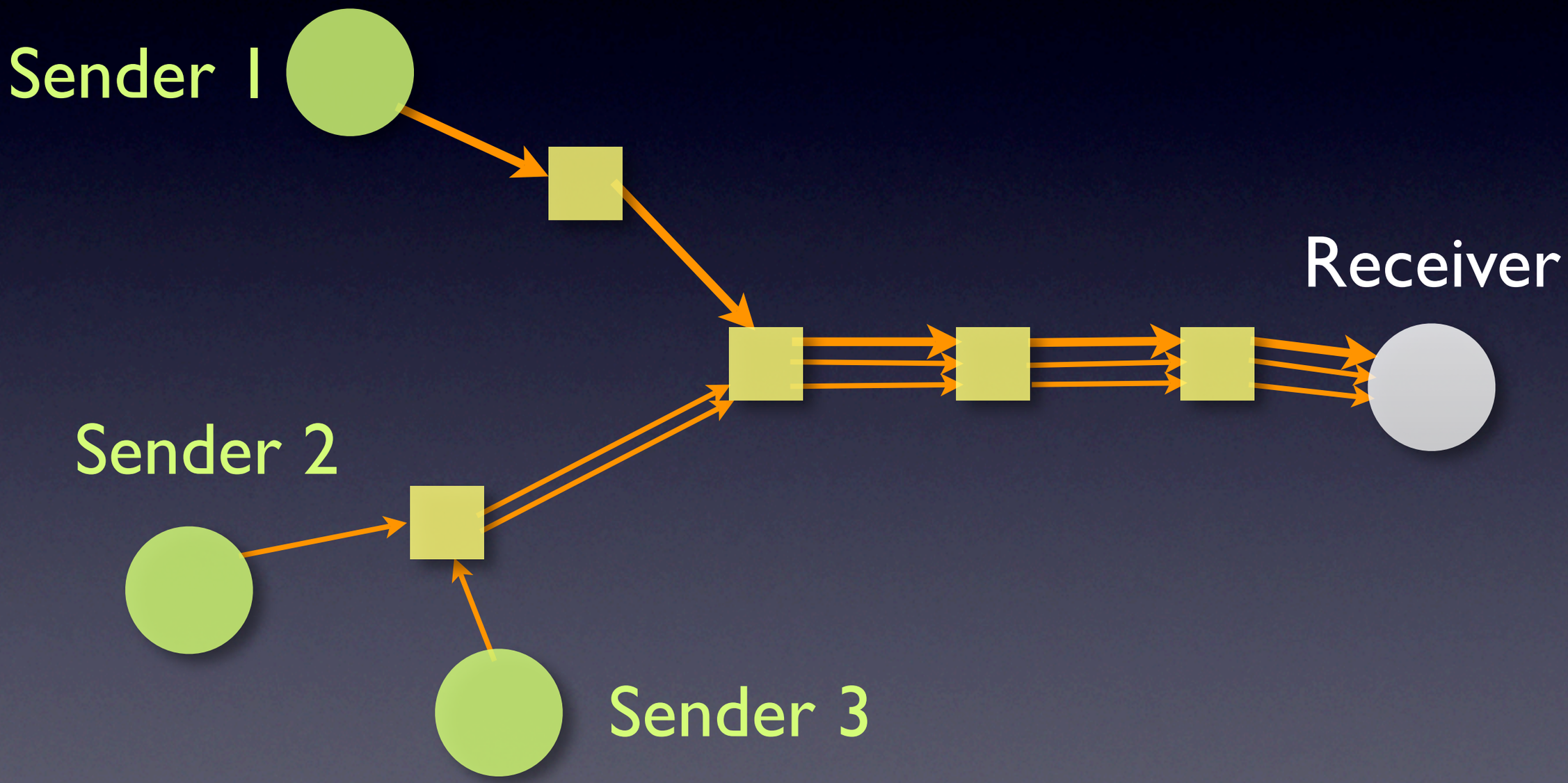
# Congestion Control in Distributed Media Streaming

Lin Ma and Wei Tsang Ooi
National University of Singapore

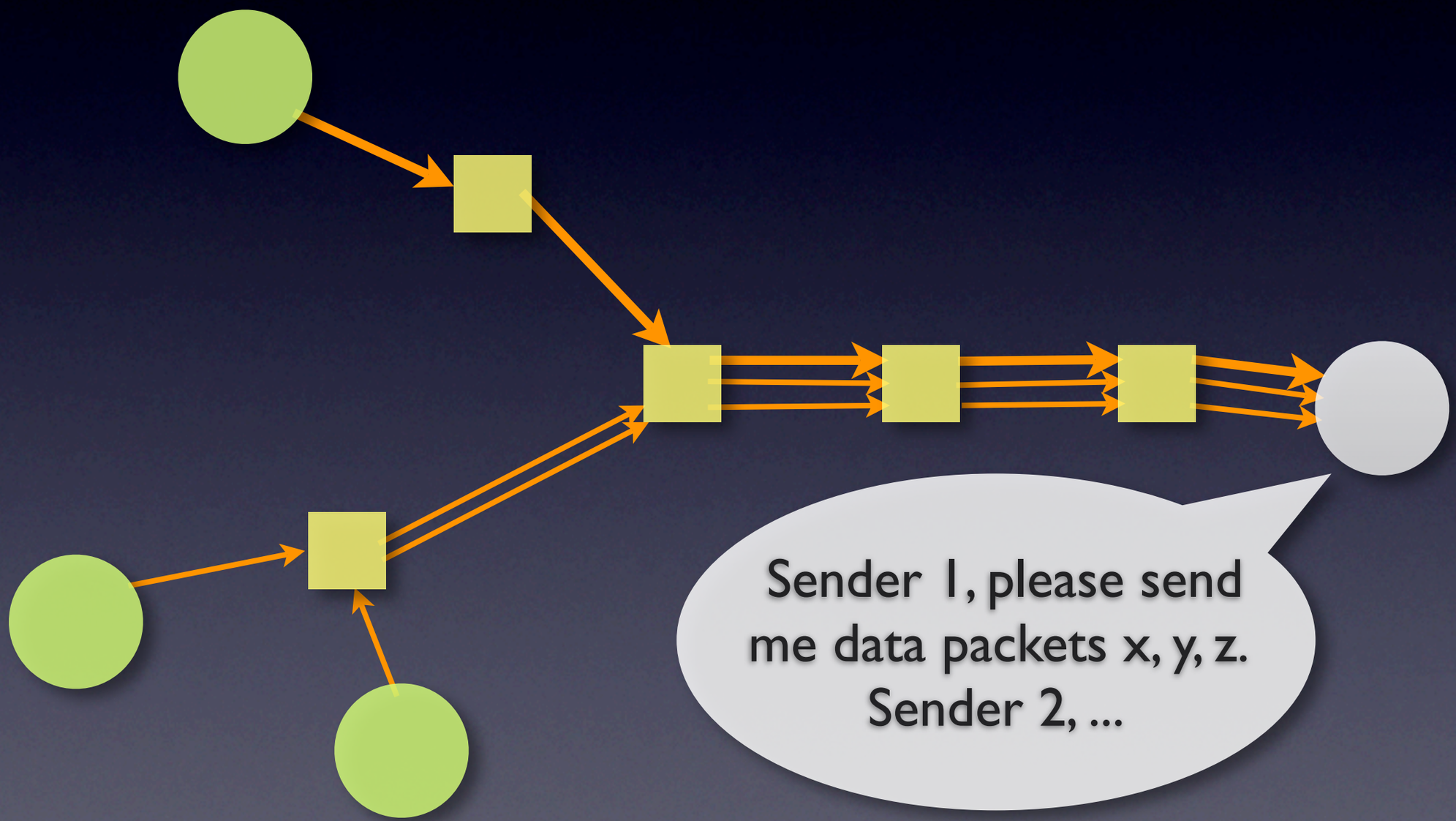
# What is “Distributed Media Streaming”?

(aka multi-source streaming)

Multiple senders collaboratively stream media content to a receiver.



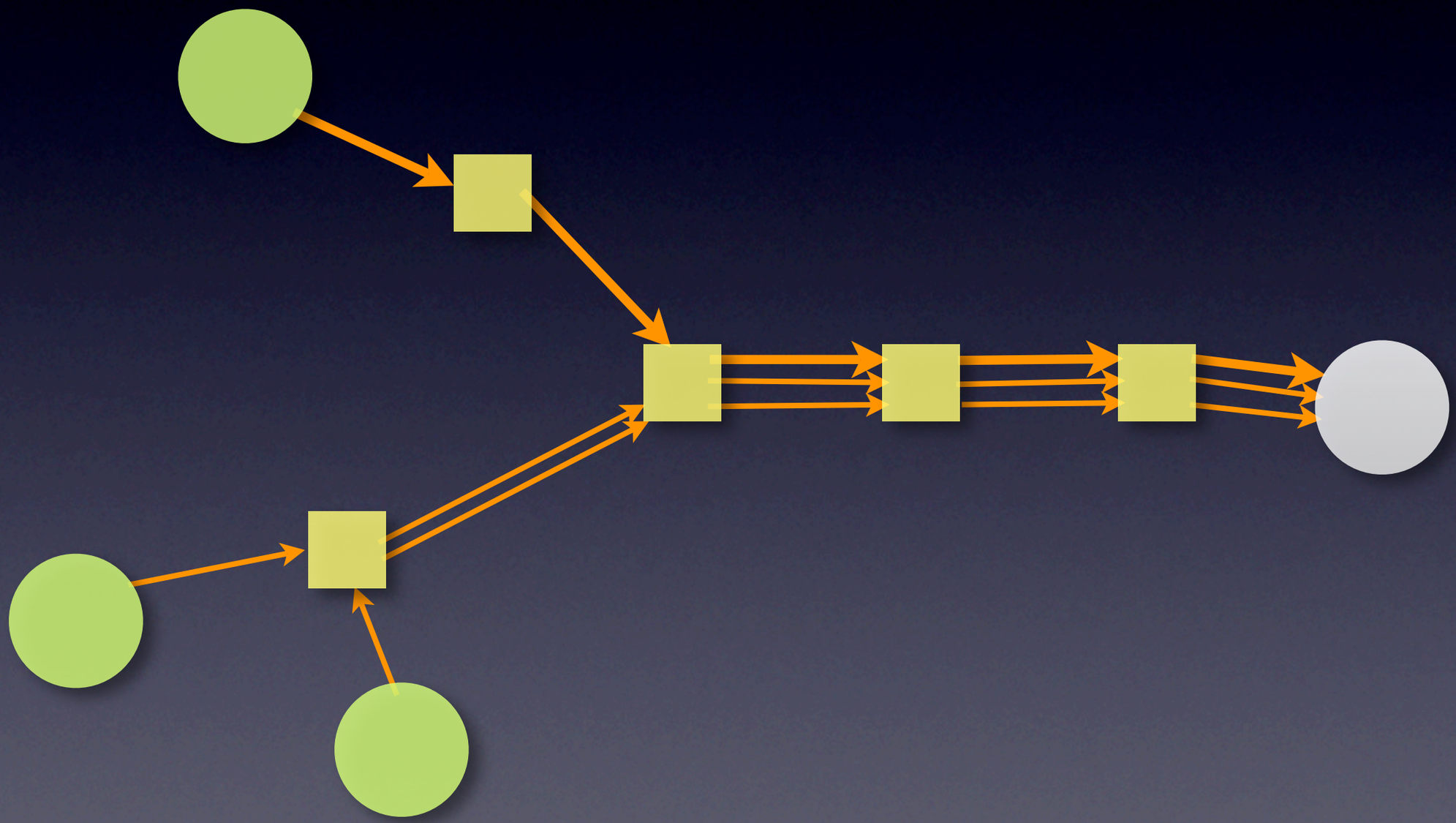
Receiver coordinates between the senders using a pull-based protocol to request different segments from different senders.



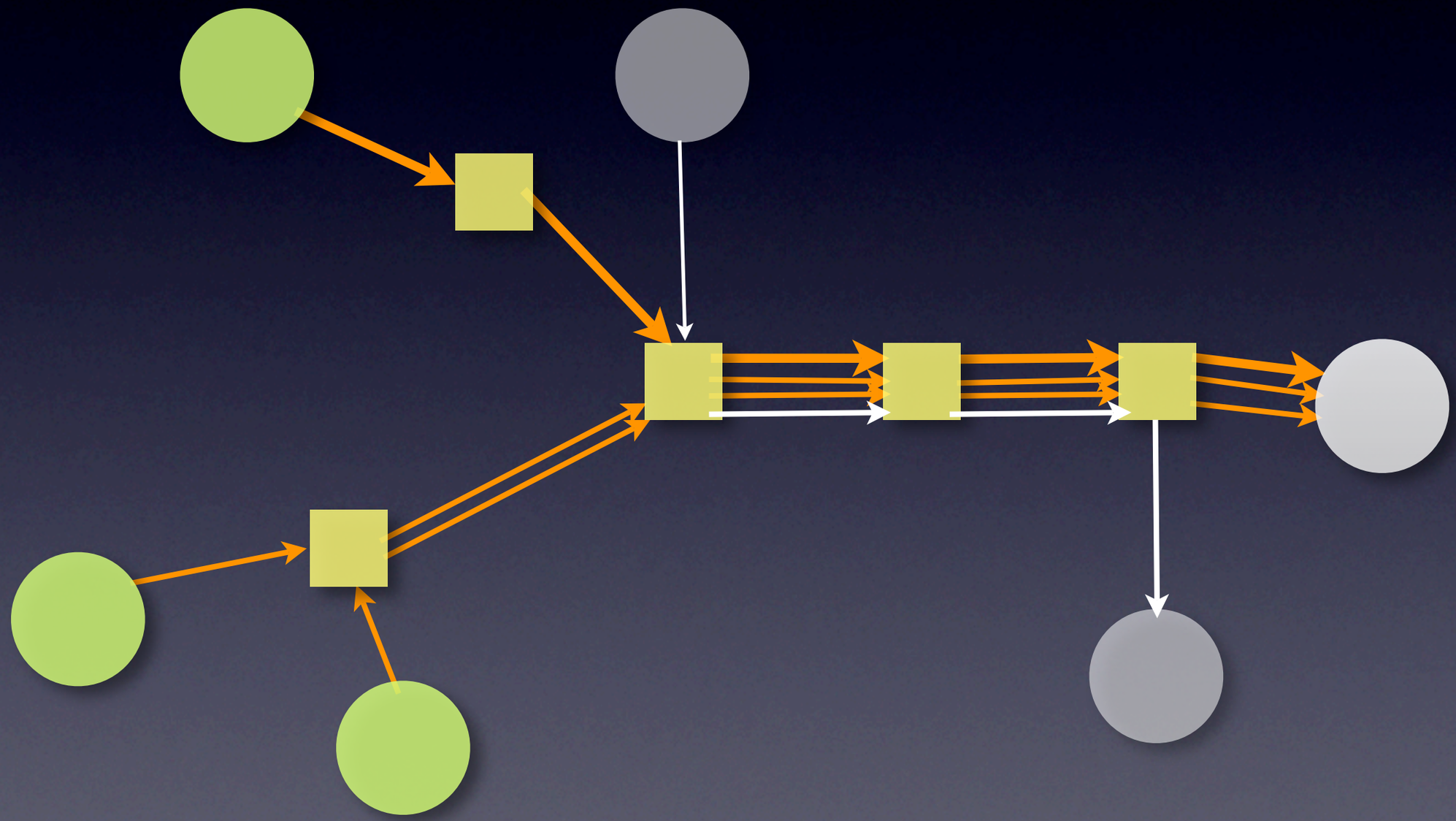
- Exploits path diversity and server diversity to increase resilient to congestion and sender failure
- Using media coding scheme such as MDC, the receiver can still playback continuously (at a lower quality) if a sender fails.

