Receiver-Based Flow Control for Networks in Overload

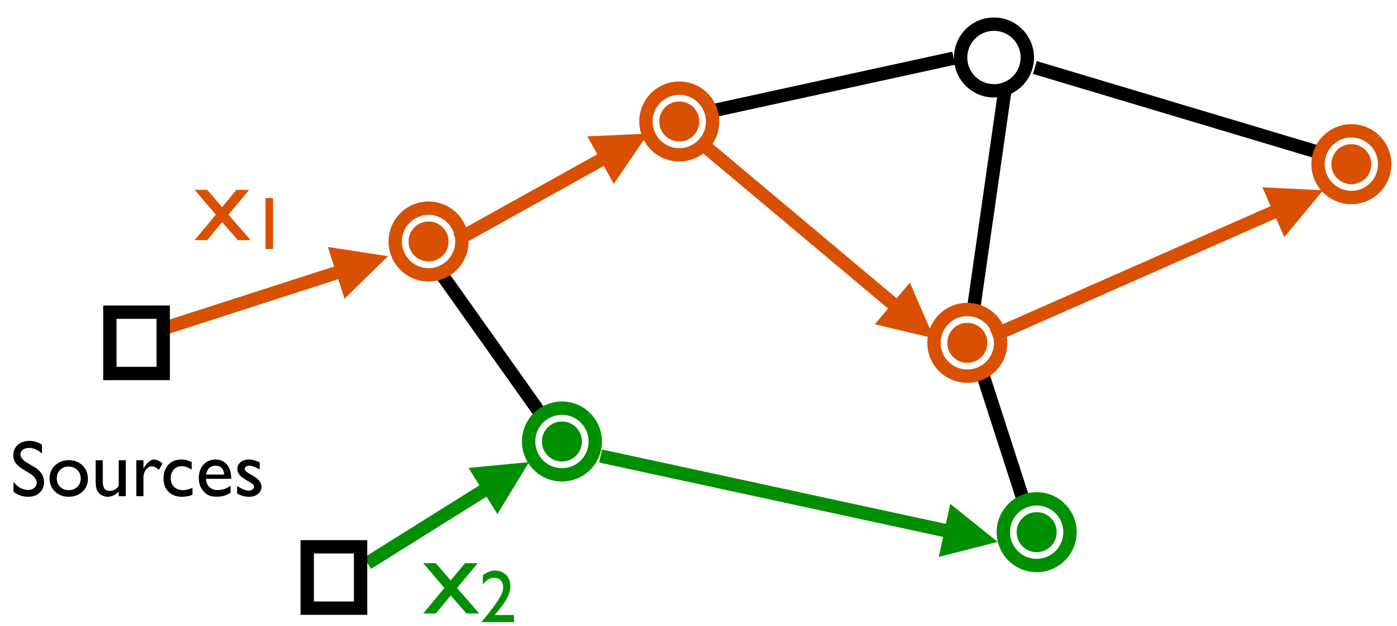
Chih-ping Li
Massachusetts Institute of Technology

joint work with Eytan Modiano (MIT)

IEEE INFOCOM   April 18, 2013

Traditional Flow Control

- Sources adjust transmission rates in response to network congestion
  - TCP flow control
  - Network utility maximization problems

