CS 268: Lecture 17
(Dynamic Packet State)

Ion Stoica

April 15, 2002

What is the Problem?

Internet has limited resources and management capabilities

- Prone to congestion, and denial of service
- Cannot provide guarantees

Existing solutions

- Stateless – scalable and robust, but weak network services
- Stateful – powerful services, but much less scalable and robust

Stateless vs. Stateful Solutions

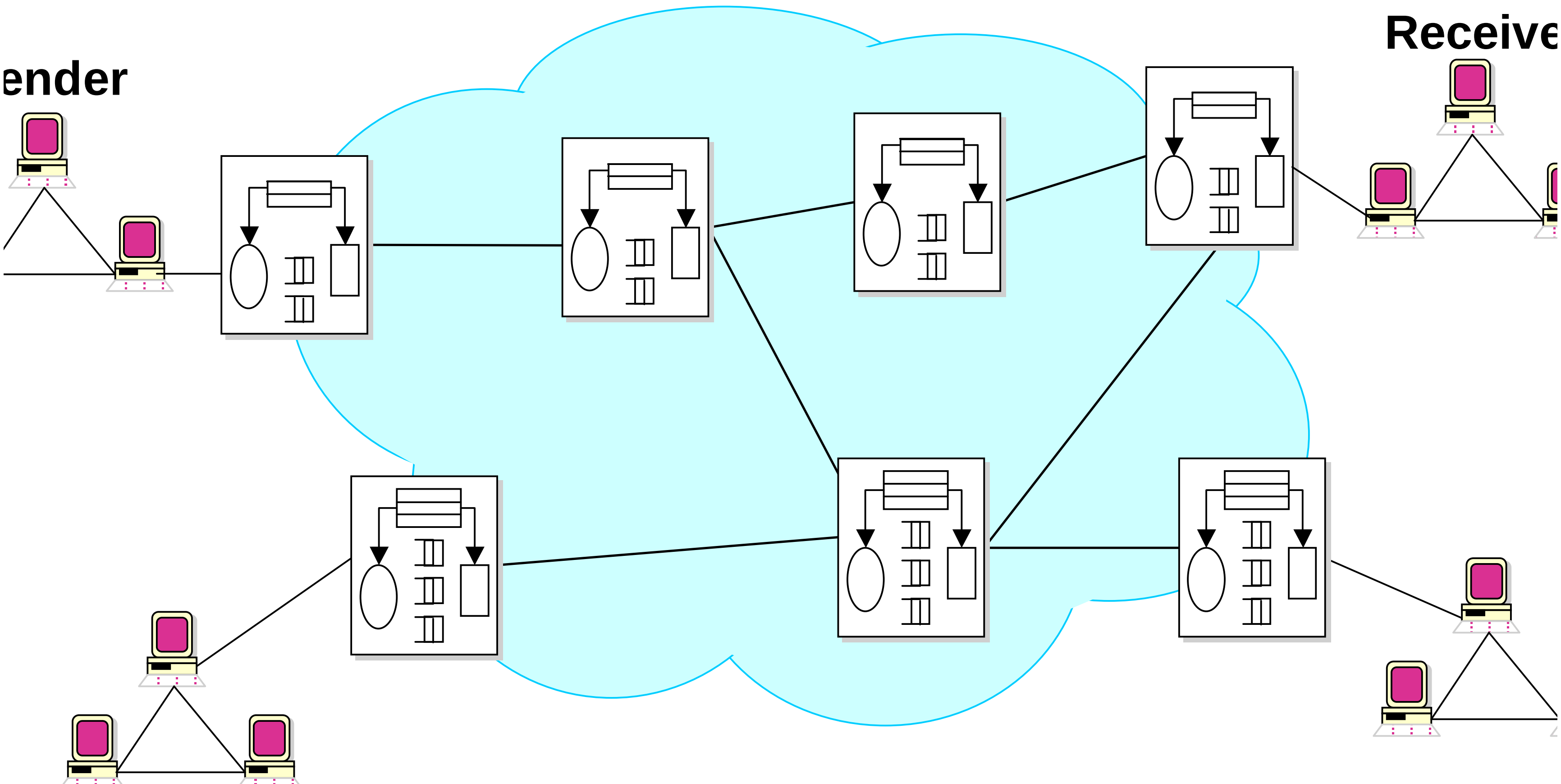
- Stateless solutions – routers maintain no fine grained state about traffic
  - ↑ scalable, robust
  - ↓ weak services
- Stateful solutions – routers maintain per-flow state
  - ↑ powerful services
    - guaranteed services + high resource utilization
    - fine grained differentiation
    - protection
  - ↓ much less scalable and robust

Existing Solutions

	Stateful	Stateless
;	- Tenet [Ferrari & Verma '89] - Intserv [Clark et al '91] - ATM [late '80s]	- Diffserv - [Clark & Wroclawski '97] - [Nichols et al '97]
vork oort for jestion :rol	- Round Robin [Nagle '85] - Fair Queueing [Demers et al '89] - Flow Random Early Drop (FRED) [Lin & Morris '97]	- DecBit [Ramkrishnan & Ja '88] - Random Early Detection (RED) [Floyd & Jacobson '! - BLUE [Feng et al '99]

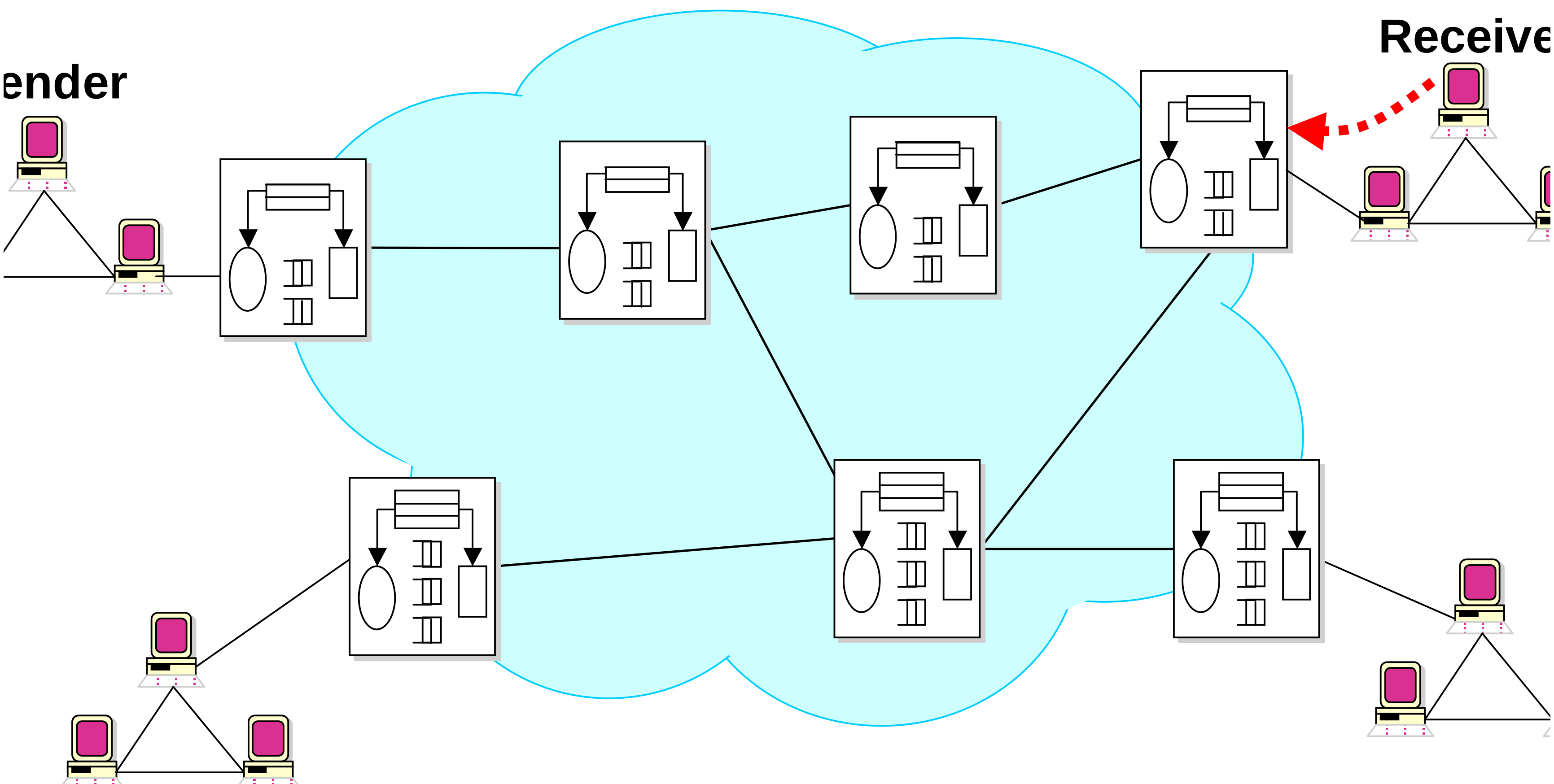
Stateful Solution: Guaranteed Services

- Achieve per-flow bandwidth and delay guarantees
  - Example: guarantee 1MBps and < 100 ms delay to a flow