# Congestion Control in Distributed Media Streaming

# Per-flow Congestion Control ?



# Using multiple flows is unfair to other single flow applications.



- Similar problem observed in parallel TCP flows:

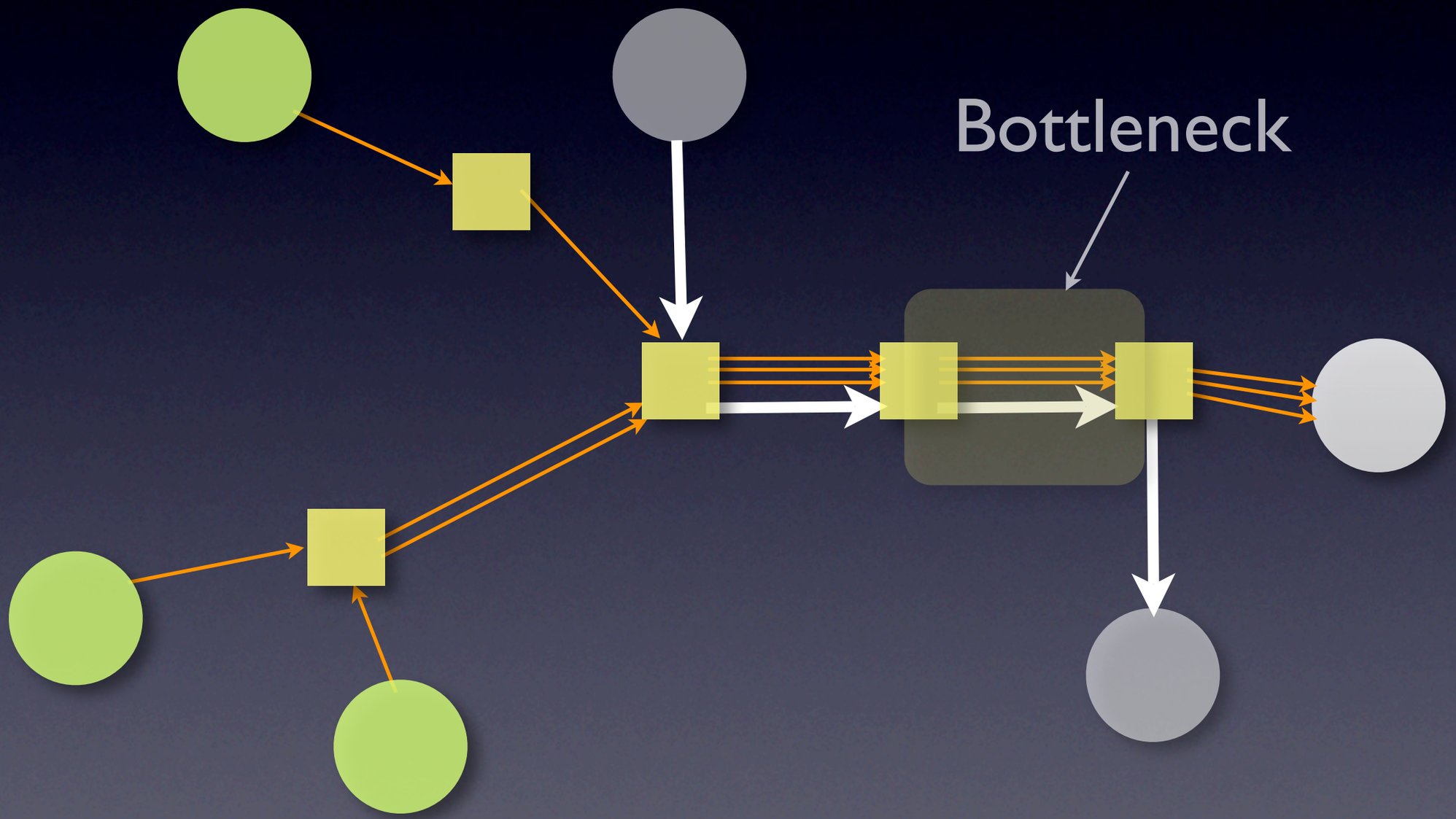
1. TCP-P (Soonghyun Cho et al)
2. TCP-ROME (Roger Karrer et al)
3. Multi-priority TCP (Ronald Tse et al)

# Task-level TCP Friendliness

- The total bandwidth of flows, belonging to the same task, on a link should be no larger than other TCP flows on the same link (experiencing similar network conditions).

$$\sum_{f_i \in L} B_i \leq B_{TCP}$$

# Task-Level TCP Friendliness

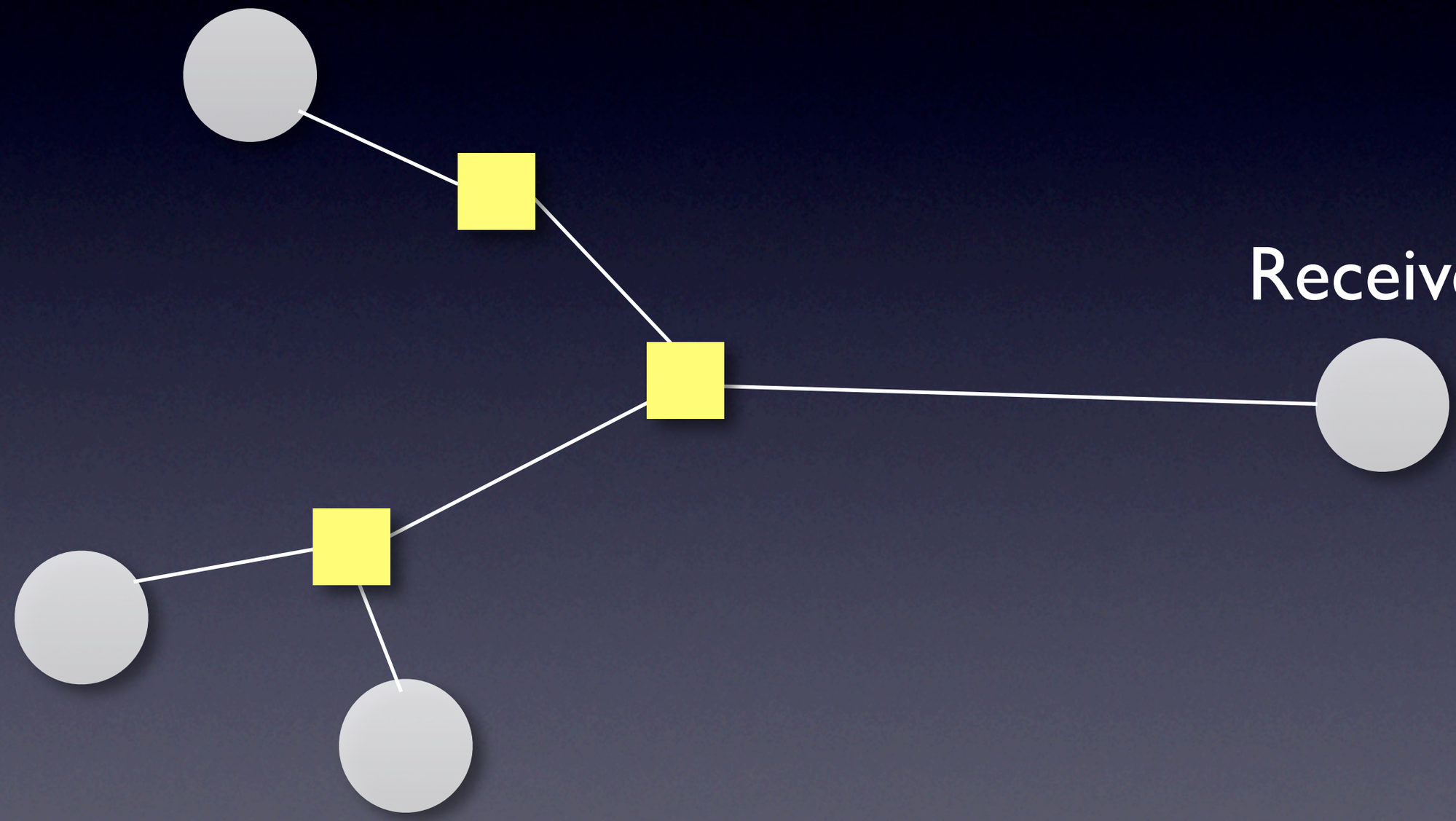


# The Challenges

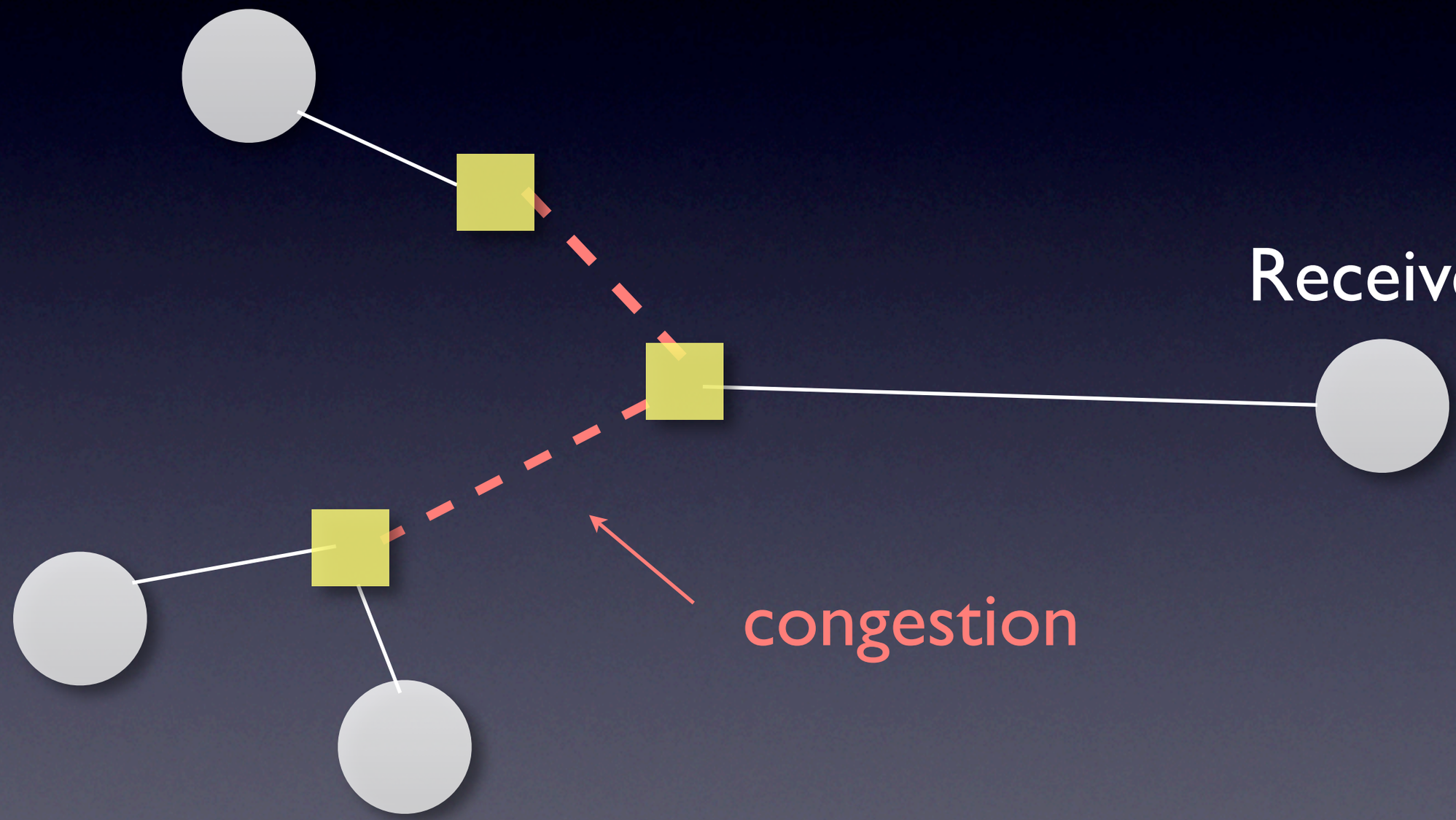
- Different media flows may experience different congested links
- How to determine the “fair” throughput of a media flow?

# DMSCC : Congestion Control Algorithm

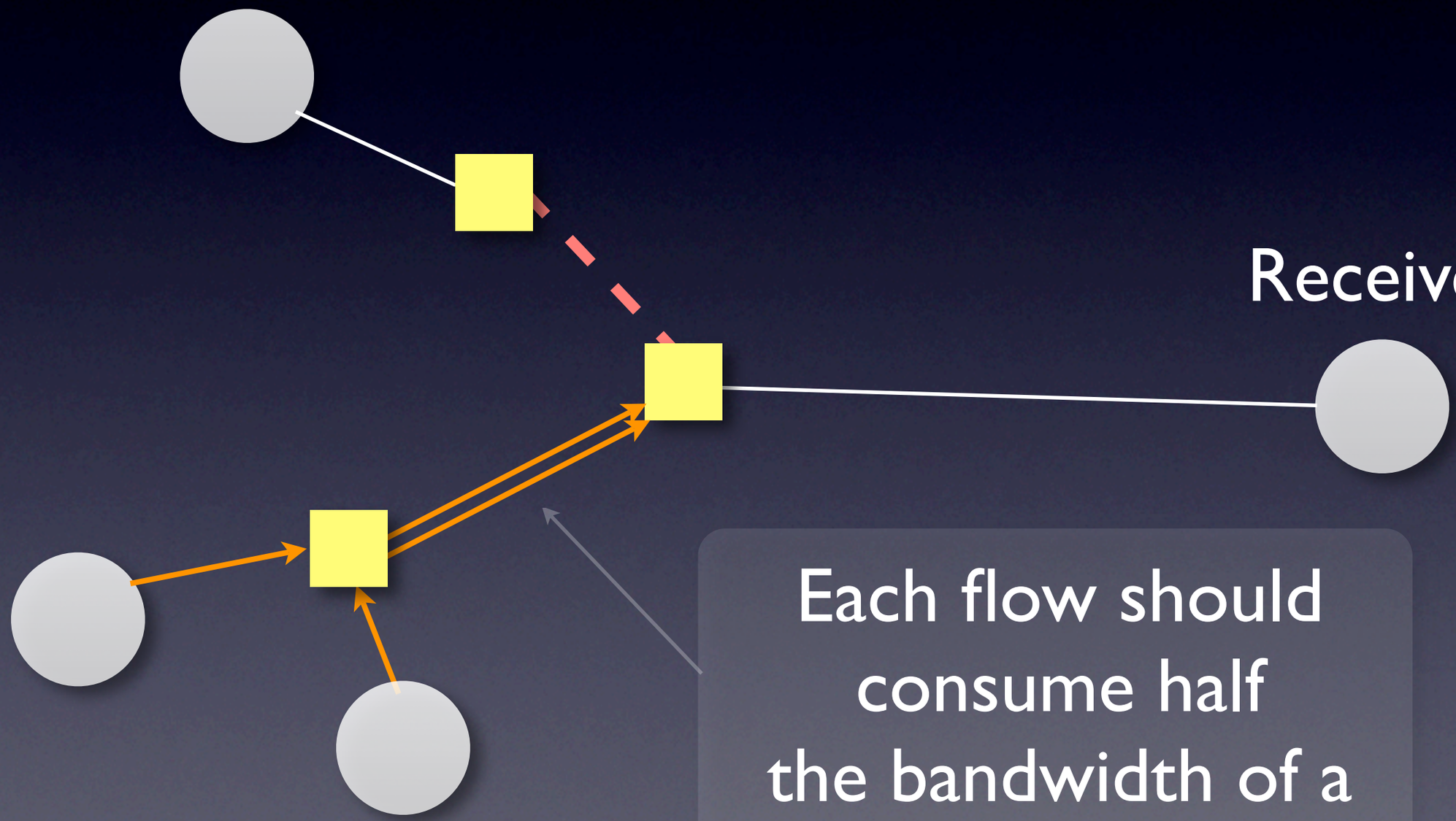
Suppose (i) we know the topology, and
(ii) the topology is a tree.



