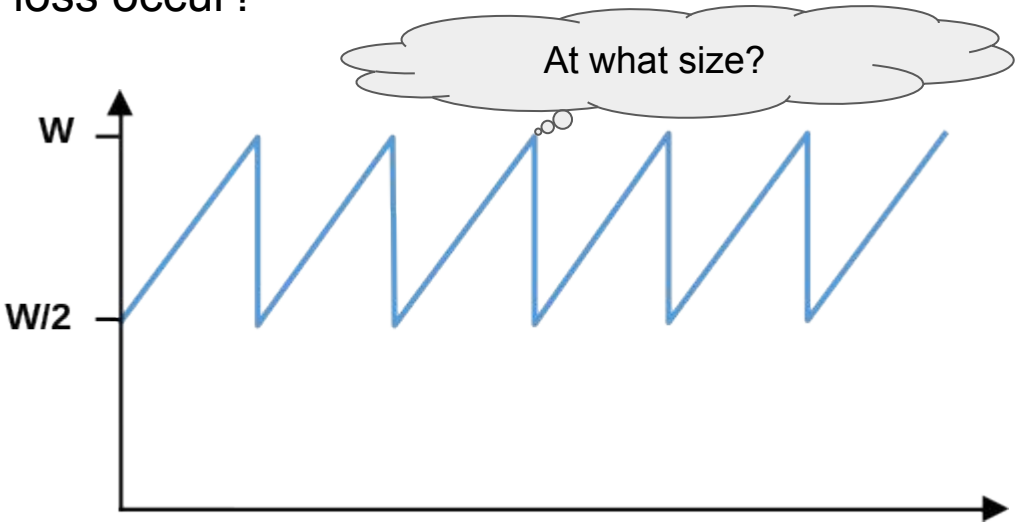
AIMD loss-based Congestion Control

- What are the problems with TCP RENO?



AIMD loss-based Congestion Control

- When does a loss occur?



AIMD loss-based Congestion Control

- Loss-based congestion control algorithms have the tendency to fill up the queues (buffers) in the network
  - Bufferbloat
  - High delay
  - Packet losses
  - A recipe for throughput collapse under incasts

Congestion Control in Datacenters

- DCTCP
- TIMELY
- HPCC
- PowerTCP

Desirable Congestion Control for Datacenters

- Low latency
  - Requires maintaining small queue lengths in the network

Desirable Congestion Control for Datacenters

- Low latency
  - Requires maintaining small queue lengths in the network
- High throughput
  - Requires maintaining high link utilization

Desirable Congestion Control for Datacenters

- Low latency
  - Requires maintaining small queue lengths in the network
- High throughput
  - Requires maintaining high link utilization
- Fast convergence
  - Requires rapidly achieving low delay & high throughput, after any sudden changes in the network (eg., after an incast)

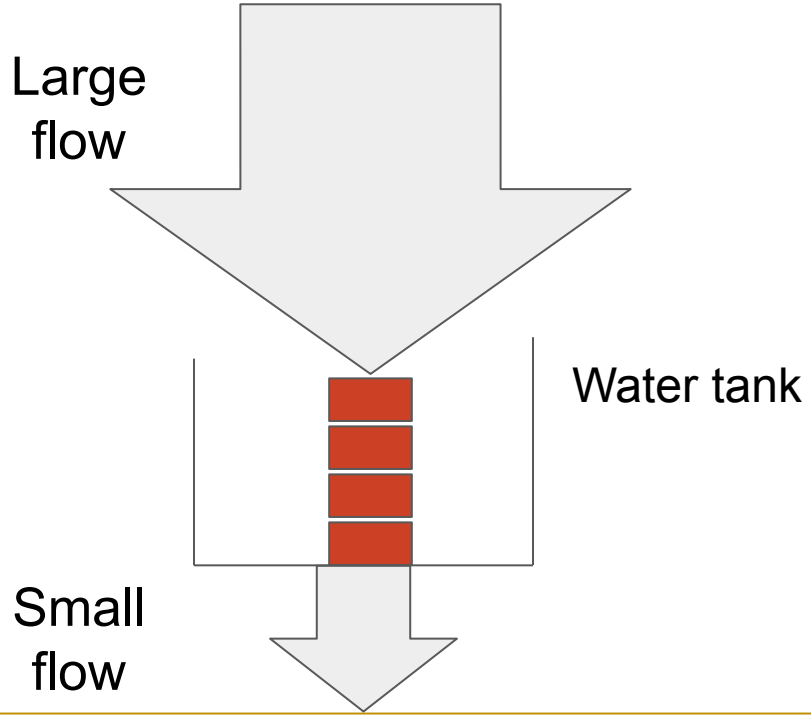
Desirable Congestion Control for Datacenters

- Low latency
  - Requires maintaining small queue lengths in the network
- High throughput
  - Requires maintaining high link utilization
- Fast convergence
  - Requires rapidly achieving low delay & high throughput, after any sudden changes in the network (eg., after an incast)
- Fairness
  - Requires fairly sharing the link bandwidth among competing flows

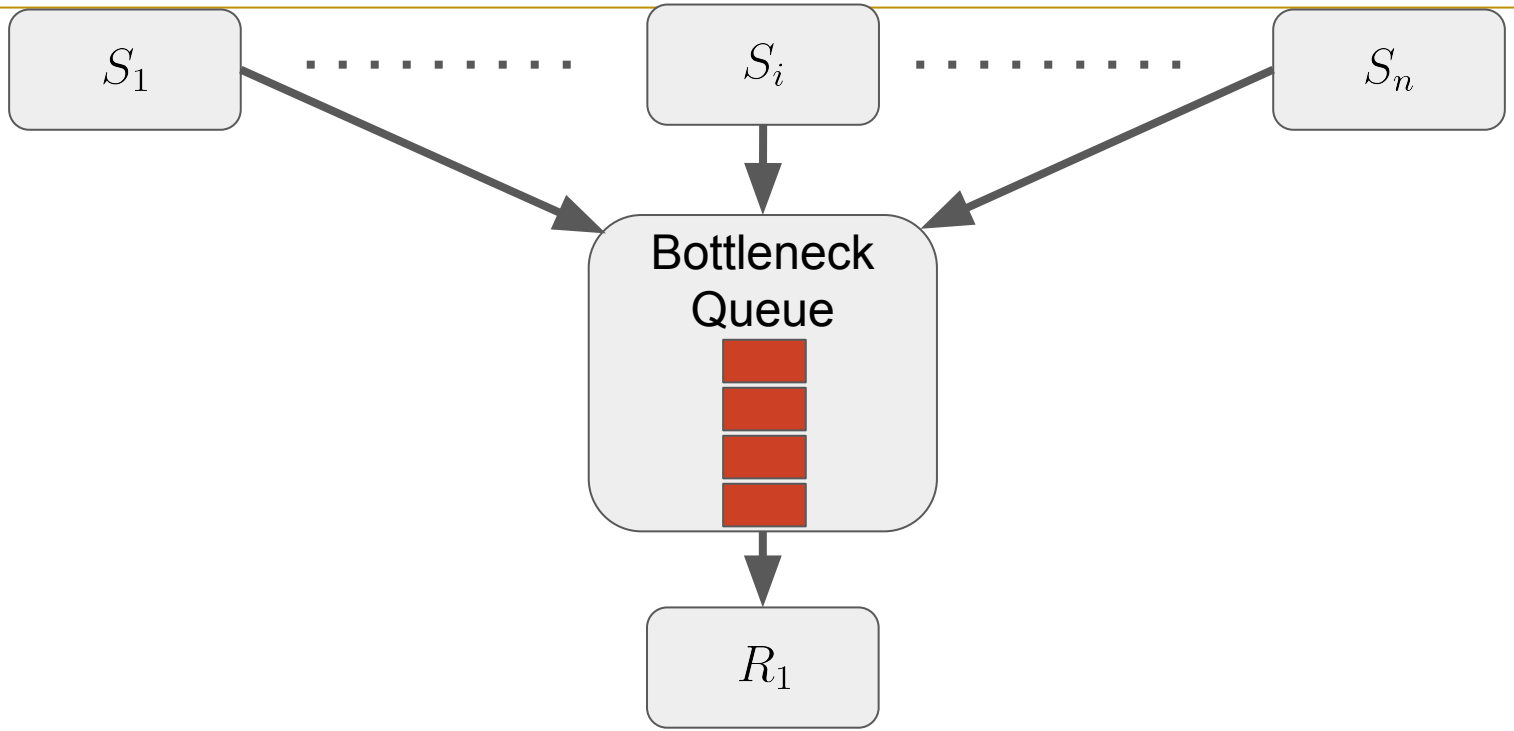
Desirable Congestion Control for Datacenters

- Low latency
  - Requires maintaining small queue lengths in the network
- High throughput
  - Requires maintaining high link utilization
- Fast convergence
  - Requires rapidly achieving low delay & high throughput, after any sudden changes in the network (eg., after an incast)
- Fairness
  - Requires fairly sharing the link bandwidth among competing flows
- Equilibrium
  - Requires eventually achieving a stable delay & throughput

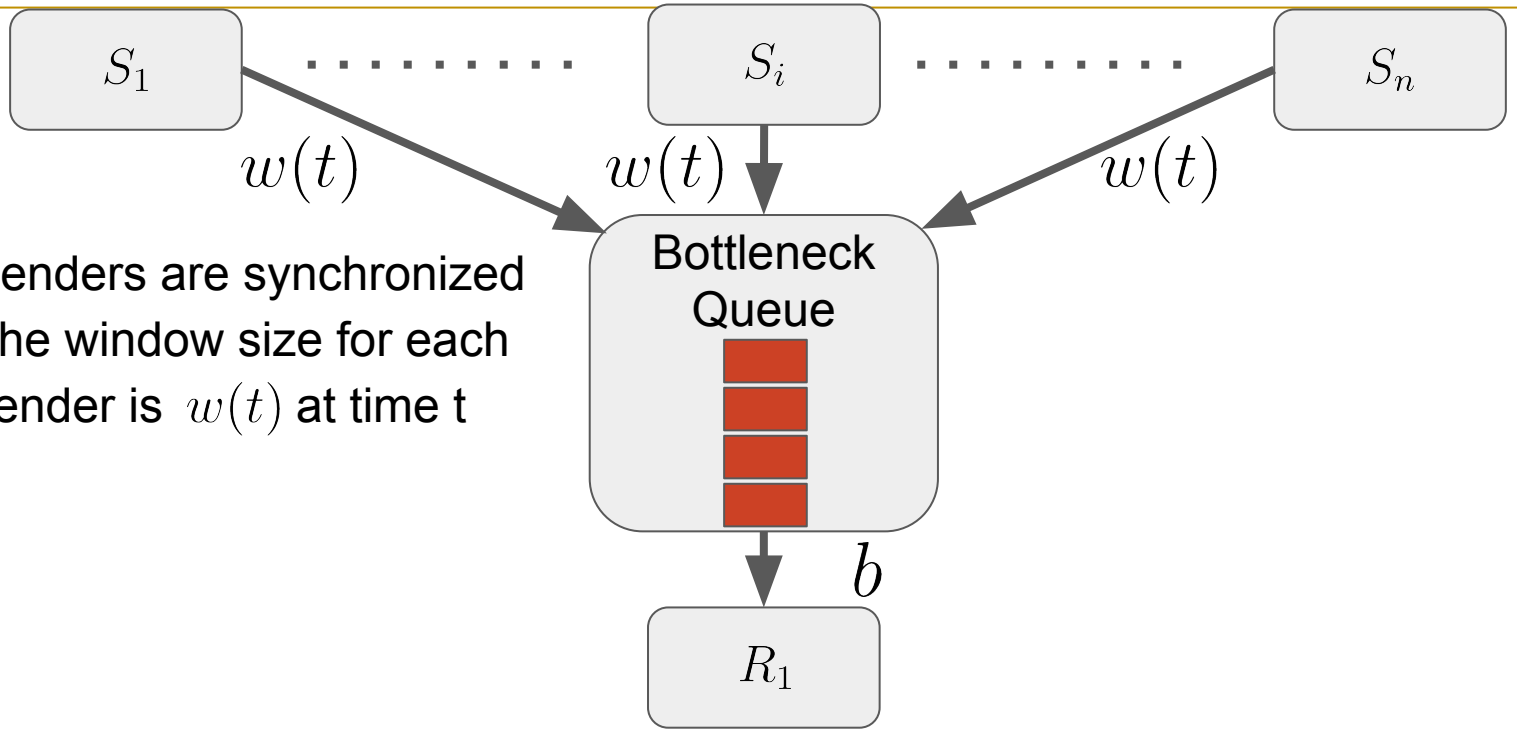
Imagine a “Pipe”



Single Bottleneck Link Simple Model

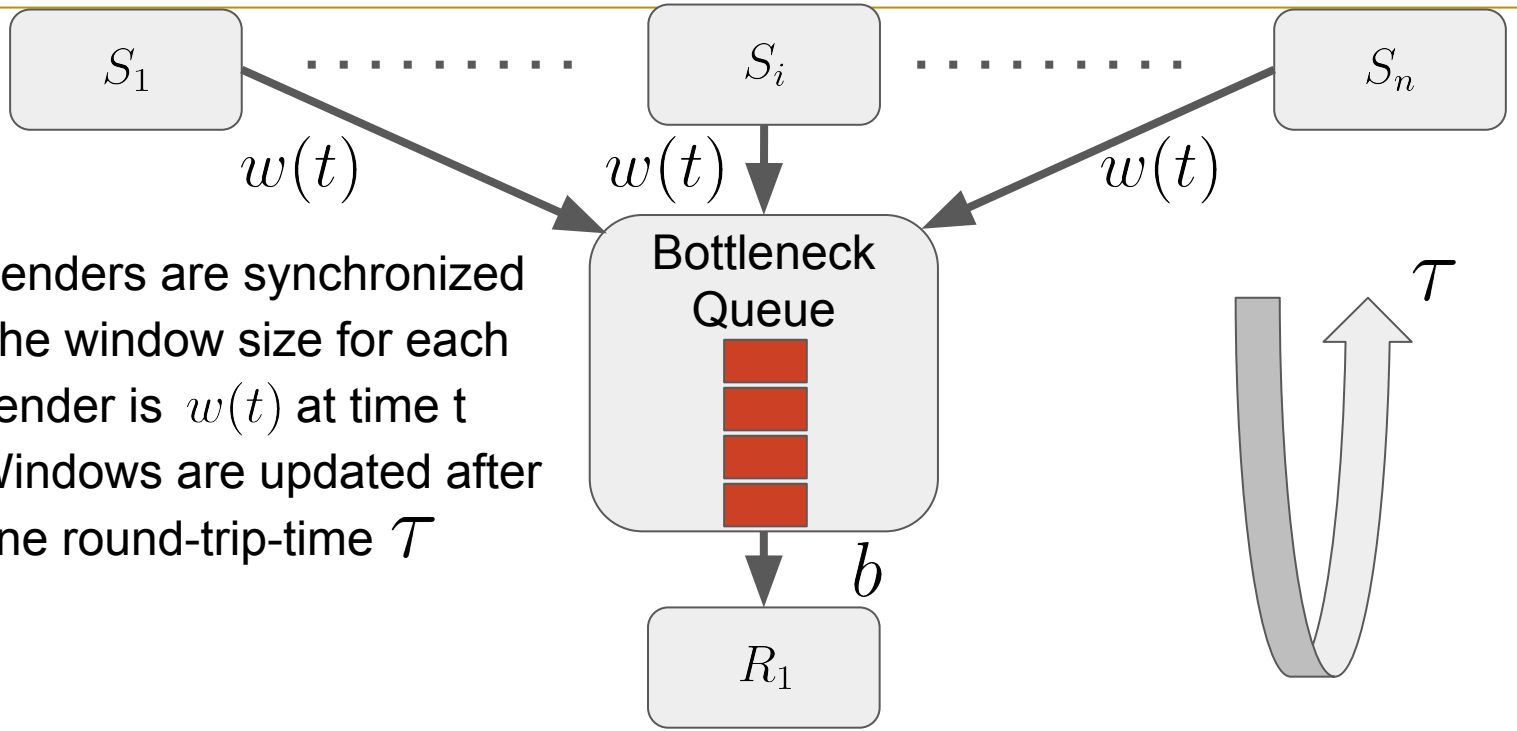


Single Bottleneck Link Simple Model



- Senders are synchronized
- The window size for each sender is $w(t)$ at time t

Single Bottleneck Link Simple Model

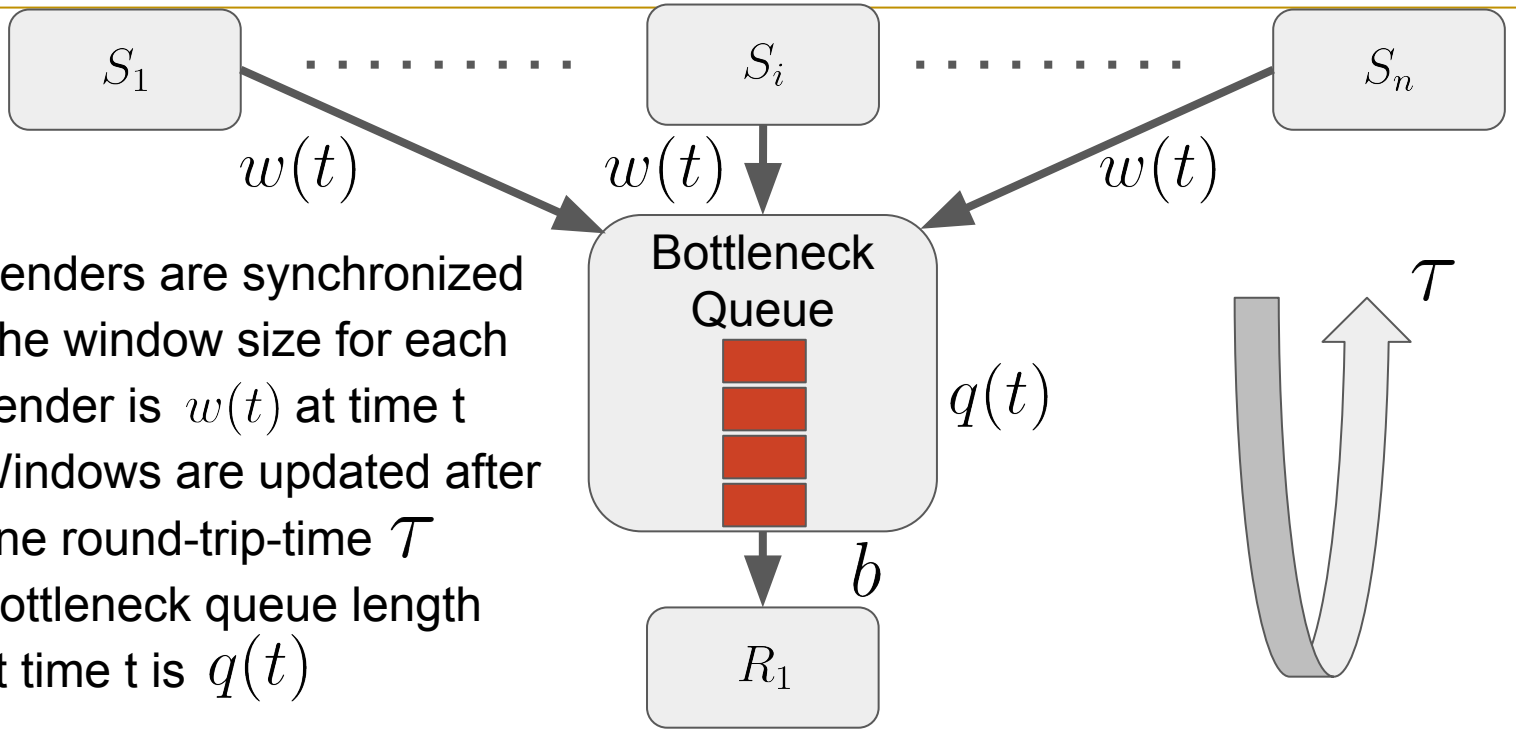


- Senders are synchronized
- The window size for each sender is $w(t)$ at time t
- Windows are updated after one round-trip-time τ