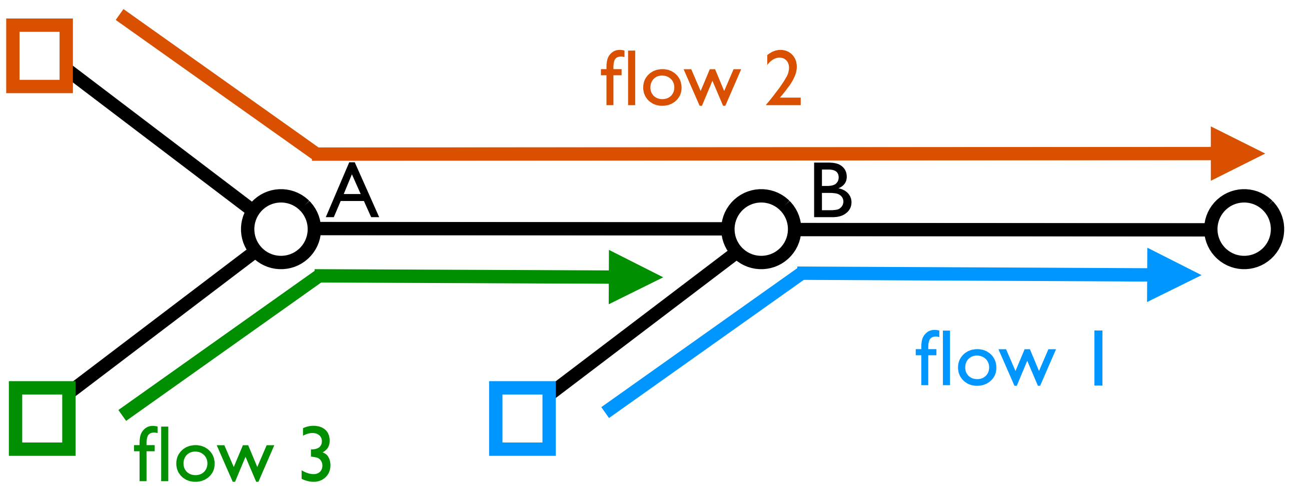


maximize: $\sum U_n(x_n)$

subject to: $(x_n) \in \Lambda$

Not All Sources Perform Flow Control

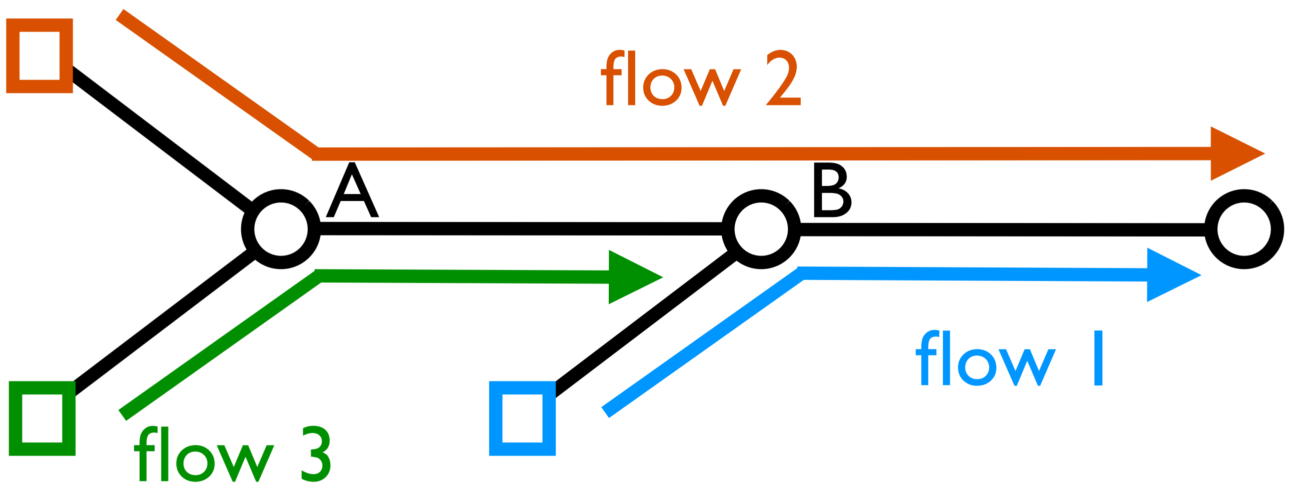
- Greedy users
- Malicious users launch attacks to crash prominent websites
- UDP-based flows



If flow 2 overloads the network, then flows 1 & 3 are adversely affected

Not All Sources Perform Flow Control

- Greedy users
- Malicious users launch attacks to crash prominent websites
- UDP-based flows