# 1. Find out where the congested link(s) are.



# 2. Control the rate of the flows on congested links.




Each flow should consume half the bandwidth of a TCP flow

# Identifying Congested Links

Given end-to-end measurements on a set of flows, determine which flows share bottleneck link(s).

# Controlling Throughput

Given a set of flows on a bottleneck link, how to control the throughput of the flows so that they satisfy

$$\sum_{f_i \in L} B_i \leq B_{TCP}$$

# Identifying Congested Links

Given end-to-end measurements on a set of flows, determine which flows share bottleneck link(s).

# Controlling Throughput

Given a set of flows on a bottleneck link, how to control the throughput of the flows so that they satisfy

$$\sum_{f_i \in L} B_i \leq B_{TCP}$$

# Identifying Congested Links

- Non-trivial problem for one shared bottleneck
  - Rubenstein (TON'02), Kim (SIGCOMM '04)
- Even harder for multiple bottlenecks.
- We use Rubenstein's method as a building block.

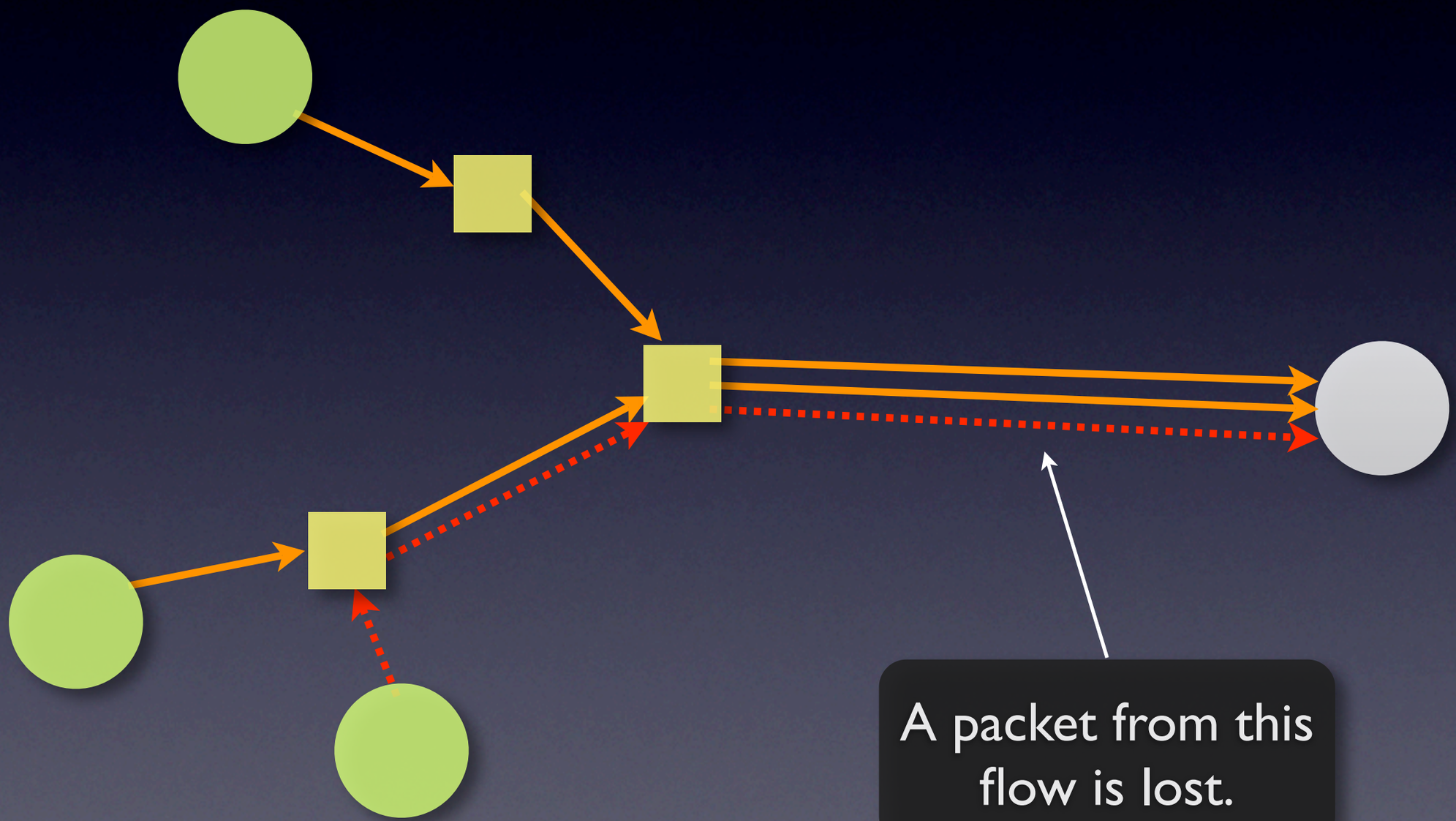
# Rubenstein's Method

- **SHARE( *f*, *g* ):** Does two flows *f* and *g* share the same bottleneck?
  - Observe the packet delay of flow *f* and *g*.
  - Yes, if cross-correlation of *f* and *g* is larger than auto-correlation of *f*.

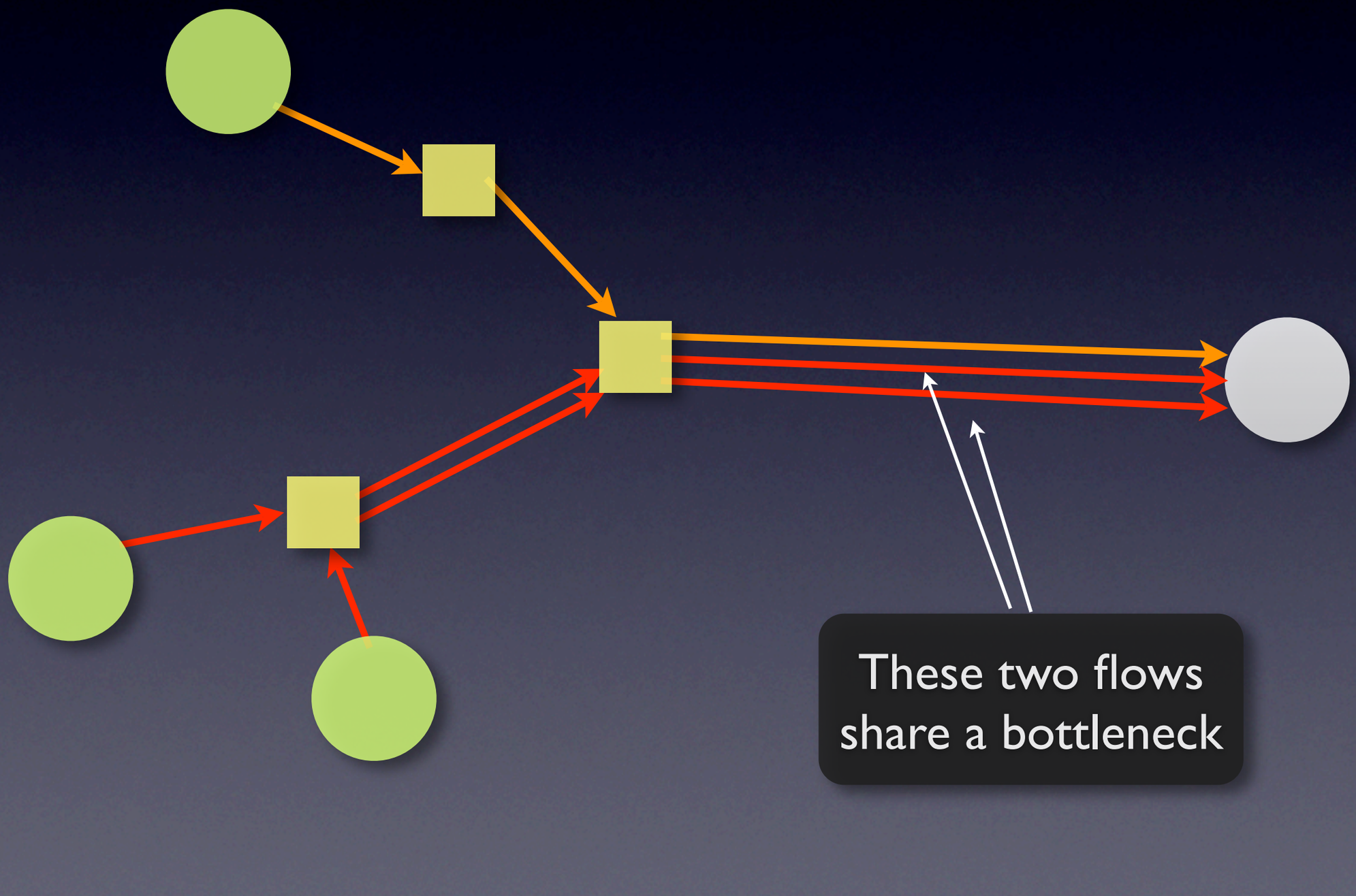
# Congestion Location (one bottleneck)

- Suppose a packet from flow *f* is lost
- Find all other flows *g* such that
  **SHARE**( f, g ) = true
- Find all common links of these flows
- Return the link furthest away from receiver

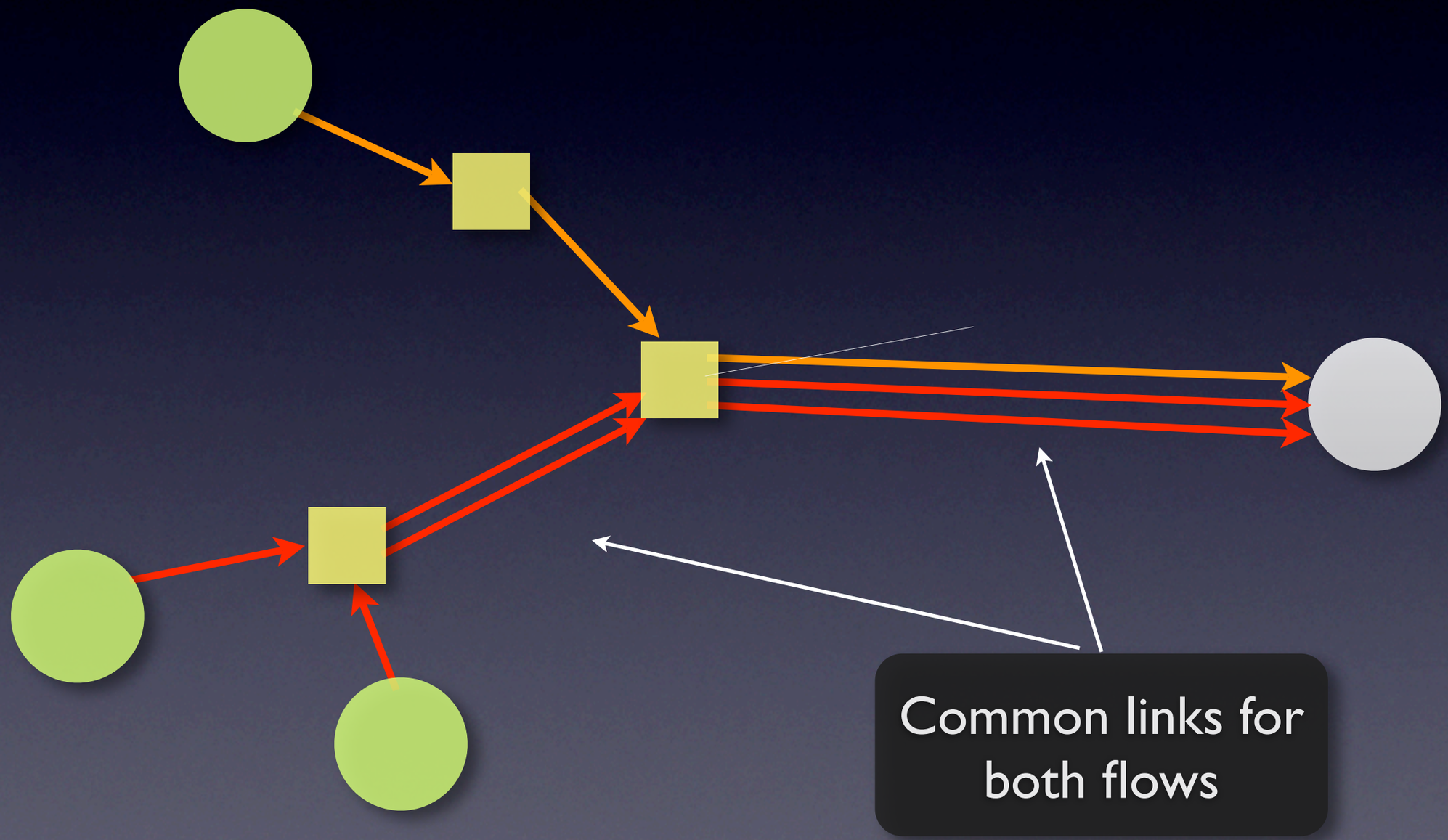
# Congestion Location



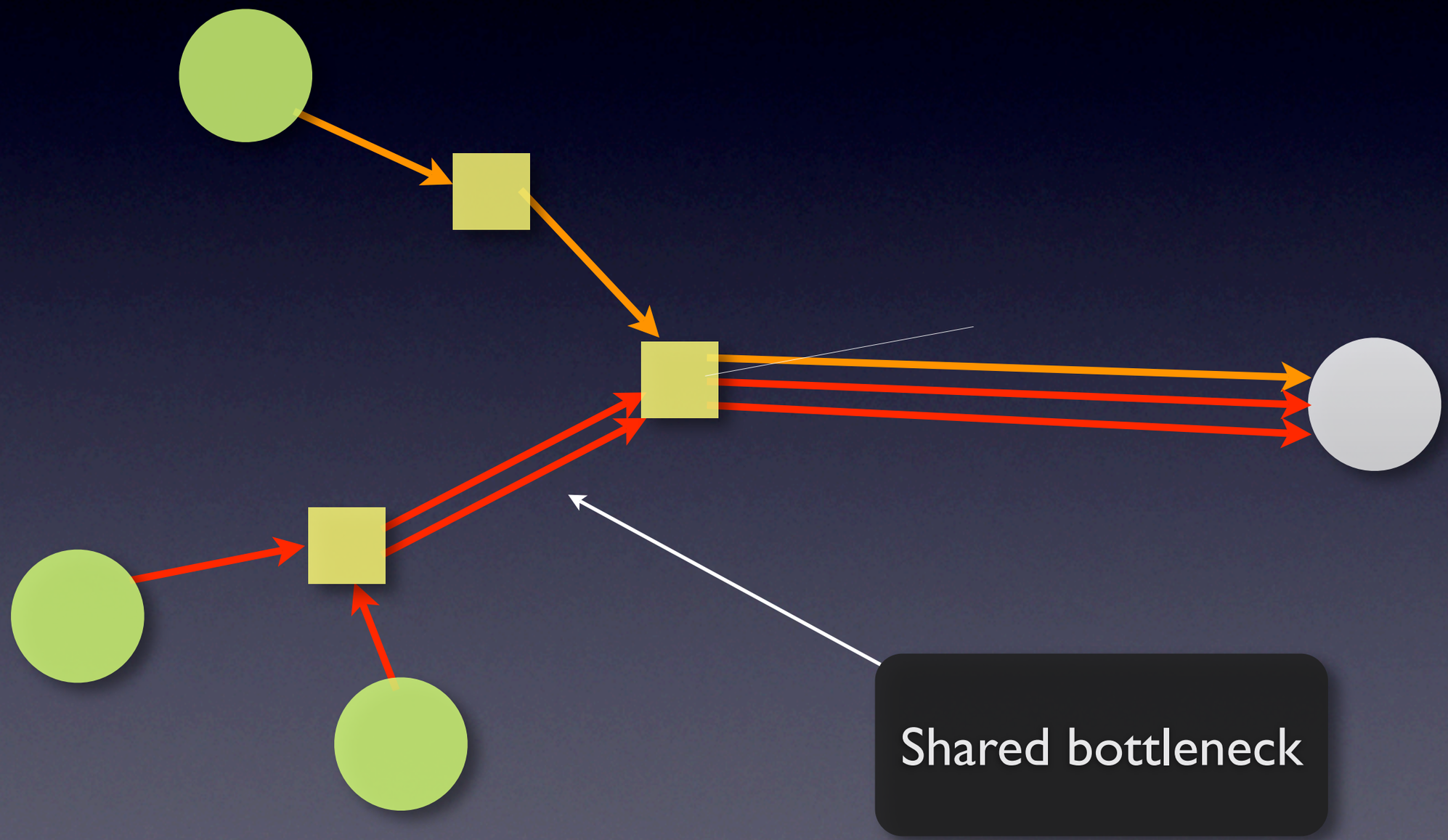
# Congestion Location



# Congestion Location



# Congestion Location



# Congestion Location (multiple bottlenecks)

- Keep a history of *h* previous bottleneck detections.
- All links in this set are presumed to be congested.