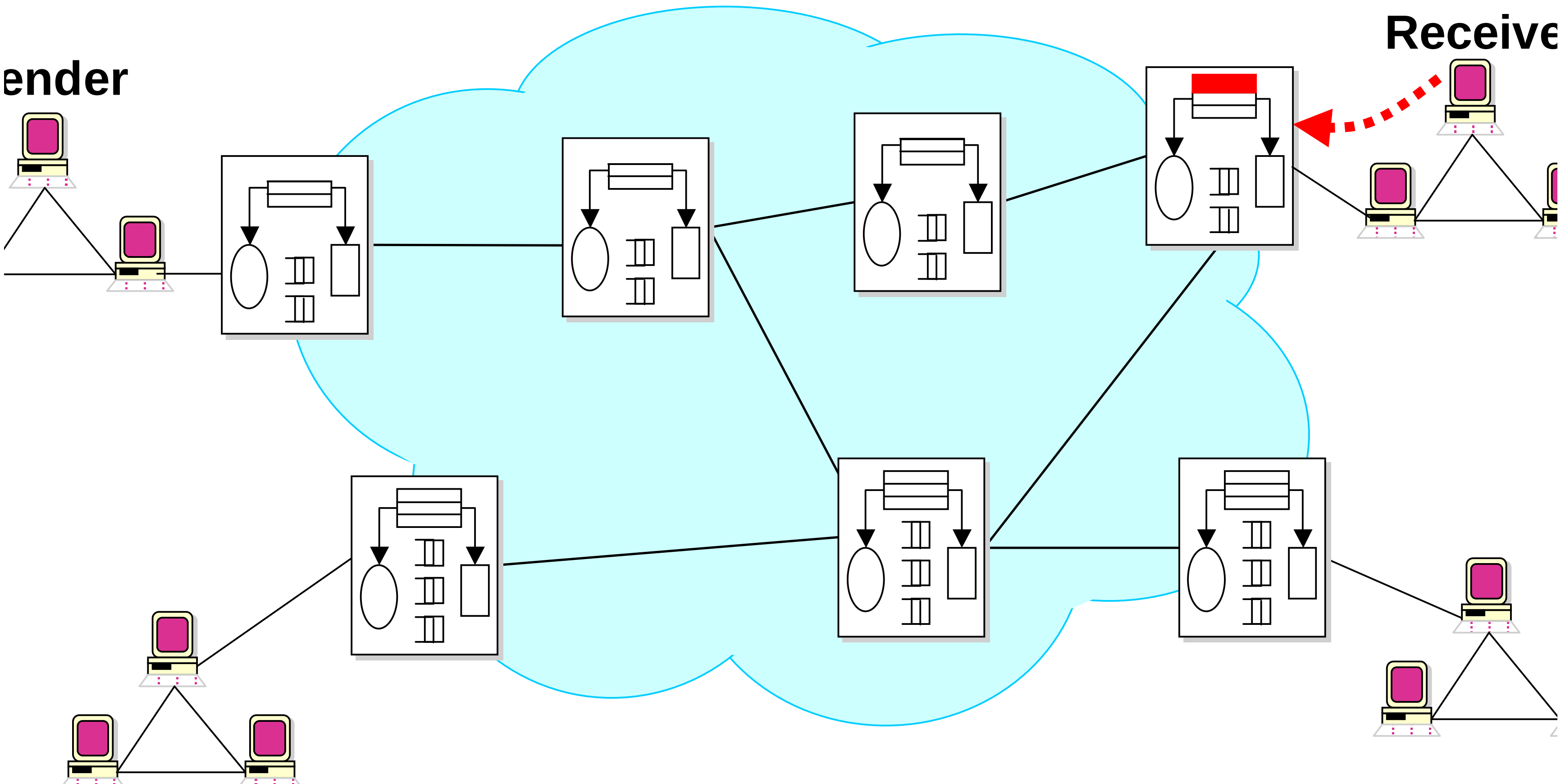
Stateful Solution: Guaranteed Services

- Allocate resources - perform per-flow admission control



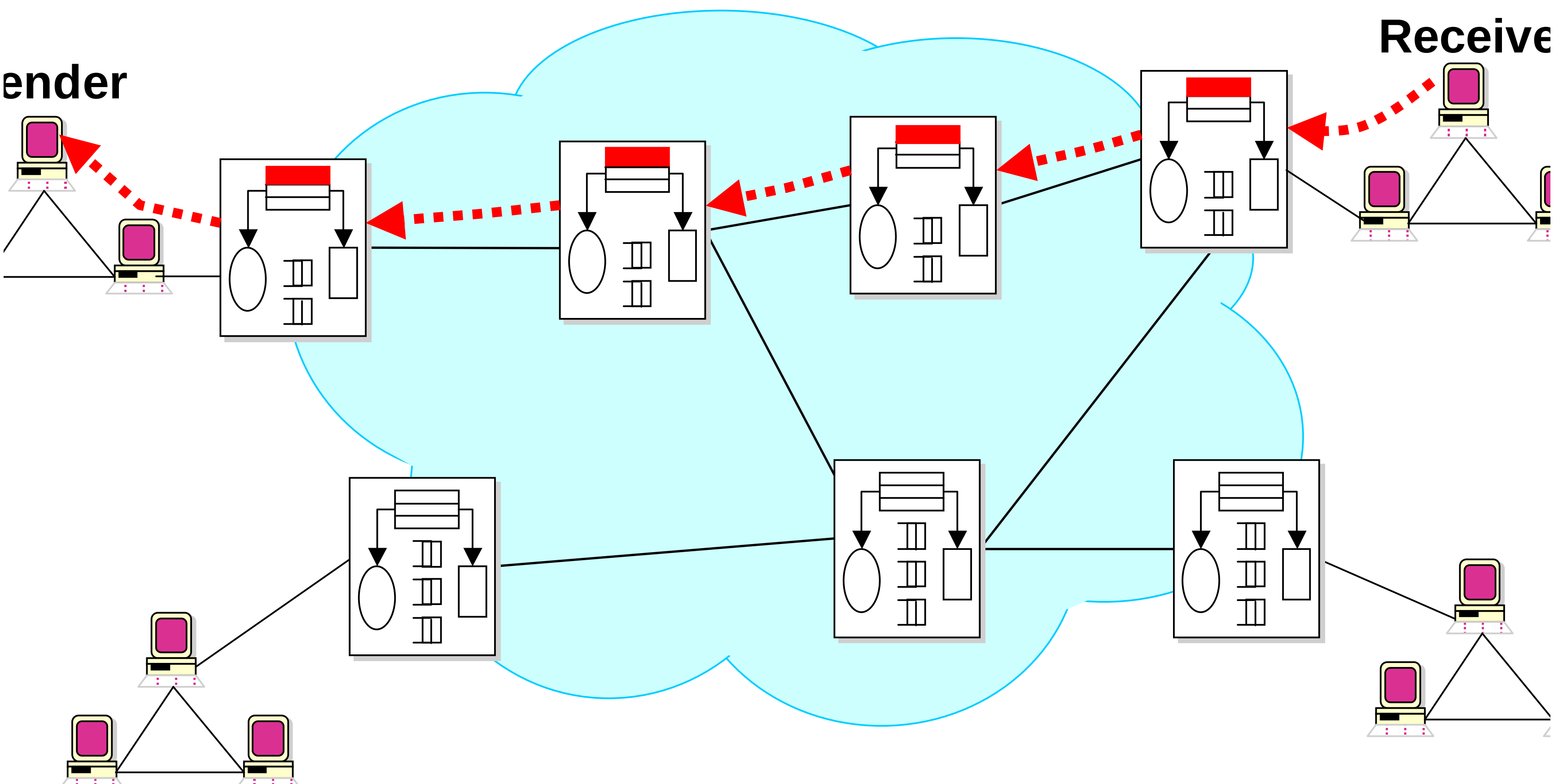
Stateful Solution: Guaranteed Services

- Install per-flow state



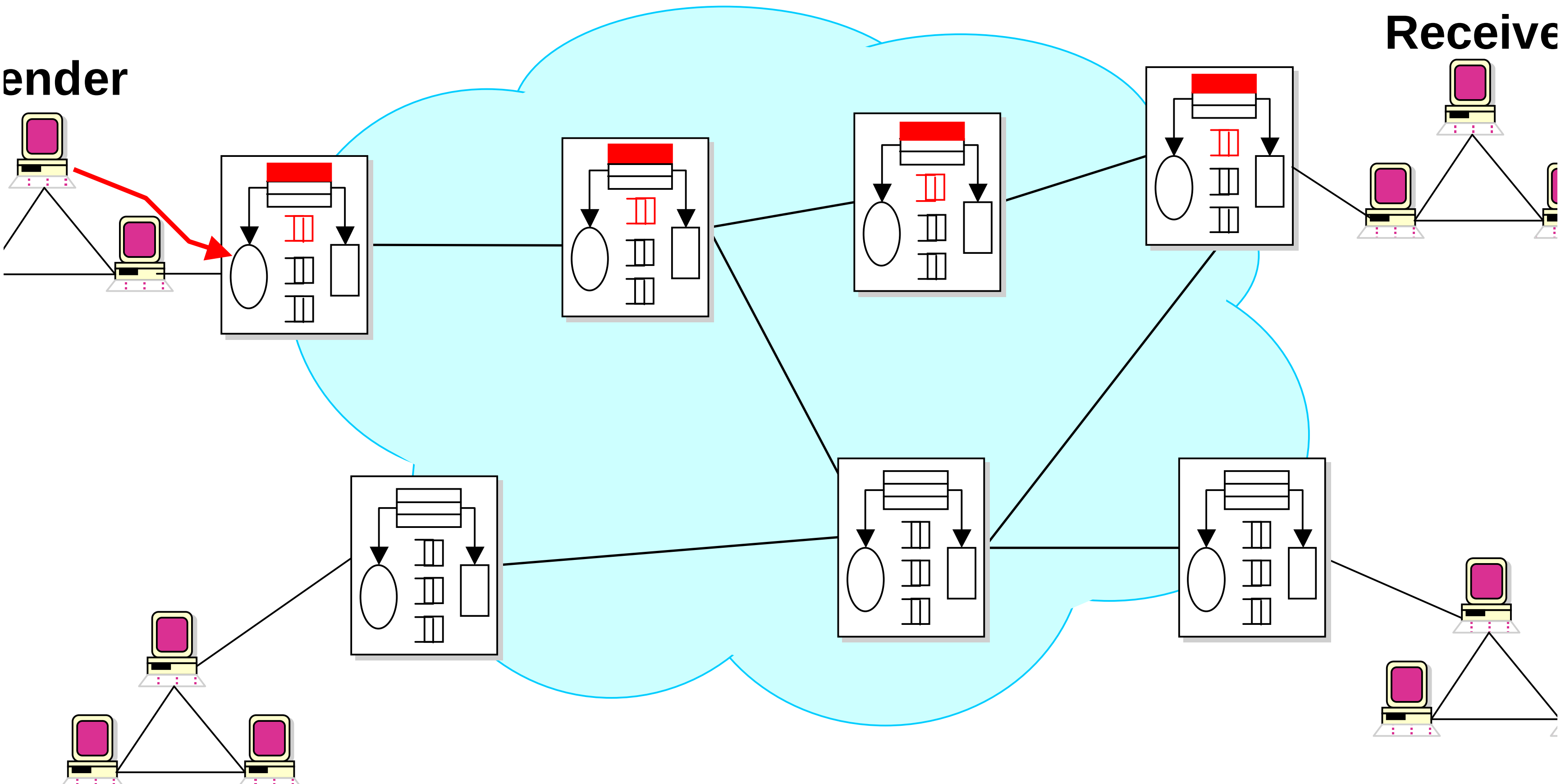
Stateful Solution: Guaranteed Services

- Challenge: maintain per-flow state consistent



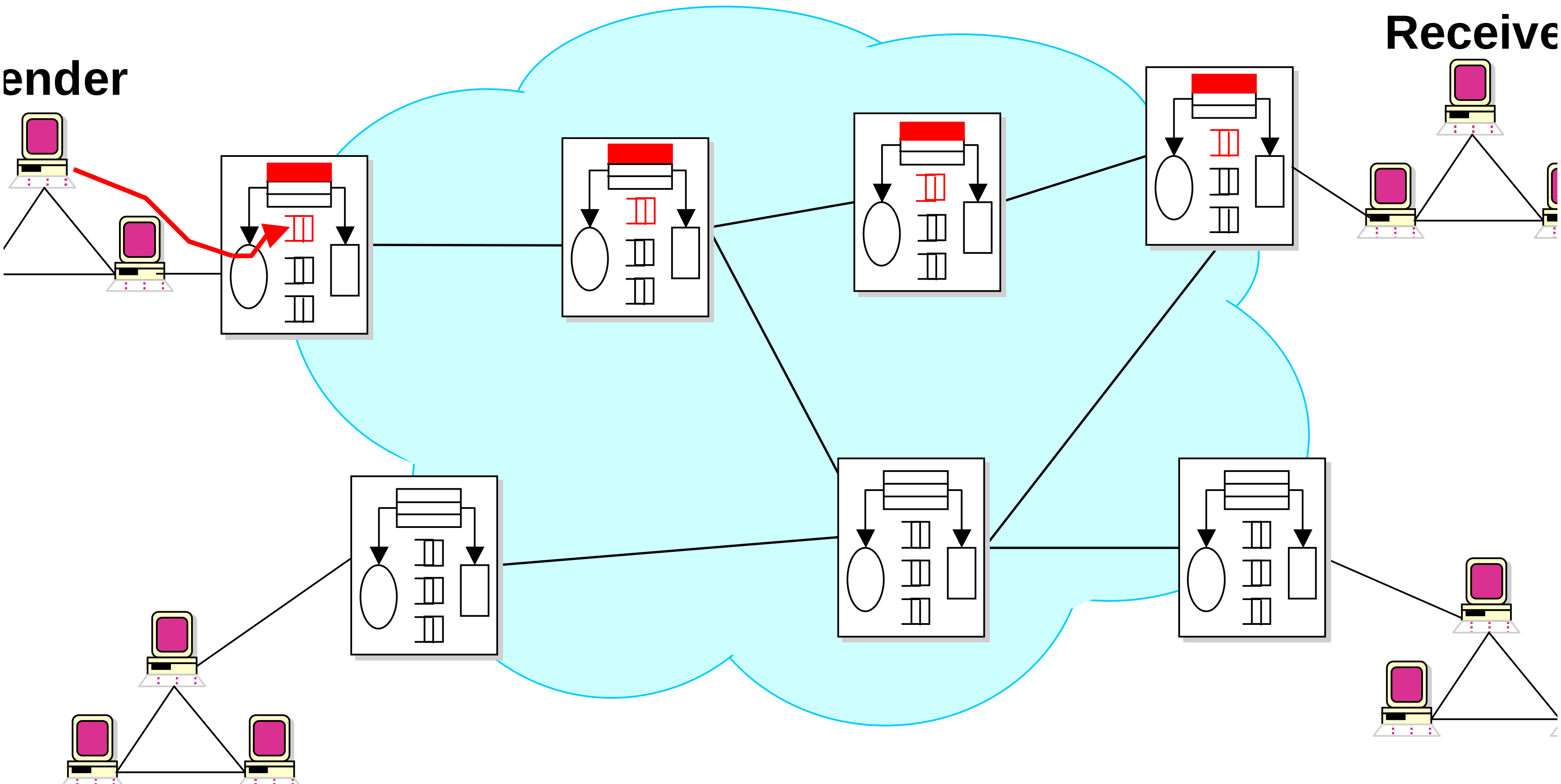
Stateful Solution: Guaranteed Services

- Per-flow classification



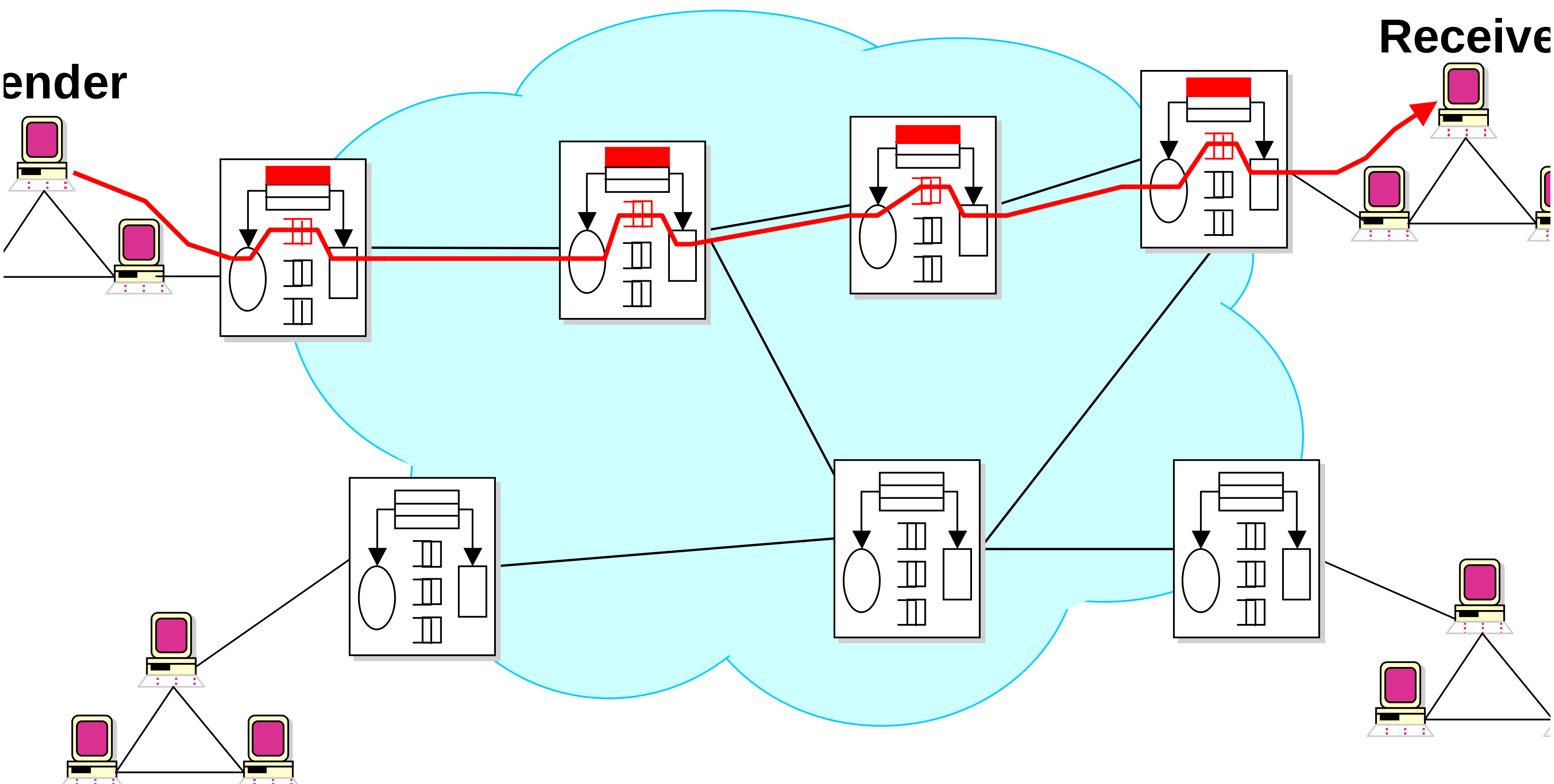
Stateful Solution: Guaranteed Services

- Per-flow buffer management



Stateful Solution: Guaranteed Services

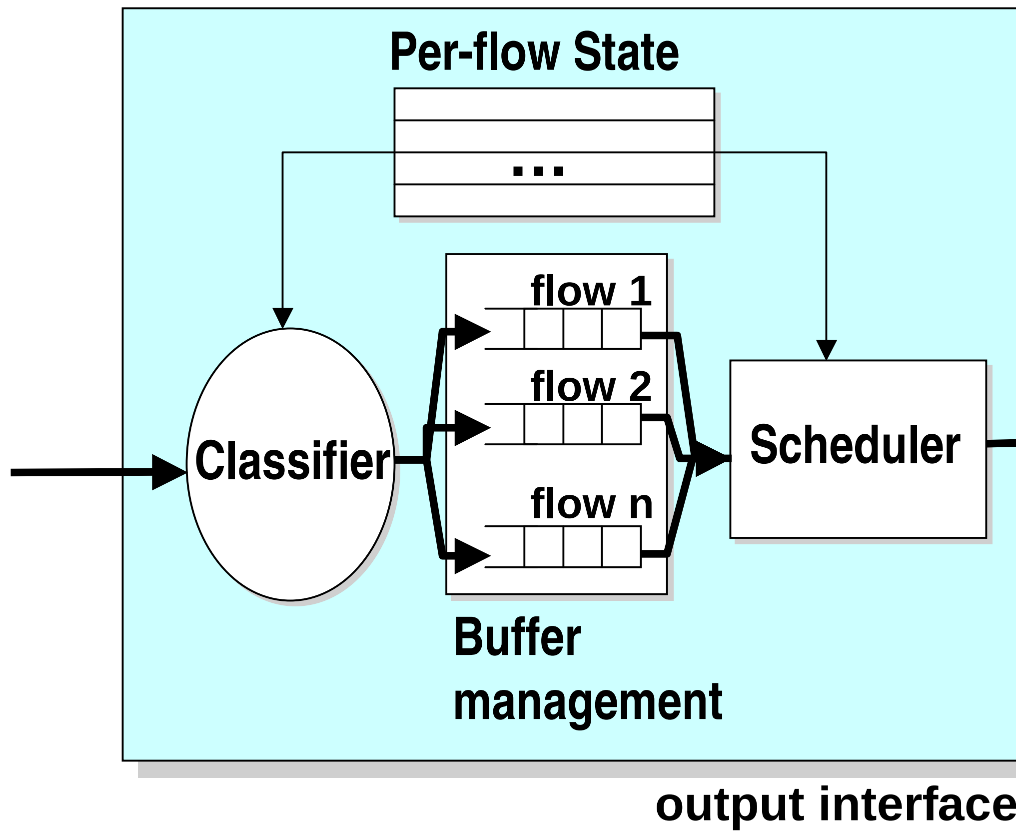
- **Per-flow scheduling**



Stateful Solution Complexity

Data path
- Per-flow classification
- Per-flow buffer management
- Per-flow scheduling

Control path
- install and maintain per-flow state for data and control paths



Stateless vs. Stateful

- Stateless solutions are more
  - scalable
  - robust
- Stateful solutions provide more powerful and flexible services
  - guaranteed services + high resource utilization
  - fine grained differentiation
  - protection

Question

- Can we achieve the best of two worlds, i.e., provide services implemented by stateful networks while maintaining advantages of stateless architectures?