PURDUE UNIVERSITY

STyGIANet

Single Bottleneck Link Simple Model

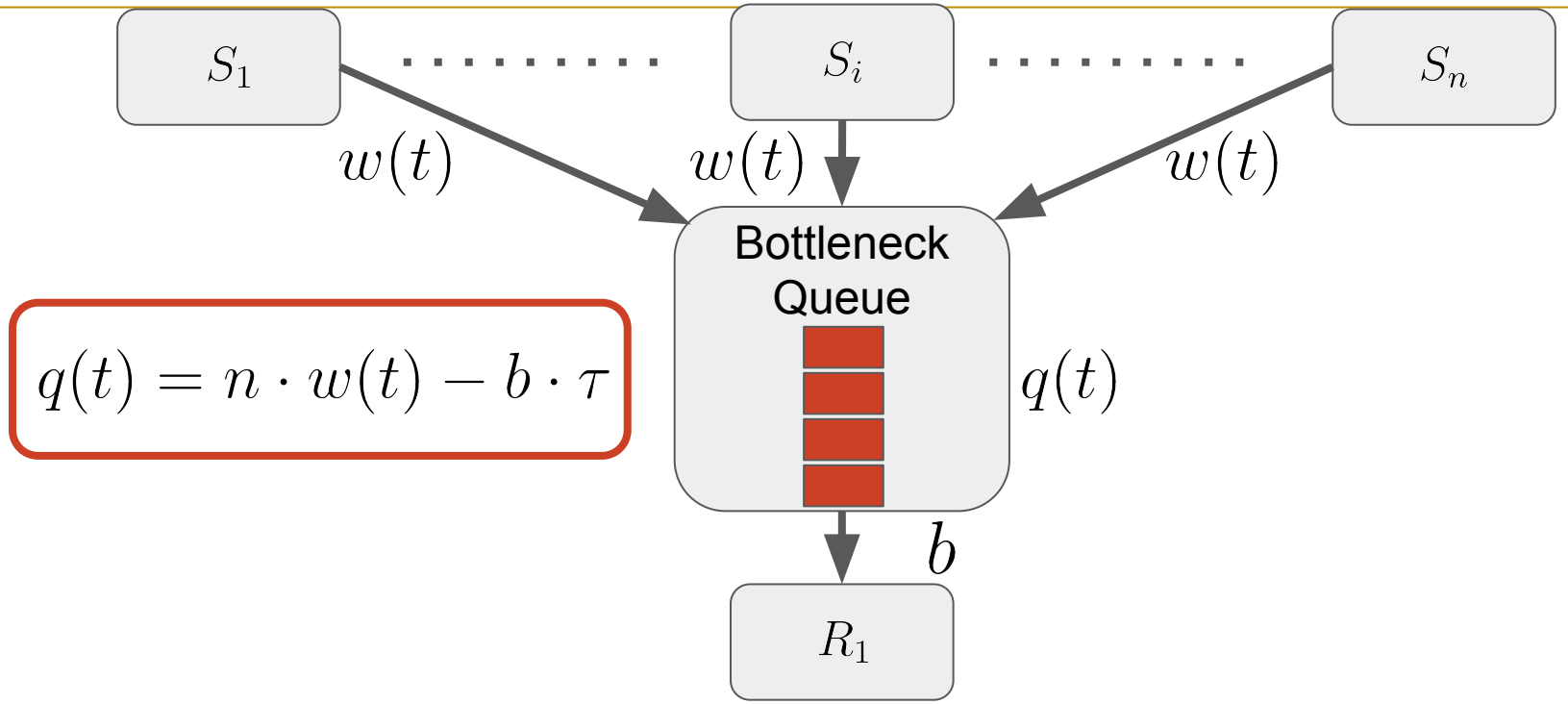


- Senders are synchronized
- The window size for each sender is $w(t)$ at time t
- Windows are updated after one round-trip-time τ
- Bottleneck queue length at time t is $q(t)$

PURDUE UNIVERSITY

STyGIANet

Single Bottleneck Link Simple Model

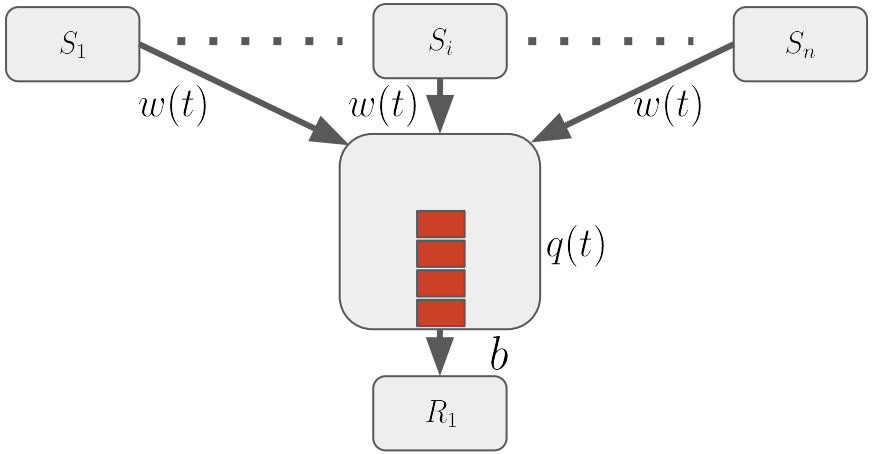


$$q(t) = n \cdot w(t) - b \cdot \tau$$

Single Bottleneck Link Simple Model

- A congestion control algorithm adjusts $w(t)$ in order to achieve the desirable properties such as low latency and high throughput

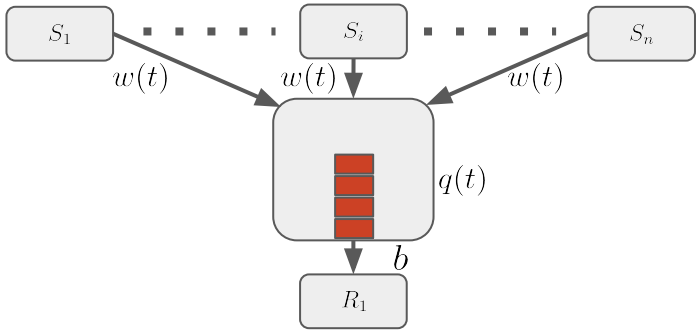
$$q(t) = n \cdot w(t) - b \cdot \tau$$



Single Bottleneck Link Simple Model

- How does a congestion control algorithm infer the network state?
- Not responding to the network state leads to bufferbloat
- eg., increasing the window size $w(t)$ blindly increases the queue size $q(t)$ significantly — leading to high latencies

$$q(t) = n \cdot w(t) - b \cdot \tau$$



Explicit Congestion Notification (ECN)

IPv4 header format

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Version				IHL				DSCP						ECN		Total length															
4	32	Identification																Flags				Fragment offset											
8	64	Time to Live								Protocol								Header checksum															
12	96	Source address																															
16	128	Destination address																															
20	160	Options (if IHL > 5)																															
⋮	⋮																																
56	448																																

2 Bits: *ECN Capable Transport (ECT bit)* and *Congestion Experienced (CE bit)*

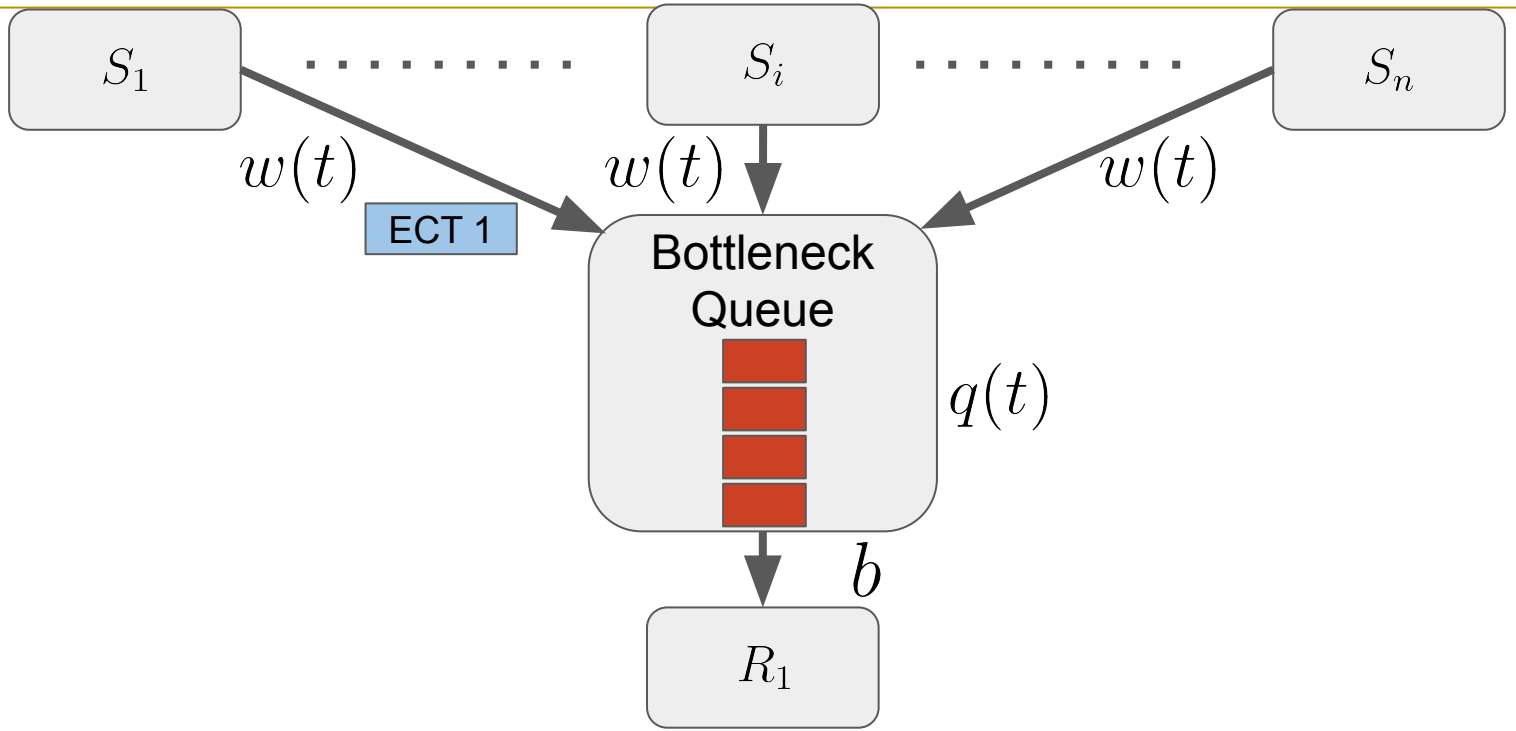
Non-ECT = 00: Non ECN-Capable

ECT-0 = 01: ECN Capable Transport (0)

ECT-1 = 10: ECN Capable Transport (1)

CE = 11: Congestion Experienced

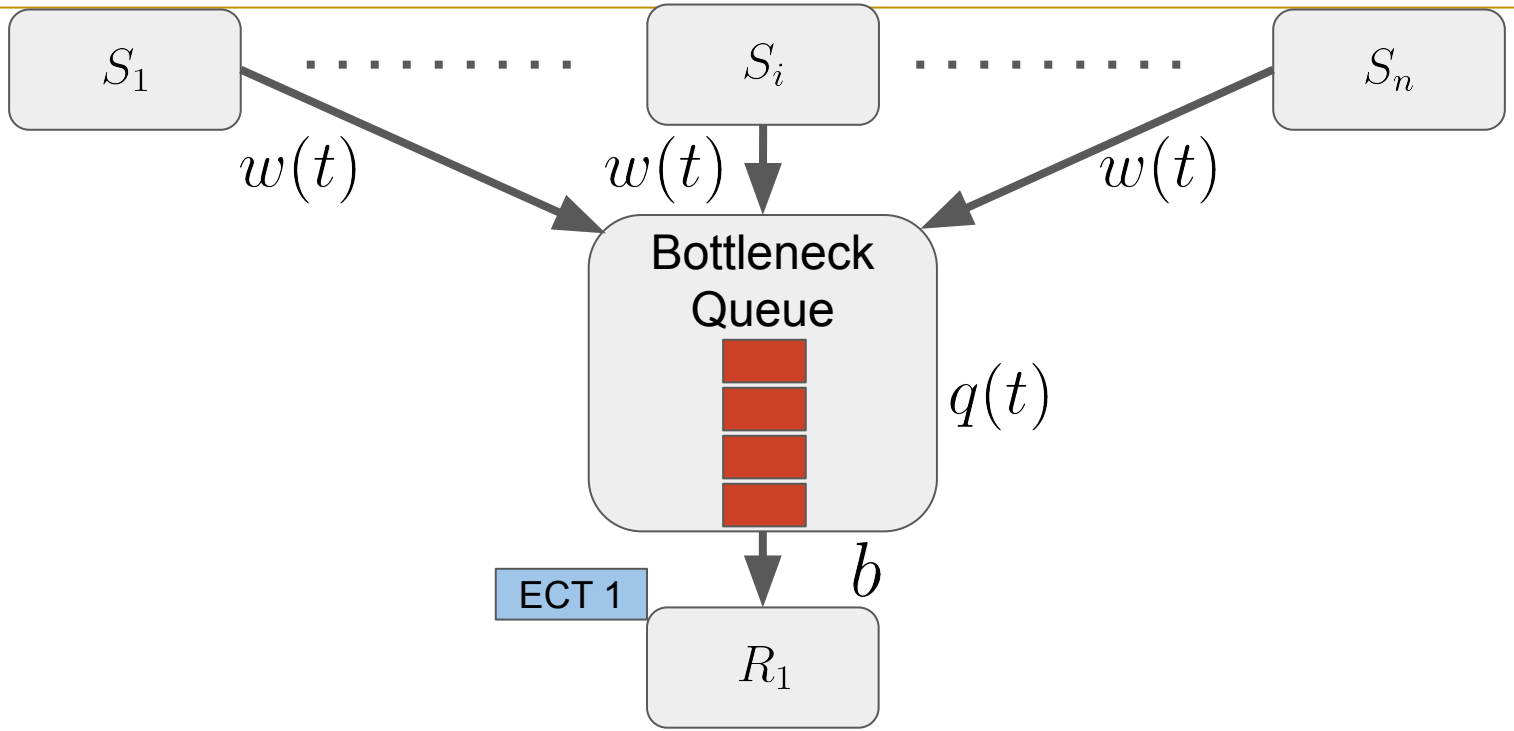
Explicit Congestion Notification (ECN)



PURDUE UNIVERSITY

STyGIANet

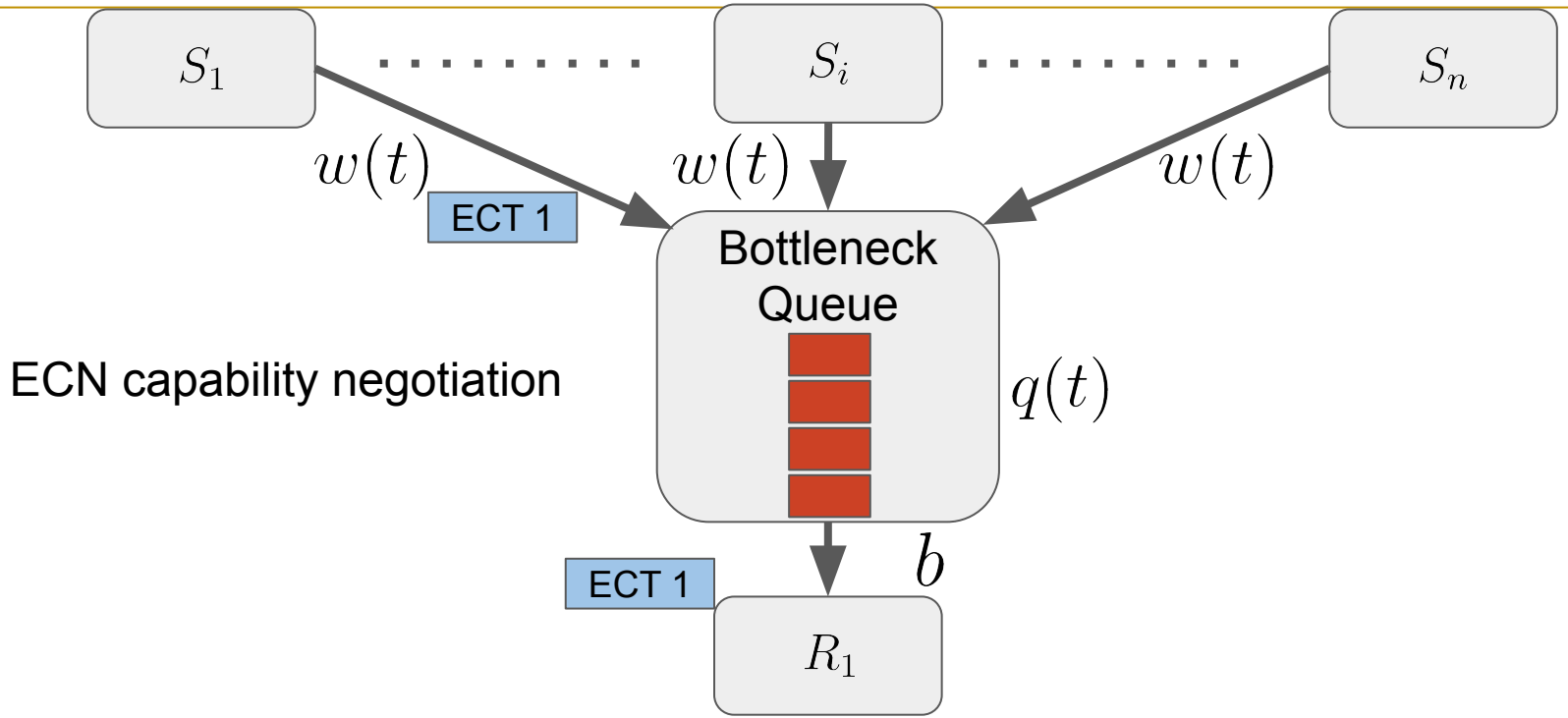
Explicit Congestion Notification (ECN)



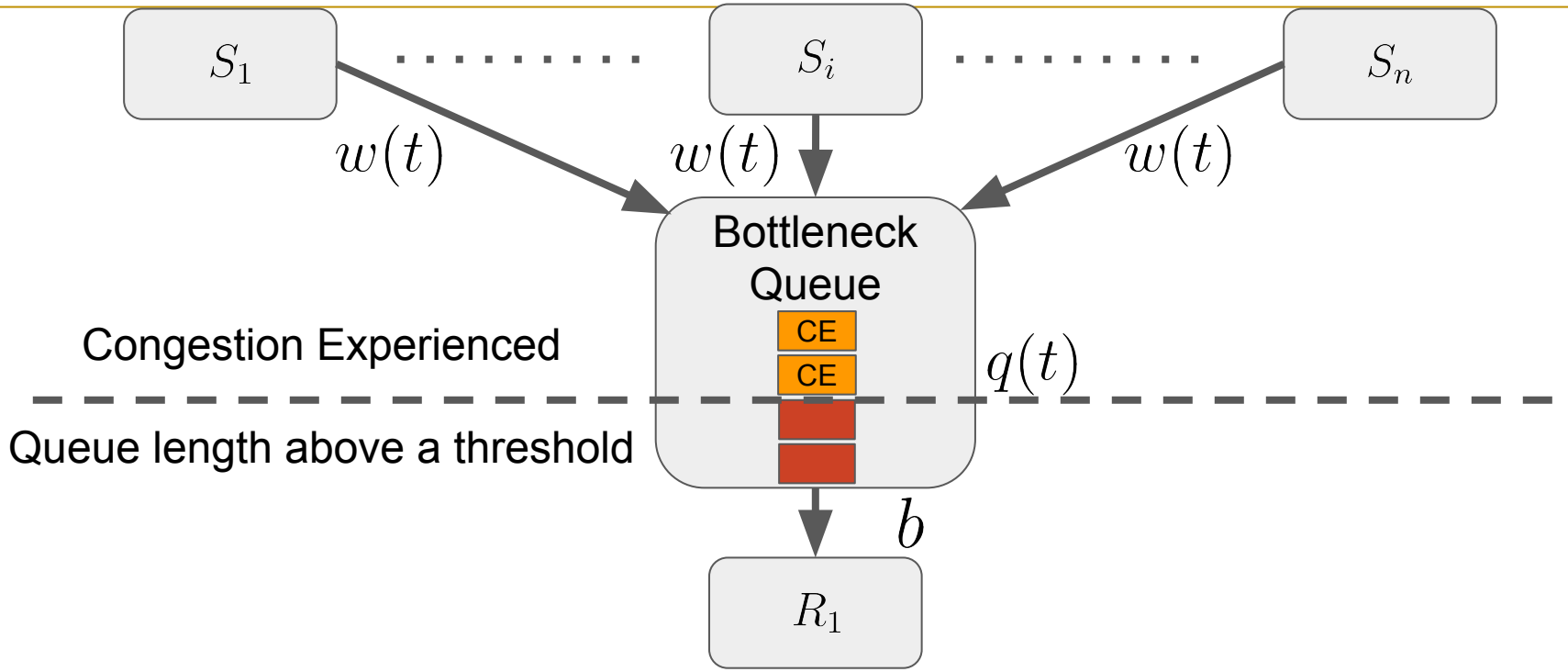
PURDUE UNIVERSITY

STyGIANet

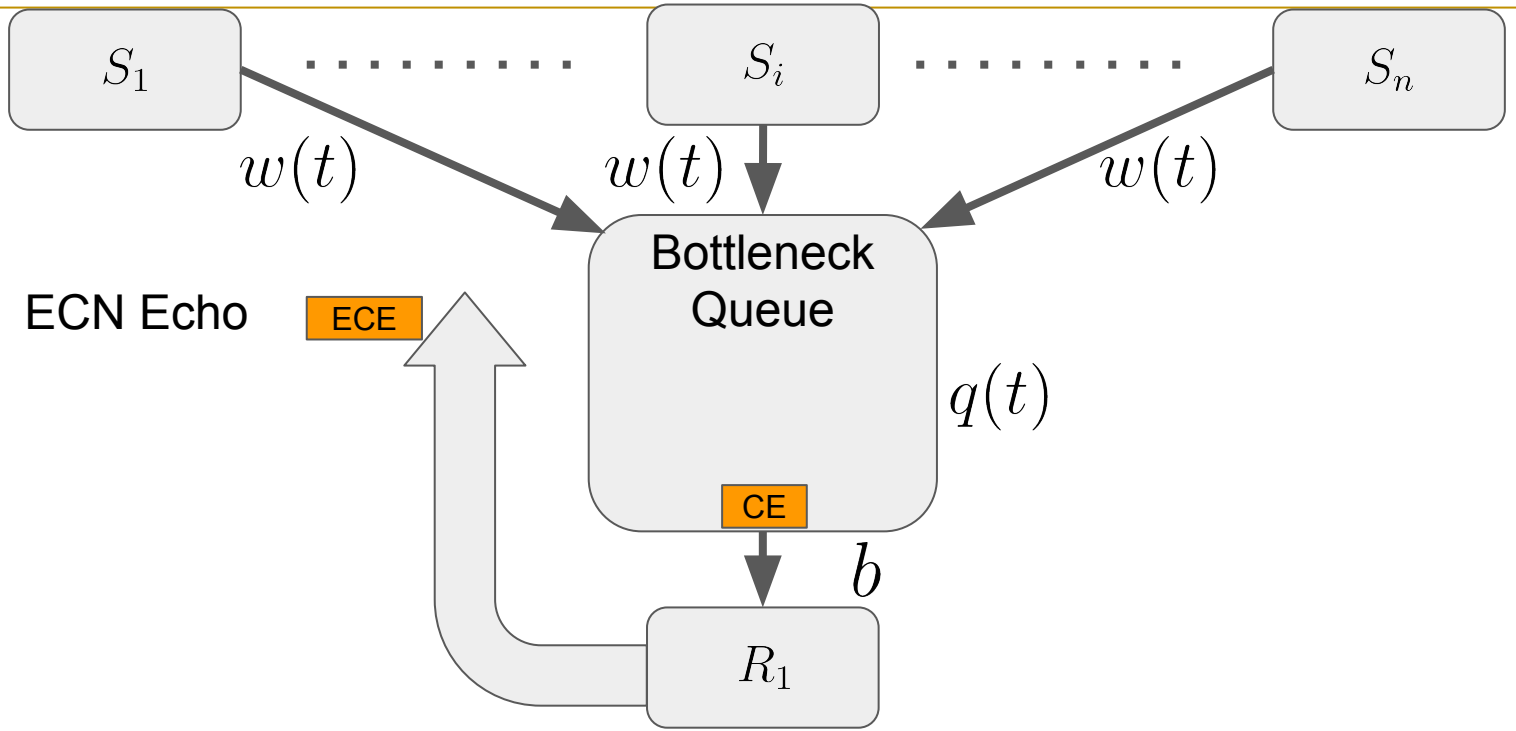
Explicit Congestion Notification (ECN)