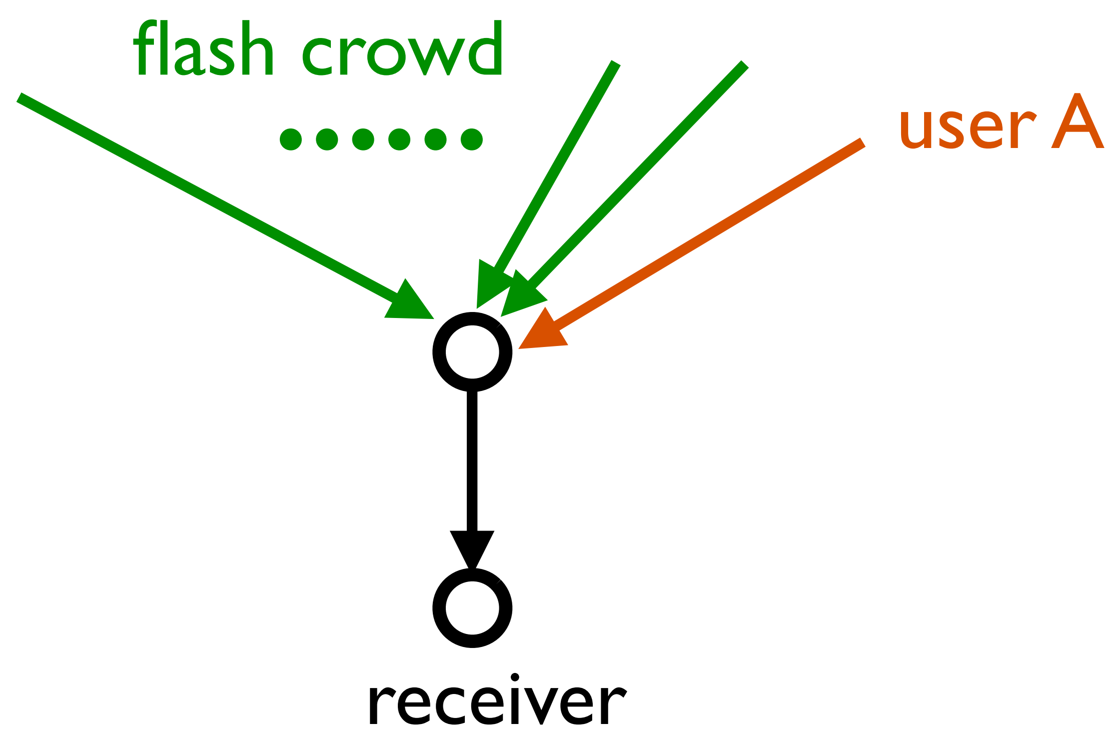


If flow 2 overloads the network, then flows 1 & 3 are adversely affected

Task 1:
In-network packet dropping by all intermediate network nodes to optimize per-flow throughput without source cooperation

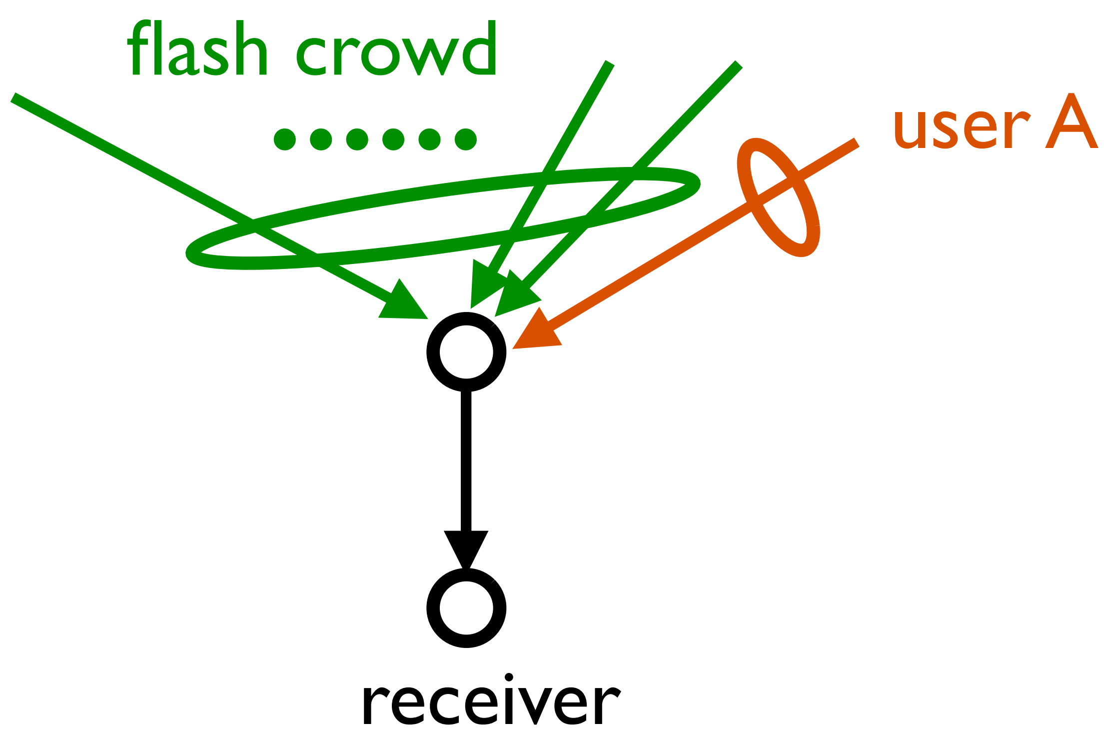
When Source-Based Flow Control is Ineffective

- Flash crowd - lots of small flows yield congestion close to a web server (e.g., news websites experience high delay after the 911 attack)