Answer

Yes, at least in some interesting cases:

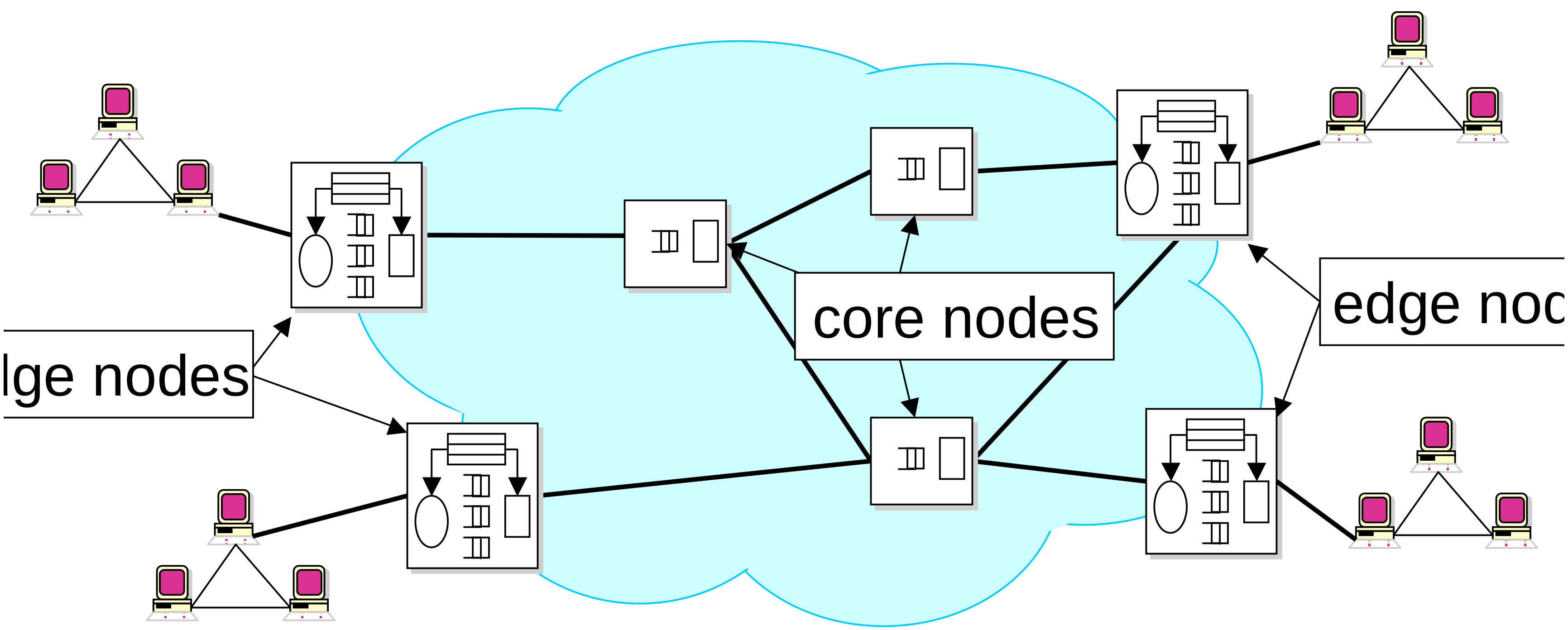
- guaranteed services [Stoica and Zhang, SIGCOMM'99]
- network support for congestion control: *Core*-Stateless Fair Queueing [Stoica et al, SIGCOMM'98]
- service differentiation [Stoica and Zhang, NOSSDAV'98]

Outline

- *Solution: SCORE architecture and DPS technique*
- Example: providing guaranteed services
- Conclusions

Scalable Core (SCORE)

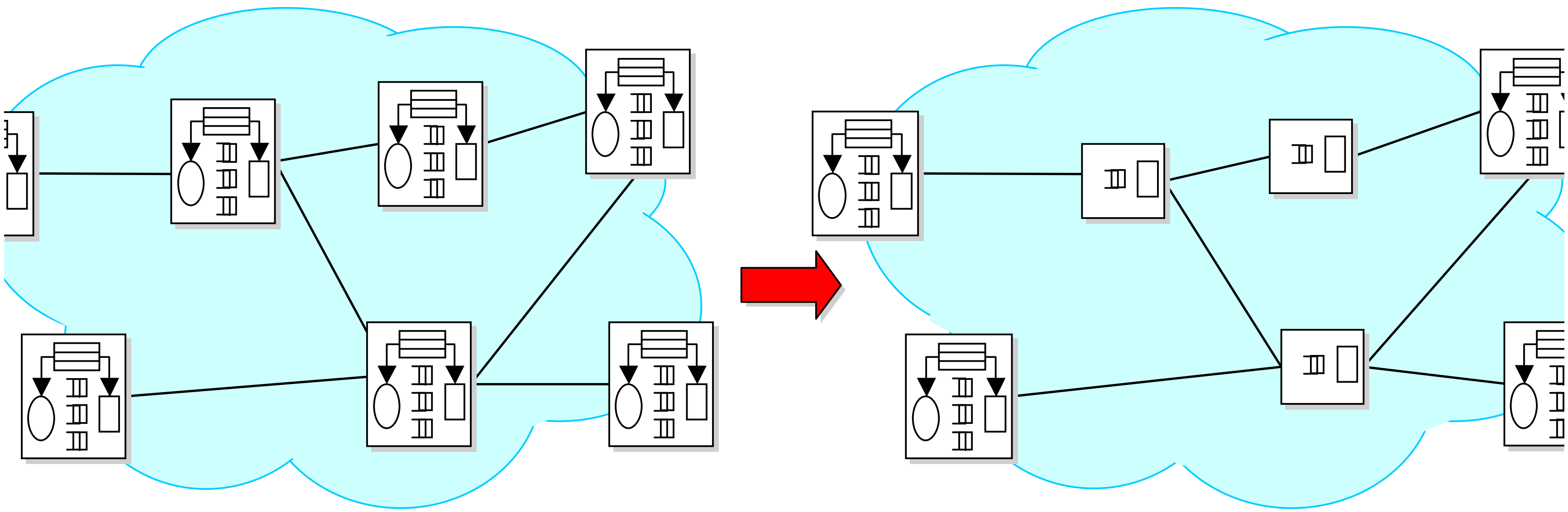
- A trusted and contiguous region of network in which
  - edge nodes – perform per flow management
  - core nodes – do not perform per flow management



The Approach

1. Define a reference stateful network that implements the desired service

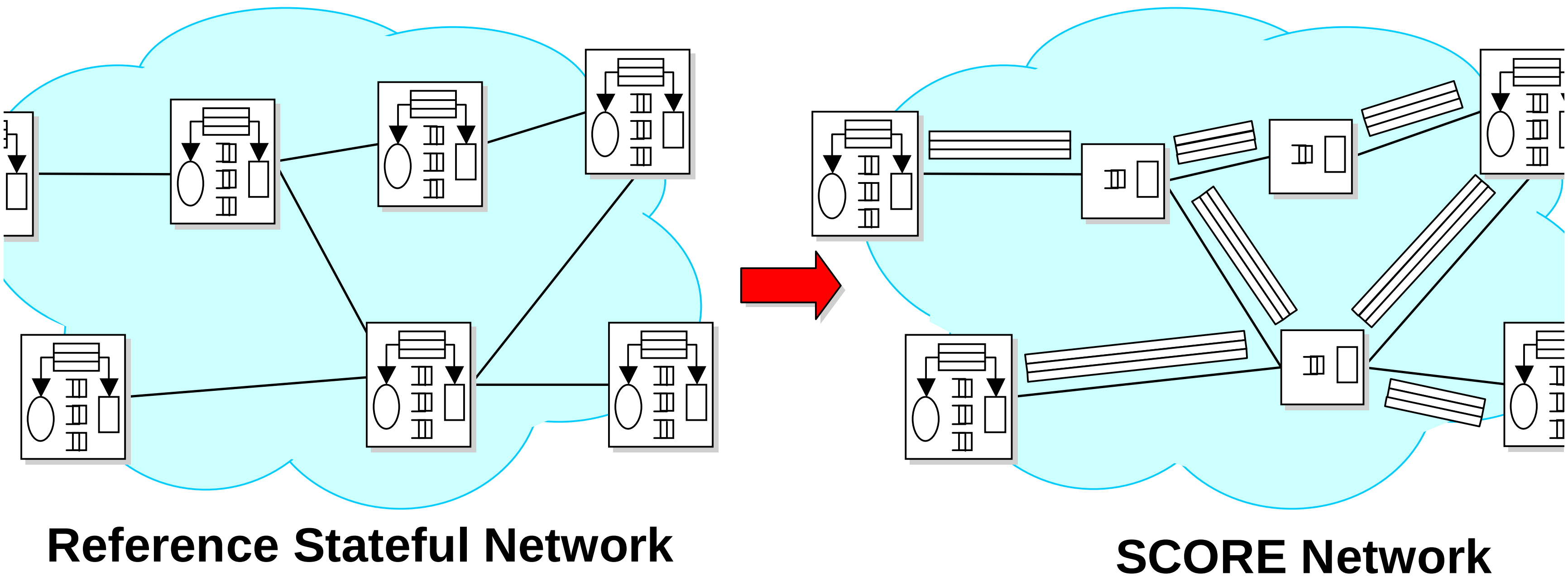
2. Emulate the functionality of the reference network in a SCORE network



Reference Stateful Network **SCORE Network**

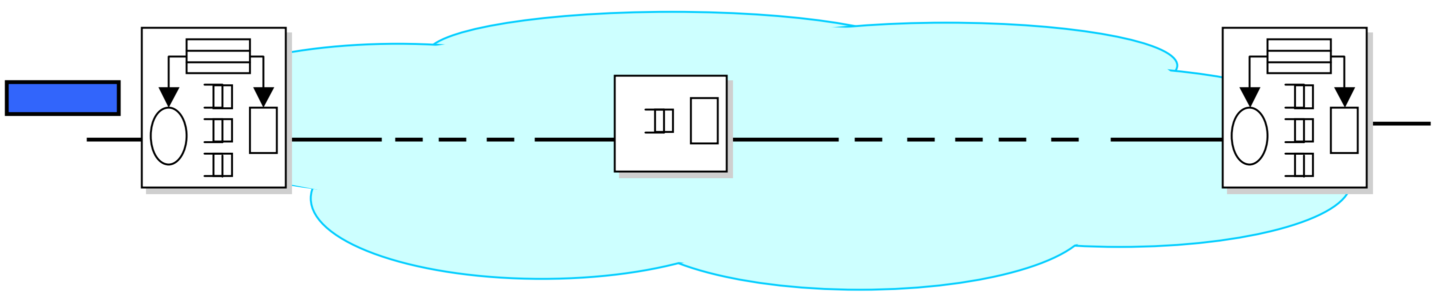
The Idea

- Instead of having core routers maintaining per-flow state have packets carry per-flow state



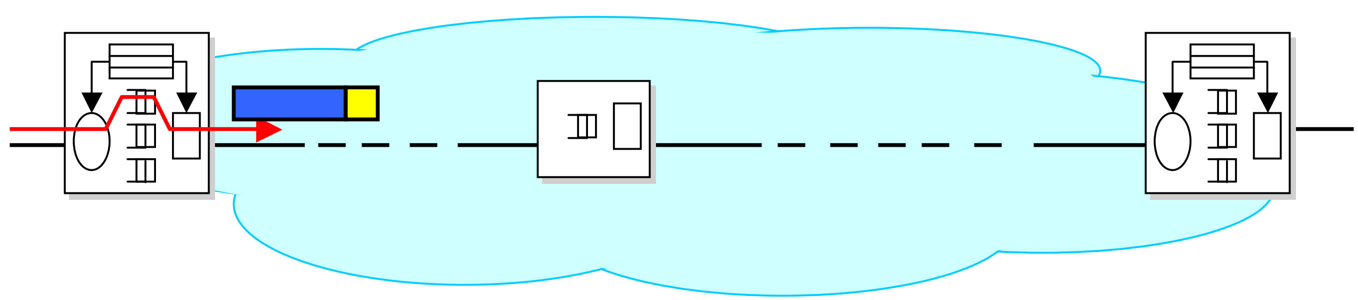
The Technique: Dynamic Packet State (DPS)

- Ingress node: compute and insert flow state in packet's header



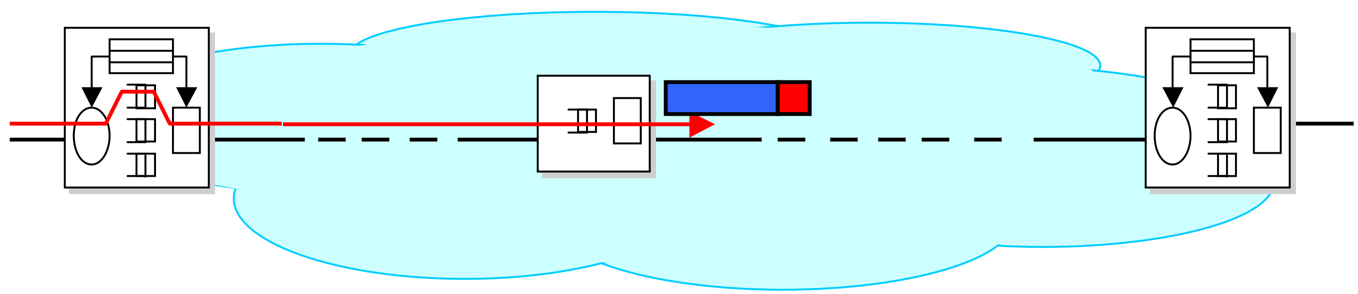
The Technique: Dynamic Packet State (DPS)