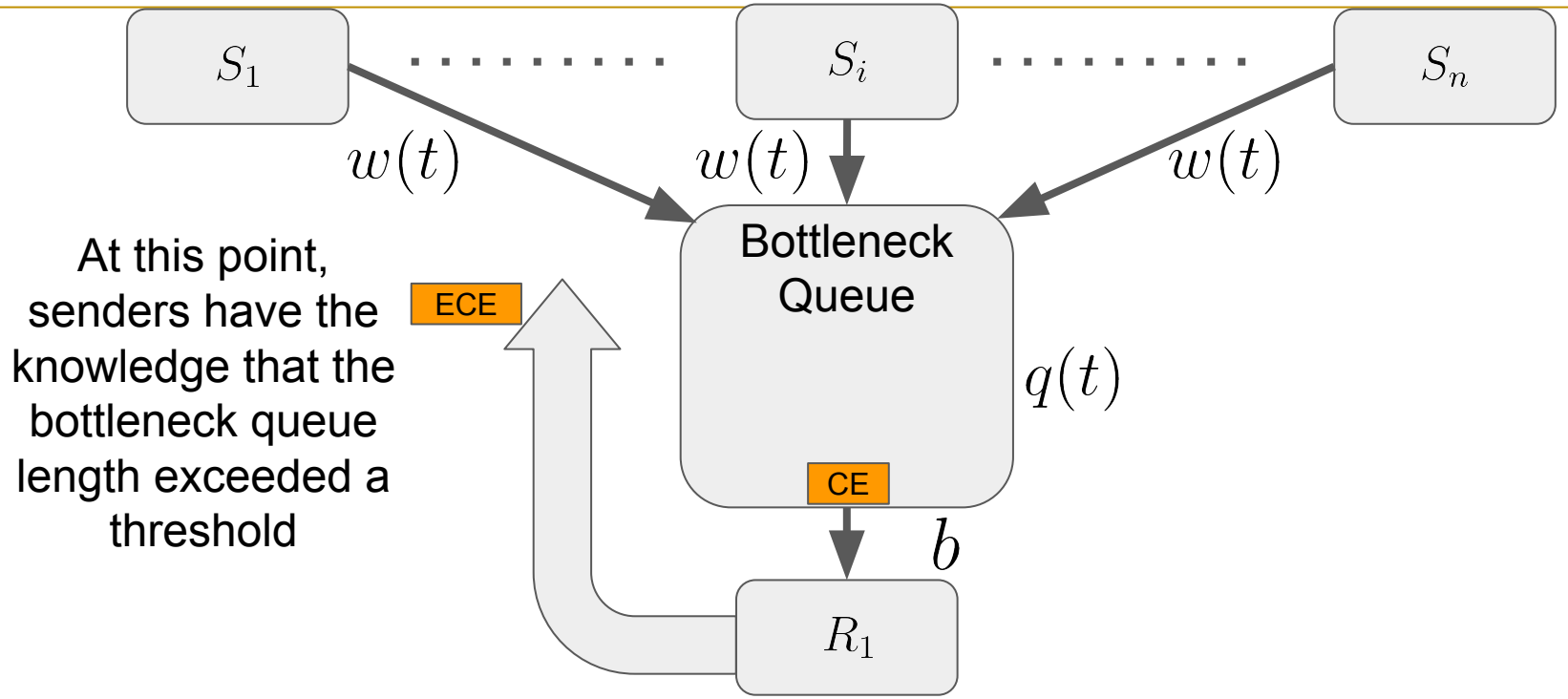
Explicit Congestion Notification (ECN)



Explicit Congestion Notification (ECN)



Explicit Congestion Notification (ECN)



At this point, senders have the knowledge that the bottleneck queue length exceeded a threshold

PURDUE UNIVERSITY

STyGIANet

DCTCP

- Same as TCP RENO
  - Slow start
  - Fast recovery
  - Retransmissions
  - Additive Increase
- But…..

DCTCP

- Same as TCP RENO
  - Slow start
  - Fast recovery
  - Retransmissions
  - Additive Increase
- But…..
  - ~~reduce the window by ½ upon packet loss like RENO~~

DCTCP

- ECN-based Congestion Control at the end-host
- Network switches (bottleneck queues) mark CE when the queue length exceeds *K*
  - *K* is a parameter

DCTCP

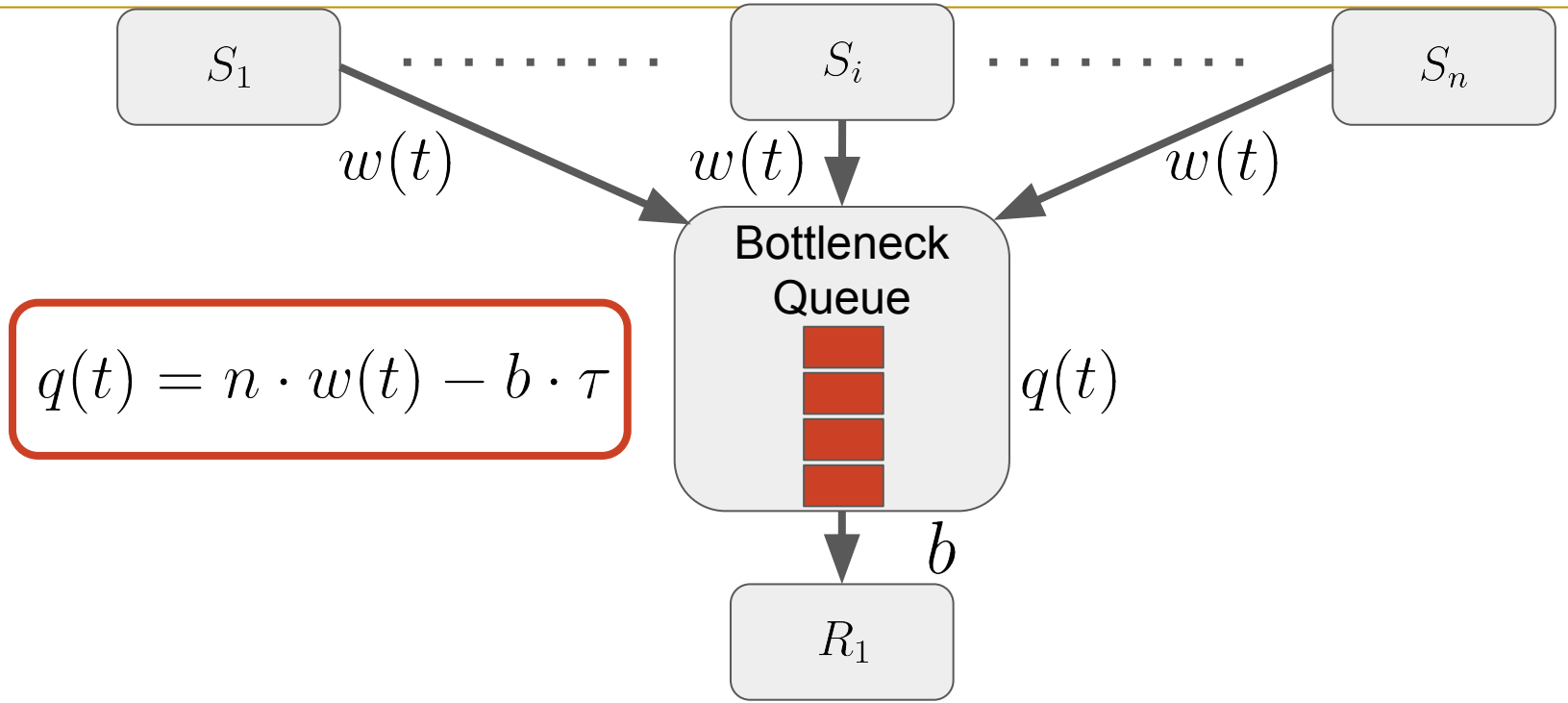
- The sender maintains an estimate of the fraction of packets that are marked, called α, which is updated once for every window of data (roughly one RTT)

$$\alpha \leftarrow (1 - g) \cdot \alpha + g \cdot F$$

- F: The fraction of packets that were marked in the last window of data
- g : Moving average parameter for the estimation of α
- Congestion window is updated based on α, the fraction of packets that experienced congestion in the network

$$cwnd \leftarrow cwnd \cdot (1 - \frac{\alpha}{2})$$

DCTCP

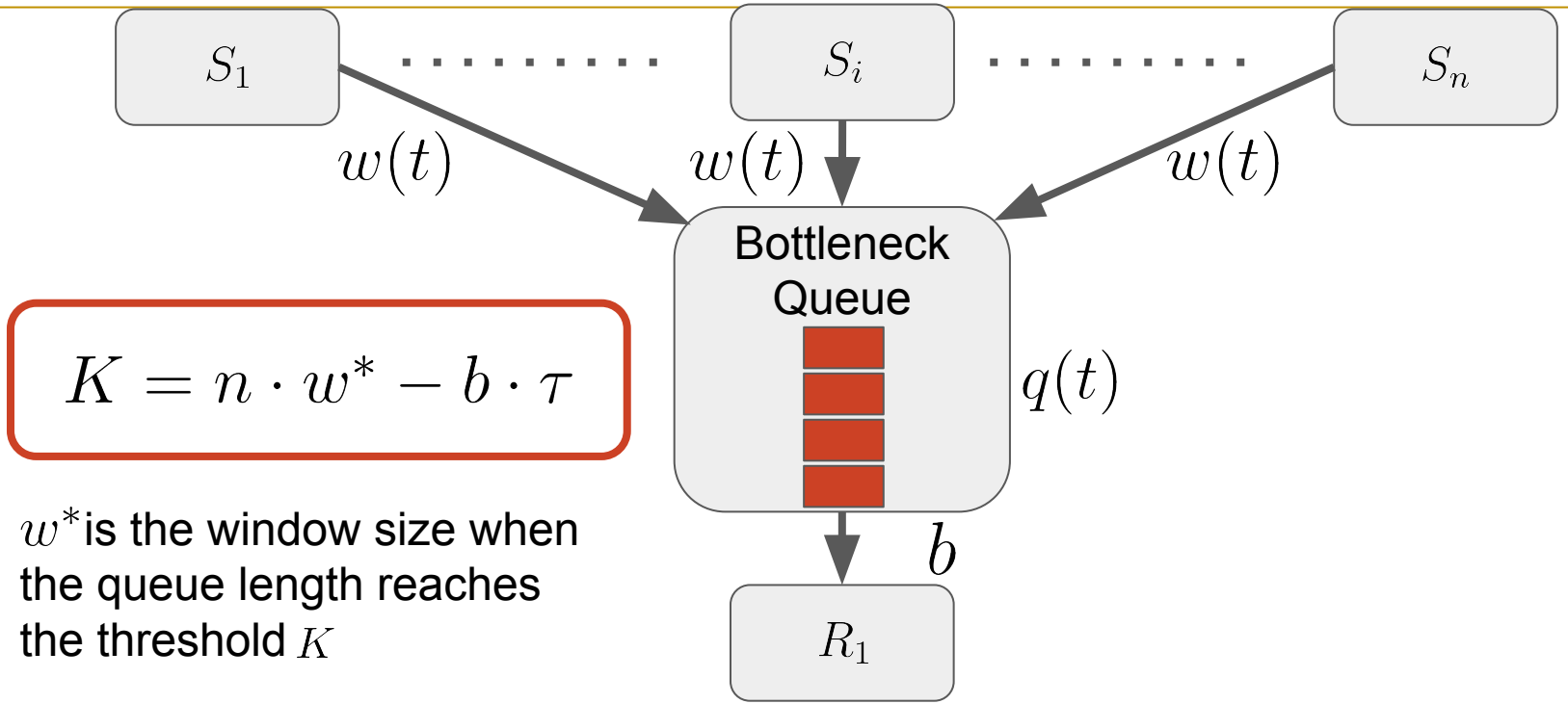


$$q(t) = n \cdot w(t) - b \cdot \tau$$

PURDUE UNIVERSITY

STyGIANet

DCTCP



$$K = n \cdot w^* - b \cdot \tau$$

$w^*$is the window size when the queue length reaches the threshold K

PURDUE UNIVERSITY

STyGIANet

DCTCP

$$K = n \cdot w^* - b \cdot \tau$$

$w^*$is the window size when the queue length reaches the threshold K

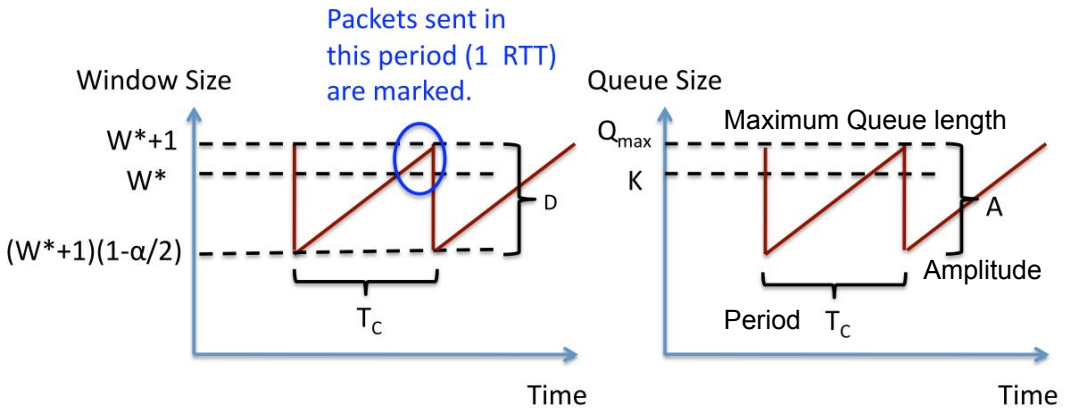


Image source: [1]

PURDUE UNIVERSITY

STyGIANet

DCTCP

- Let $S(w(t_1), w(t_2))$ be the number of packets transmitted between the window sizes $w(t_1)$and $w(t_2)$
- It takes $w(t_2) - w(t_1)$ RTTs
- Avg. window size during this period is $\frac{w(t_2)+w(t_1)}{2}$

$$s(w(t_1), w(t_2)) = \frac{w(t_2)^2 - w(t_1)^2}{2}$$

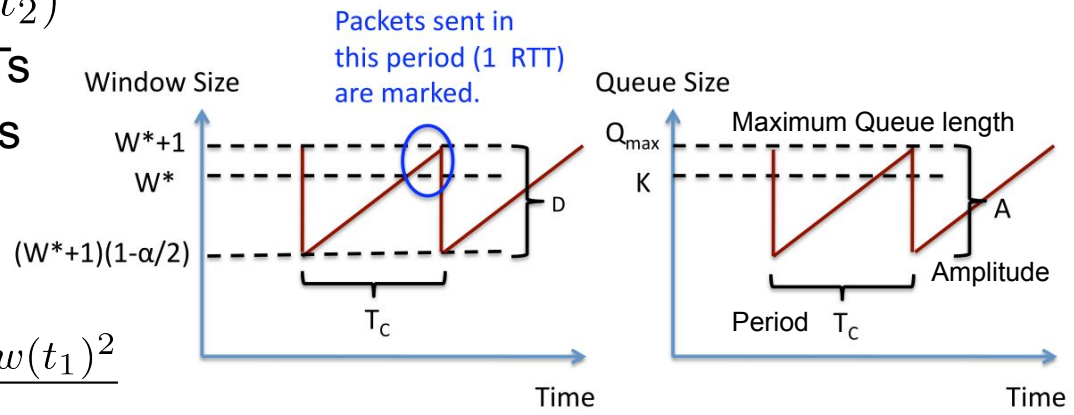


Image source: [1]

DCTCP

- The sender maintains an estimate of the fraction of packets that are marked, called α, which is updated once for every window of data (roughly one RTT)

$$\alpha = \frac{S(w^*, w^*+1)}{S((w^*+1)\cdot(1-\frac{\alpha}{2}), w^*+1)}$$

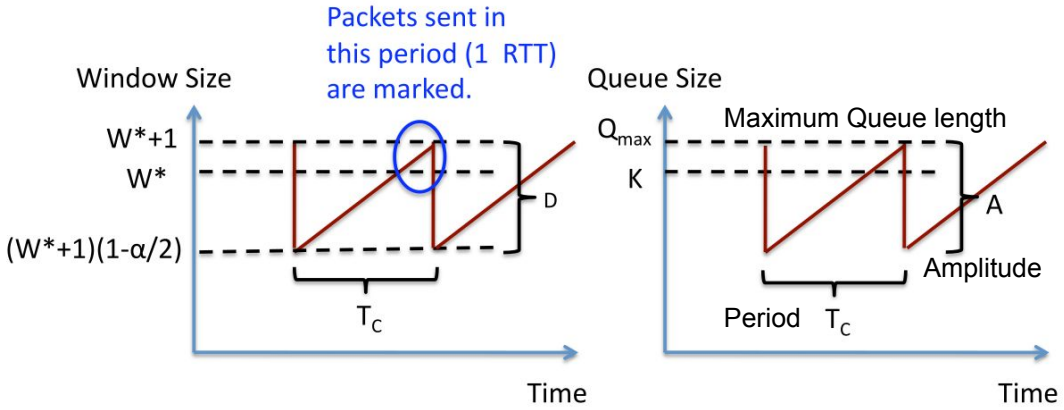


Image source: [1]

PURDUE UNIVERSITY

STyGIANet

DCTCP

- The sender maintains an estimate of the fraction of packets that are marked, called α, which is updated once for every window of data (roughly one RTT)

$$\alpha = \frac{S(w^*, w^*+1)}{S((w^*+1)\cdot(1-\frac{\alpha}{2}), w^*+1)}$$

$$\alpha \approx \sqrt{\frac{2}{w^*}}$$

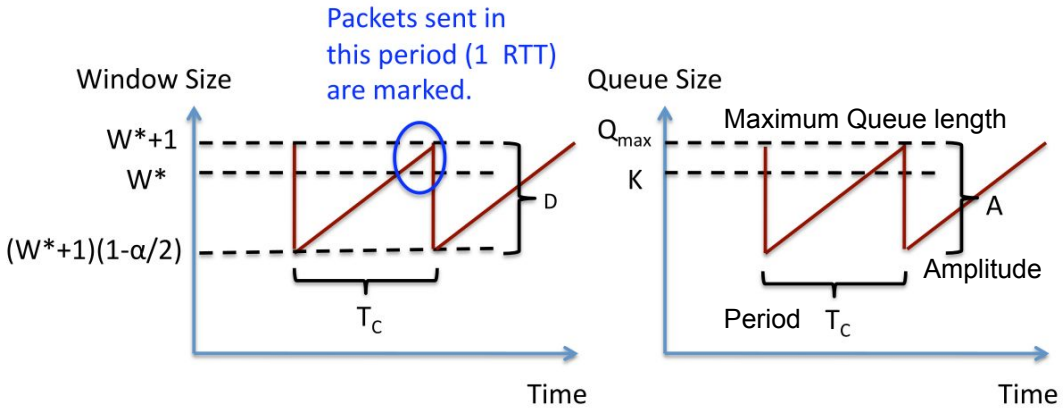


Image source: [1]