When Source-Based Flow Control is Ineffective

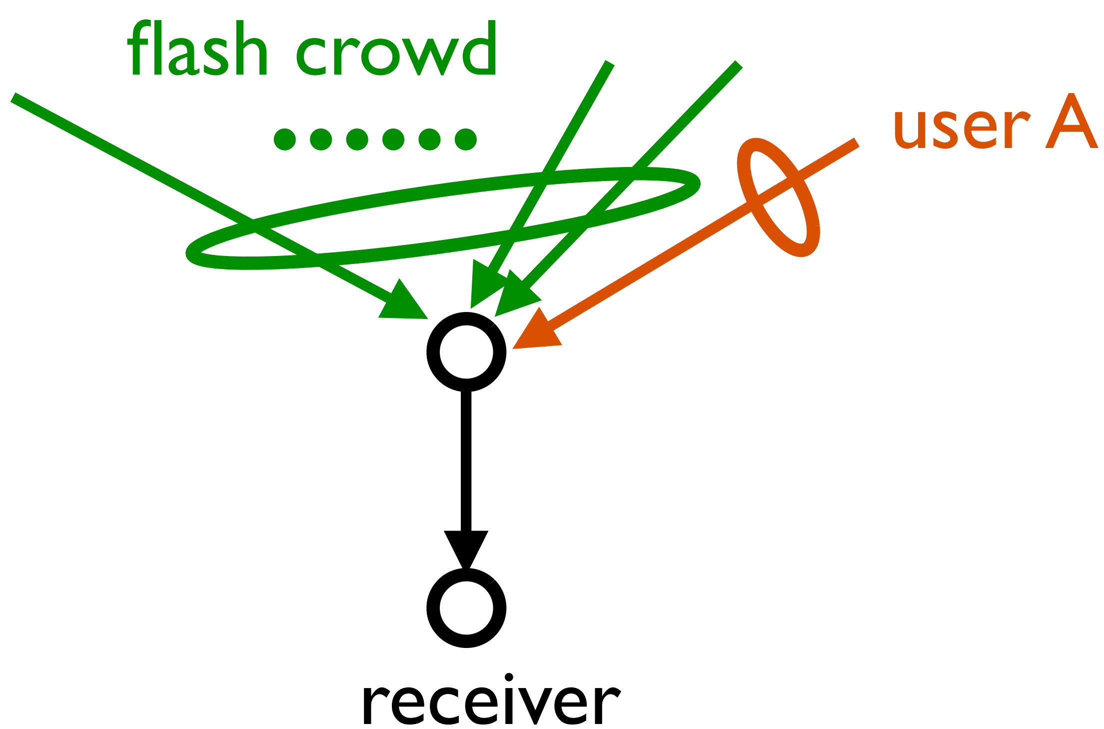
- Flash crowd - lots of small flows yield congestion close to a web server (e.g., news websites experience high delay after the 911 attack)



Task 2:
Let the receiver control the aggregate rate of a class of flows without source cooperation

When Source-Based Flow Control is Ineffective

- Flash crowd - lots of small flows yield congestion close to a web server (e.g., news websites experience high delay after the 911 attack)

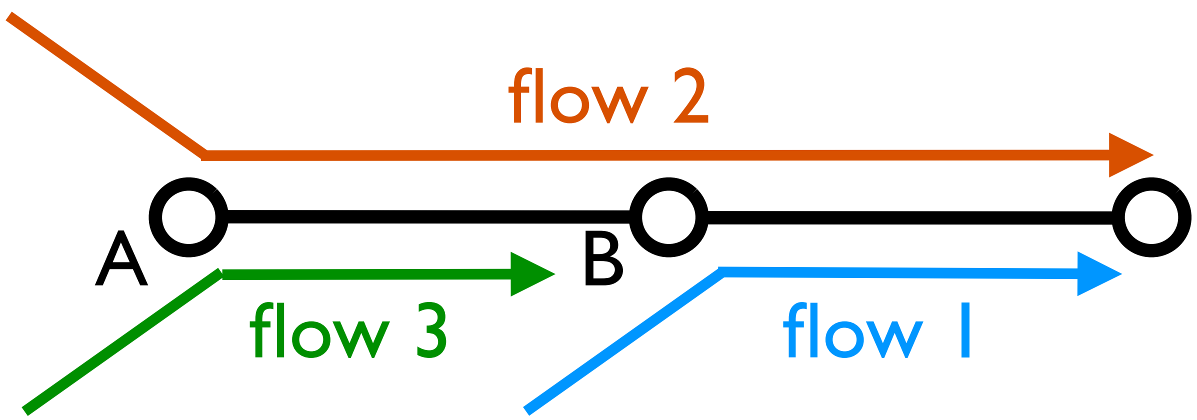


Task 2:
Let the receiver control the aggregate rate of a class of flows without source cooperation

Provide fairness (differentiated services) over classes of flows, instead of individual flows