- Ingress node: compute and insert flow state in packet's header



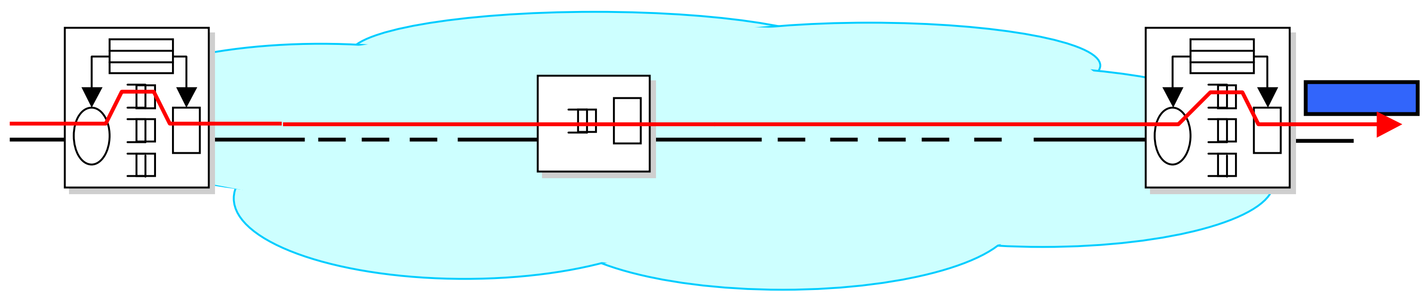
The Technique: Dynamic Packet State (DPS)

- Core node:
  - process packet based on state it carries and node's state
  - update both packet and node's state



The Technique: Dynamic Packet State (DPS)

- Egress node: remove state from packet's header



Outline

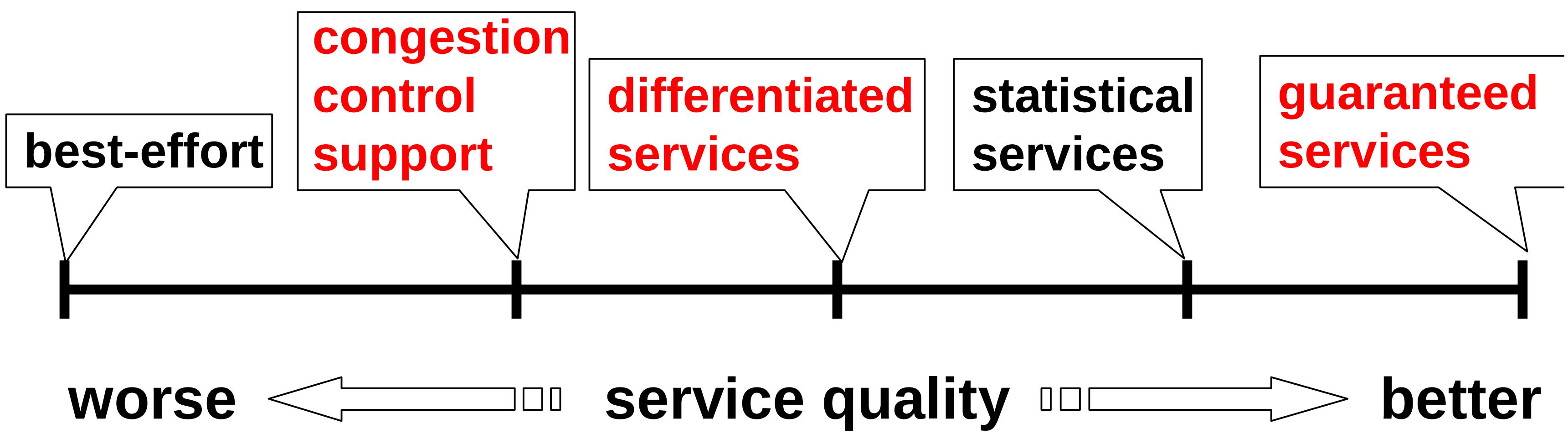
Solution: SCORE architecture and DPS technique

Example: providing guaranteed services

Conclusions

Why Guaranteed Service Example?

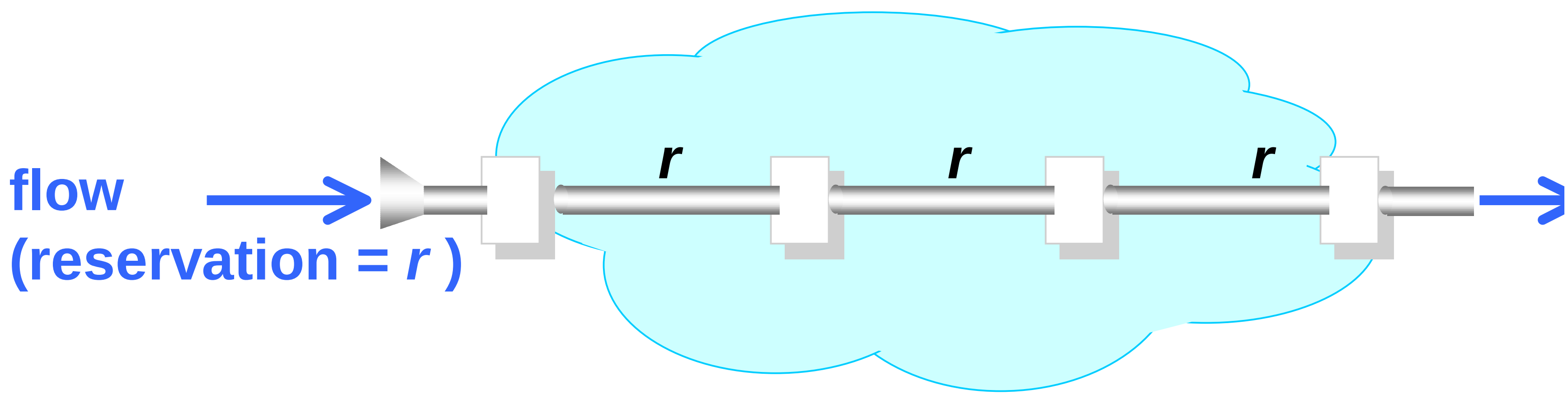
- Illustrate power and flexibility of our solution
  - guaranteed service - strongest semantic service proposed in context of stateful networks



Example: Guaranteed Services

Goal: provide per-flow delay and bandwidth guarantees

How: emulate ideal model in which each flow traverses dedicated links of capacity *r*

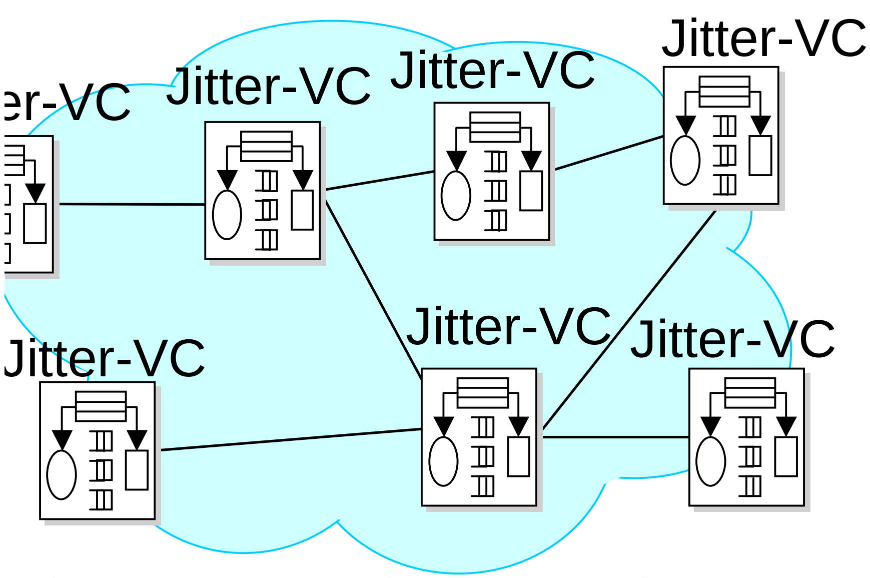


Per-hop packet service time = (packet length) / *r*

Guaranteed Services

Define reference network to implement service

- control path: per-flow admission control, reserve capacity *r* on each link
- data path: enforce ideal model, by using Jitter Virtual Clock (Jitter-VC) scheduler

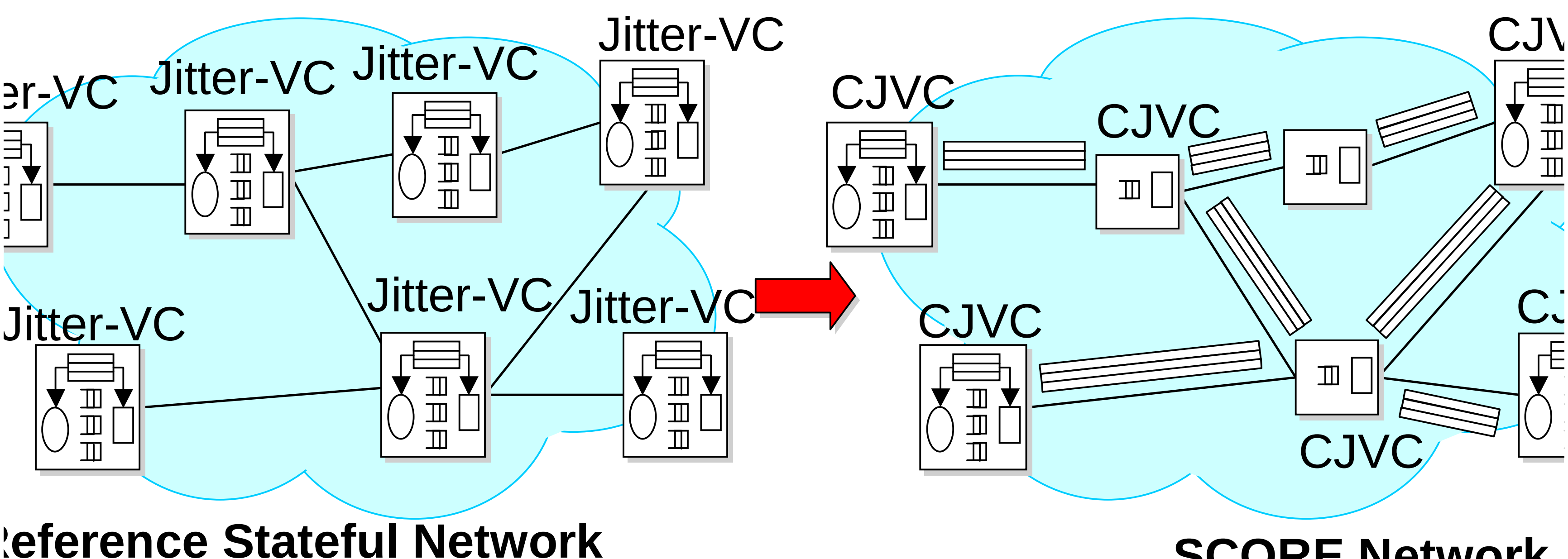


Reference Stateful Network

Guaranteed Services

Use DPS to eliminate per-flow state in core

- control path: emulate per-flow admission control
- data path: emulate Jitter-VC by Core-Jitter Virtual Clock (CJVC)



Outline

Solution: SCORE architecture and DPS technique

Example: providing guaranteed services

- ➡ *Eliminate per-flow state on data path*
- Eliminate per-flow state on control path
- Implementation and experimental results