# Identifying Congested Links

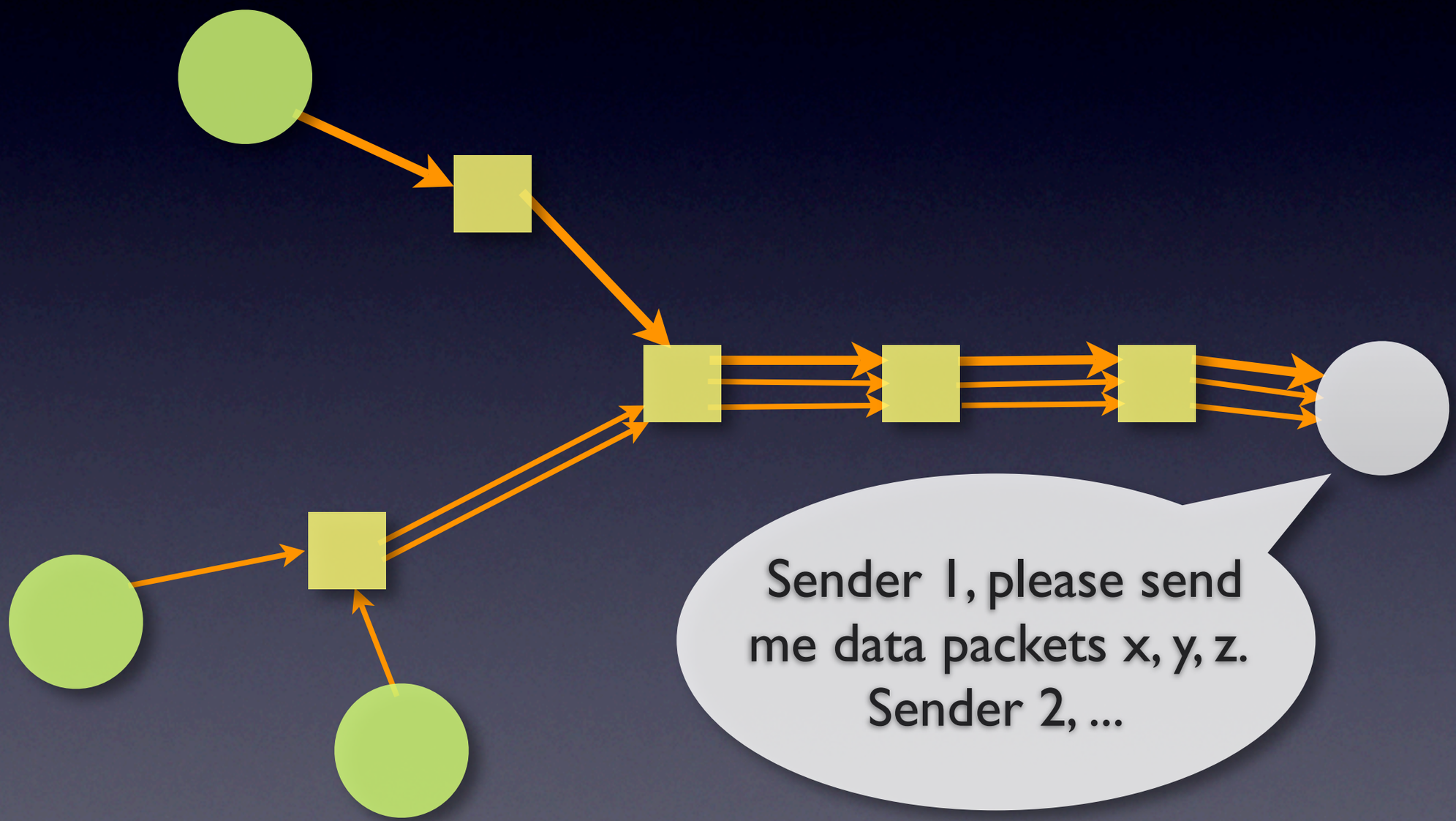
Given end-to-end measurements on a set of flows, determine which flows share bottleneck link(s).

# Controlling Throughput

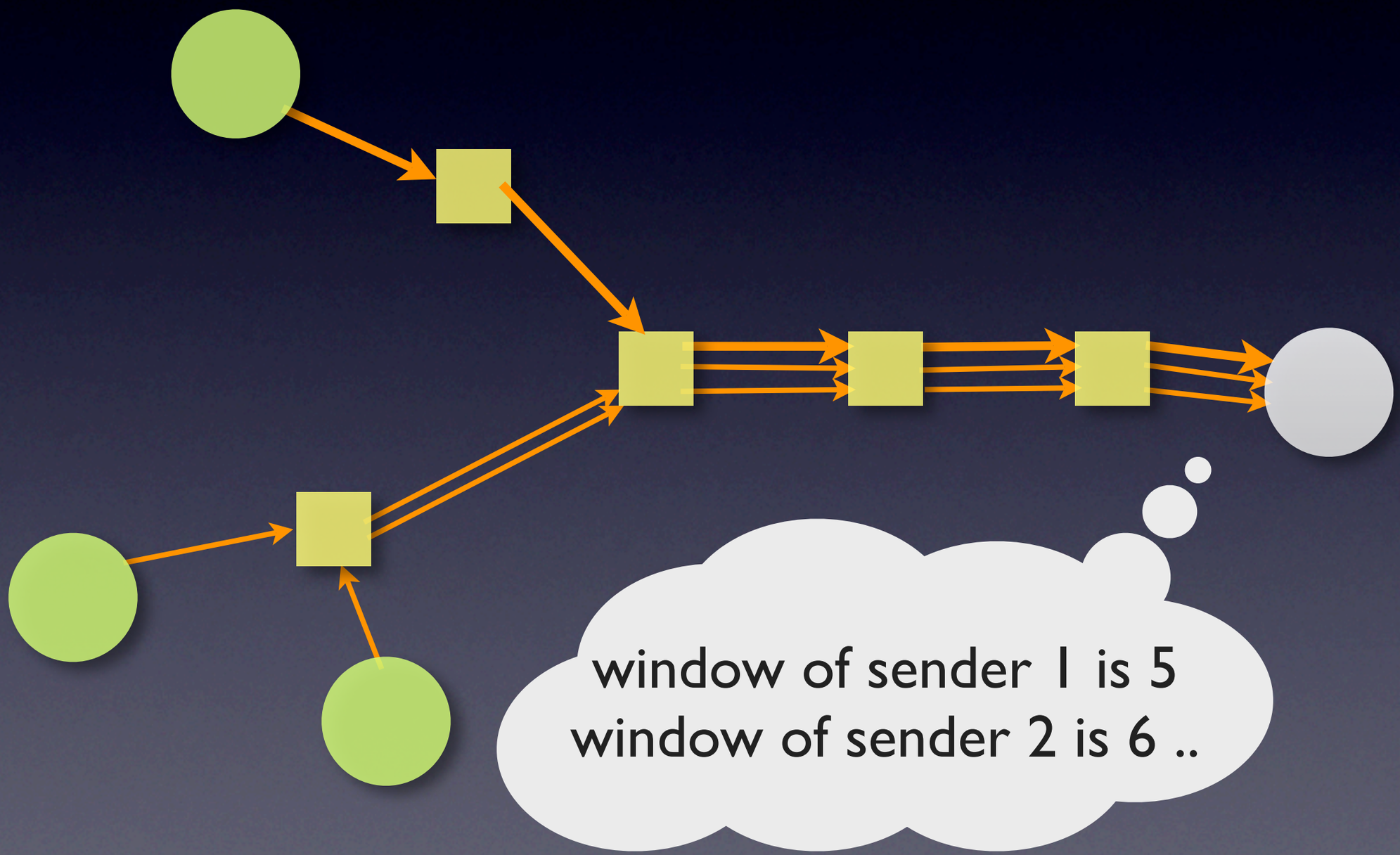
Given a set of flows on a bottleneck link, how to control the throughput of the flows so that they satisfy

$$\sum_{f_i \in L} B_i \leq B_{TCP}$$

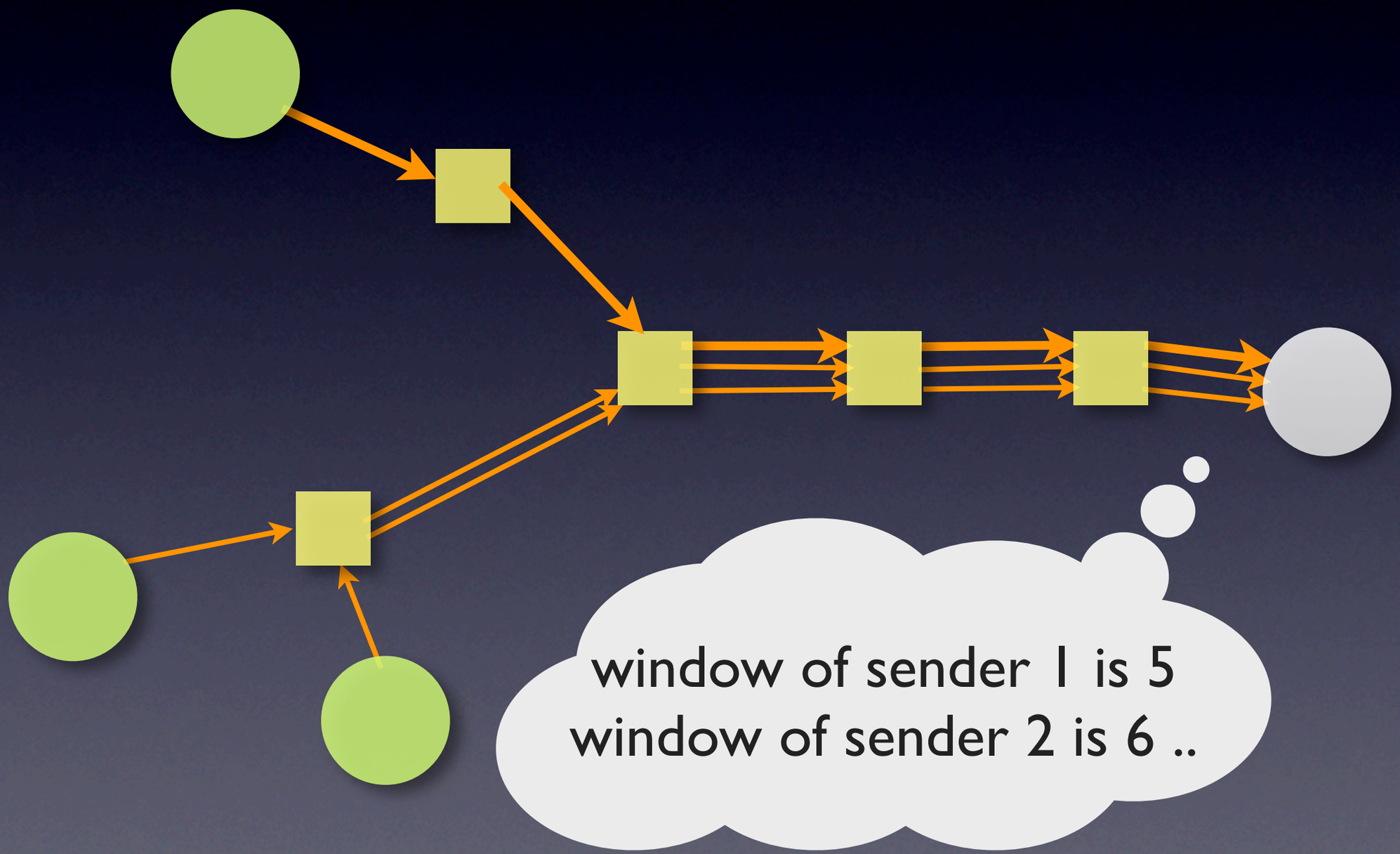
# Recall that we are running a pull-based protocol



To control the throughput, the receiver maintains a “congestion window” for each sender and never pulls more than the window allows.

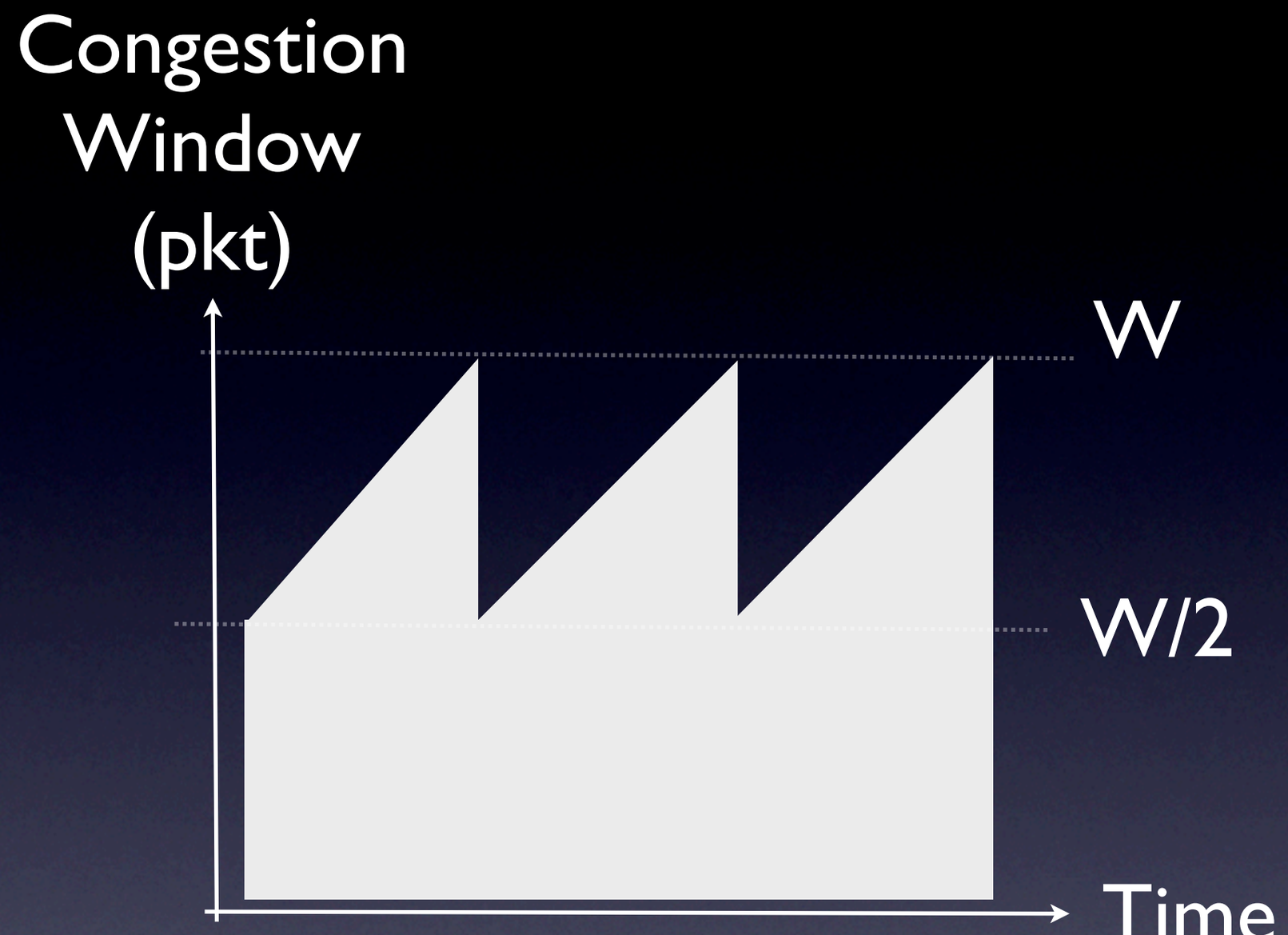


The window is adjusted according to AIMD when packet transmission is successful or lost.