Task I: Maximizing weighted sum throughput by in-network packet dropping

Maximum Weighted Sum Throughput



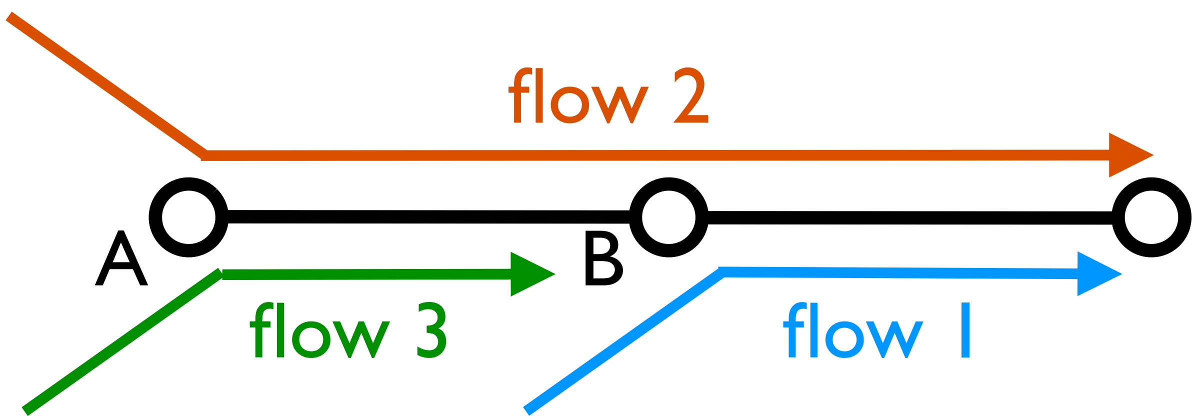
Unit capacity links
Arrival rates (2, 2, 2)

r_i - long-term throughput of flow i

Maximize: 3 r_1+ 2 r_2+ 1 r_3 ⟶ Optimal throughput (1, 0, 1)

B serves flow 1, A serves flow 3

Maximum Weighted Sum Throughput



Unit capacity links
Arrival rates (2, 2, 2)

r_i - long-term throughput of flow i

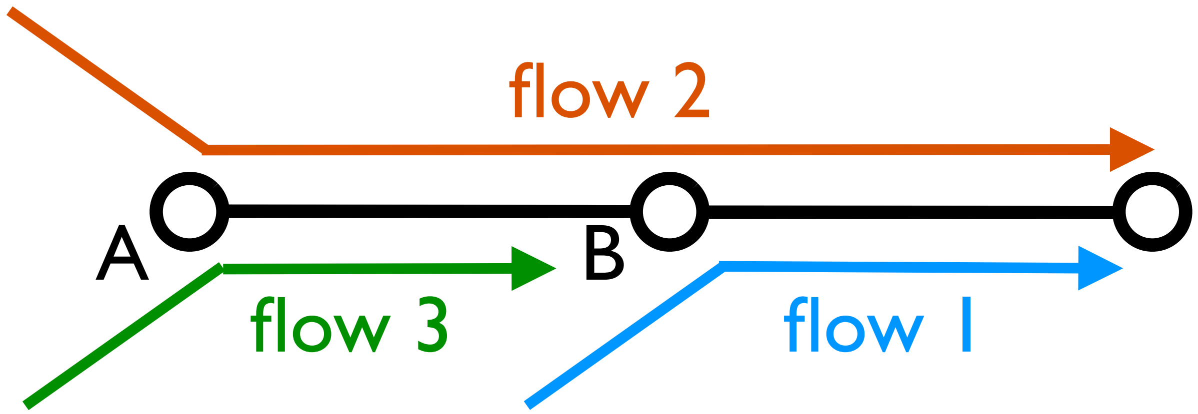
Maximize: $3\ r_1 + 2\ r_2 + 1\ r_3 \longrightarrow$ Optimal throughput (1, 0, 1)

B serves flow 1, A serves flow 3

Maximize: $3\ r_1 + 5\ r_2 + 1\ r_3 \longrightarrow$ Optimal throughput (0, 1, 0)