Conclusions

Data Path

Ideal Model

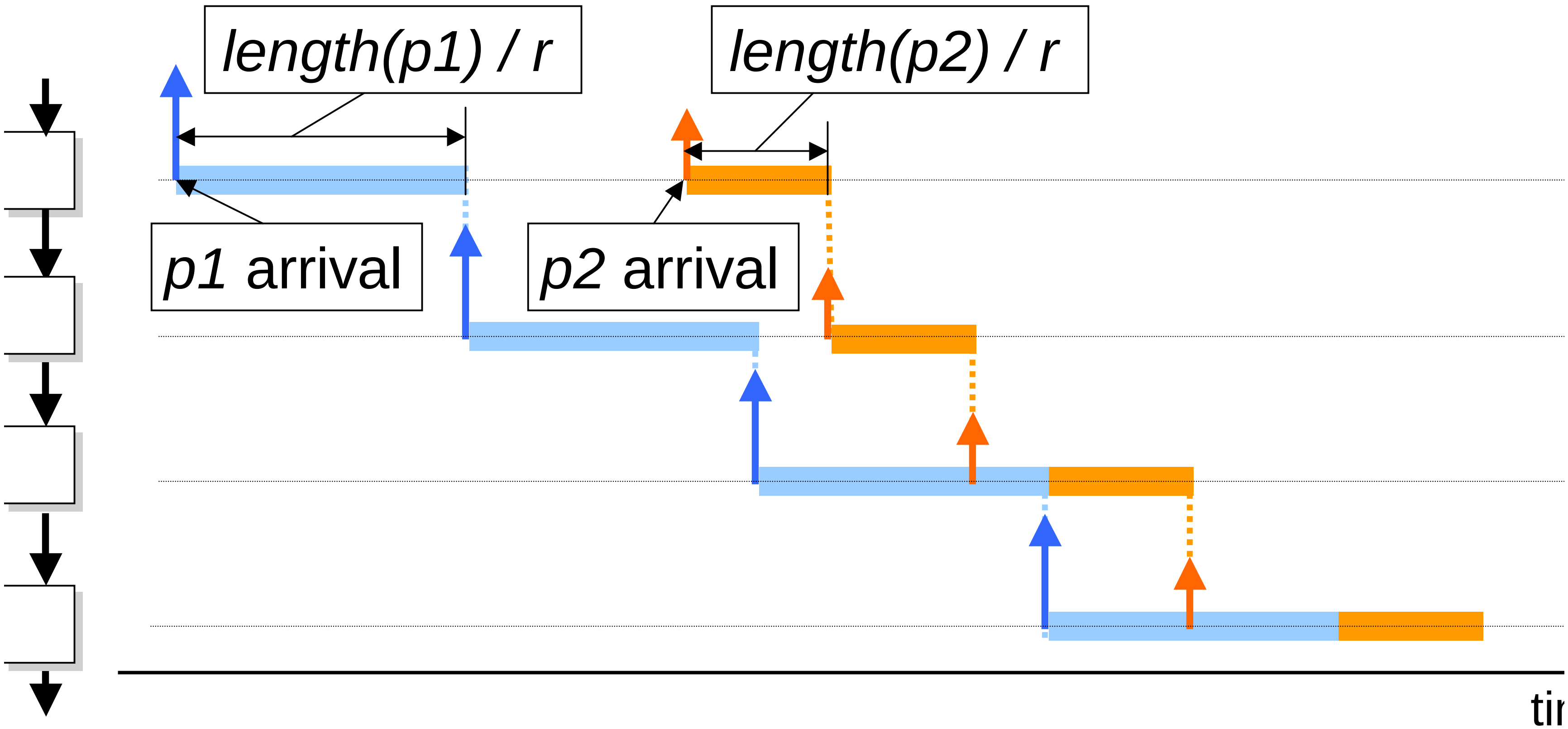
↓

Stateful solution: Jitter Virtual Clock

↓

Stateless solution: Core-Jitter Virtual Clock

Ideal Model: Example



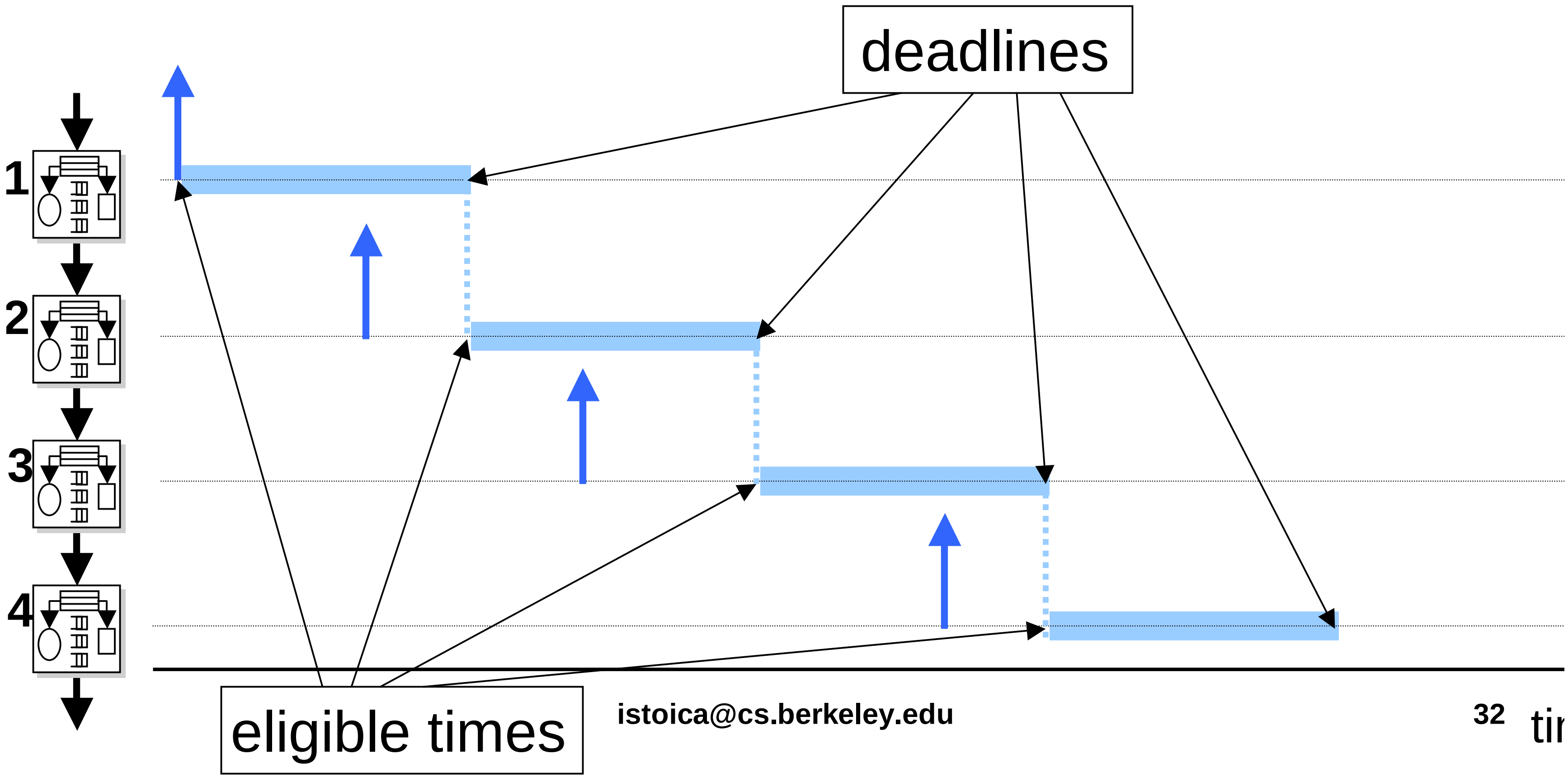
packet arrival time

packet transmission time (service) in ideal model

Stateful Solution: Jitter Virtual Clock (Jitter-VC)

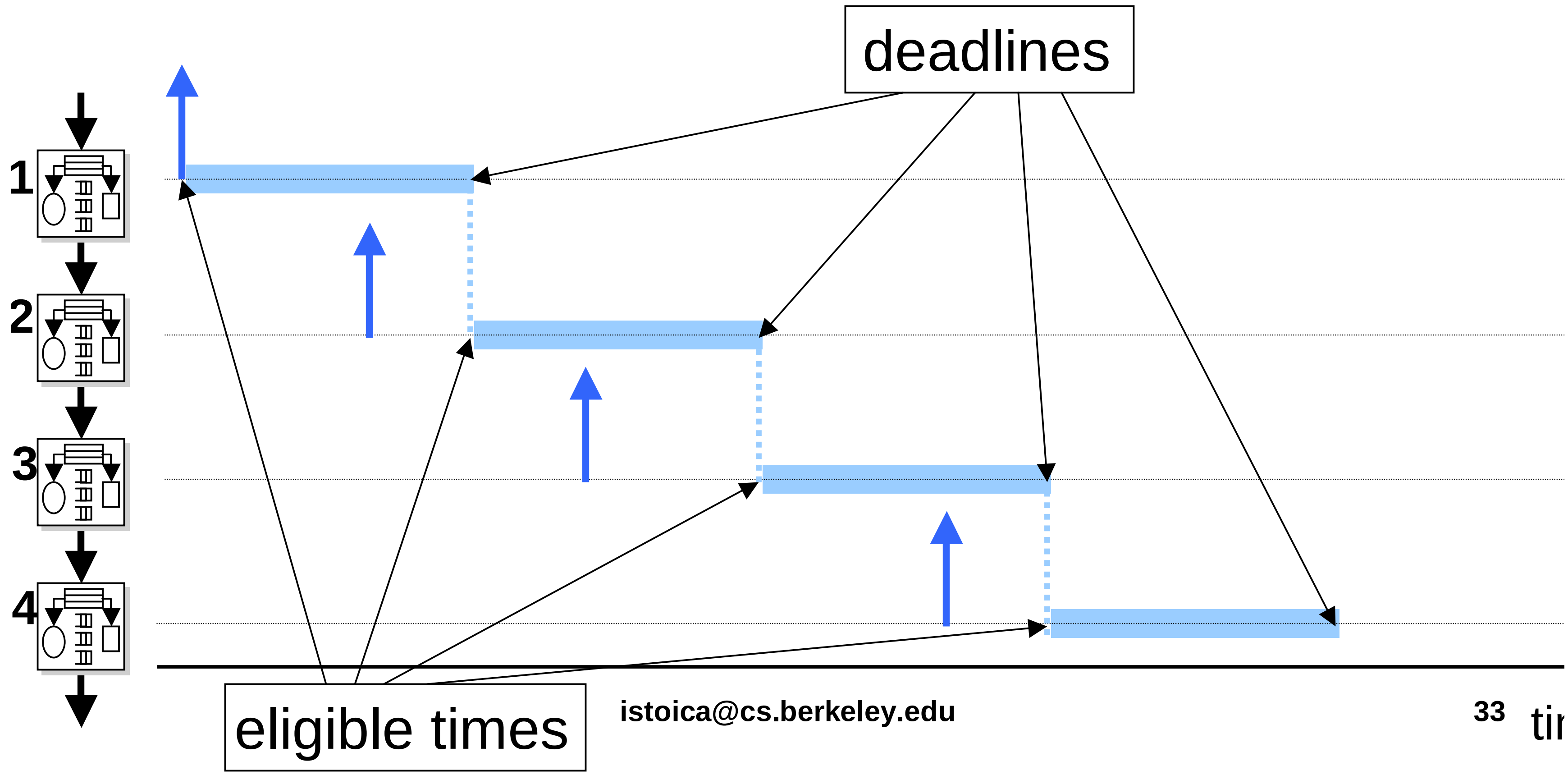
With each packet associate

- eligible time – start time of serving packet in ideal mod
- deadline – finish time of serving packet in ideal model



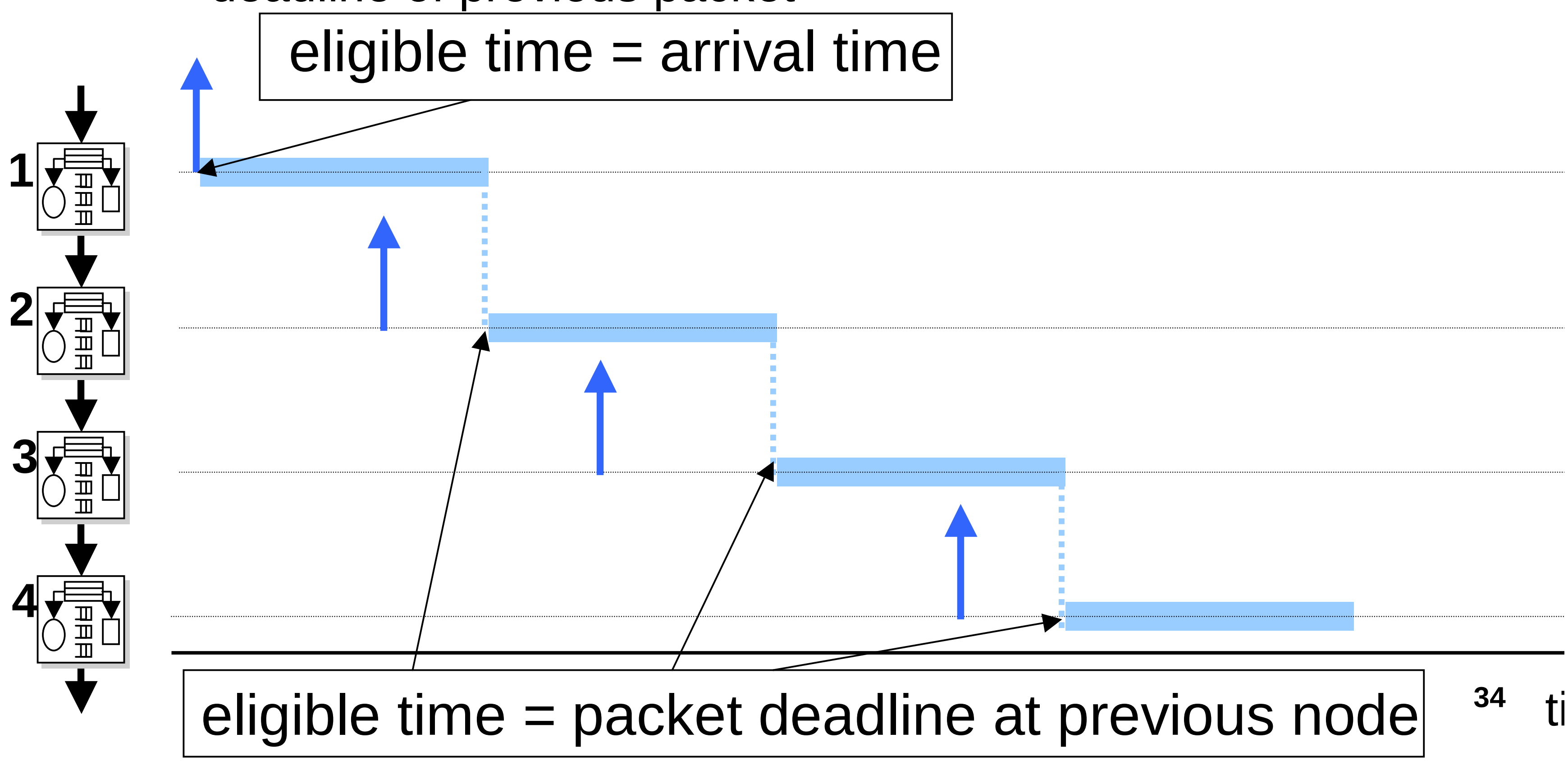
Jitter-VC

- lgorithm: schedule eligible packets in increasing order c
heir deadlines
- roperty: guarantees that all packets meet their deadlines



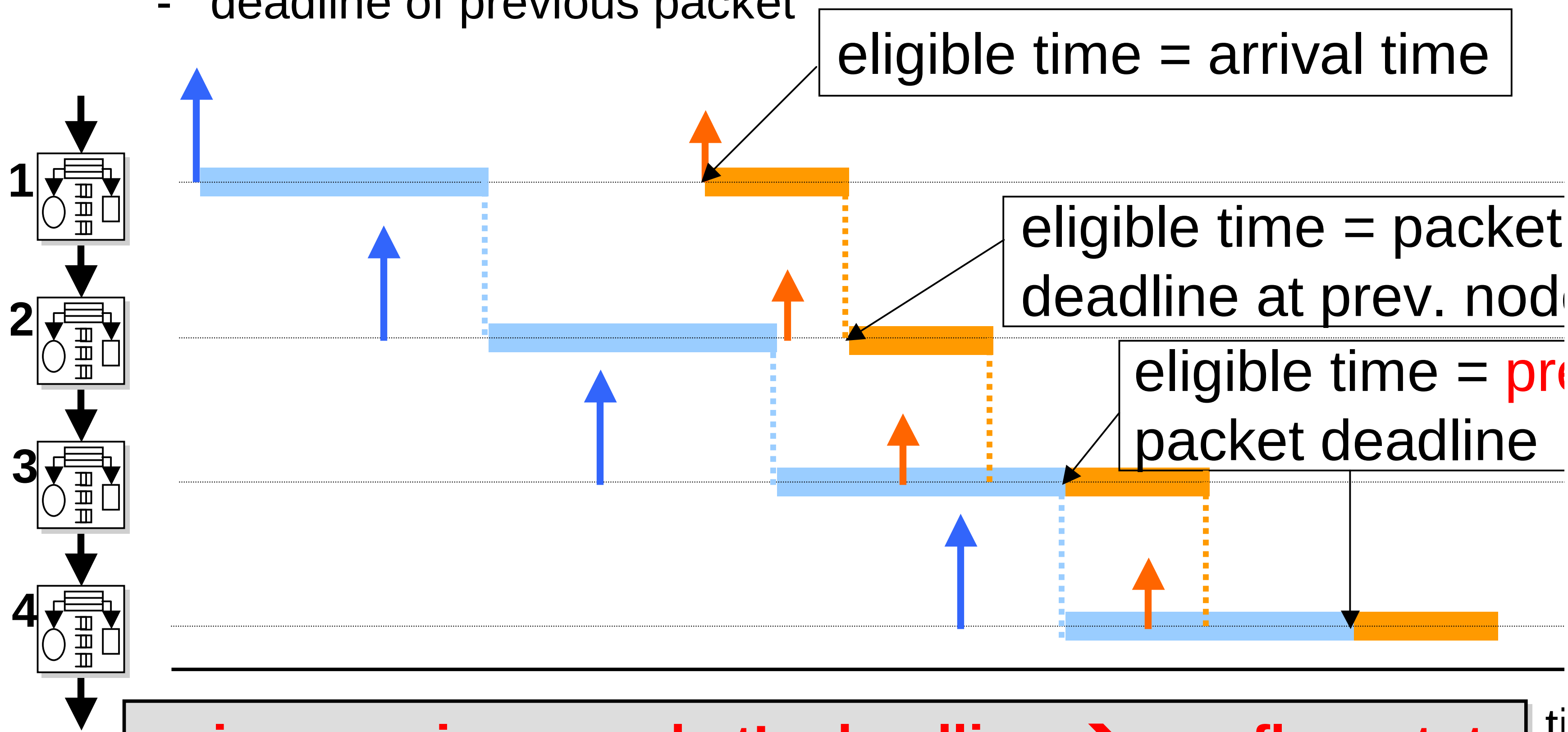
Jitter-VC: Eligible Time Computation

- Minimum between
  - arrival time
  - deadline at previous node + propagation delay
  - deadline of previous packet