DCTCP

- Amplitude: n flows, each oscillating between $w^* + 1$ and $(w^* + 1) \cdot (1 - \frac{\alpha}{2})$

$$A = n \cdot (w^* + 1) \cdot \frac{\alpha}{2} \approx \frac{n}{2} \cdot \sqrt{2 \cdot w^*}$$

$$= \frac{1}{2} \cdot \sqrt{2 \cdot n \cdot (b \cdot \tau + K)}$$

$$T_c = \frac{A}{n}$$

$$Q_{max} = n \cdot (w^* + 1) - b \cdot \tau = K + n$$

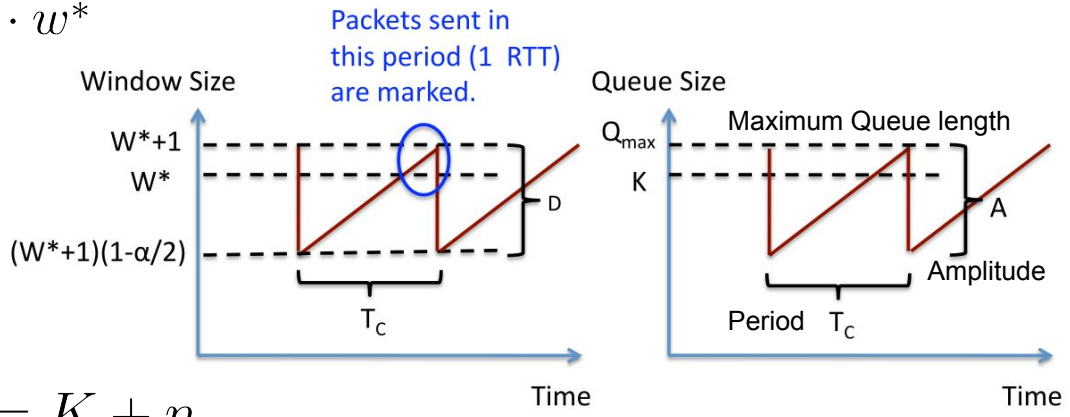


Image source: [1]

PURDUE UNIVERSITY

STyGIANet

DCTCP

- Guidelines for choosing parameters:
  - Choose *K* such that $K > \frac{b \cdot \tau}{7}$
  - Set *K* to 20 packets for 1Gbps links
  - Set *K* to 65 packets for 10Gbps links

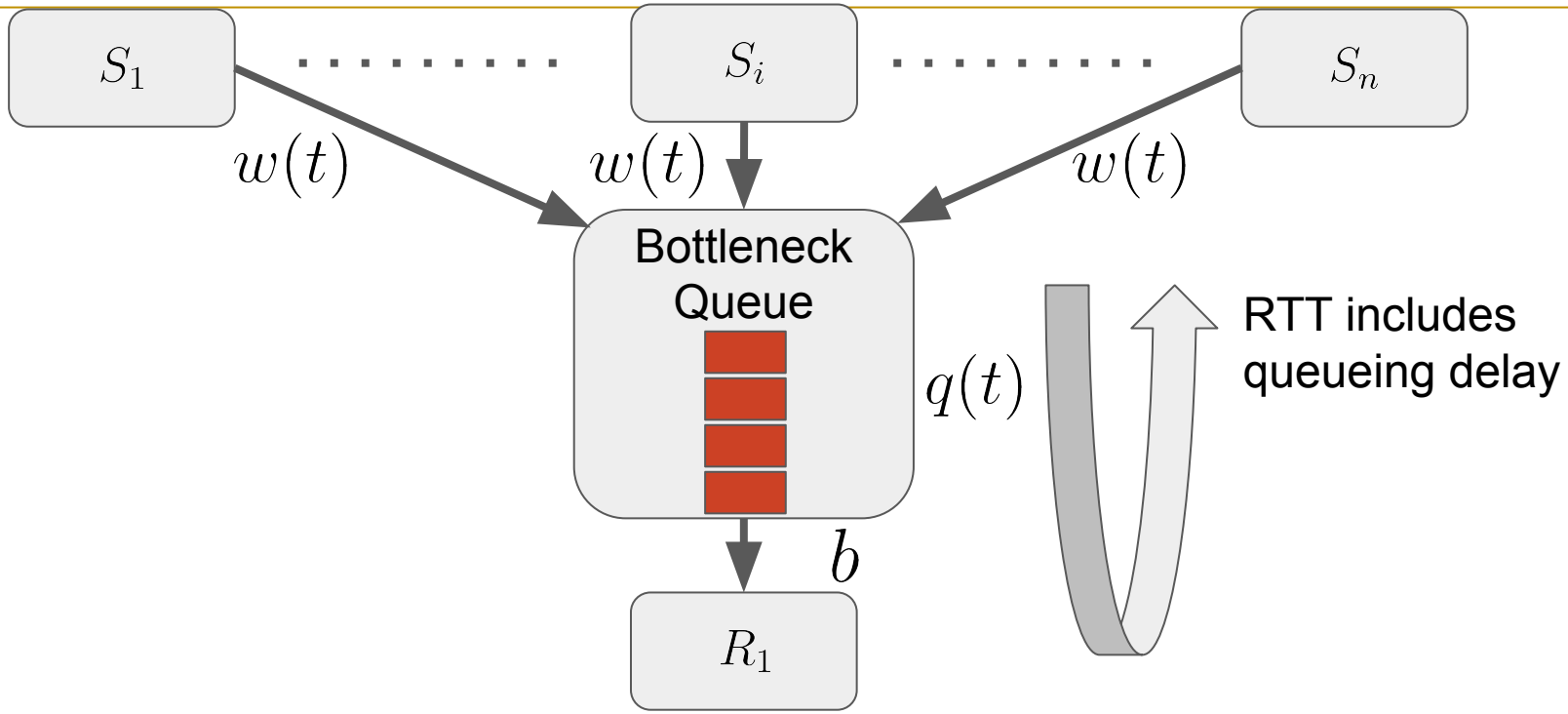
DCTCP

- Advantages:
  - Simple and effective
  - Nearly all datacenter switches support ECN marking
  - Reduces queueing delays compared to loss-based TCP eg., RENO
  - High throughput due to proportional decrease based on α

DCTCP

- Drawbacks:
  - Higher queue lengths that the desired queue length of near-zero
  - Optimally configuring the parameter *K* is hard in practice
  - ECN is queue **length-**based and does not necessarily indicate queueing delay
  - RTT variability in the topology (eg., fattree) requires different *K* parameter for different type of flows
    - Notice that our analysis assumes a single RTT value for all the flows

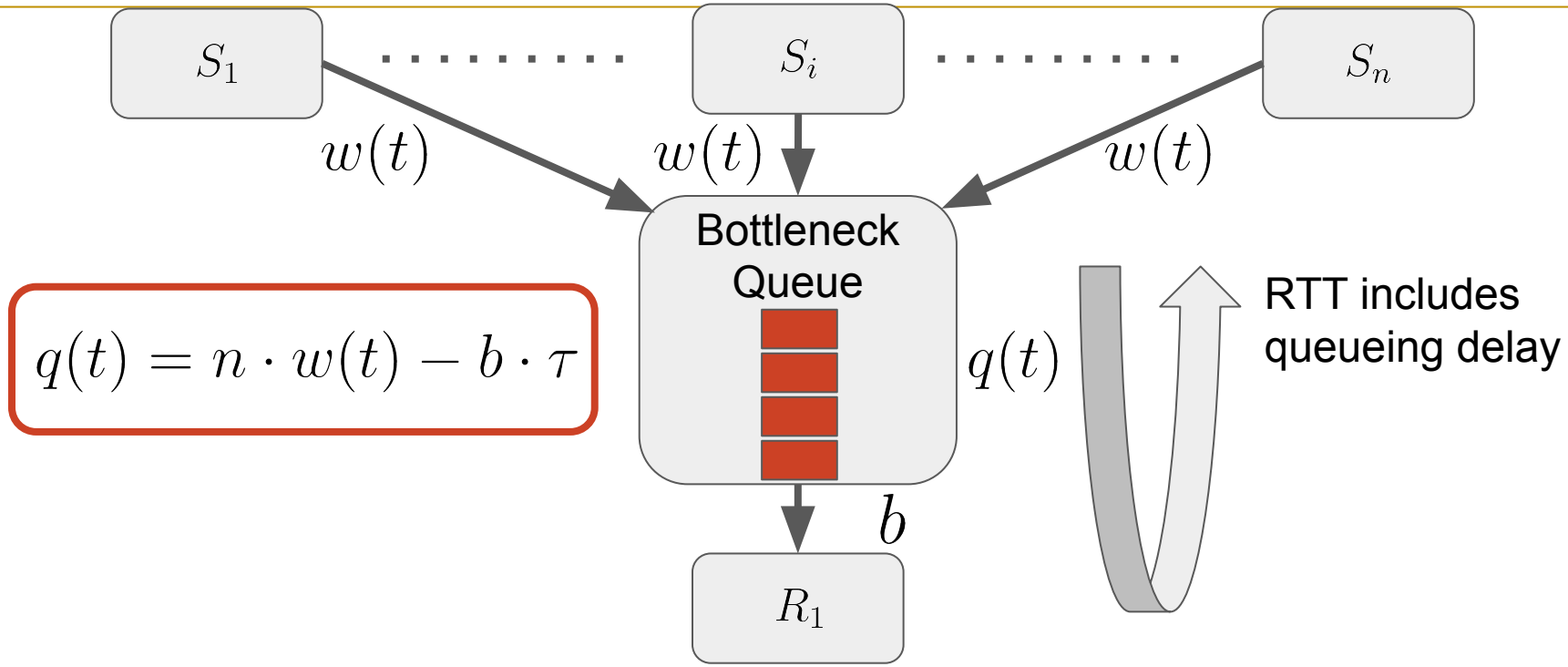
Round-trip-time (RTT) for Congestion Control



PURDUE UNIVERSITY

STyGIANet

Round-trip-time (RTT) for Congestion Control



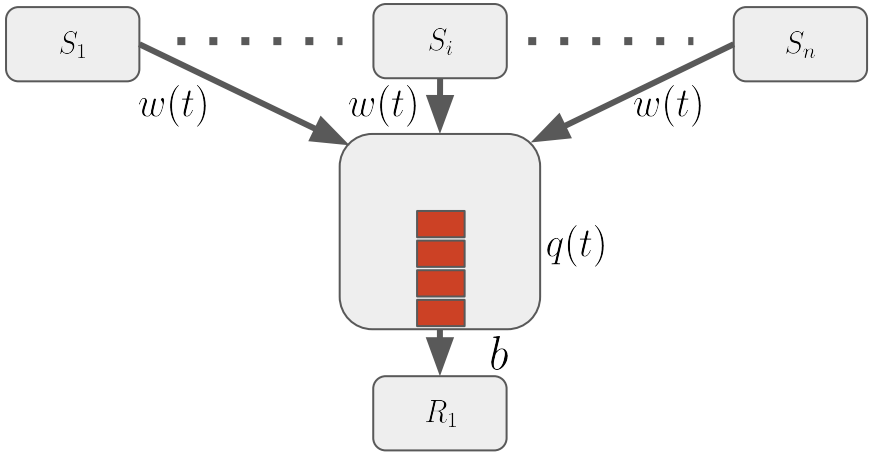
PURDUE UNIVERSITY

STyGIANet

Round-trip-time (RTT) for Congestion Control

- How can we use RTT to adjust $w(t)$ in order to achieve the desirable properties such as low latency and high throughput?

$$q(t) = n \cdot w(t) - b \cdot \tau$$



TIMELY

- Rate-based congestion control
  - Transmission rate is controlled directly, instead of using cwnd
- RTT-gradient approach
- High precision timestamps are obtained from Network Interface Cards (NIC)

TIMELY

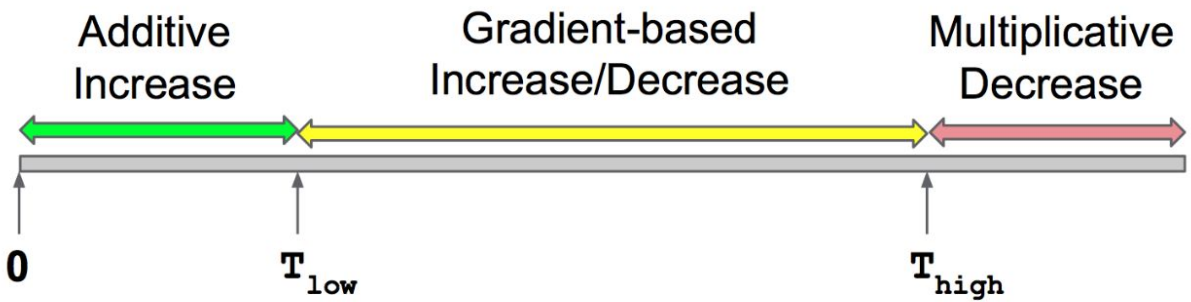


Image source: [2]

PURDUE UNIVERSITY

STyGIANet

TIMELY

- θ_{new} : Current round-trip-time
- θ_{prev} : Previous round-trip-time
- $\dot{\theta}_{new} = \theta_{new} - \theta_{prev}$: round-trip-time gradient
- $\dot{\theta} = (1 - \alpha) \cdot \dot{\theta} + \alpha \cdot \dot{\theta}_{new}$: Moving averaged RTT gradient
  - α is the moving average parameter

TIMELY

- If θ_{new} is less than a lower threshold T_{low}
  - Additive increase: transmission rate is increased by a constant
- If θ_{new} is greater than a higher threshold T_{high}
  - Multiplicative decrease: transmission rate is decreased multiplicatively

- Else
  - Rate is adjusted based on the normalized **RTT gradient**
  - $rate \leftarrow rate \cdot (1 - \beta \cdot \frac{\dot{\theta}}{\tau})$
  - β is a parameter and τ is the baseRTT (minimum RTT)
- Other corner case actions.

TIMELY

- Advantages:
  - High precision rate control based on NIC timestamps

Note: Deployed in Google datacenters nearly 10 years ago but not used anymore.

TIMELY

- Drawbacks:
  - RTT includes the entire path delay and does not necessarily relate to the congestion experienced by data packets
  - RTT gradient-based congestion control has no unique equilibrium
    - Latency and throughput stabilize at arbitrary points

HPCC

- High Precision Congestion Control (HPCC)
- Inband Network Telemetry provides congestion feedback to the end-hosts
- Inflight-based congestion control
  - Inflight bytes: The total unacknowledged bytes that are inflight in the network
- Deployed in Alibaba

HPCC

- Inband Network Telemetry

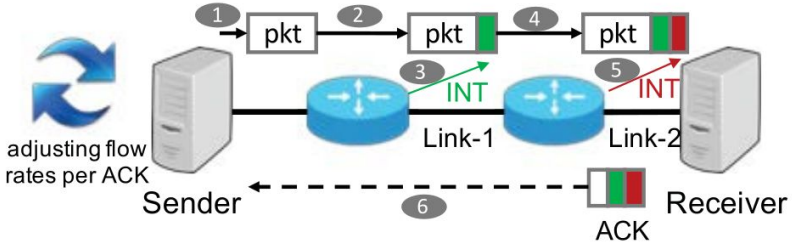


Image source: [3]

HPCC

- INT metadata inserted at each hop along the path
  - Bandwidth
  - Timestamp
  - Transmitted bytes (Tx bytes) counter value
  - Queue length

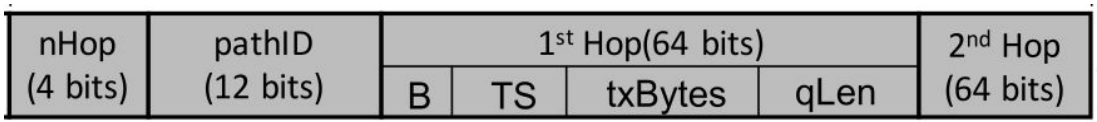
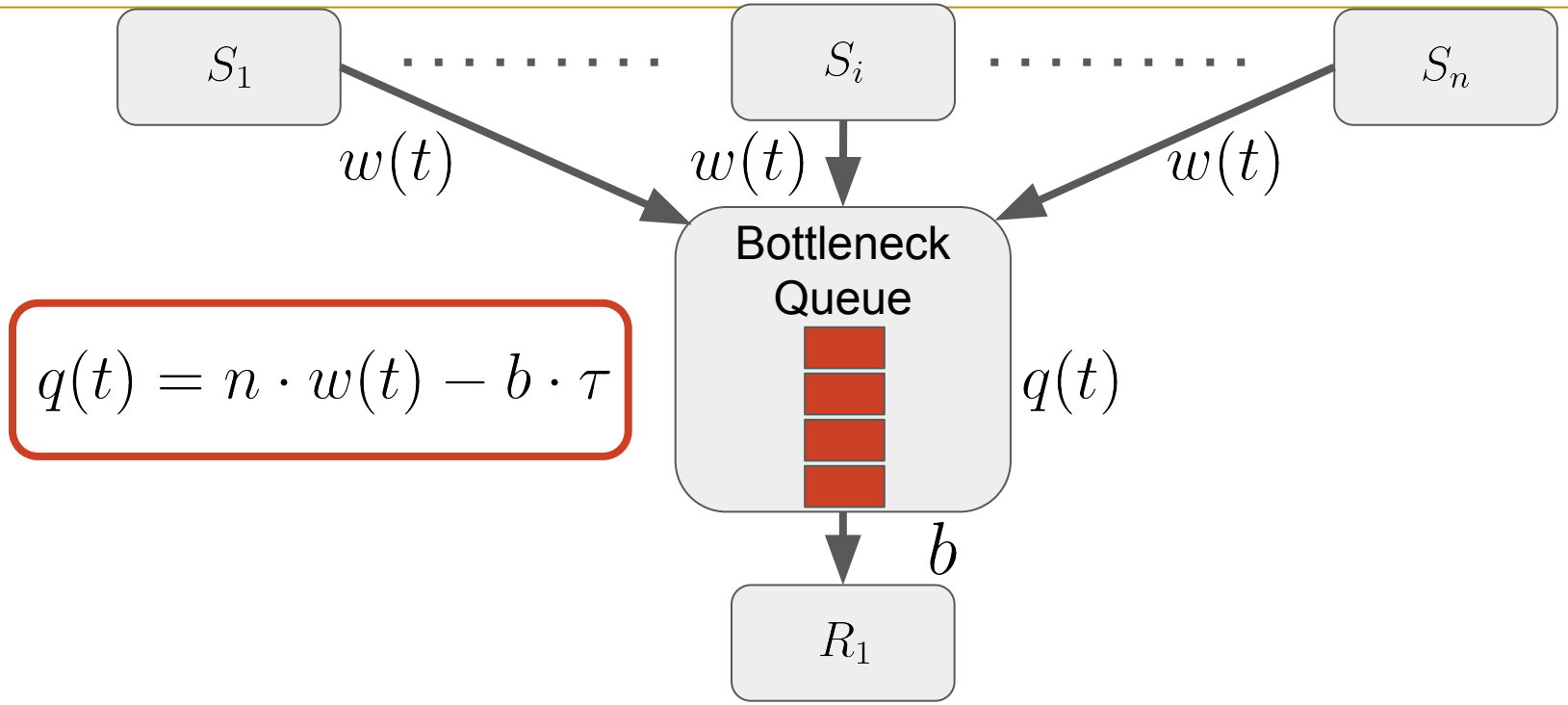