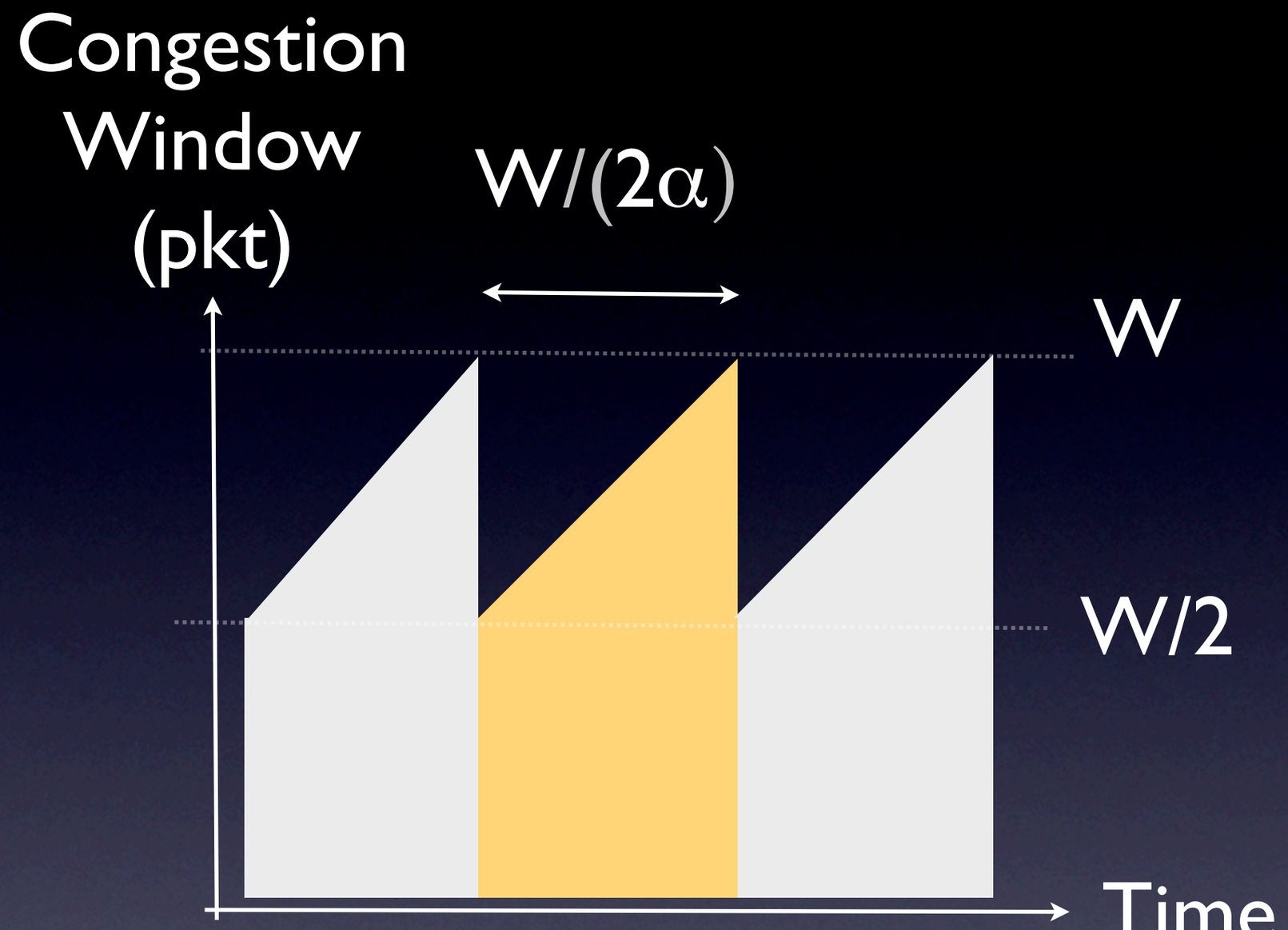


# How to adjust window?

- If we follows TCP's algorithm, then we will achieve similar throughput to a single TCP flow.
- To achieve k (k < 1) times the throughput of a TCP flows, we need to be less aggressive in increasing our window.




The window increases by $\alpha$ for every RTT;
Packet loss occur every 1/p packet .




The window increases by $\alpha$ for every RTT;
Packet loss occur every 1/p packet .

Considering the area under the curve, we get

$$\frac{3W^2}{8\alpha} = \frac{1}{p}$$

$$W = \sqrt{\alpha}\sqrt{\frac{8}{3p}}$$

To get k times the throughput of a TCP flow,
the increasing factor α should be $k^2$

# Identifying Congested Links

Given end-to-end measurements on a set of flows, determine which flows share bottleneck link(s).

# Controlling Throughput

Given a set of flows on a bottleneck link, how to control the throughput of the flows so that they satisfy

$$\sum_{f_i \in L} B_i \leq B_{TCP}$$

# DMSCC : Congestion Control Algorithm

# DMSCC Algorithm

**On packet loss**

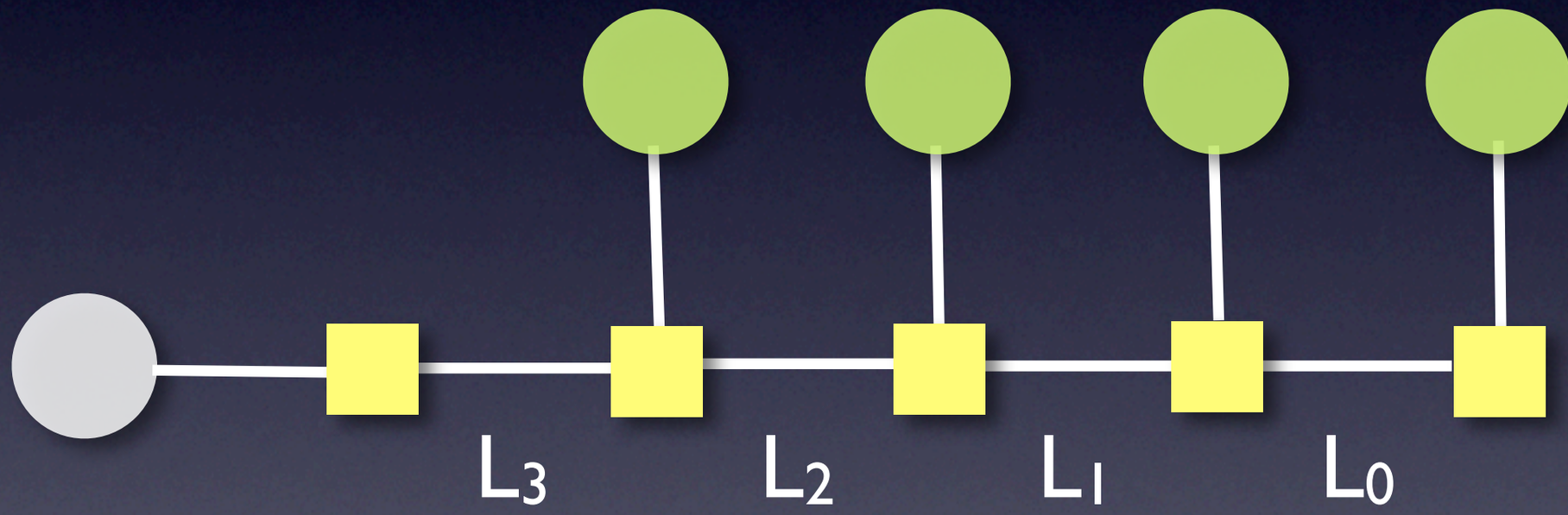
- Find the set of bottleneck links
- For each bottleneck links *l*
  let *n* be number of flows on *l*
  set $\alpha$ of each flow on *l* to $\min(\alpha, 1/n^2)$