Jitter-VC: Eligible Time Computation

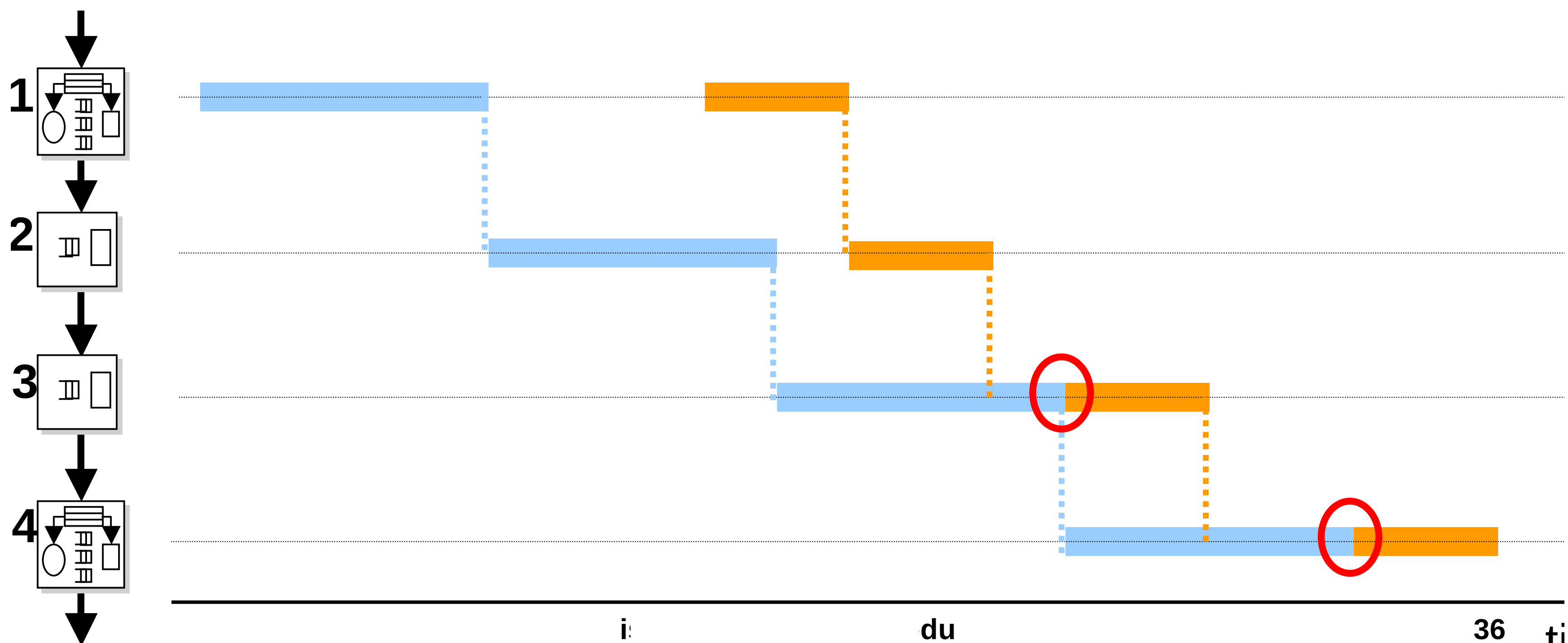
- Minimum between
  - arrival time
  - deadline at previous node + propagation delay
  - deadline of previous packet



using previous packet's deadline → per flow state

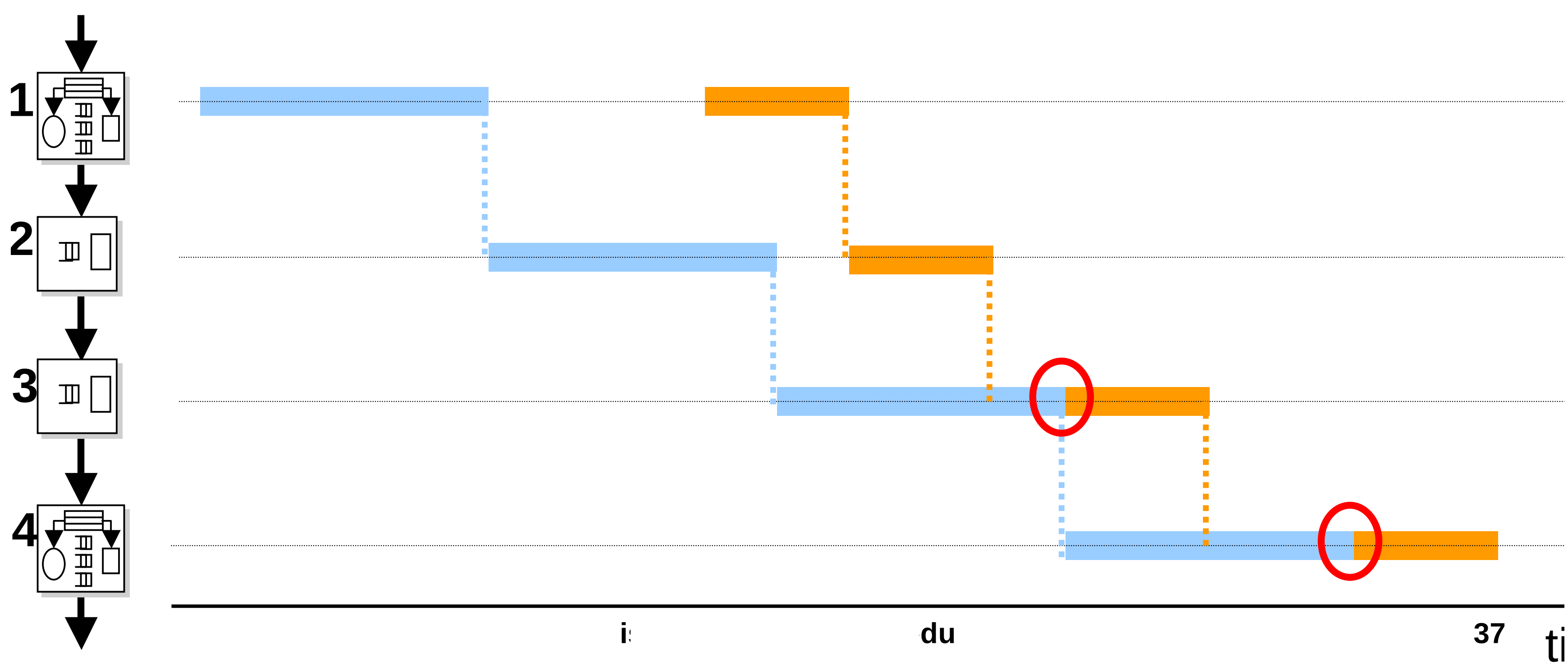
Stateless Solution: Core-Jitter Virtual Clock (CJVC)

- Goal: eliminate per-flow state
  - eliminate dependency on previous packet deadline



Core-Jitter Virtual Clock (CJVC)

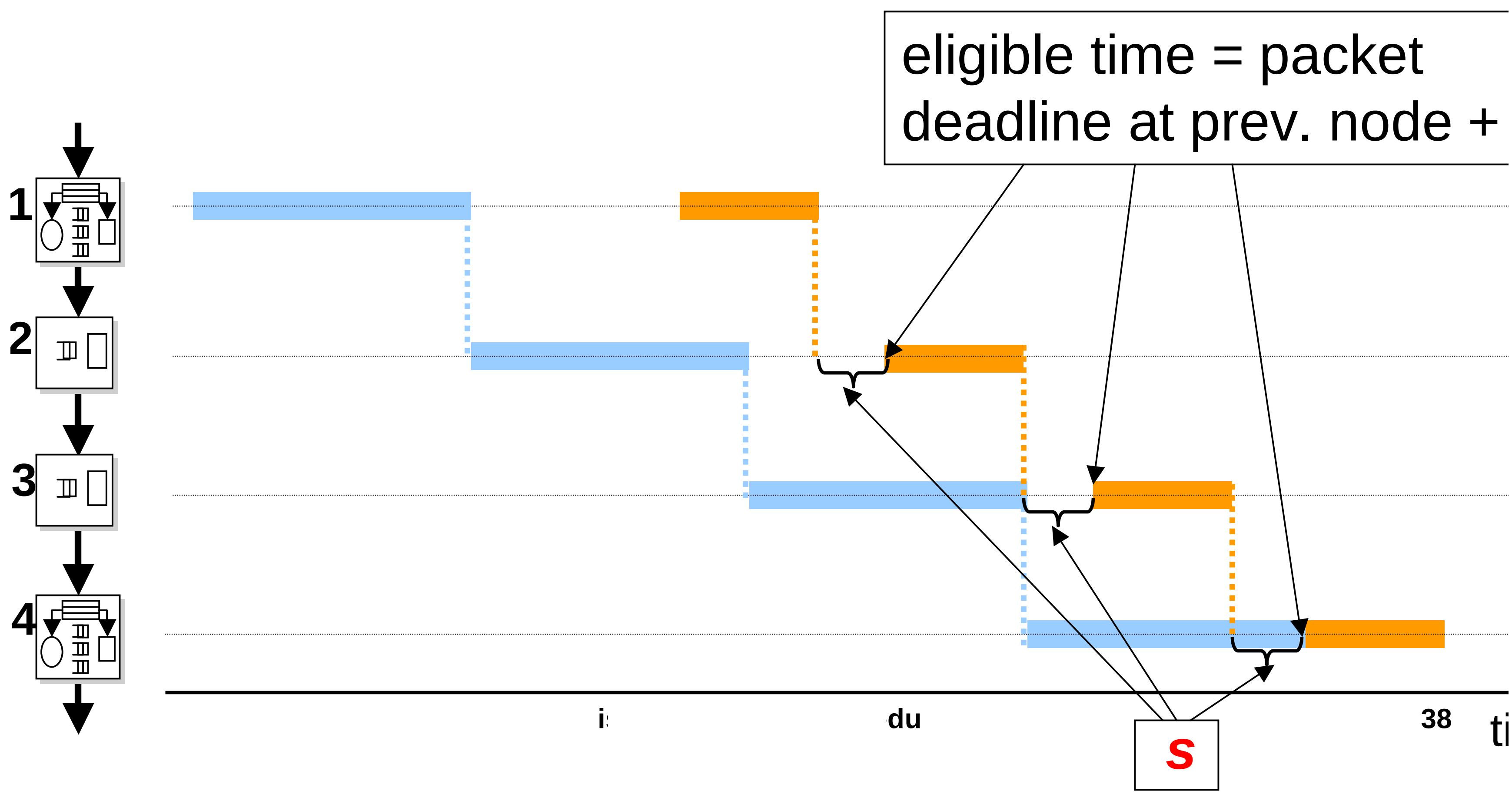
- Solution: *make* eligible time greater or equal to previous packet deadline



Core-Jitter Virtual Clock (CJVC)

How: associate to each packet a slack variable *s*

Delay eligible time at each node by *s*



CJVC Properties

Theorem: CJVC and Jitter-VC provide the same end-to-end delay bounds

s can be computed at ingress: depends on

- current and previous packet eligible times (e and e_p)
- current and previous packet lengths (l_p and l)
- slack variable associated to previous packet (s_p)
- flow reservation (r)
- number of hops (h) – computed at admission time

$$s = \max\left(0, s_p + \frac{l_p - l}{r} + \frac{e_p - e + l_p / r}{h - 1}\right)$$

CJVC Algorithm

Each packet carries in its header three variable

- slack variable s (computed and inserted by ingress)
- flow's reserved rate r (inserted by ingress)
- ahead of schedule a (inserted by previous node)

Eligible time = arrival time + a + s

Deadline = eligible time + (packet length) / r

NOTE:

- using a instead of the deadline at previous node → no need for synchronized clocks

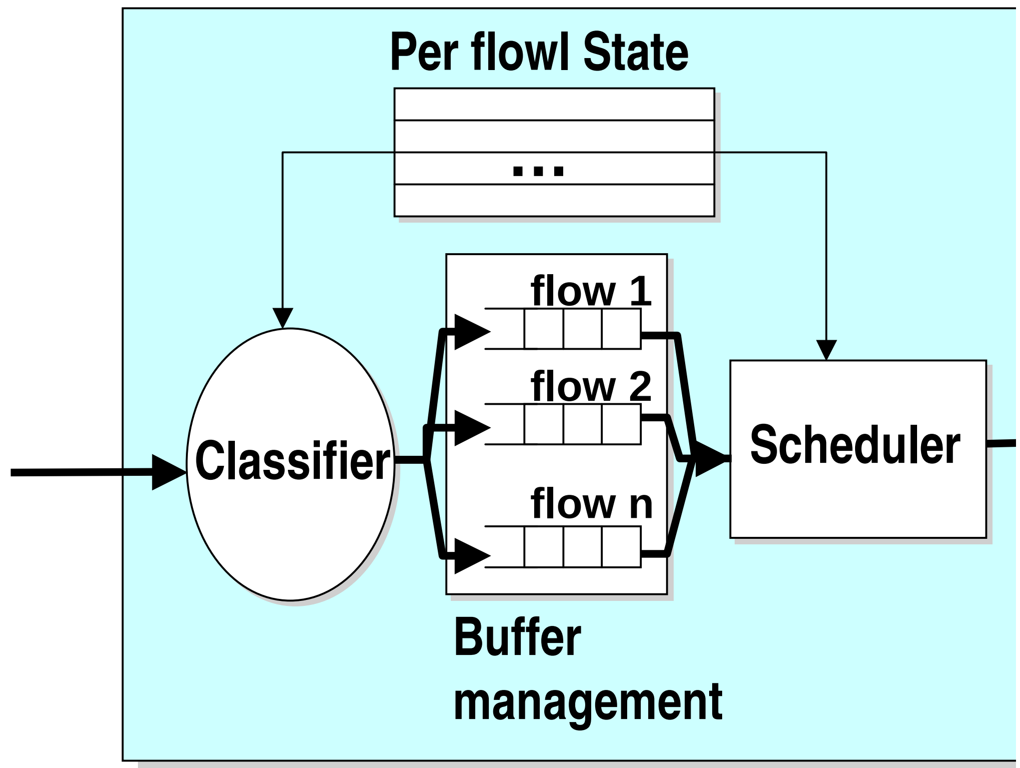
Jitter-VC: Core Router

Data path

- Per-flow classification
- Per-flow buffer management
- Per-flow scheduling

Control path

- install and maintain per-flow state for data and control paths



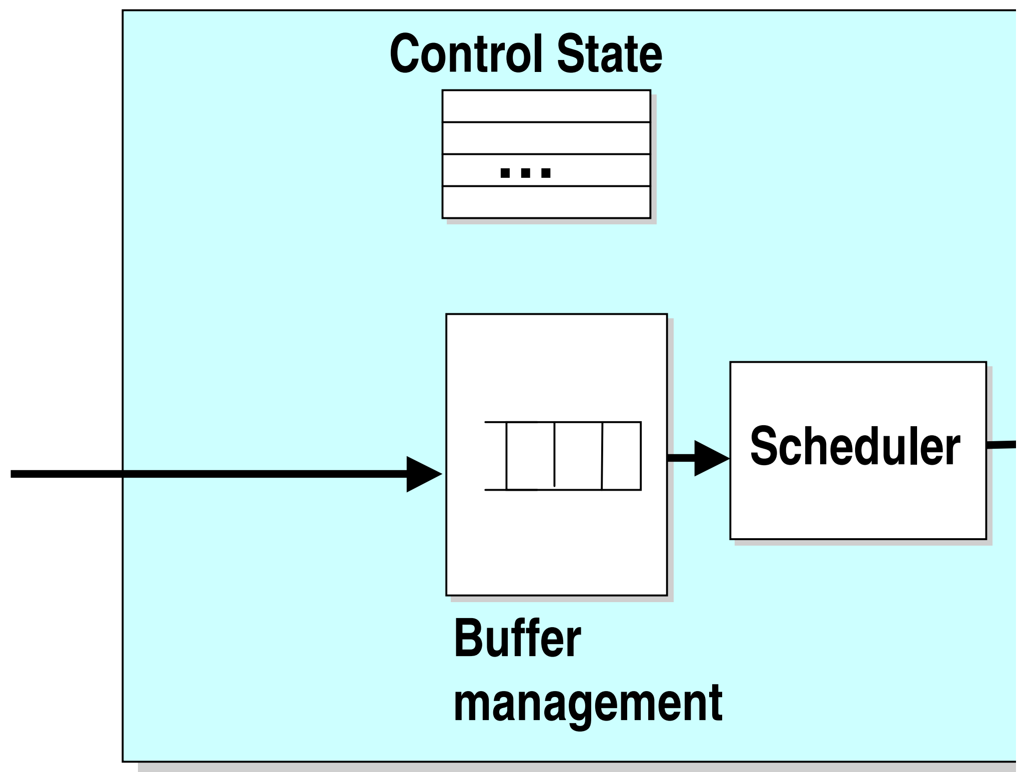
CJVC: Core Router

Data path

- ~~Per-flow classification~~
- ~~Per-flow~~ buffer management
- Per-packet scheduling