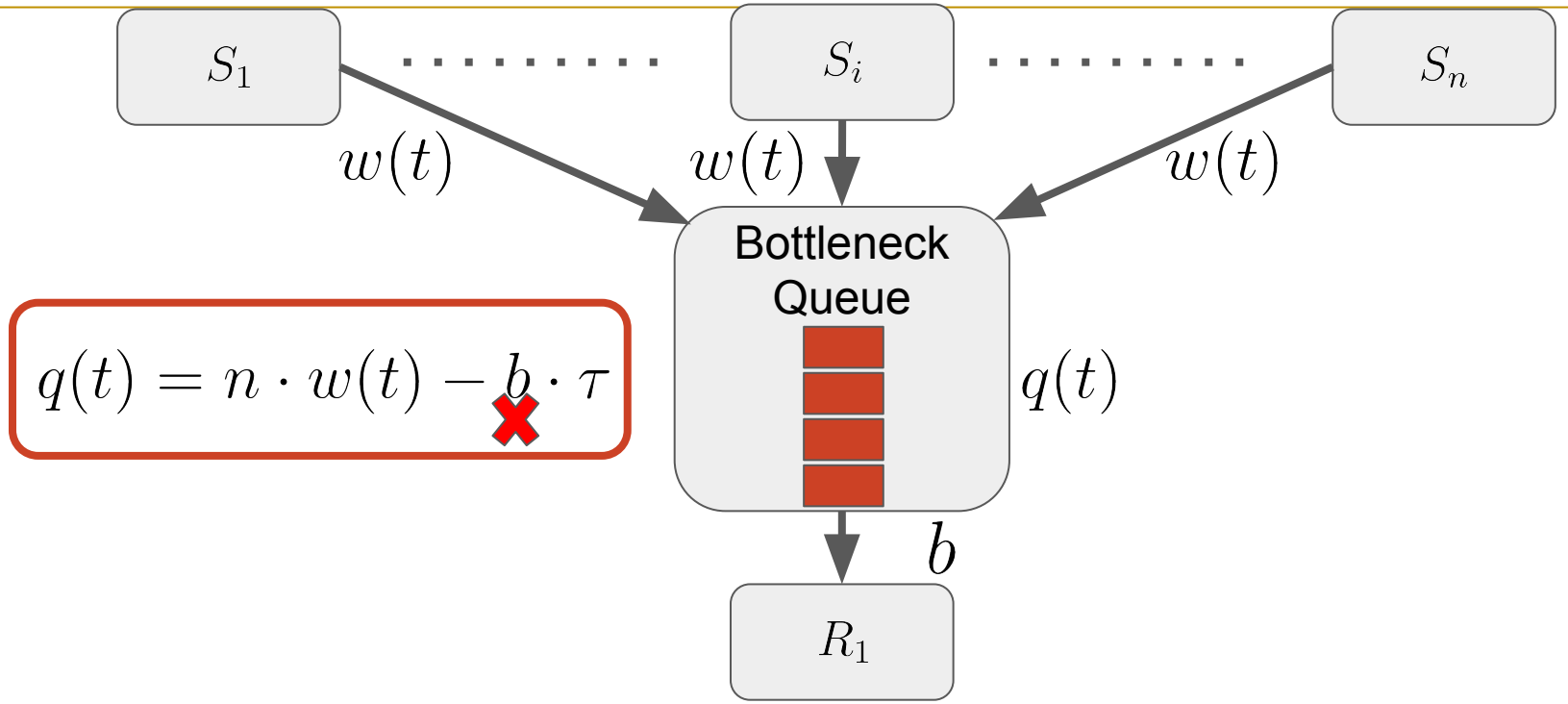
Image source: [3]

Recap: Single Bottleneck Link Simple Model



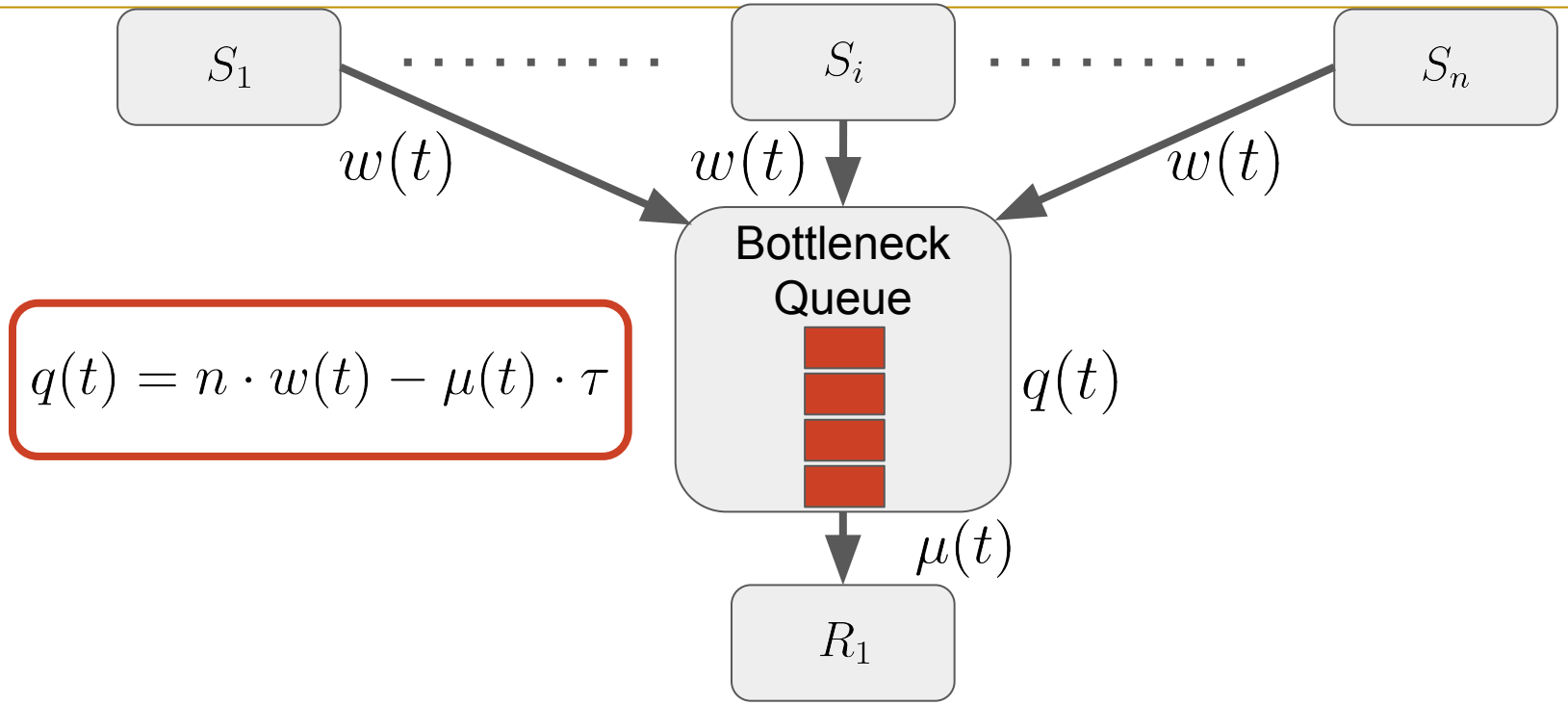
$$q(t) = n \cdot w(t) - b \cdot \tau$$

Recap: Single Bottleneck Link Simple Model



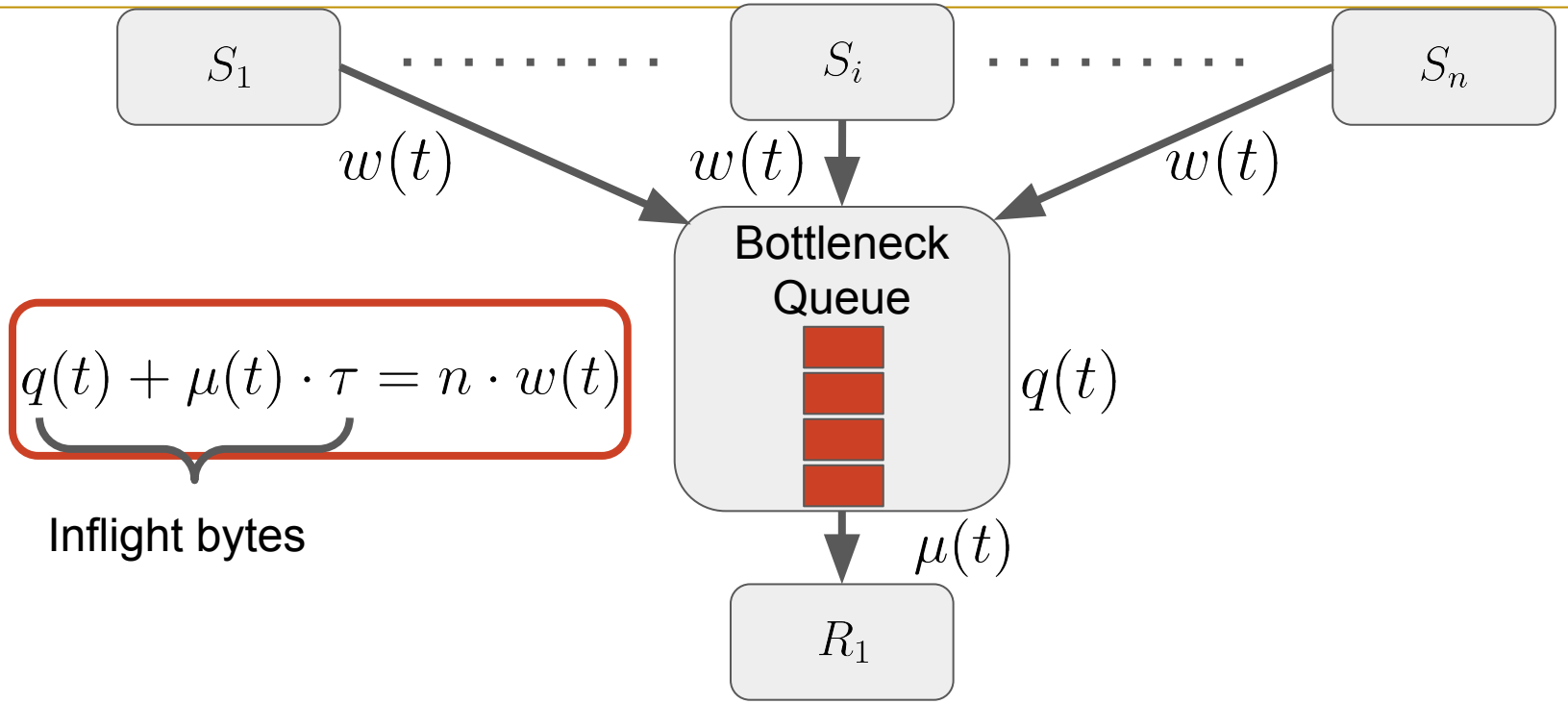
$$q(t) = n \cdot w(t) - b \cdot \tau$$

Recap: Single Bottleneck Link Simple Model



$$q(t) = n \cdot w(t) - \mu(t) \cdot \tau$$

Recap: Single Bottleneck Link Simple Model



HPCC

- Main idea:
  - Multiplicative increase and multiplicative decrease based on inflight bytes

$$\frac{q(t)+\mu(t)\cdot\tau}{b\cdot\tau}$$

$q(t)+\mu(t)\cdot\tau$ → Inflight bytes

$b\cdot\tau$ → *Ideal* Inflight bytes with maximum transmission rate and zero queue lengths.

HPCC

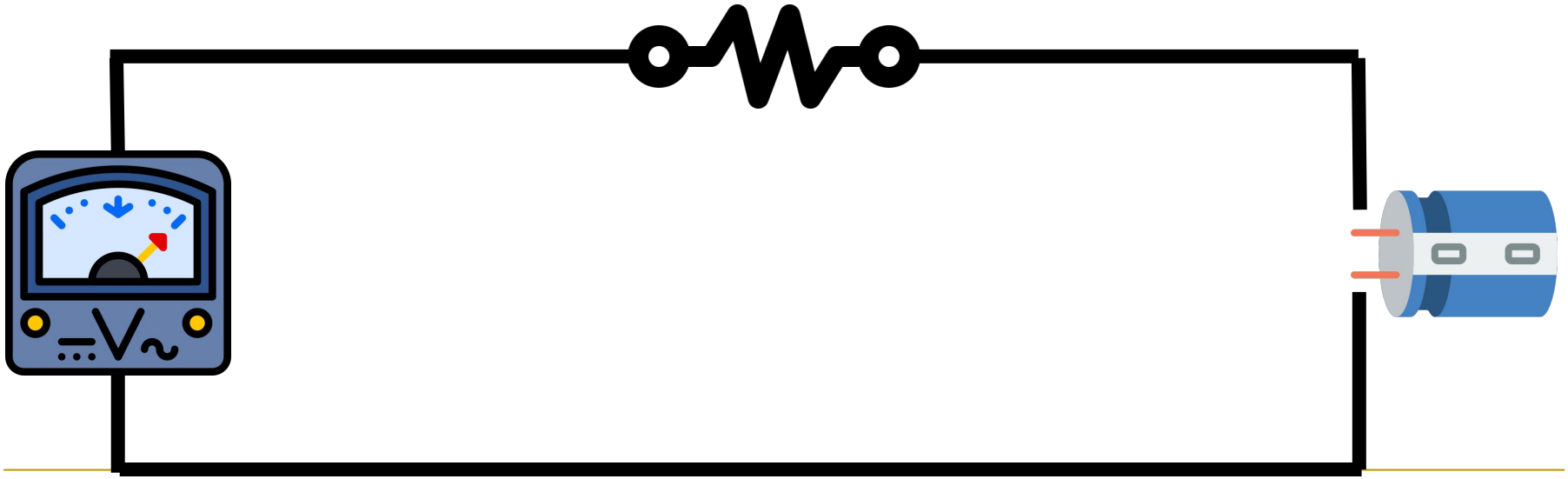
- Advantages:
  - The precision of feedback (INT) is much better than RTT measurements and ECN
  - Inflight-based congestion control achieves several properties including low latency, high throughput, equilibrium and fairness

HPCC

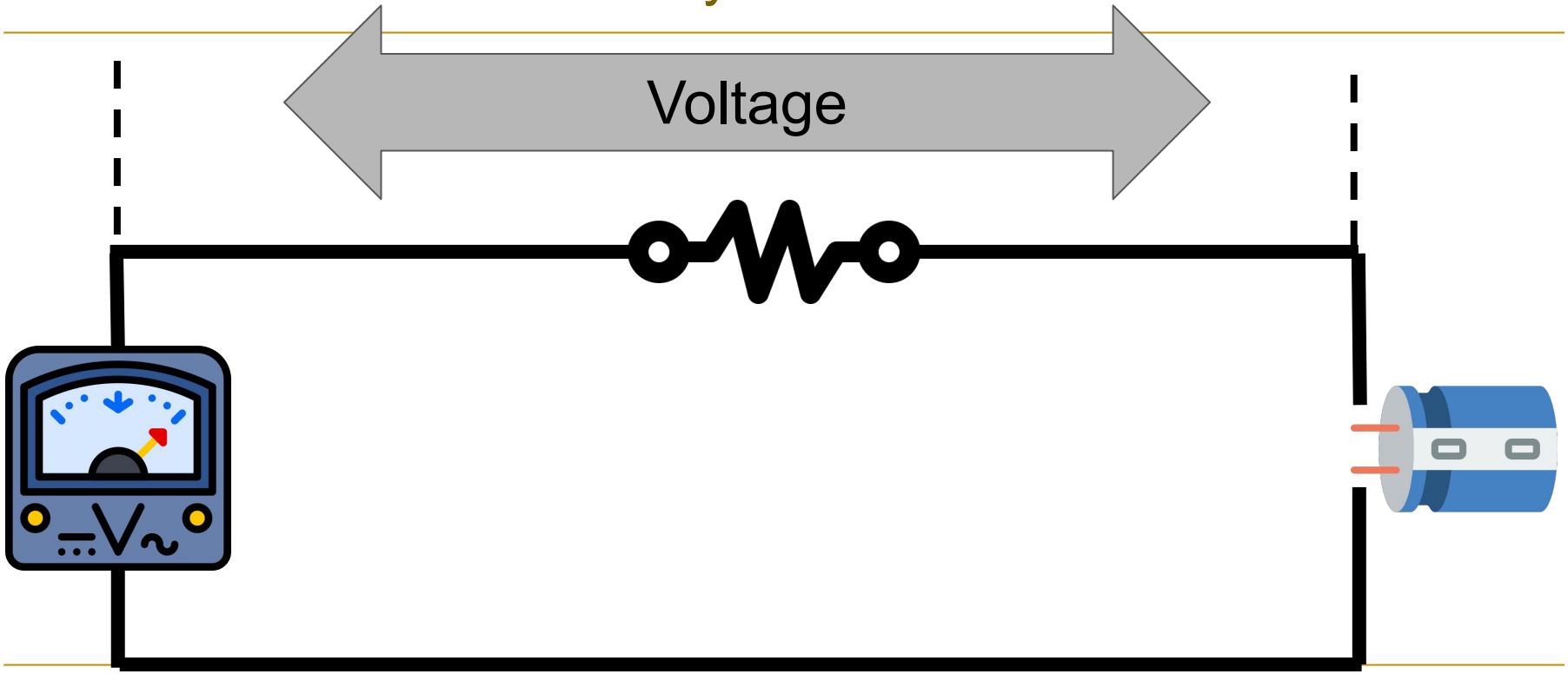
- Disadvantages:
  - Network overhead due to INT headers and per-hop metadata
  - Inflight-based congestion control does not react promptly until the queue lengths grow

Can we do better?