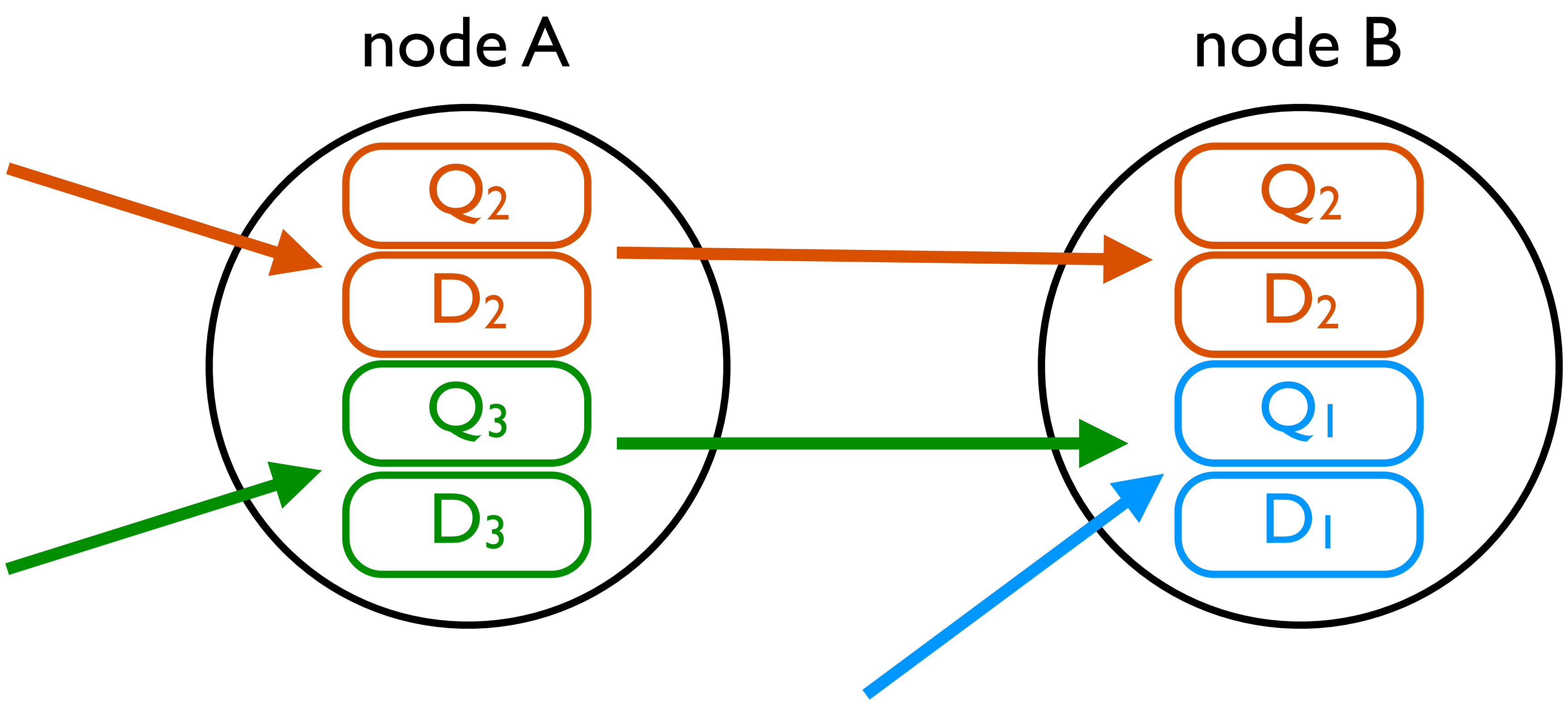
A & B serve flow 2

Threshold-Based Packet Dropping



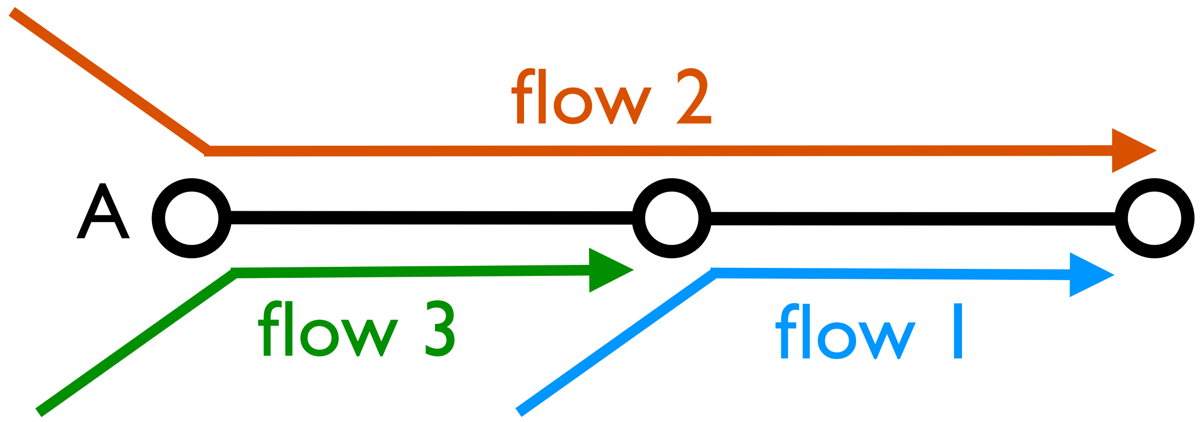
Maximize: $\theta_1 r_1 + \theta_2 r_2 + \theta_3 r_3$

Each node has a network-layer queue Q(t) and a drop queue D(t) for each flow



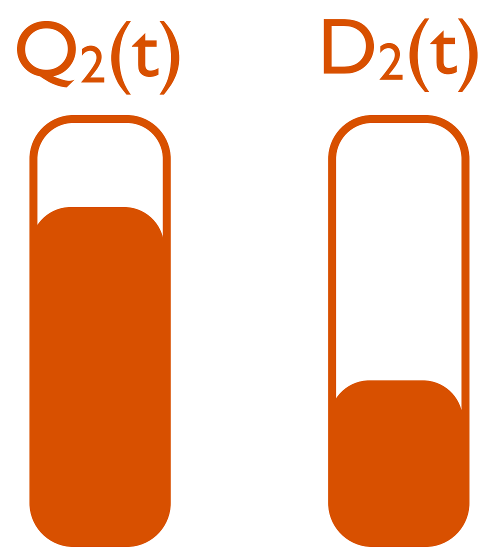
The use of D(t): directly control average packet dropping rate