**If no packet loss for some time *t***

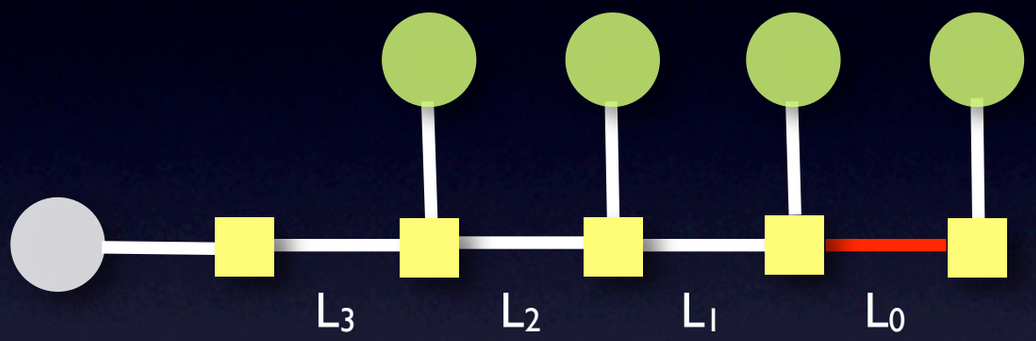
- Reset all $\alpha$ to 1

# Simulation and Results

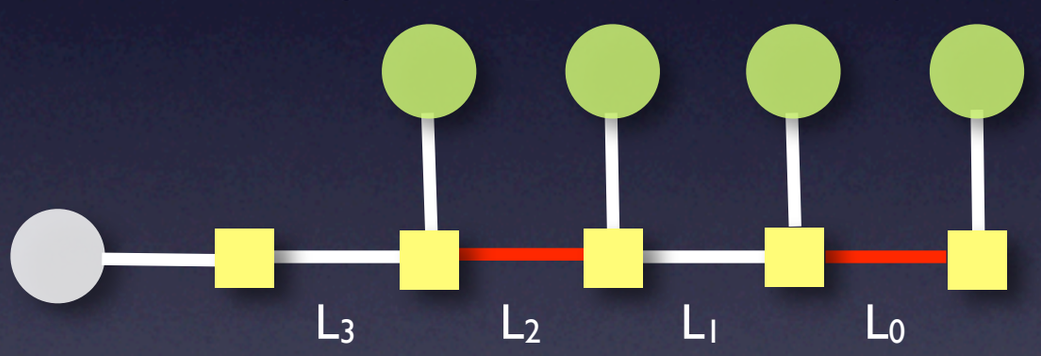
# Network Topology



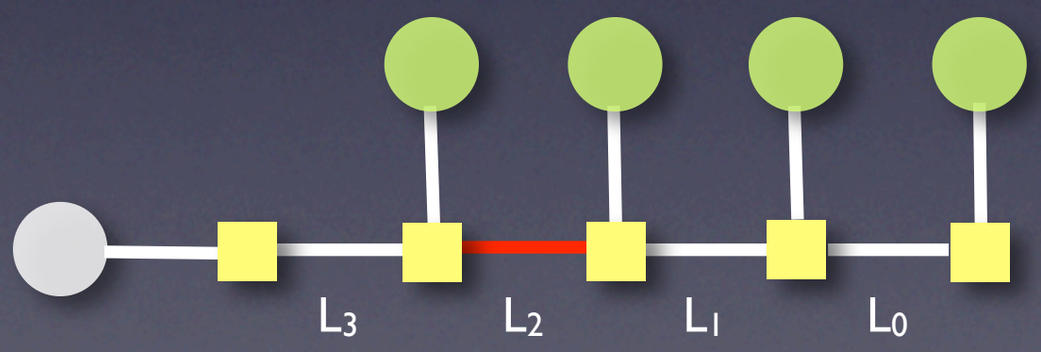
# Background Traffic




Time 0 to 50

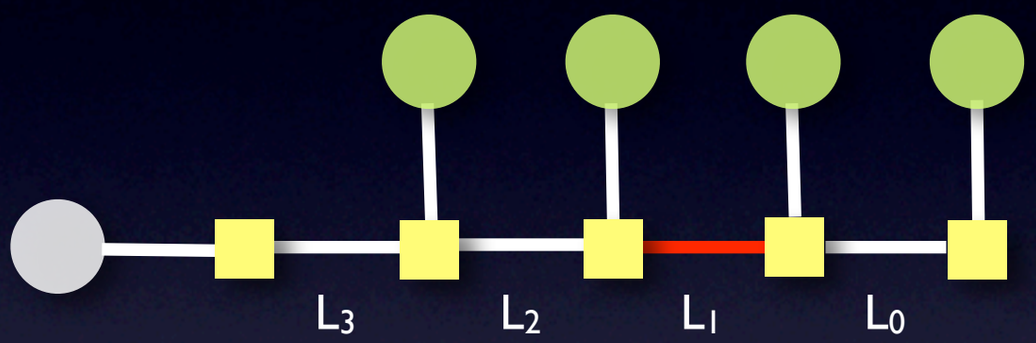



Time 50 to 100

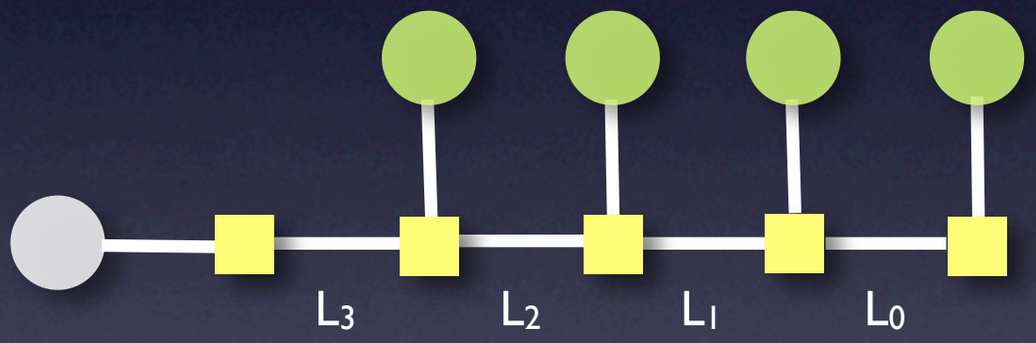



Time 100 to 150

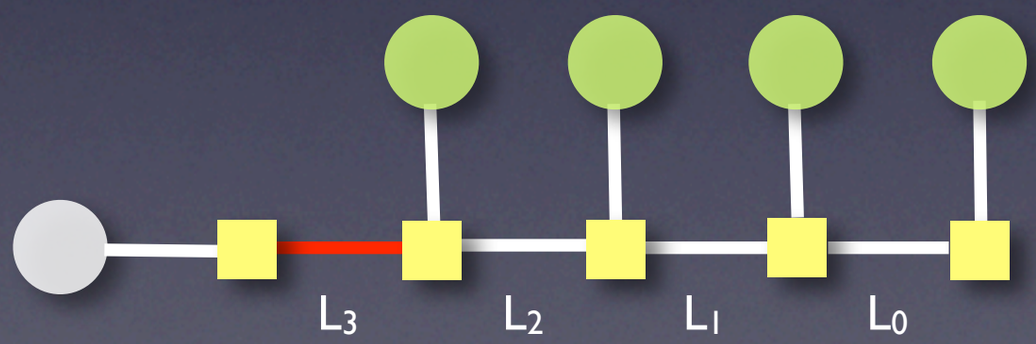
# Background Traffic



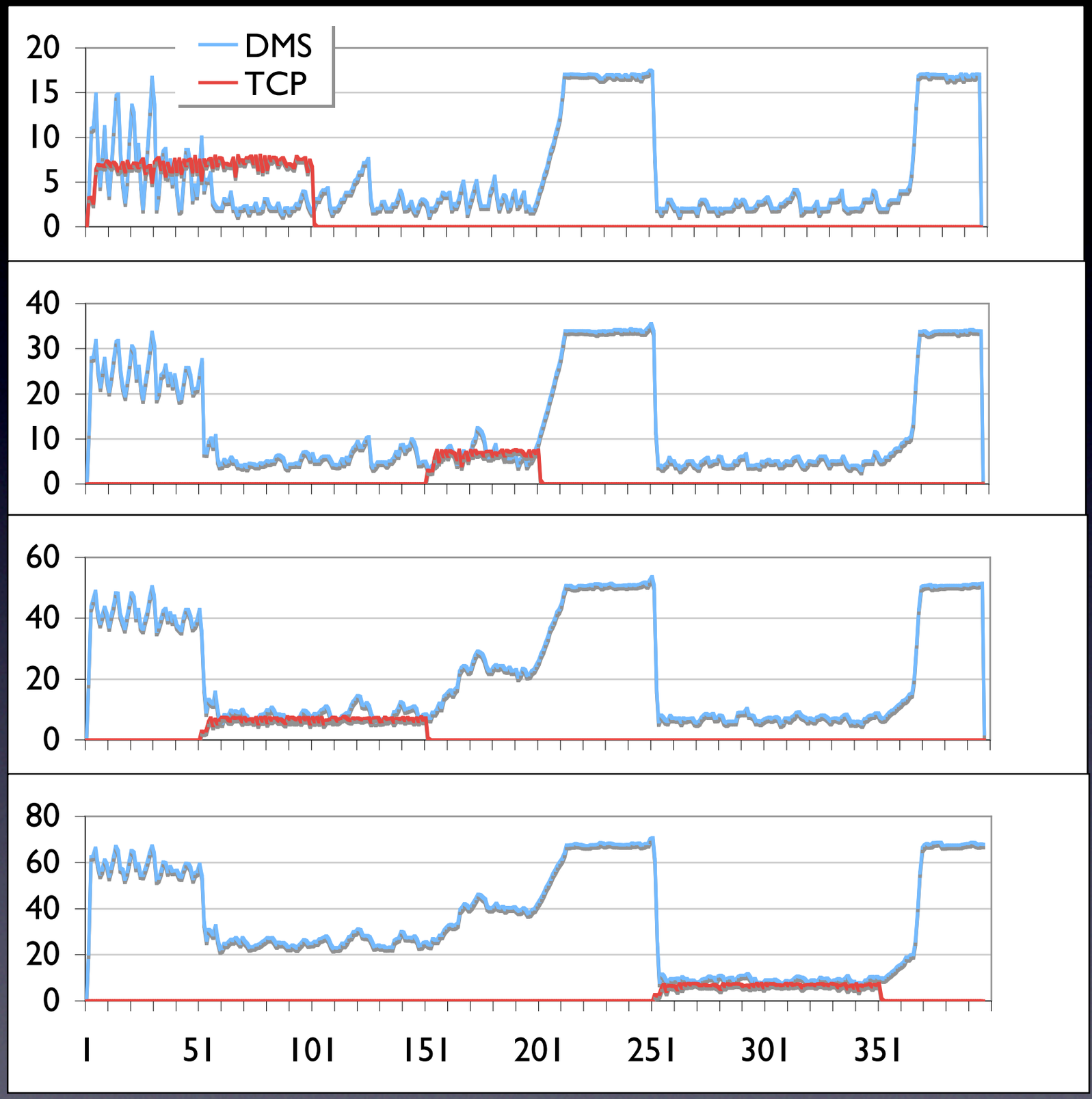

Time 150 to 200




Time 200 to 250




Time 250 to 350

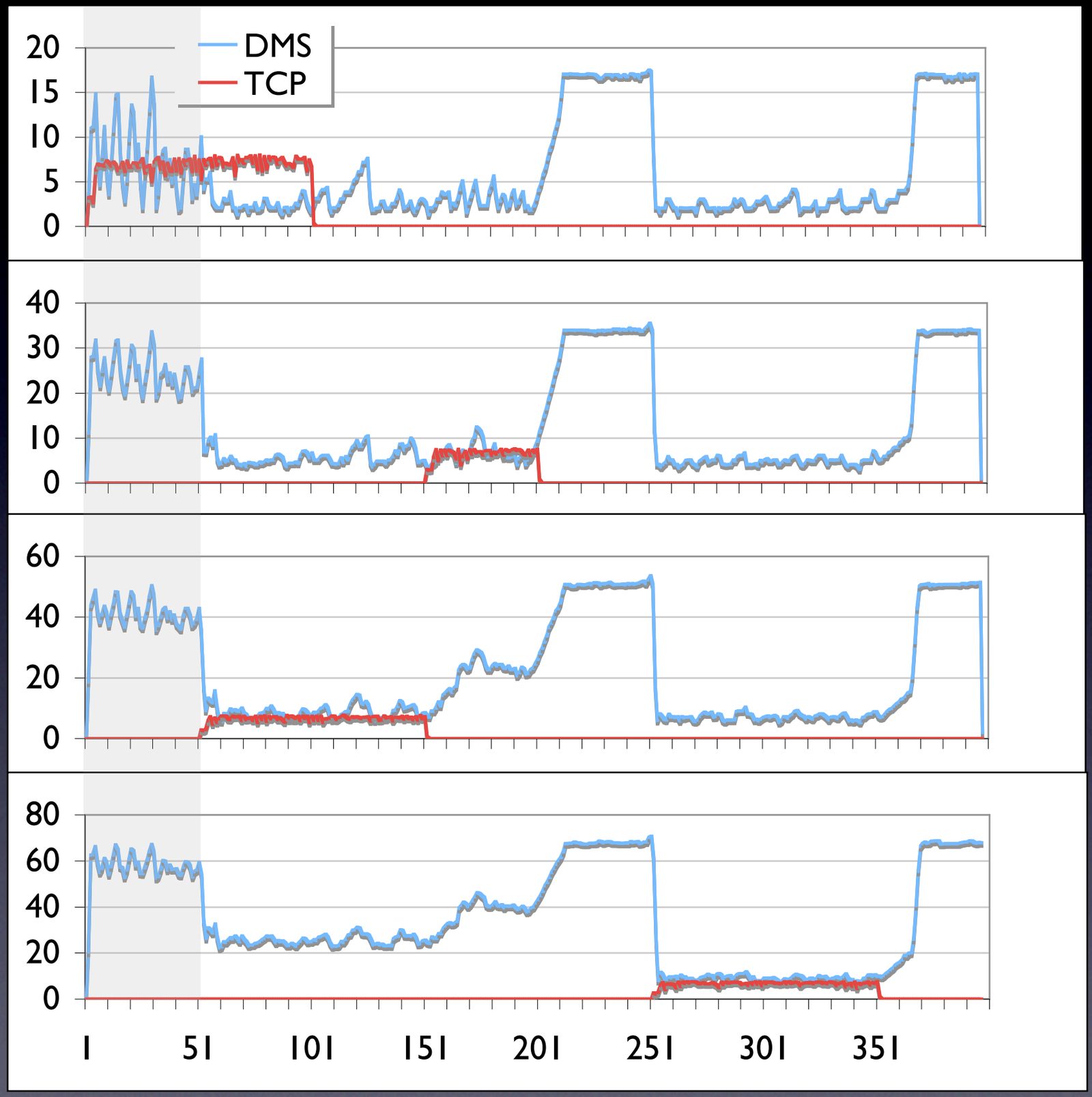

DMS
TCP
20
15
10
5
0
40
30
20
10
0
60
40
20
0
80
60
40
20
0
1
51
101
151
201
251
301
351

$L_0$

$L_1$

$L_2$

$L_3$

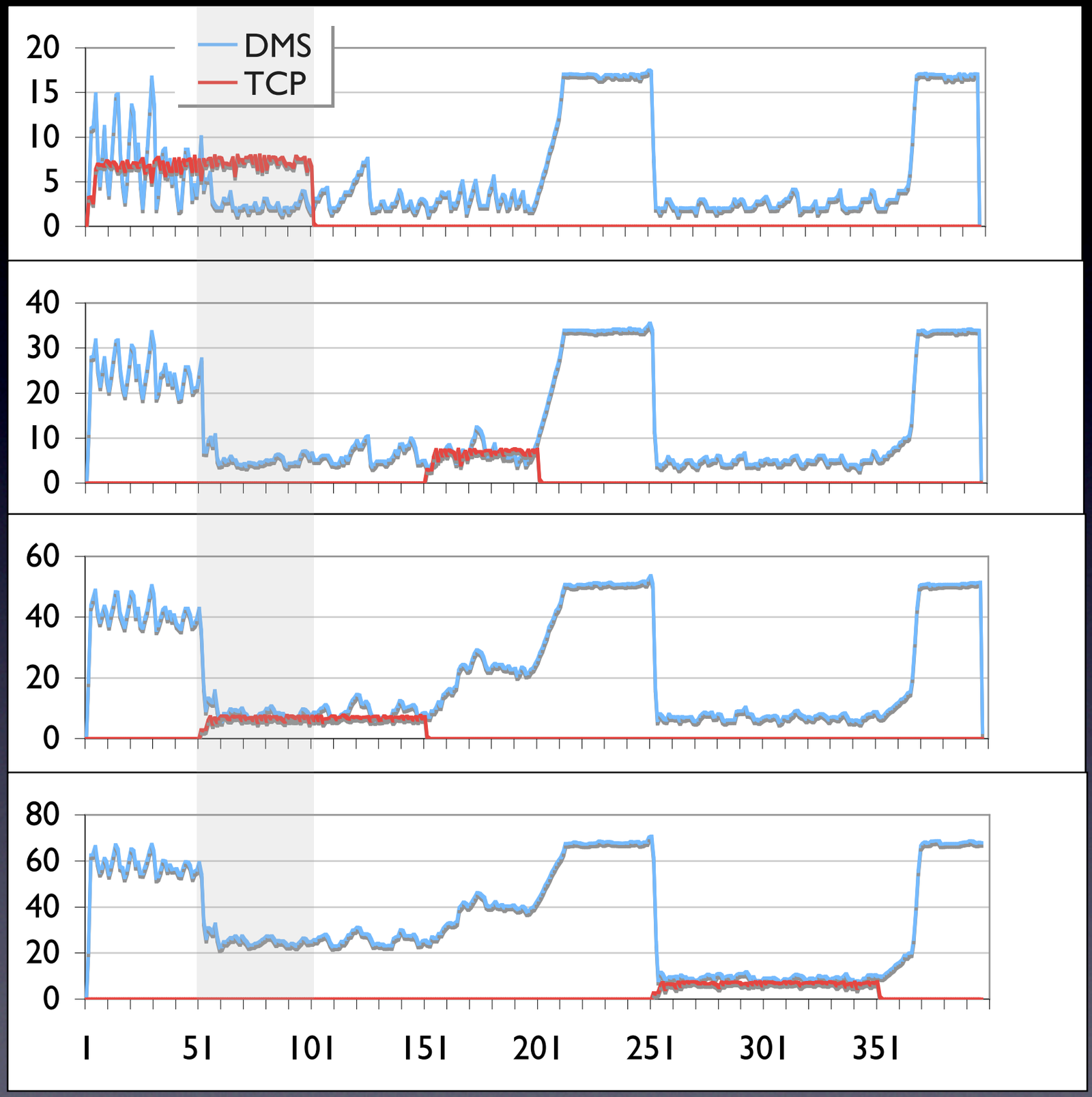

DMS
TCP
20
15
10
5
0
40
30
20
10
0
60
40
20
0
80
60
40
20
0
1
51
101
151
201
251
301
351