Control path

- Install and maintain per-flow state for ~~data and~~ control paths

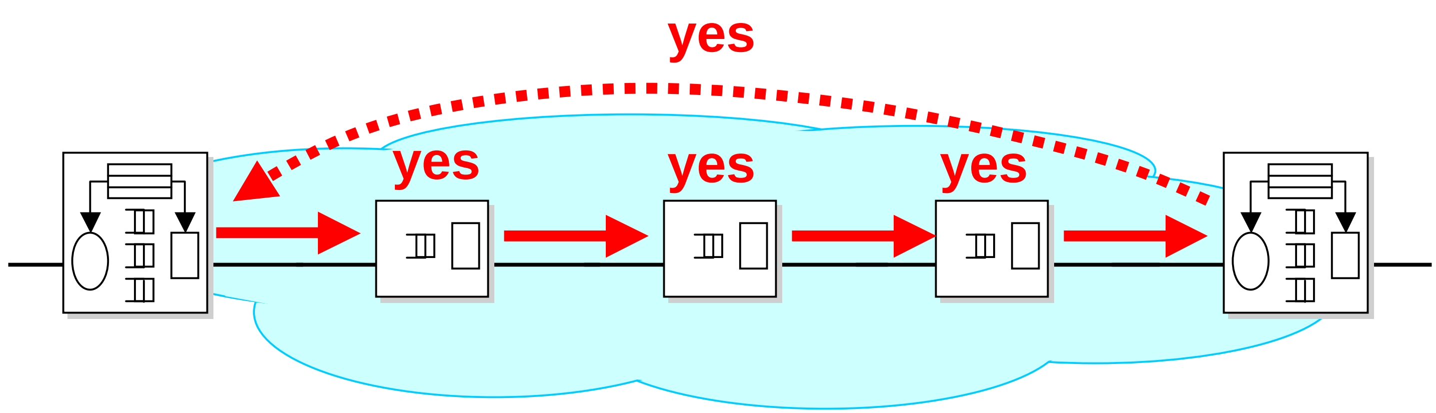


Outline

- Motivations: what is the problem and why it is important?
- Existing solutions
- Solution: SCORE architecture and DPS technique
- *Example: providing guaranteed services*
  - Eliminate per-flow state on data path
  - ➡ *Eliminate per-flow state on control path*
  - Implementation and experimental results
- Conclusions

Control Path: Admission Control

- Goal: reserve resources (bandwidth) for each flow along its path
- Approach: light-weight protocol that does not require core nodes to maintain per-flow state



Per-hop Admission Control

- A node admits a reservation r, if $r \leq C - R$
  - C – output link capacity
  - R – aggregate reservation: $R = \sum_i r_i$
- Need: maintain aggregate reservation R
- Problem: it requires per flow state to handle partial reservation failures and message loss

Solution

1. Estimate aggregate reservation R_{est}
2. Account for approximations and compute an upper bound R_{bound} , i.e., $R_{bound} >= R$
3. Use R_{bound} , instead of R, to perform admission control, i.e., admit a reservation r if

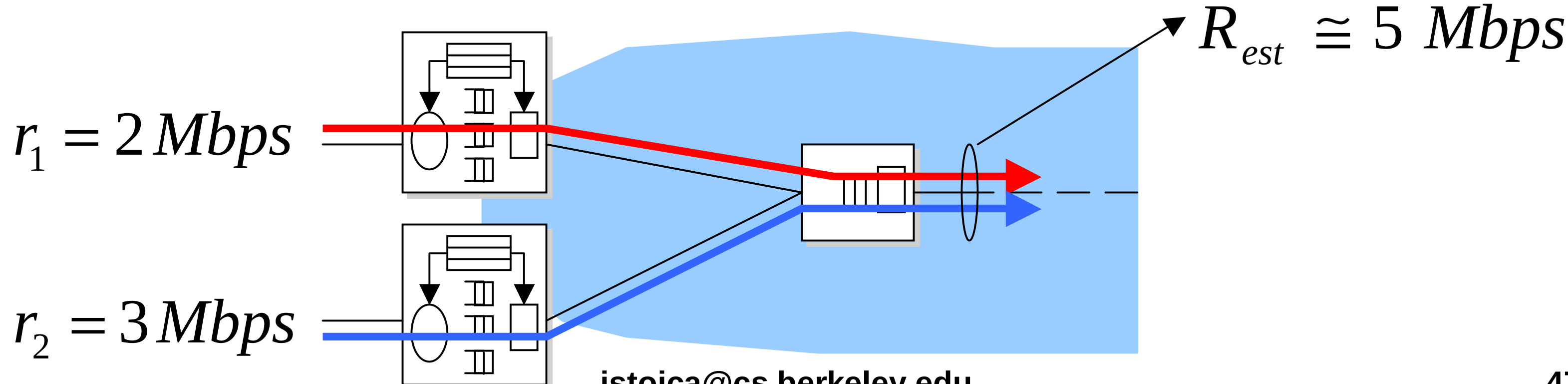
$$r \leq C - R_{bound}$$

Estimating Aggregate Reservation (R_{est})

- Observation: If all flows were sending at their reserved rates, computing R_{est} is trivial:
  - just measure the traffic throughput, e.g.,

$$R_{est} = \frac{\sum_{i \in S(a,a+T)} length(i)}{T}$$

where $S(a, a+T)$ contains all packets of all flows received during $[a, a+T)$



Virtual Length

Problem: What if flows do not send at their reserved rates ?

Virtual Length

Problem: What if flows do not send at their reserved rates ?

Solution: associate to each packet a virtual length such that

- *if lengths of all packets of a flow were equal to their virtual lengths, the flow sends at its reserved rate*

Then, use virtual lengths instead of actual packet lengths to compute R_{est}

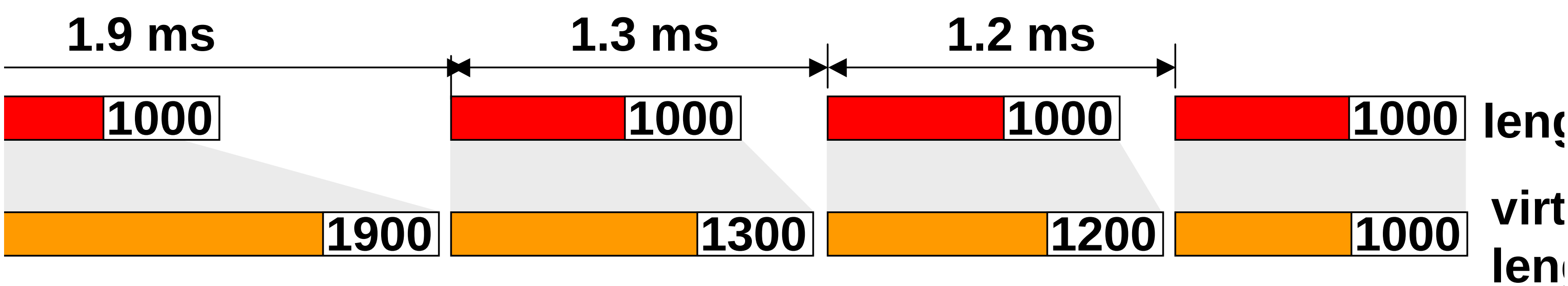
Virtual Length

Definition:

$$virtualLength = r \times (crt\_time - prev\_time)$$

- r – flow reserved rate
- crt_time – transmission time of current packet
- $prev\_time$ – transmission time of previous packet

Example: assume a flow with reservation r = 1 Mbps sending 1000 bit packets



Estimating Aggregate Reservation (R_{est})

- Use Dynamic Packet State (DPS)
- Ingress node: upon each packet departure computes the virtual length and inserts it in the packet header
- Core node: Estimate R_{est} on each output link as

$$R_{est} = \frac{\sum_{i \in S(a, a+T)} virtualLength(i)}{T}$$

  - where $S(a, a+T)$ contains of all packets of all flows received during $[a, a+T)$

Aggregate Reservation Estimation: Discussion

The estimation algorithm is robust in presence of control message loss and duplication

- their effect is “forgotten” after one estimation interval

f no packet of a flow departs during a predefined interval (i.e. maximum inter-departure time), ingress node generates a dummy packet

Utilization <= 1 – f ,

- where f = (max. inter-departure time) / (estimation int.)
- e.g.: max. inter-departure time = 5s; estimation int. = 30s → utilization 0.83

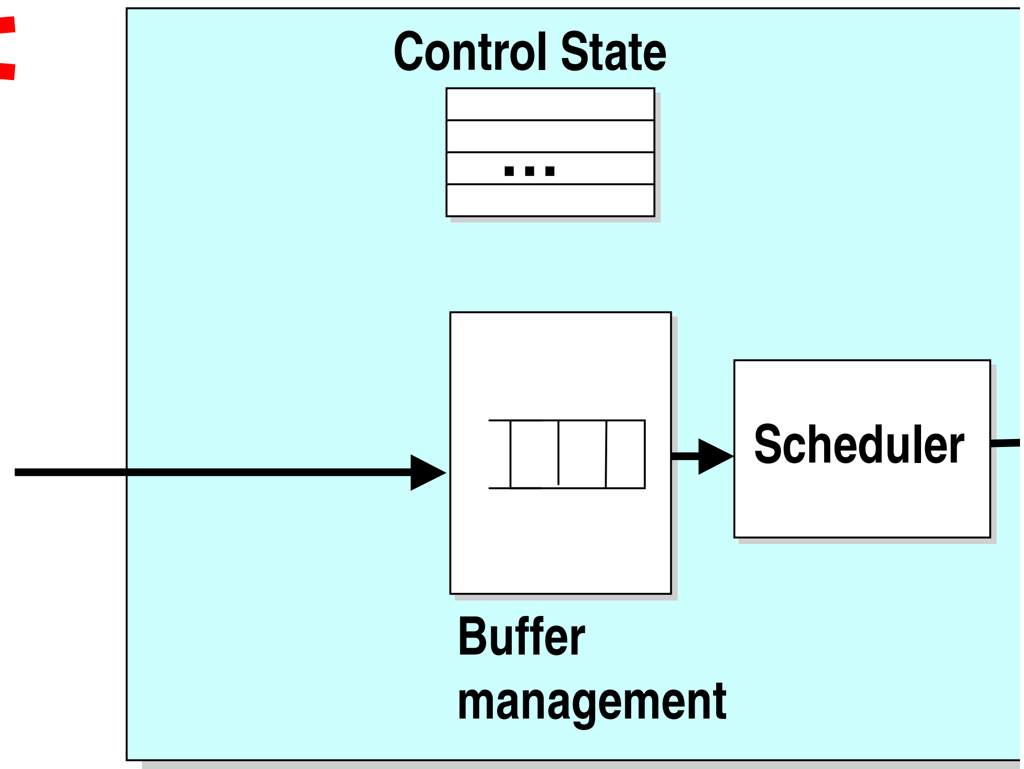
Core Router

Data path

- Per-flow classification
- Per-flow buffer management
- Per-packet scheduling

Control path

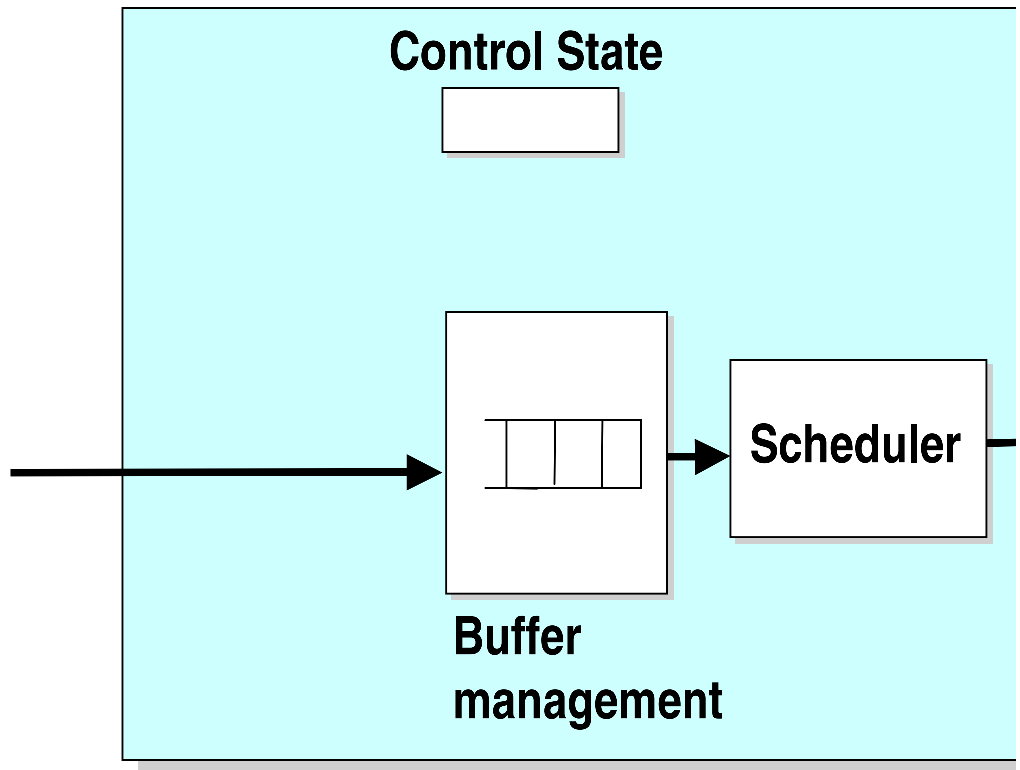
- Install and maintain per flow state for data and control paths



Core Router

Data path
- ~~Per-flow classification~~
- ~~Per-flow~~ buffer management
- Per-packet scheduling