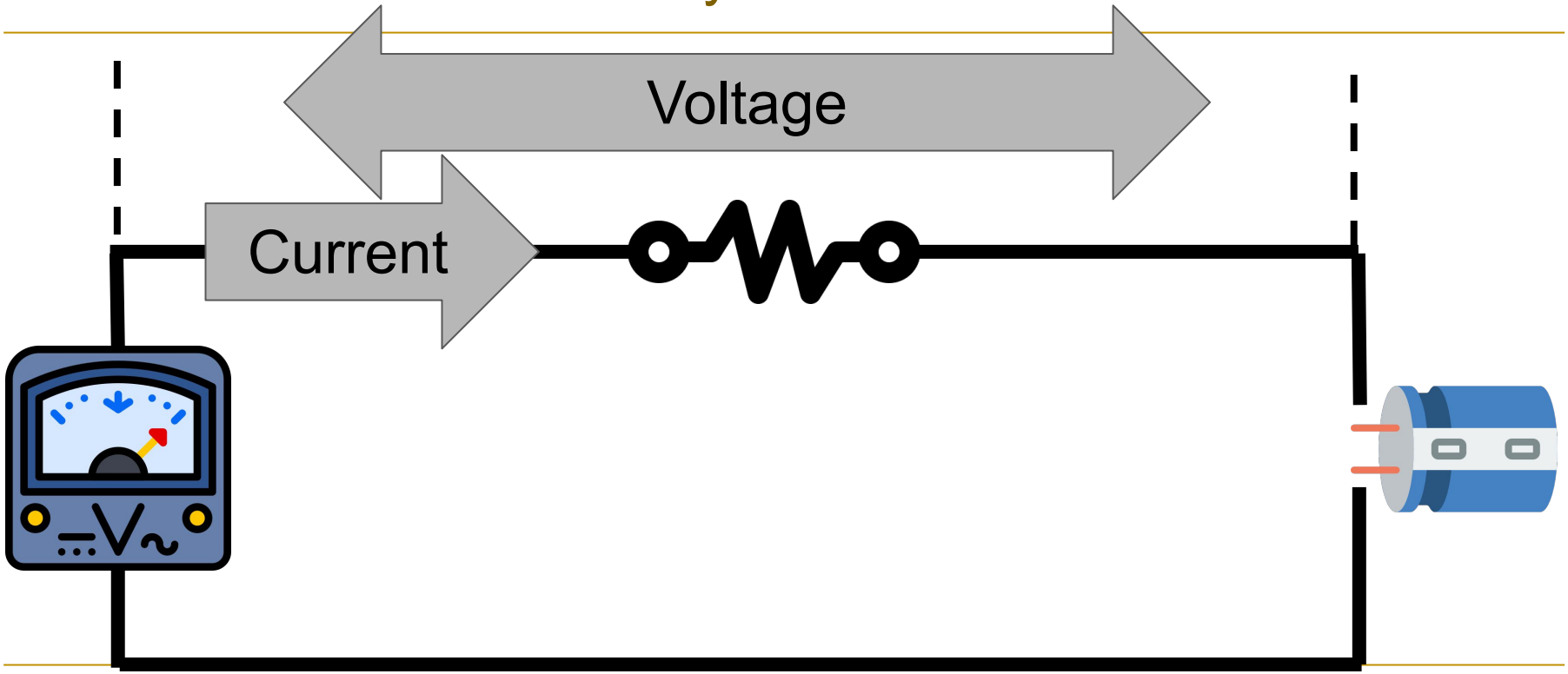
Brief context of electrical systems



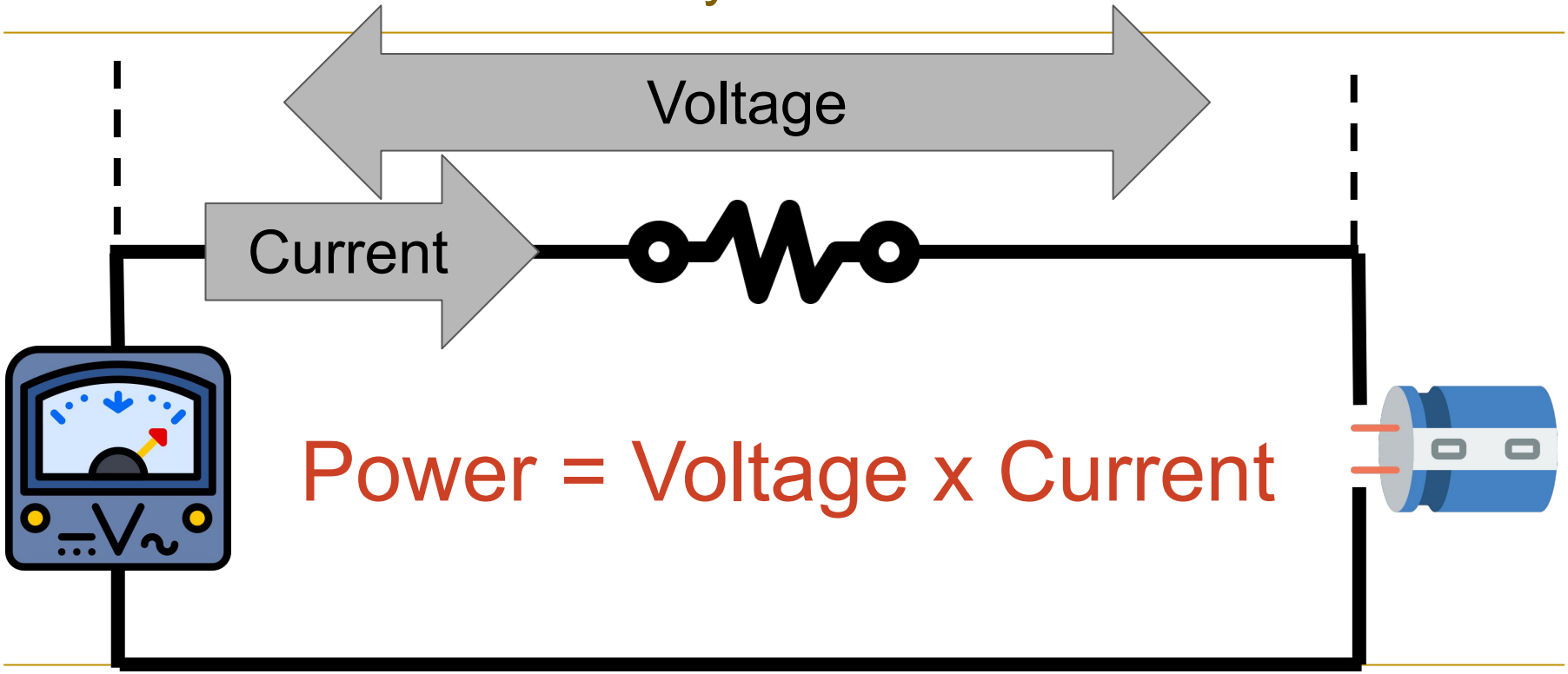
Brief context of electrical systems



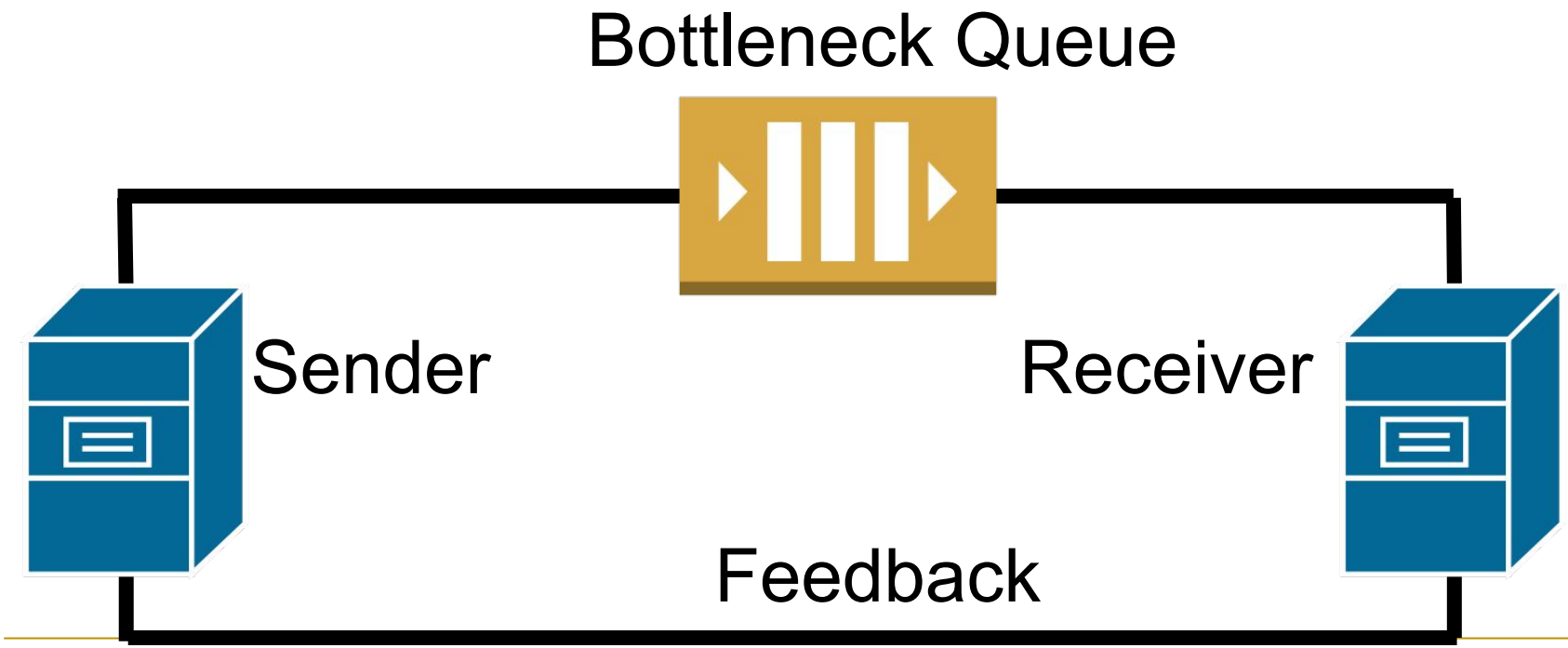
Brief context of electrical systems



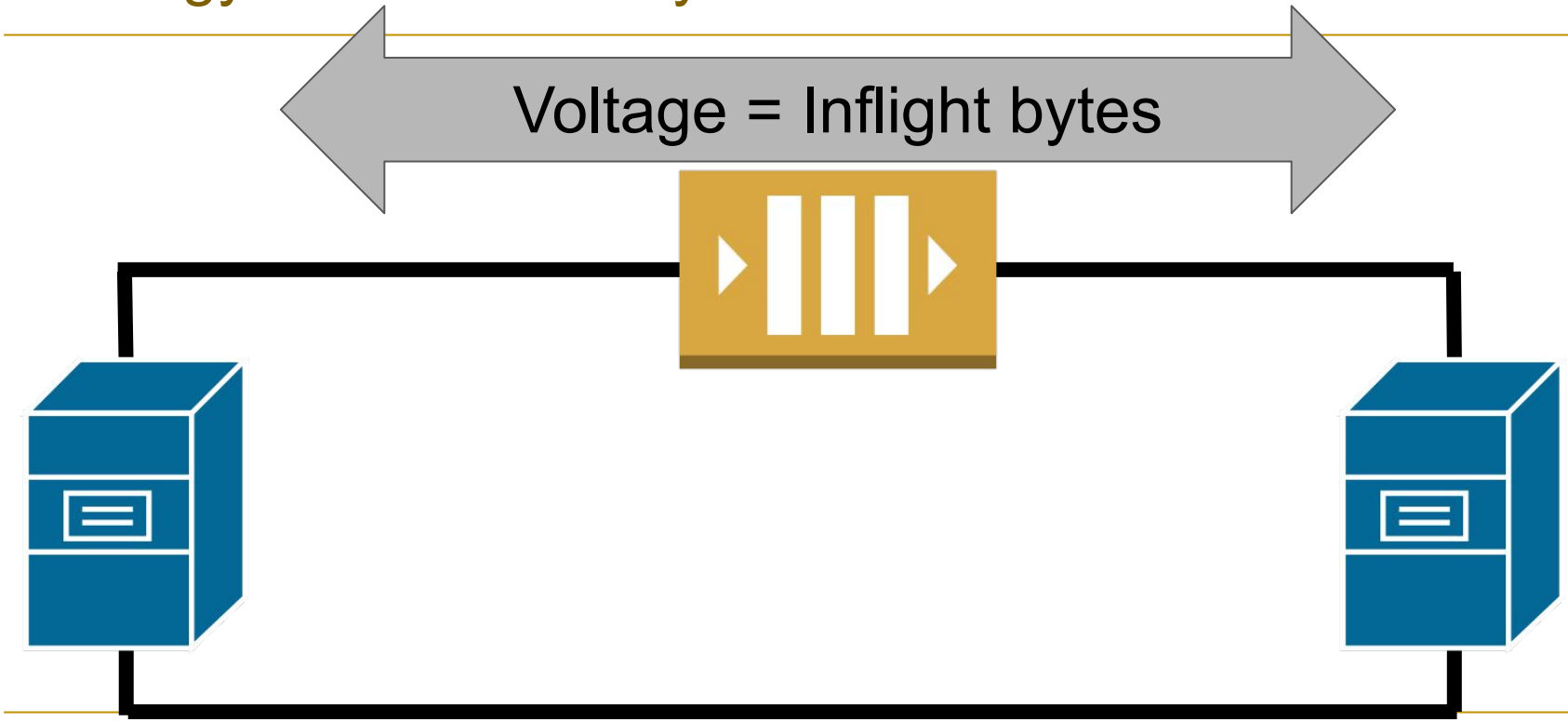
Brief context of electrical systems



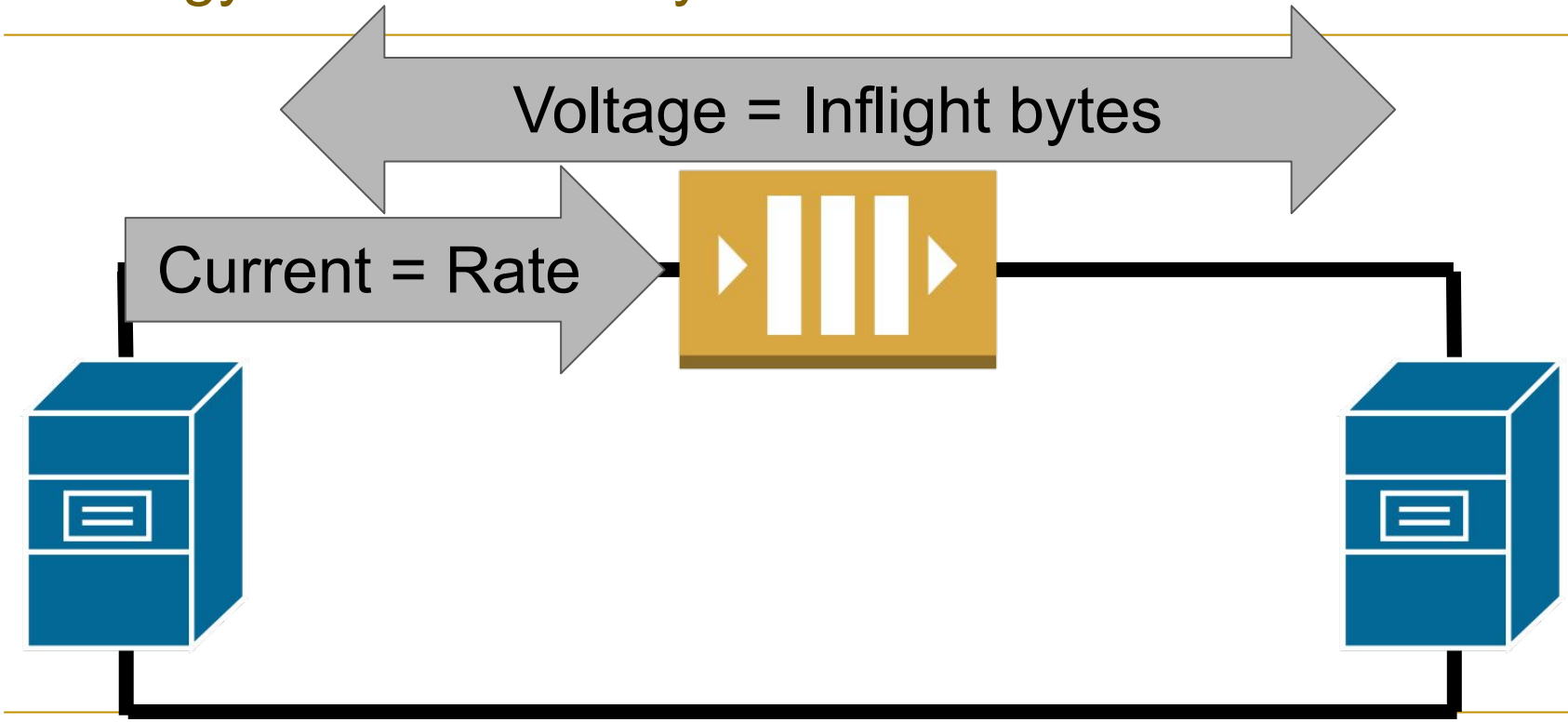
Analogy to networked systems



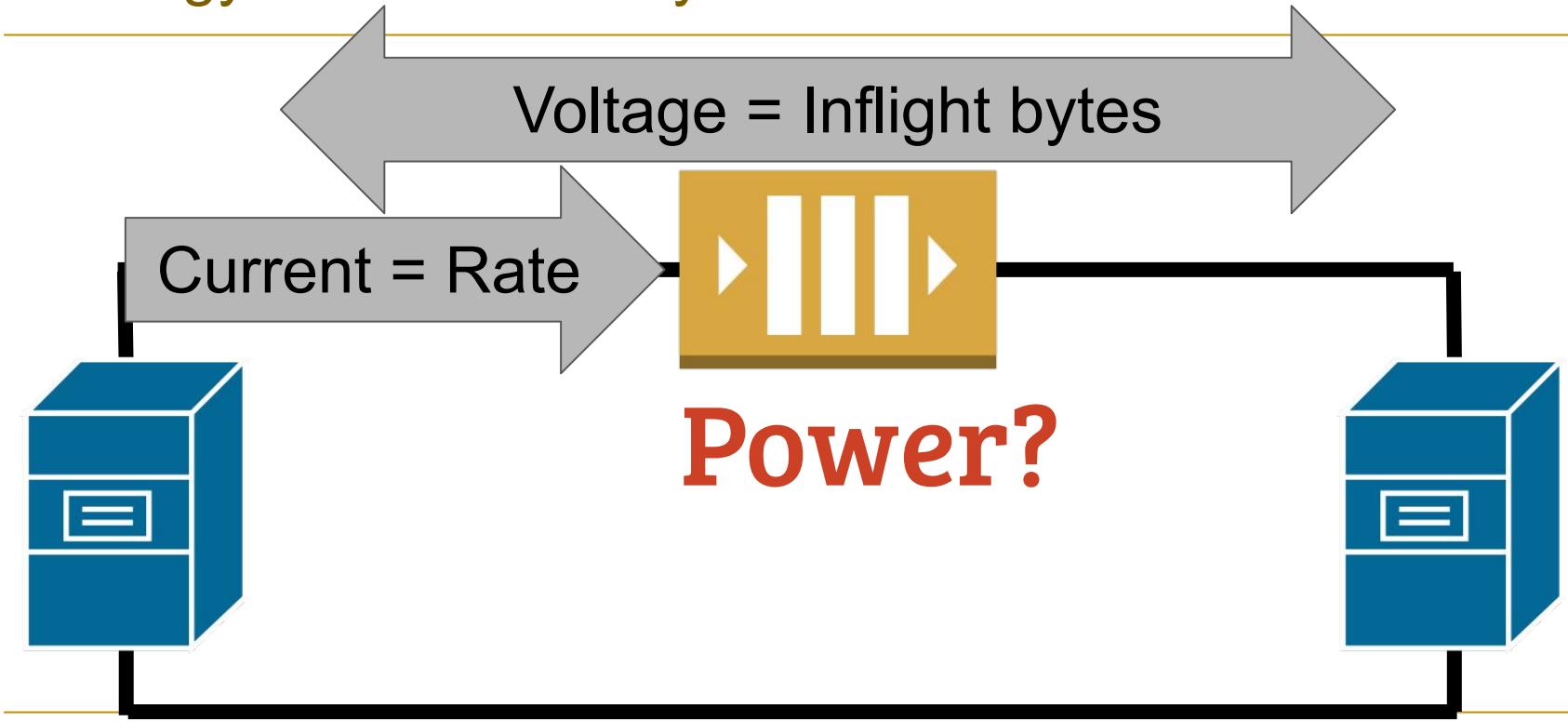
Analogy to networked systems



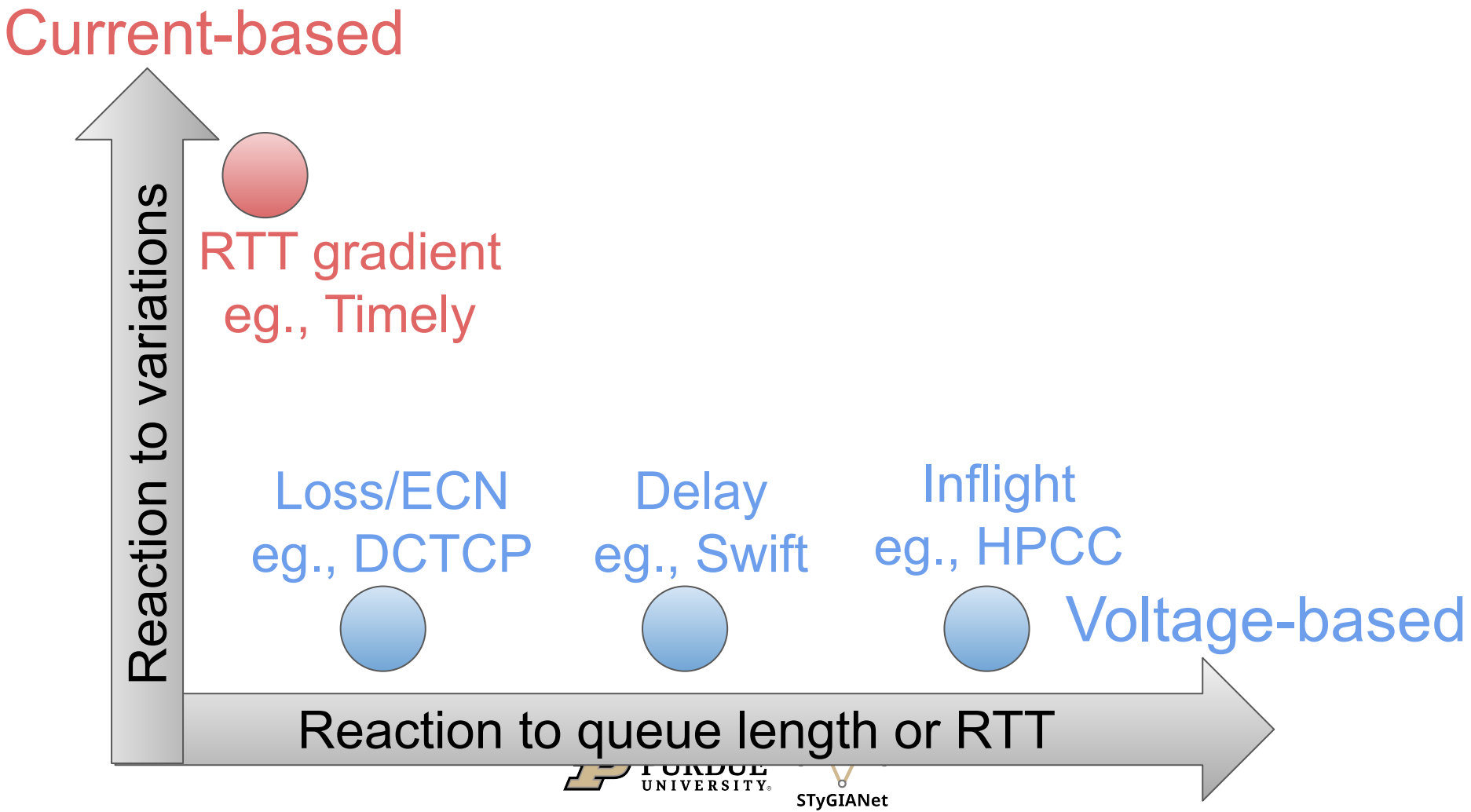
Analogy to networked systems