$L_0$

$L_1$

$L_2$

$L_3$

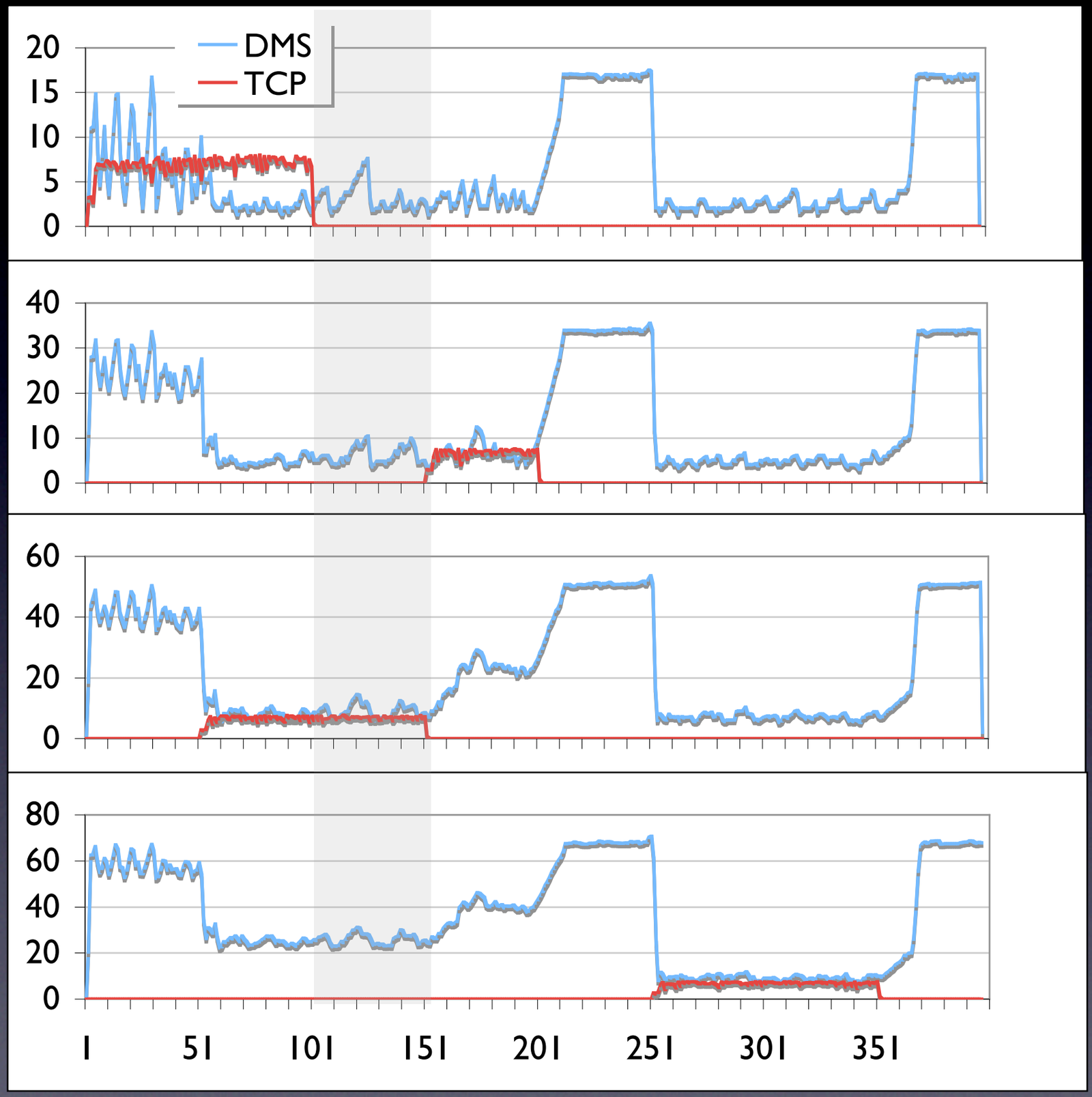

DMS
TCP
20
15
10
5
0
40
30
20
10
0
60
40
20
0
80
60
40
20
0
1
51
101
151
201
251
301
351

$L_0$

$L_1$

$L_2$

$L_3$

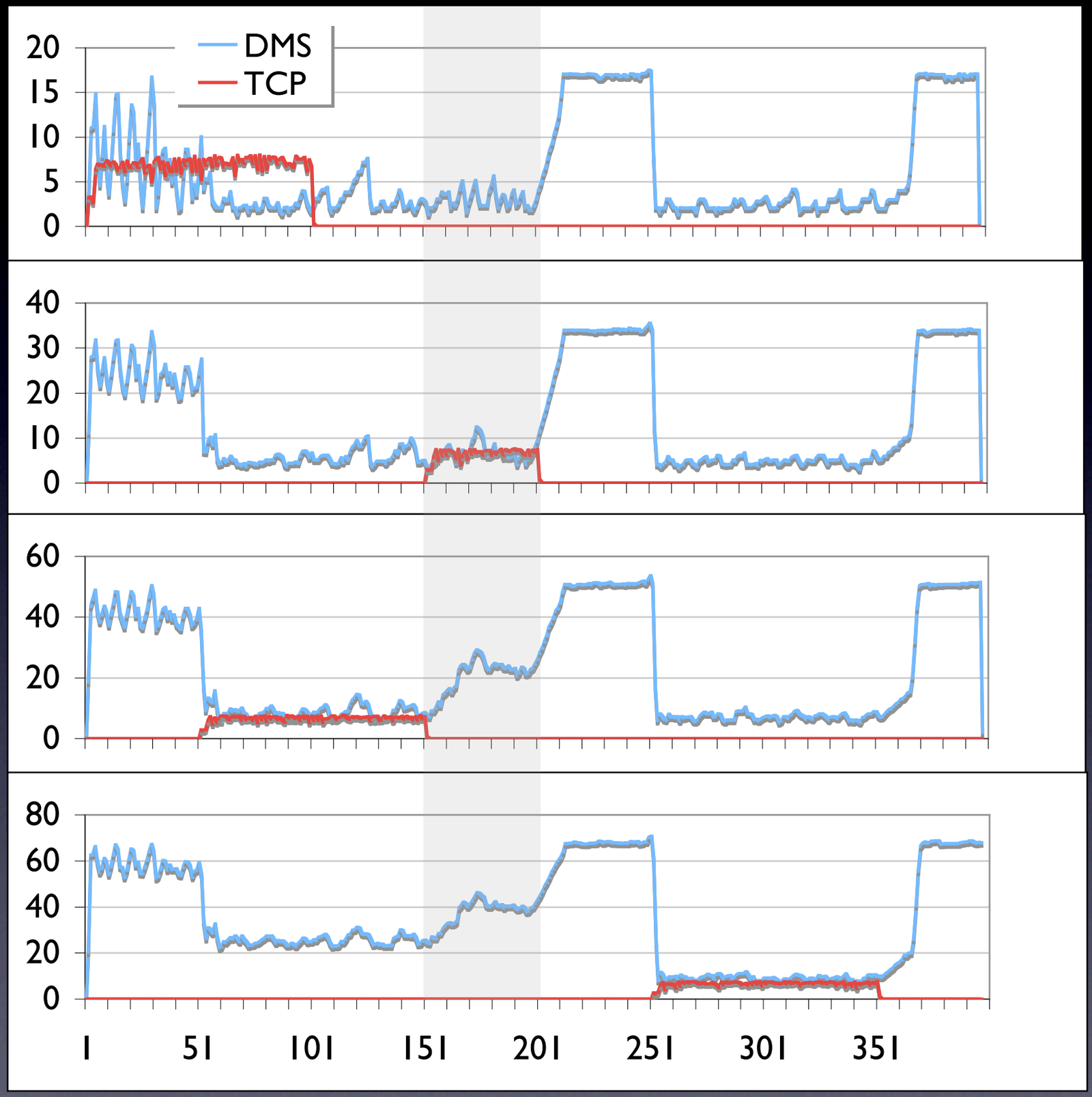

DMS
TCP
20
15
10
5
0
40
30
20
10
0
60
40
20
0
80
60
40
20
0
1
51
101
151
201
251
301
351

$L_0$

$L_1$

$L_2$

$L_3$

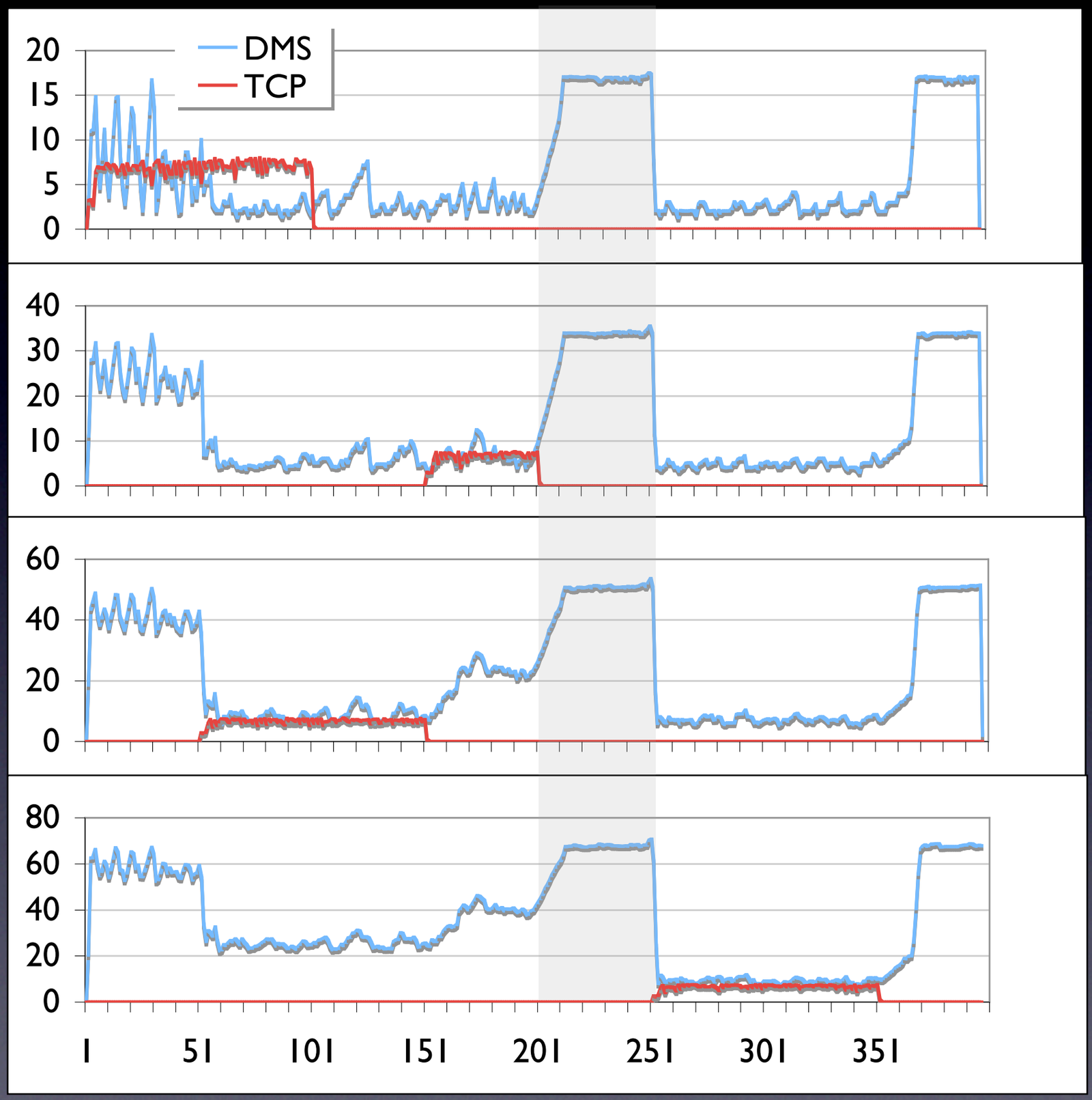

DMS
TCP
20
15
10
5
0
40
30
20
10
0
60
40
20
0
80
60
40
20
0
1
51
101
151
201
251
301
351

$L_0$

$L_1$

$L_2$

$L_3$

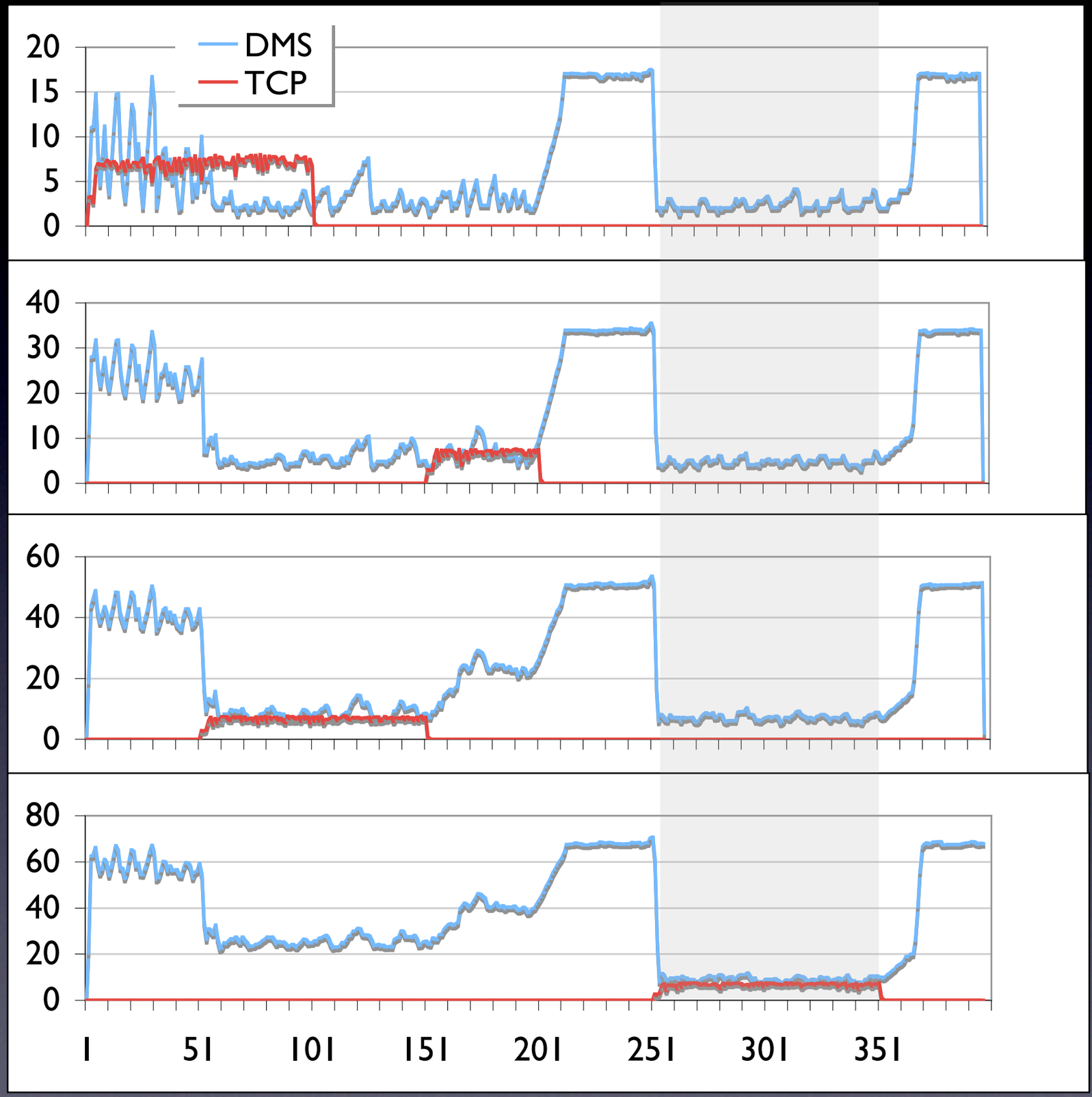

DMS
TCP
20
15
10
5
0
40
30
20
10
0
60
40
20
0
80
60
40
20
0
1
51
101
151
201
251
301
351

$L_0$

$L_1$

$L_2$

$L_3$

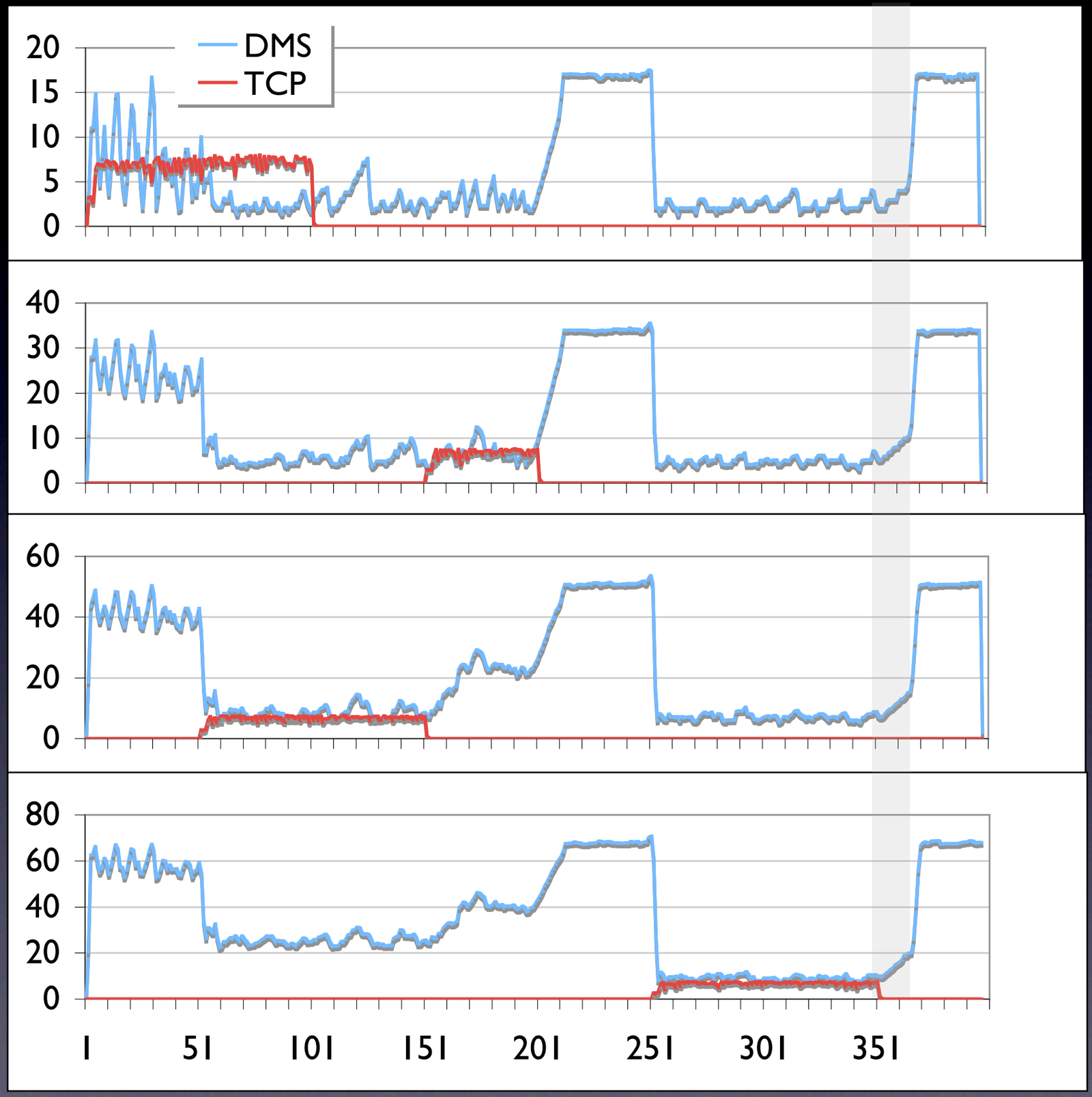

DMS
TCP
20
15
10
5
0
40
30
20
10
0
60
40
20
0
80
60
40
20
0
1
51
101
151
201
251
301
351

$L_0$

$L_1$

$L_2$

$L_3$

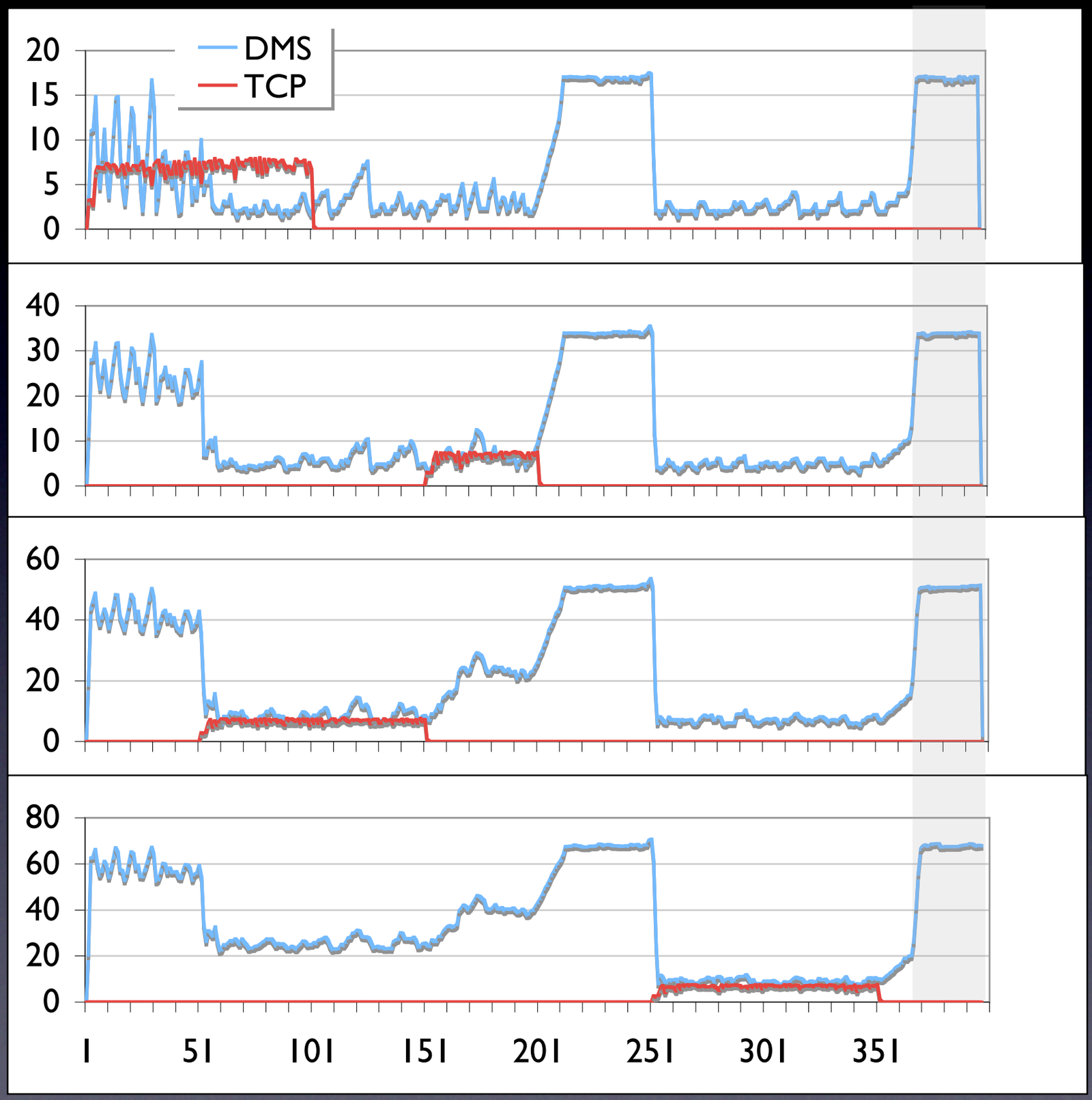