Control path
- ~~Install and maintain per-flow state for data and control paths~~



no need to maintain consistency of per-flow state

Outline

- Motivations: what is the problem and why is it important?
- Existing solutions
- Solution: SCORE architecture and DPS technique
- *Example: providing guaranteed services*
  - Eliminate per-flow state on data path
  - Eliminate per-flow state on control path
  - ➡ *Implementation and experimental results*
- Conclusions

Implementation: State Encoding

- Problem: Where to insert the state ?
- Possible solutions:
  - between link layer and network layer headers
  - as an IP option (IP option 23 allocated by IANA)
  - find room in IP header

Implementation: State Encoding

- Current solution
  - 4 bits in DS field (belong to former TOS)
  - 13 bits by reusing fragment offset
- Encoding techniques
  - Take advantage of implicit dependencies between state values
  - Temporal multiplexing: use one field to encode two states, if these states do not need to be simultaneously presented in each packet

Implementation

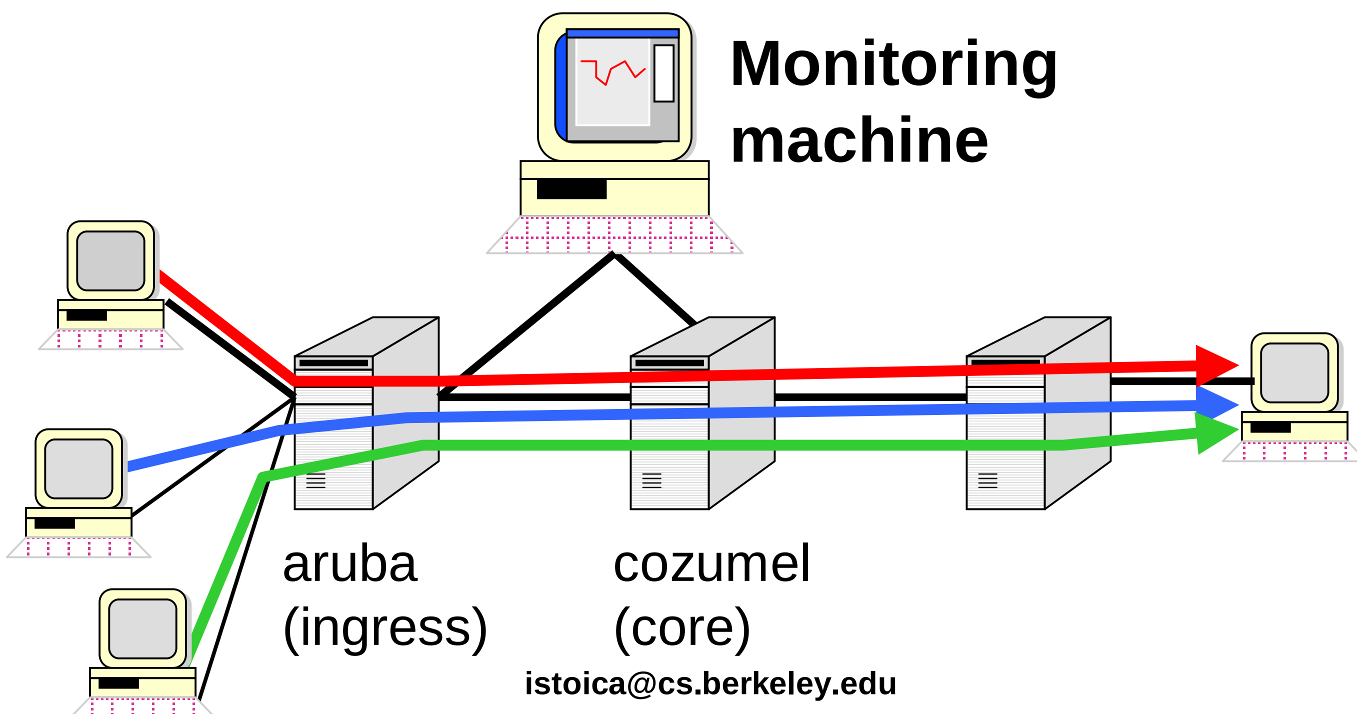
- FreeBSD 2.2.6
- Pentium II 400 MHz
- ZNYX network cards 10/100 Mbps Ethernet
- Fully implements control and data path functionalities
- Management and monitoring infrastructure

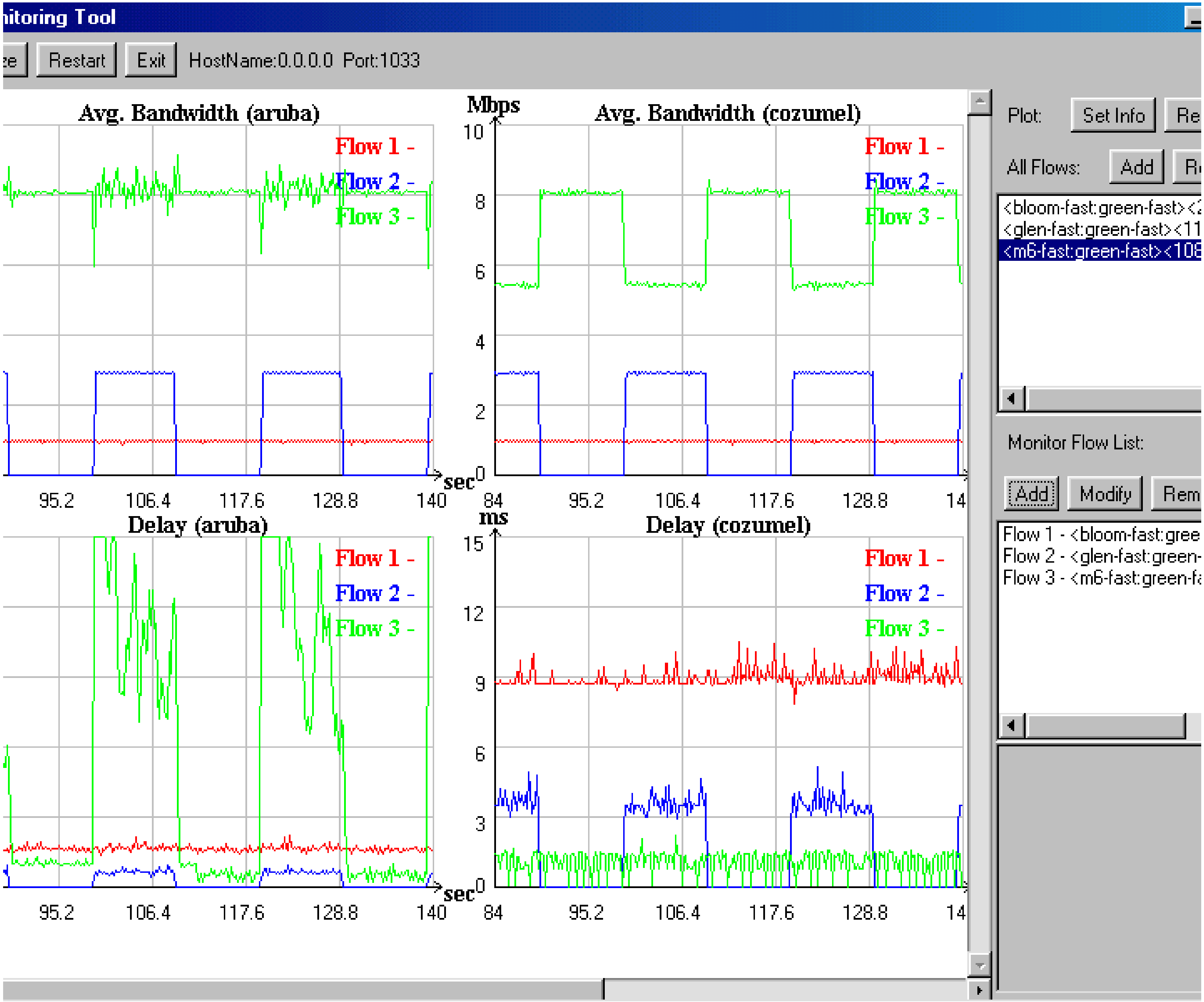
Monitoring Infrastructure

- Light weight mechanism that allows continuous monitoring at packet level
- Implementation
  - Record each packet (28 bytes)
    - IP header and port numbers
    - arrival, departure or drop times
  - Use raw IP to send this information to a monitoring site

A Simple Experiment

- Three flows sharing a 10 Mbps link
  - Flow 1: 1 Mbps reservation
  - Flow 2: 3 Mbps reservation with ON/OFF traffic
  - Flow 3: best-effort UDP sending at > 8 Mbps



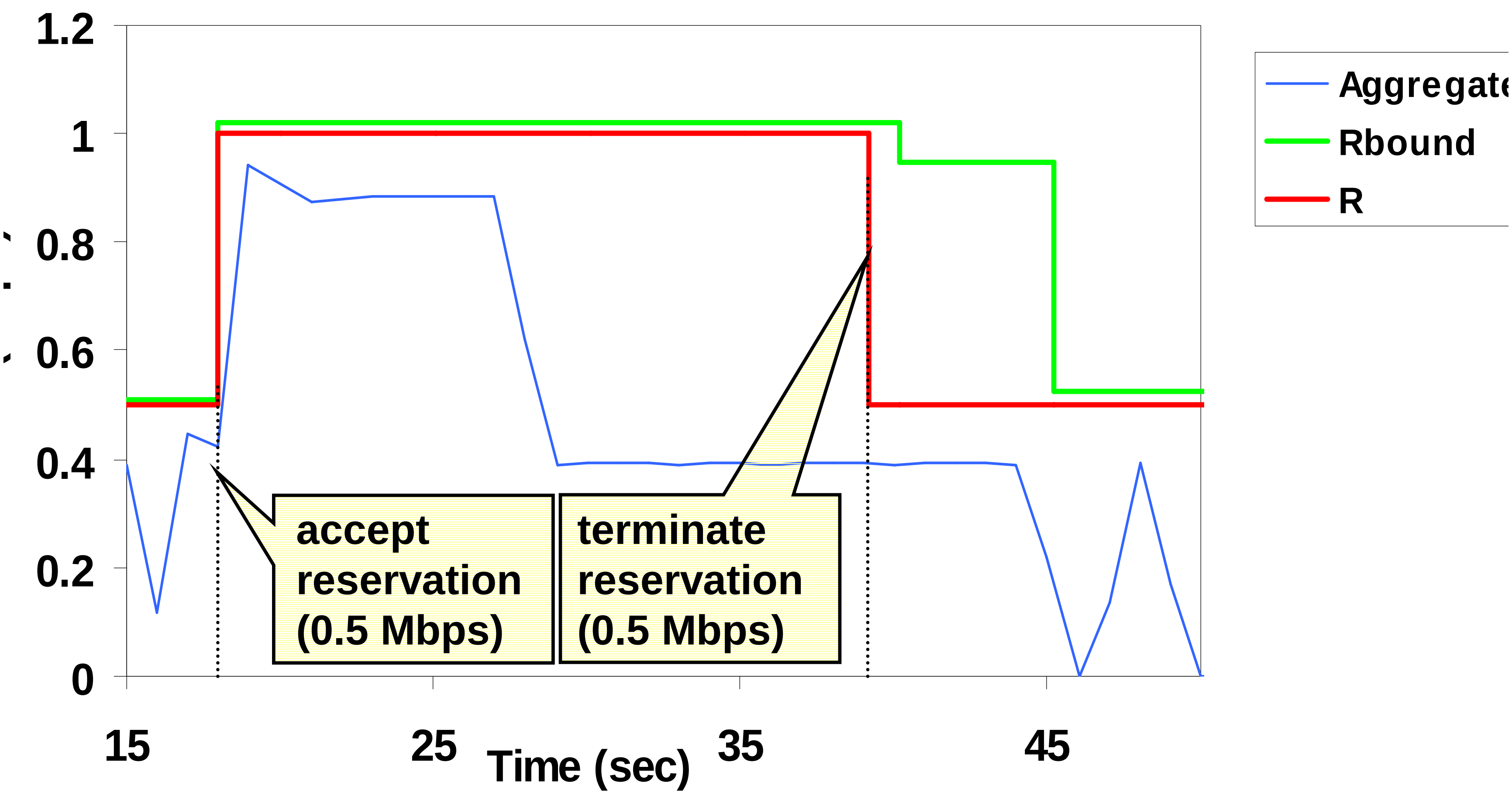


nitoring Tool
Restart
Exit
HostName:0.0.0.0 Port:1033
Avg. Bandwidth (aruba)
Avg. Bandwidth (cozumel)
Mbps
Flow 1 -
Flow 2 -
Flow 3 -
sec
95.2
106.4
117.6
128.8
140
84
Delay (aruba)
Delay (cozumel)
ms
Plot:
Set Info
All Flows:
Add
<bloom-fast:green-fast>
<glen-fast:green-fast>
<m6-fast:green-fast>
Monitor Flow List:
Modify
Flow 1 - <bloom-fast:gree
Flow 2 - <glen-fast:green-
Flow 3 - <m6-fast:green-fa

Aggregate Reservation Computation

0.5 Mbps reservation active during entire interval

0.5 Mbps reservation starting at 18 sec; ending at 39 sec

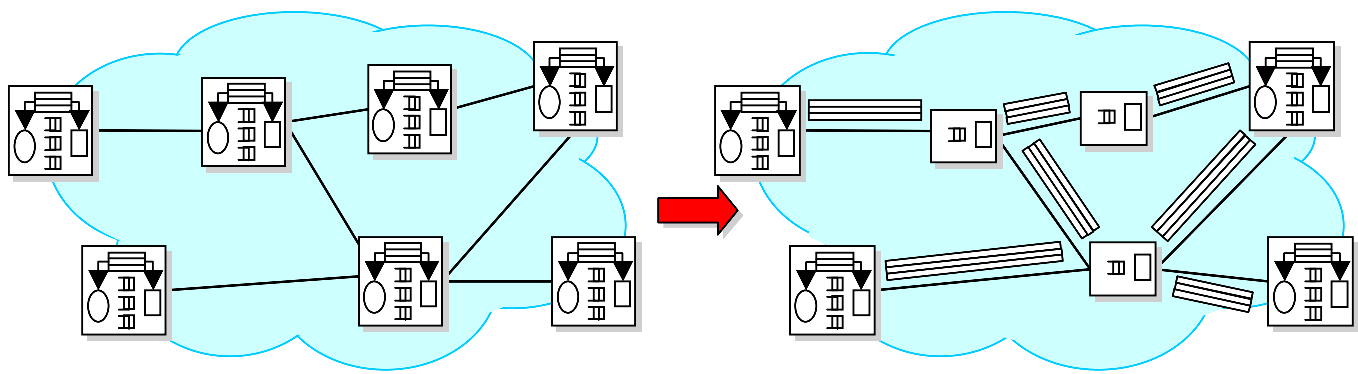


Conclusions

Propose a network architecture (SCORE) and a technique (DPS) that bridge longstanding gap between stateless and stateful solutions

Key ideas

- Instead of core routers maintain per-flow state have packets carry this state
- Use state to coordinate edge and core router actions



Reference Stateful Network   SCORE Network

Conclusions

Develop first scalable solutions to provide:

- Service guarantees
- Network support for congestion control
- Service differentiation

DPS compatible with Diffserv: can greatly enhances the functionality while requiring minimal changes