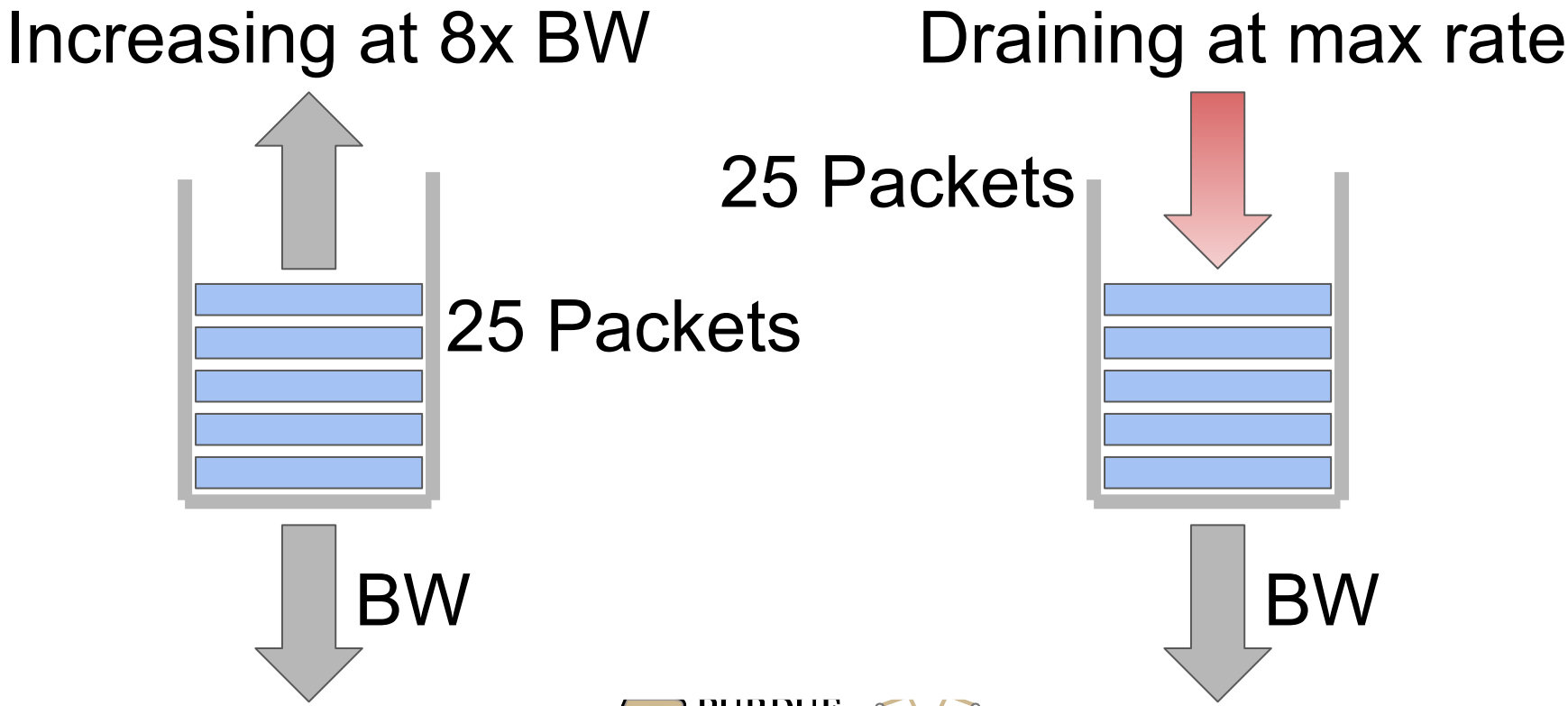
Analogy to networked systems



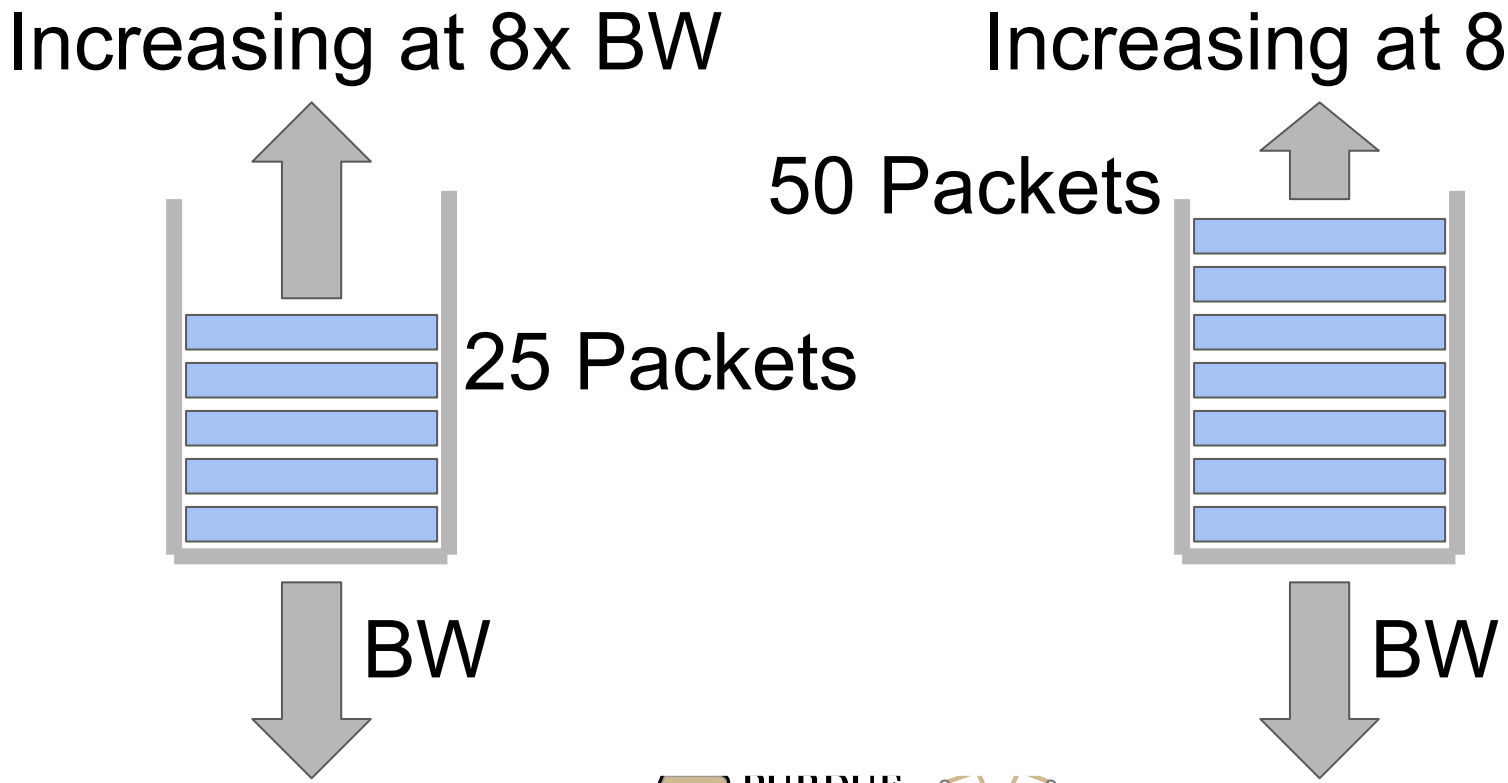
PowerTCP


Current-based
Reaction to variations
RTT gradient
eg., Timely
Loss/ECN
eg., DCTCP
Delay
eg., Swift
Inflight
eg., HPCC
Voltage-based
Reaction to queue length or RTT
PURDUE UNIVERSITY
STyGIANet

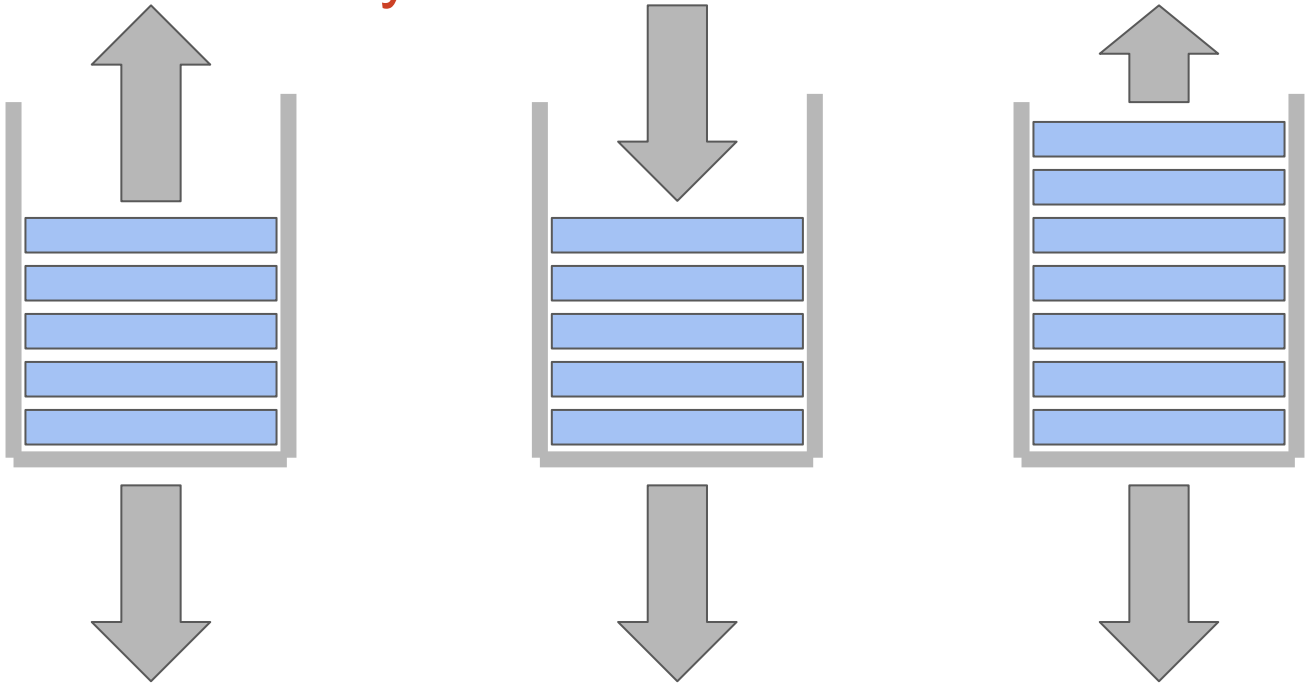
Indistinguishable by HPCC



Indistinguishable by TIMELY

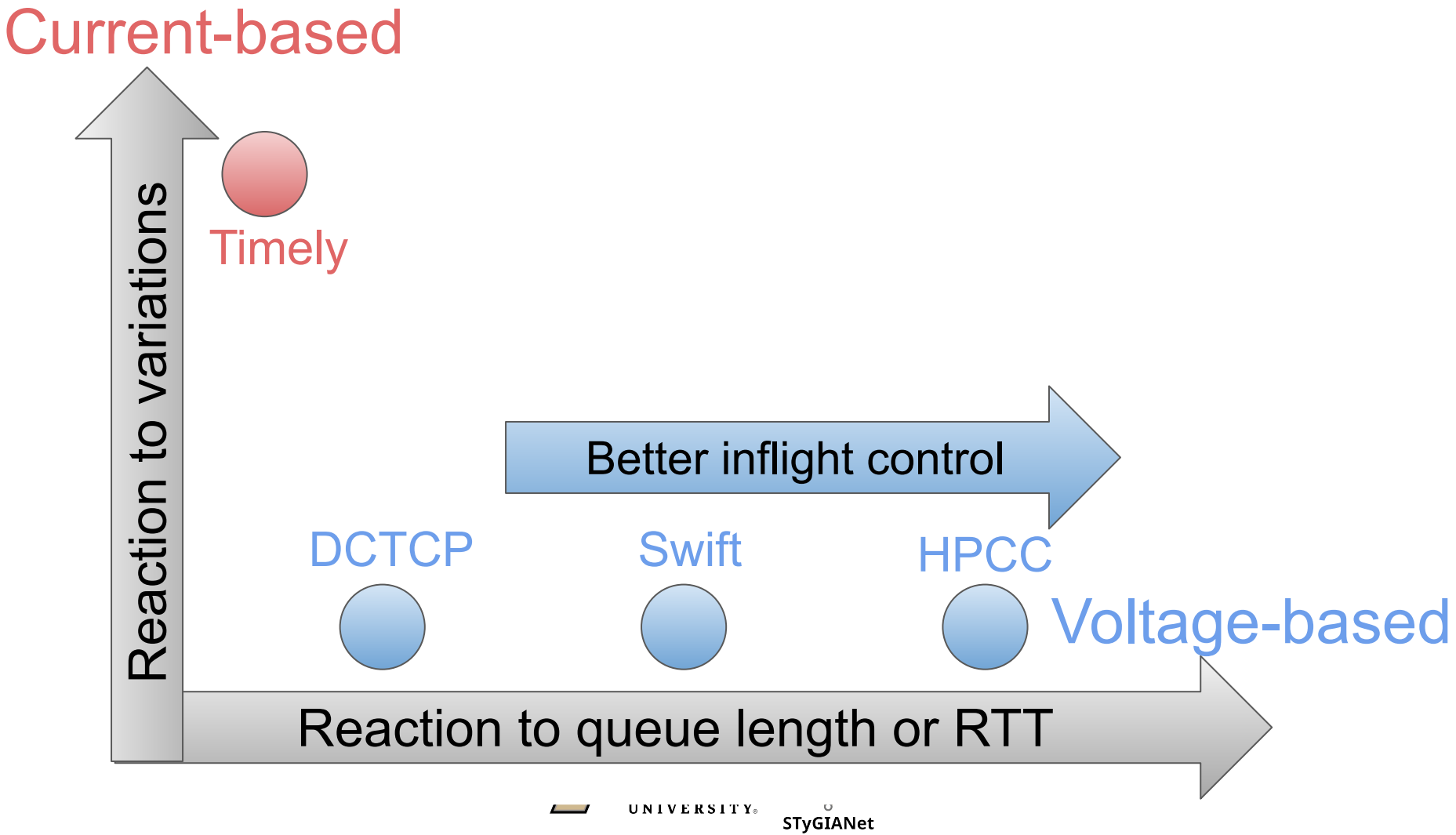


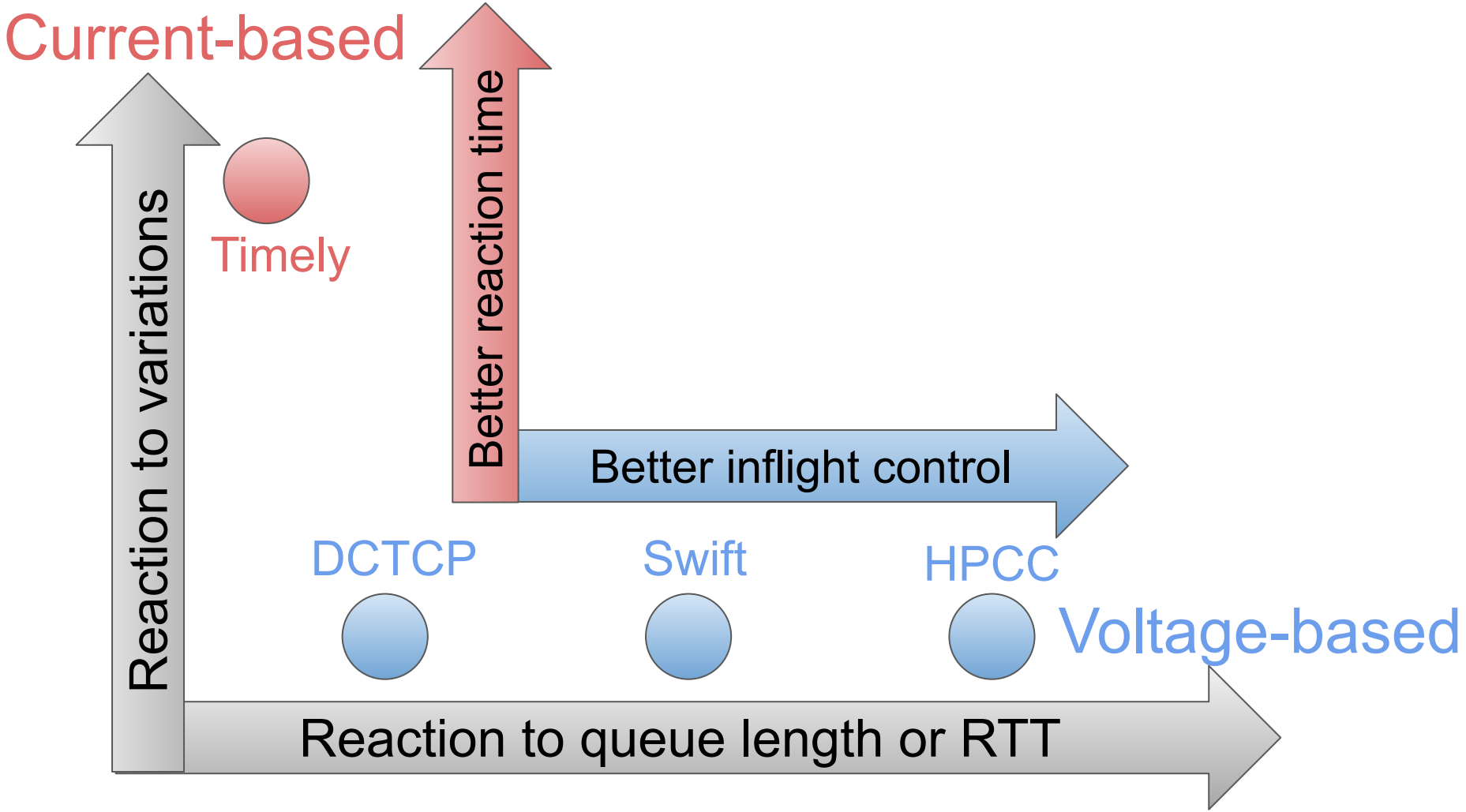
DCTCP, TIMELY, HPCC cannot distinguish all the three cases simultaneously