DMS
TCP
20
15
10
5
0
40
30
20
10
0
60
40
20
0
80
60
40
20
0
1
51
101
151
201
251
301
351

$L_0$

$L_1$

$L_2$

$L_3$


DMS
TCP
20
15
10
5
0
40
30
20
10
0
60
40
20
0
80
60
40
20
0
1
51
101
151
201
251
301
351

$L_0$

$L_1$

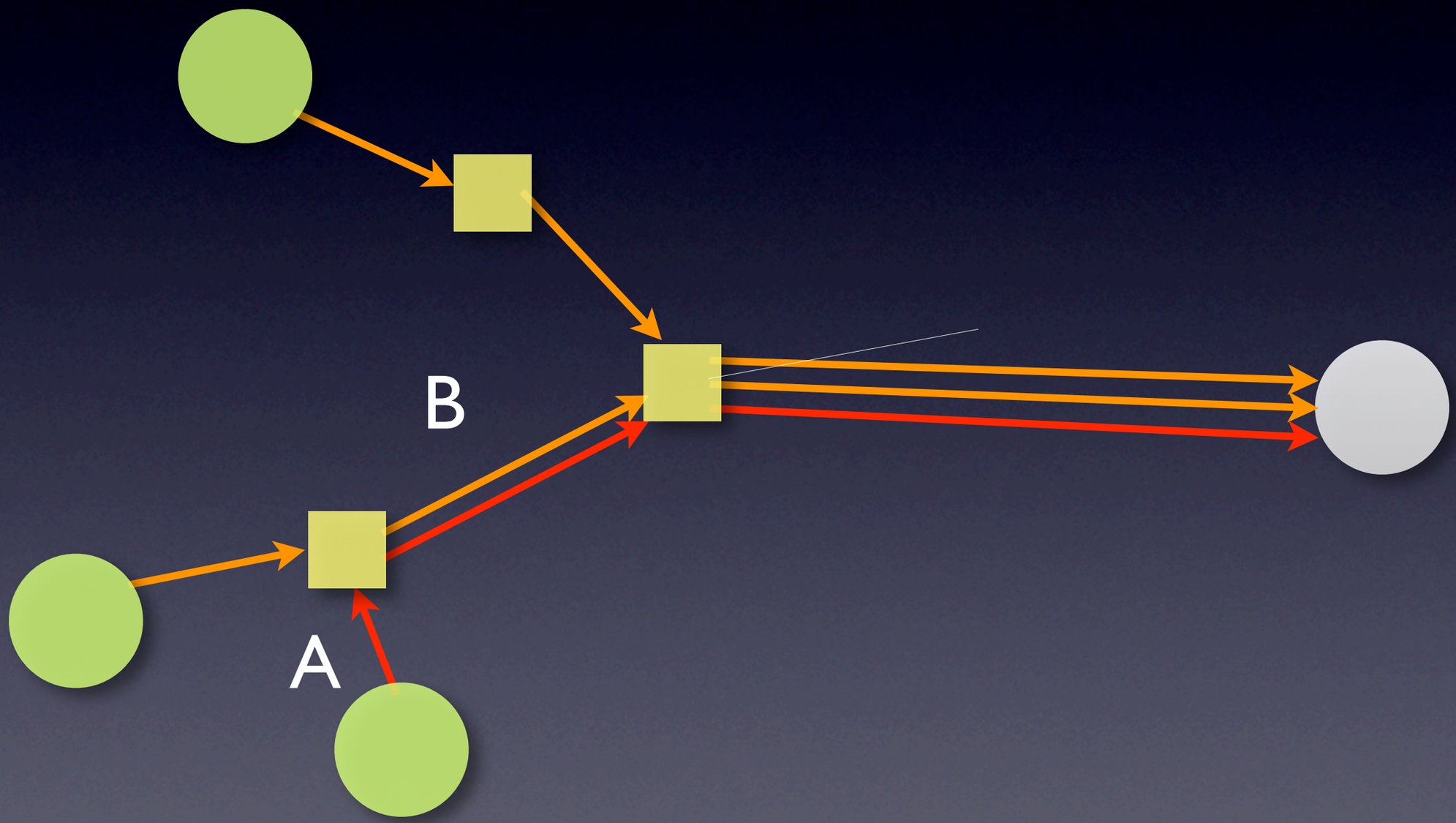
$L_2$

$L_3$

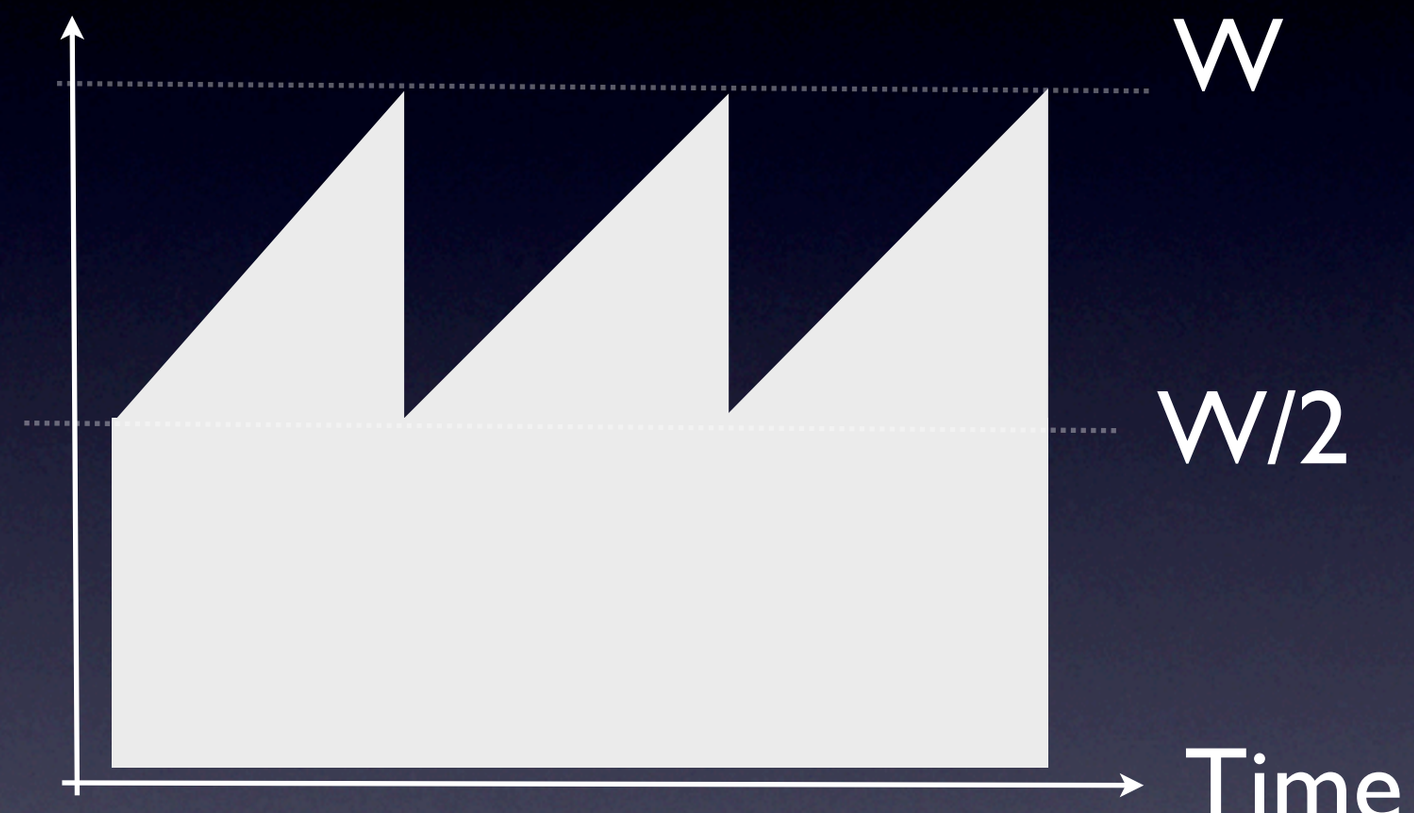
# Summary

- Distributed media streaming needs task-level congestion control.
- Two sub-problems: identify congested links and control sending rates.

If link A and B are congested at the same time, shared congestion at B might not be detected.

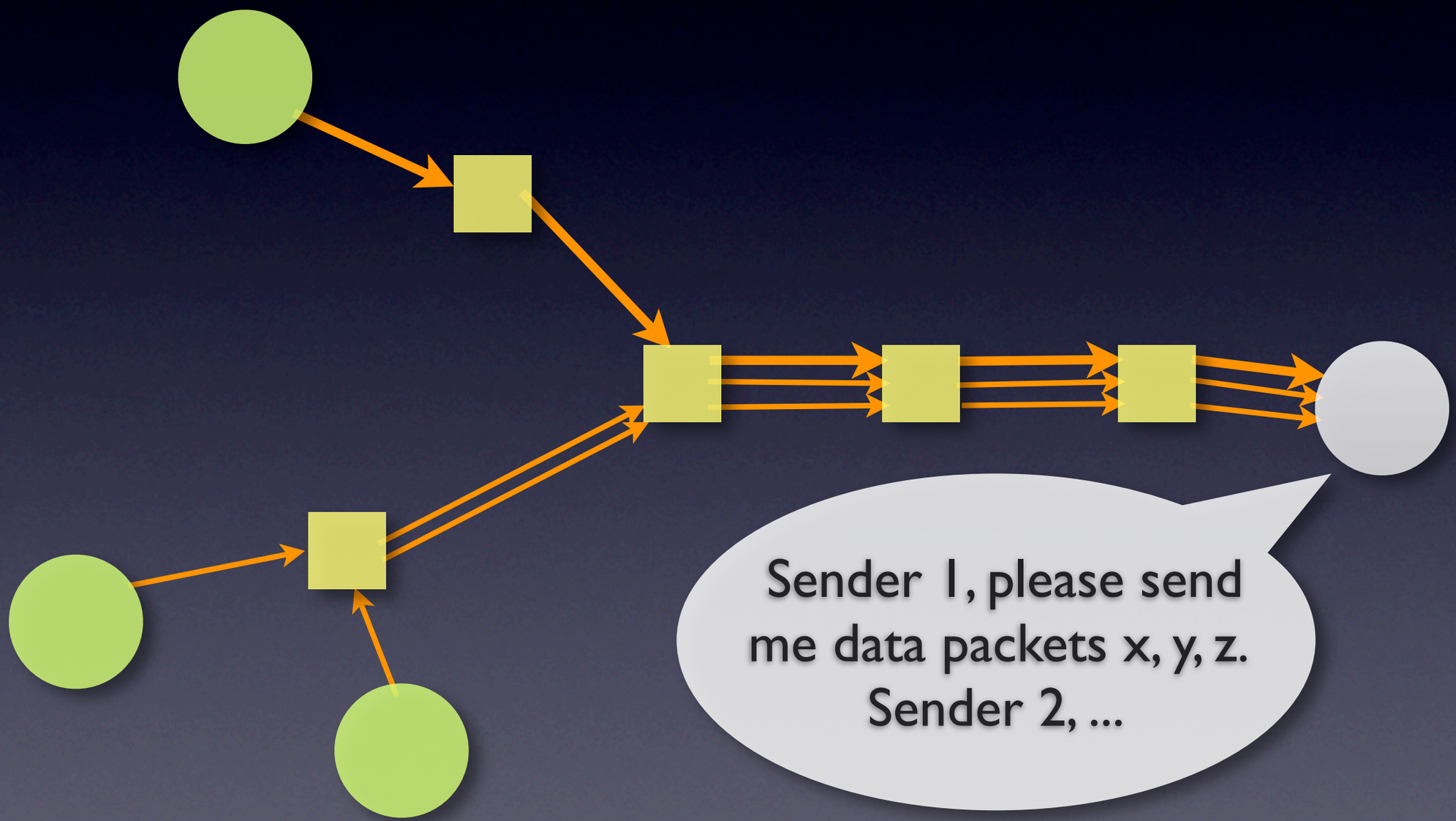


# Throughput control not as accurate when packet losses are bursty.




The window increases by $\alpha$ for every RTT;
Packet loss occur every 1/p packet .

Pull-based protocol might not be the right thing to do.



# The End