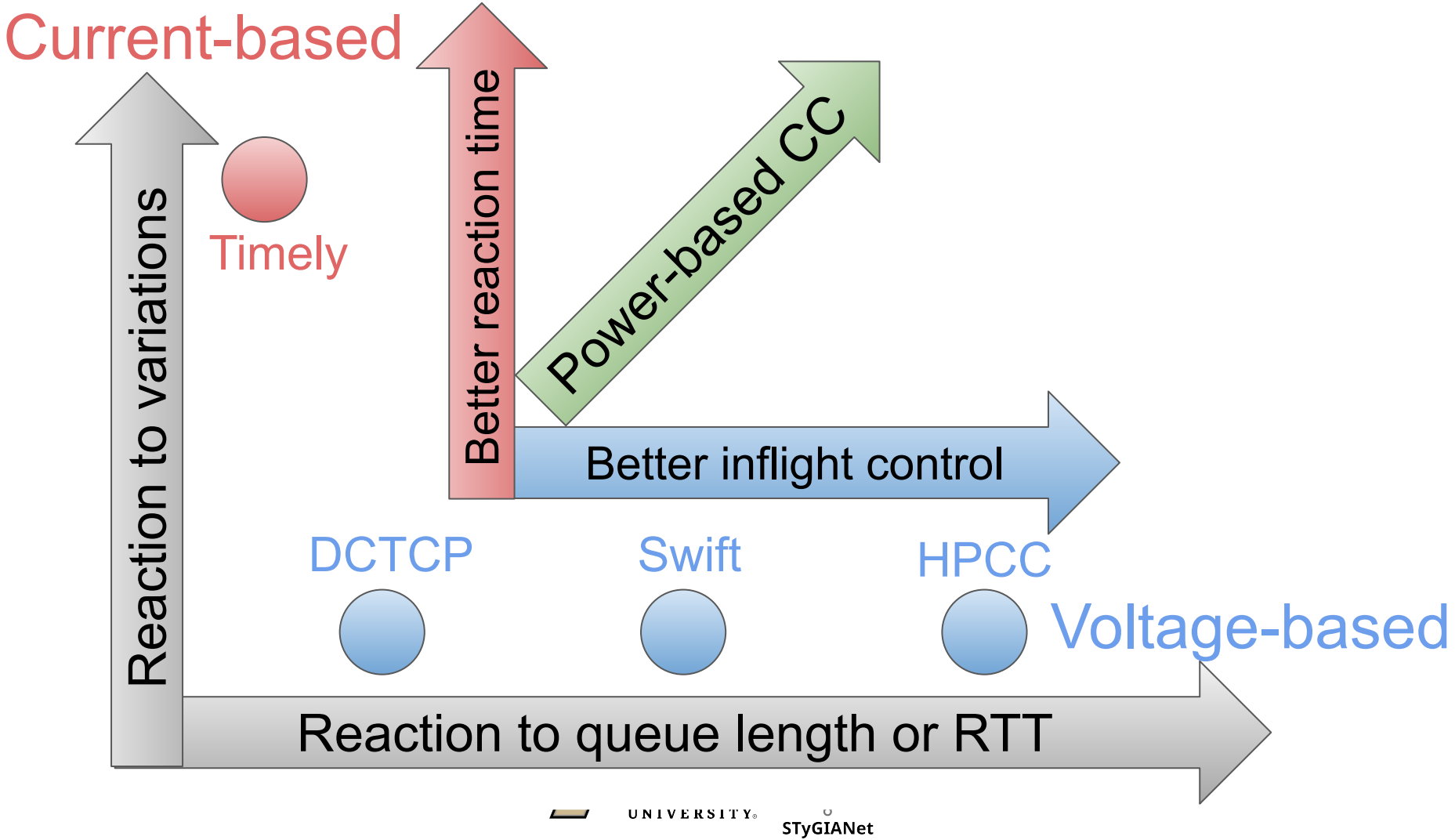


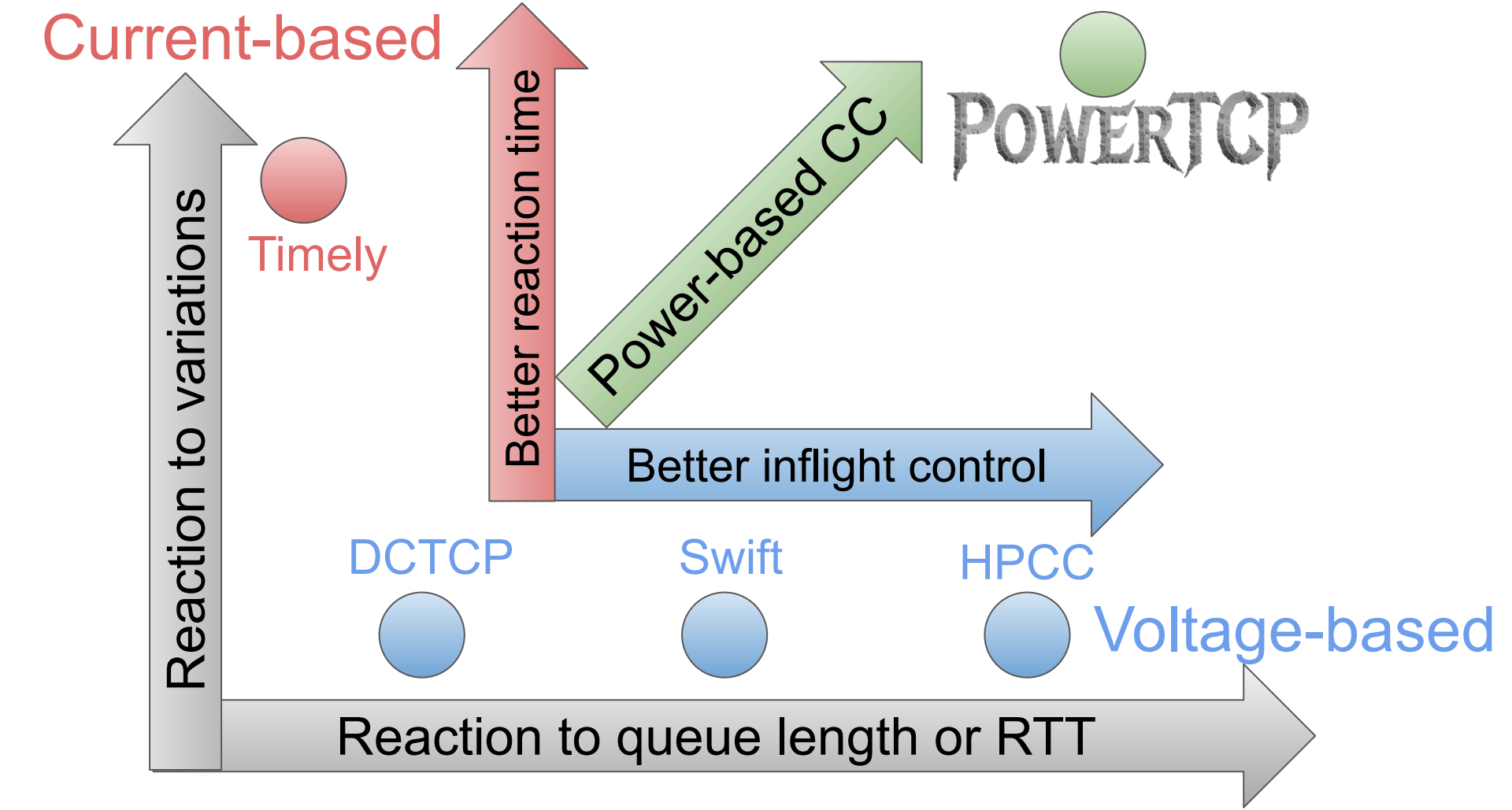
Summary of Prior Approaches

- Voltage-based (eg., HPCC)
  - Can in-principle achieve near-zero queue equilibrium
  - Slow reaction
- Current-based (eg., TIMELY)
  - Unstable with no equilibrium
  - Fast Reaction


Current-based
Timely
Reaction to variations
Better inflight control
DCTCP
Swift
HPCC
Voltage-based
Reaction to queue length or RTT


Current-based
Reaction to variations
Timely
Better reaction time
Better inflight control
DCTCP
Swift
HPCC
Voltage-based
Reaction to queue length or RTT


Current-based
Timely
Reaction to variations
Better reaction time
Power-based CC
Better inflight control
DCTCP
Swift
HPCC
Voltage-based
Reaction to queue length or RTT


Current-based
Timely
Reaction to variations
Better reaction time
Power-based CC
POWERTCP
Better inflight control
DCTCP
Swift
HPCC
Voltage-based
Reaction to queue length or RTT

The notion of power

- Power = Voltage x Current

$$\underbrace{\Gamma}_{\text{Power}} = \underbrace{(q(t) + b \times \tau)}_{\text{Voltage}} \times \underbrace{(\dot{q}(t) + \mu(t))}_{\text{Current}}$$

The notion of power

- A function of both queue length and variations
  - Detects increased queue lengths
  - Detects congestion onset and intensity
  - Detects rapid drop in queue lengths

PowerTCP

$$w_i(t+\delta t) = \gamma \cdot \left(w_i(t) \cdot \frac{e}{f(t)} + \beta \right) + (1-\gamma) \cdot w_i(t)$$

New window size

PowerTCP

$$w_i(t+\delta t) = \gamma \cdot \left(w_i(t) \cdot \frac{e}{f(t)} + \beta \right) + (1-\gamma) \cdot w_i(t)$$

Old window size

PowerTCP

$$w_i(t+\delta t) = \gamma \cdot \left(w_i(t) \cdot \frac{e}{f(t)} + \beta \right) + (1-\gamma) \cdot w_i(t)$$

MIMD based on Power

(Multiplicative increase - multiplicative decrease)

PURDUE UNIVERSITY

STyGIANet

PowerTCP

$$w_i(t+\delta t) = \gamma \cdot \left(w_i(t) \cdot \frac{e}{f(t)} + \beta \right) + (1-\gamma) \cdot w_i(t)$$

- $e = b^2 \cdot \tau$
- $f(t)$ is power

MIMD based on Power

(Multiplicative increase - multiplicative decrease)

PowerTCP

$$w_i(t+\delta t) = \gamma \cdot \left(w_i(t) \cdot \frac{e}{f(t)} + \beta \right) + (1-\gamma) \cdot w_i(t)$$

Additive increase
(constant)

PowerTCP

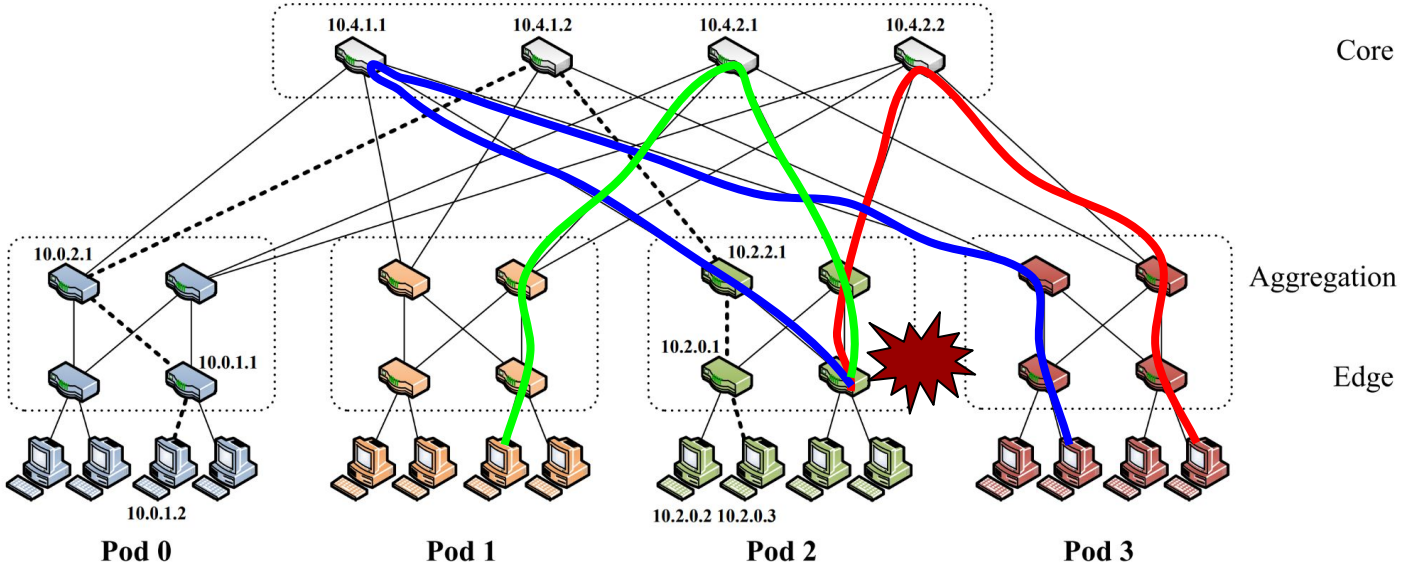
$$w_i(t+\delta t) = \gamma \cdot \left(w_i(t) \cdot \frac{e}{f(t)} + \beta \right) + (1-\gamma) \cdot w_i(t)$$

Exponential Weighted Moving Average (EWMA)

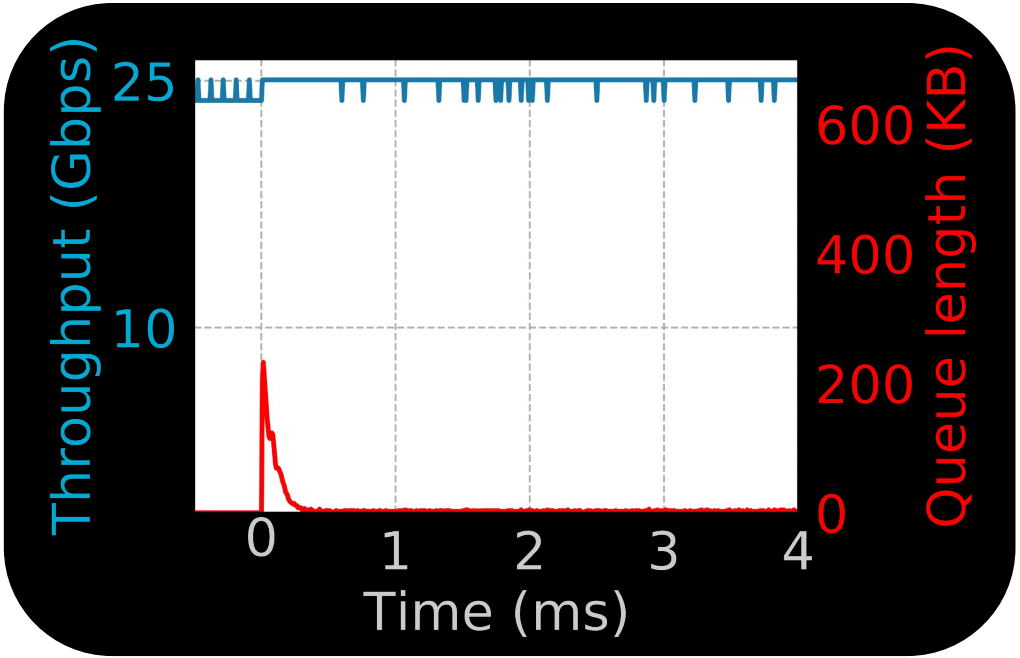
PowerTCP

- Power-based congestion control
- Quickly reacts to congestion without losing throughput
- Rapidly converges within 1 RTT
- Fair and asymptotically stable

Incast


10.4.1.1
10.4.1.2
10.4.2.1
10.4.2.2
Core
10.0.2.1
10.0.1.1
10.0.1.2
10.2.2.1
10.2.0.1
10.2.0.2 10.2.0.3
Aggregation
Edge
Pod 0
Pod 1
Pod 2
Pod 3

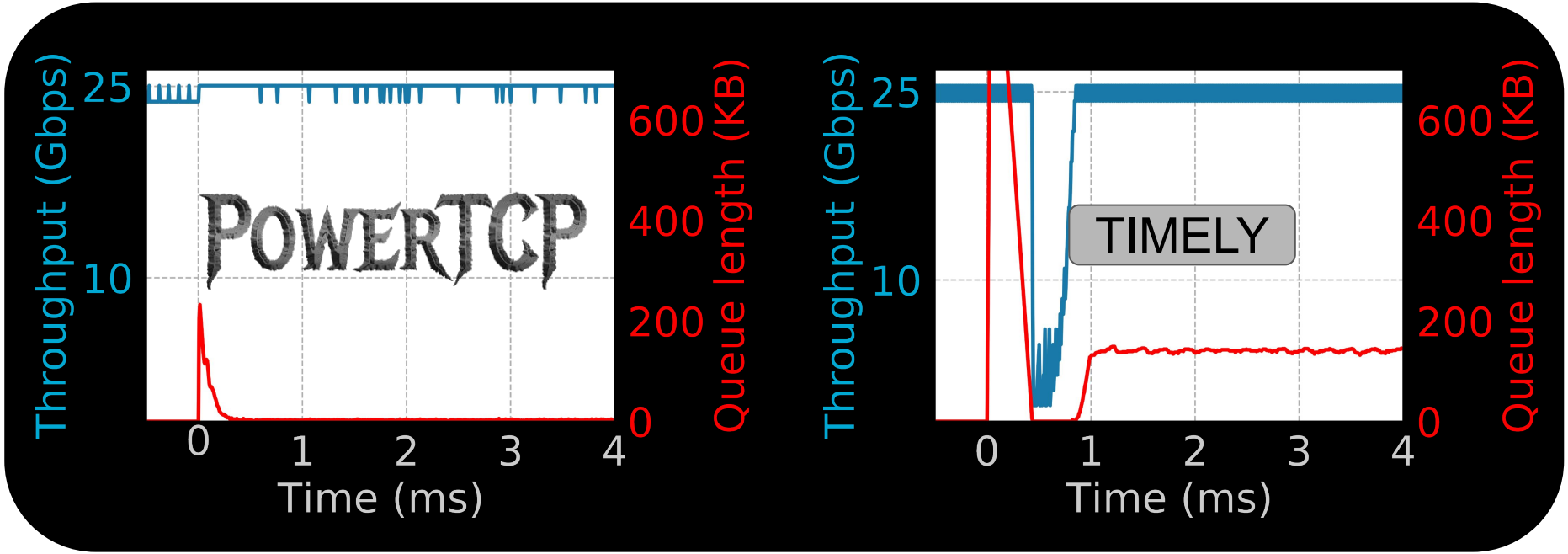
Incast



PURDUE UNIVERSITY

STyGIANet

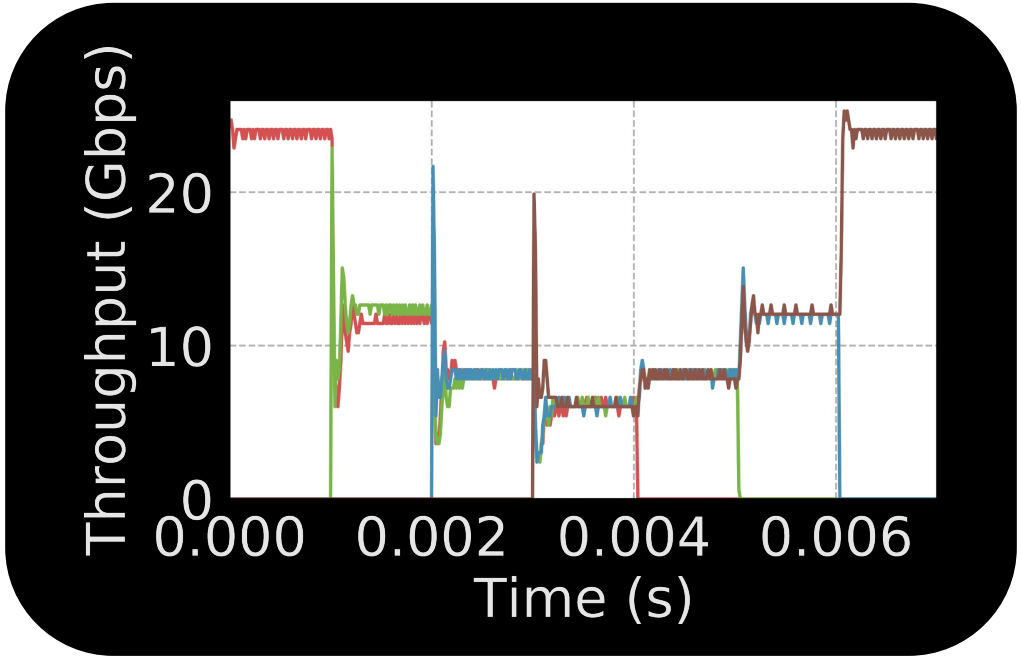
Incast



PURDUE UNIVERSITY

STyGIANet

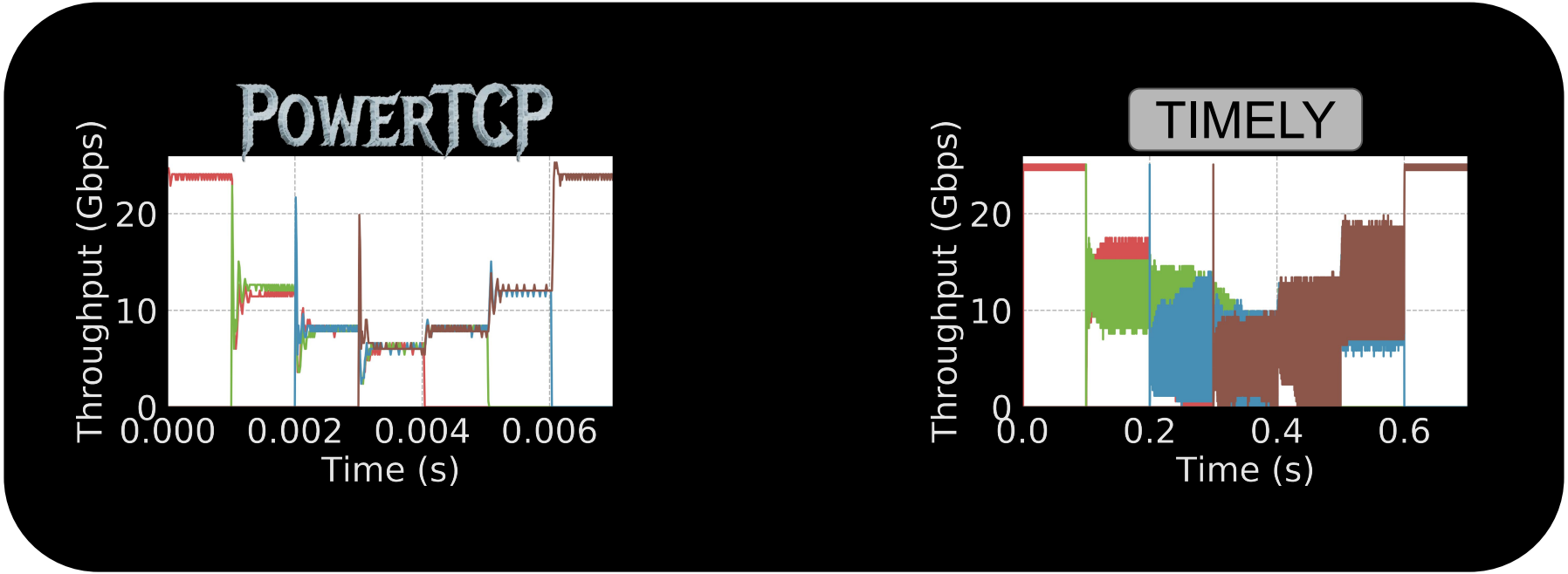
Fairness



PURDUE UNIVERSITY

STyGIANet

Fairness

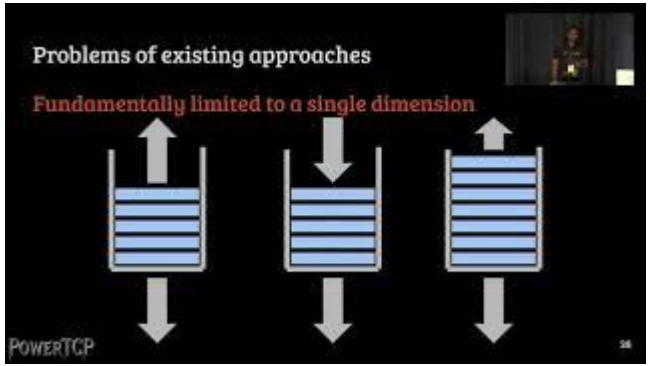


PURDUE UNIVERSITY

STyGIANet

PowerTCP

Further analysis and large-scale evaluations:
https://www.usenix.org/system/files/nsdi22-paper-addanki_3.pdf



PURDUE UNIVERSITY

STyGIANet

References

[1] Alizadeh M, Greenberg A, Maltz DA, Padhye J, Patel P, Prabhakar B, Sengupta S, Sridharan M. Data center tcp (dctcp). InProceedings of the ACM SIGCOMM 2010 Conference 2010 Aug 30 (pp. 63-74).

[2] Mittal R, Lam VT, Dukkipati N, Blem E, Wassel H, Ghobadi M, Vahdat A, Wang Y, Wetherall D, Zats D. TIMELY: RTT-based congestion control for the datacenter. ACM SIGCOMM Computer Communication Review. 2015 Aug 17;45(4):537-50.

[3] Li Y, Miao R, Liu HH, Zhuang Y, Feng F, Tang L, Cao Z, Zhang M, Kelly F, Alizadeh M, Yu M. HPCC: High precision congestion control. InProceedings of the ACM Special Interest Group on Data Communication 2019 Aug 19 (pp. 44-58).

[4] Addanki V, Michel O, Schmid S. {PowerTCP}: Pushing the performance limits of datacenter networks. In19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22) 2022 (pp. 51-70).