Threshold-Based Packet Dropping

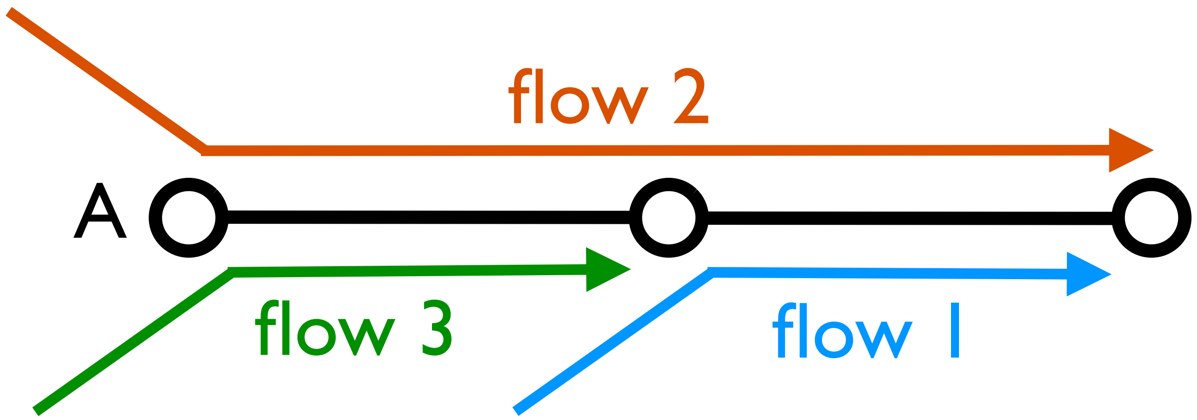


Maximize: $\theta_1 r_1 + \theta_2 r_2 + \theta_3 r_3$

Dropping policy for flow 2 at node A:



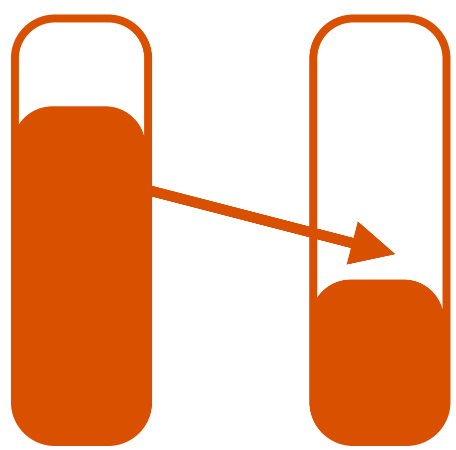
Threshold-Based Packet Dropping



Maximize: $\theta_1 r_1 + \theta_2 r_2 + \theta_3 r_3$

Dropping policy for flow 2 at node A:

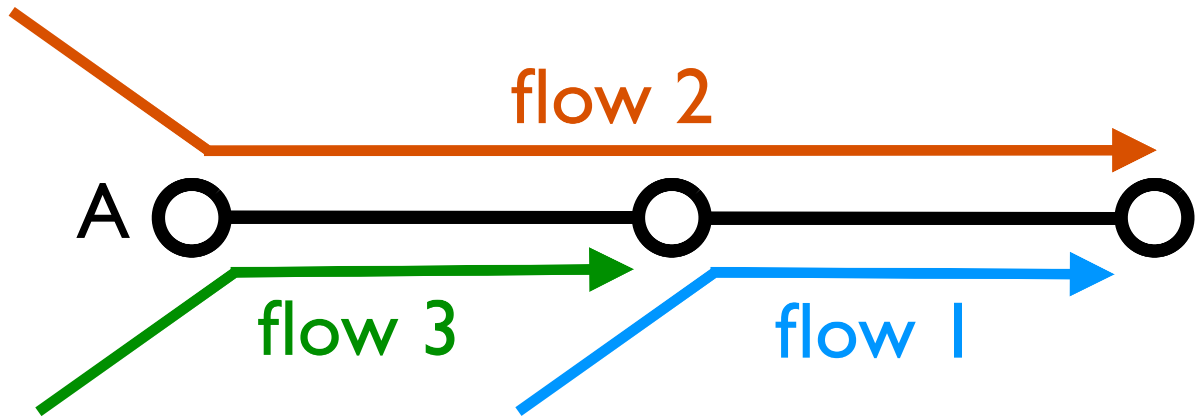
$Q_2(t)$ $D_2(t)$



drop d_{max} if $Q_2(t) > D_2(t)$

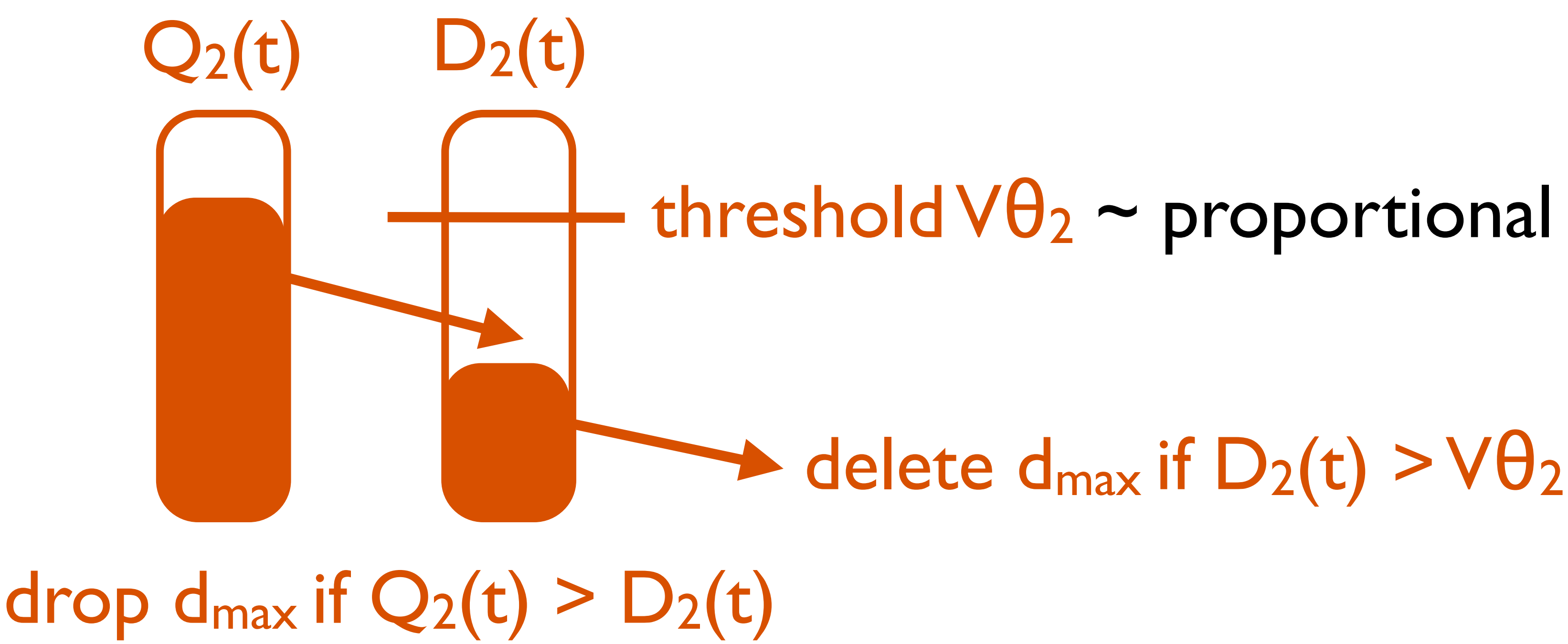
d_{max} : a finite constant guaranteeing the network can be stabilized by packet dropping

Threshold-Based Packet Dropping



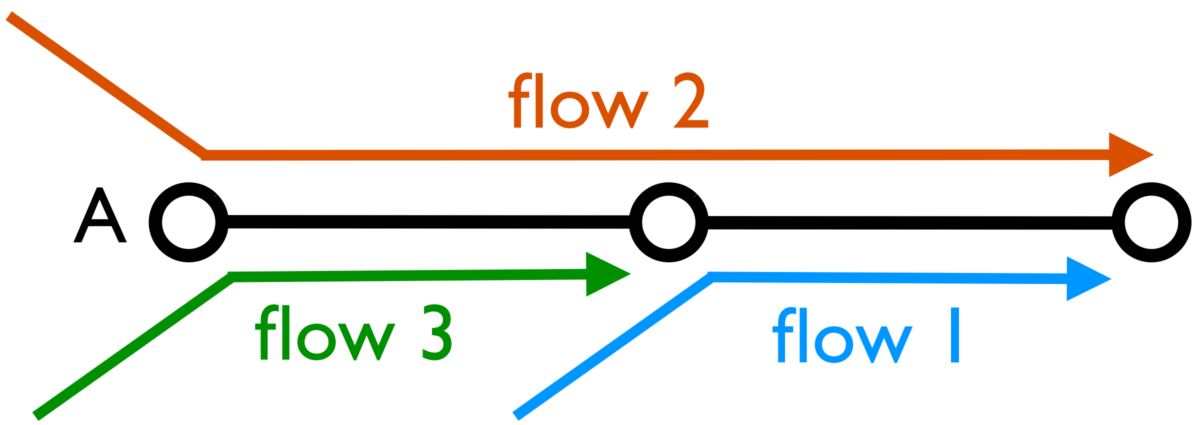
Maximize: $\theta_1 r_1 + \theta_2 r_2 + \theta_3 r_3$

Dropping policy for flow 2 at node A:

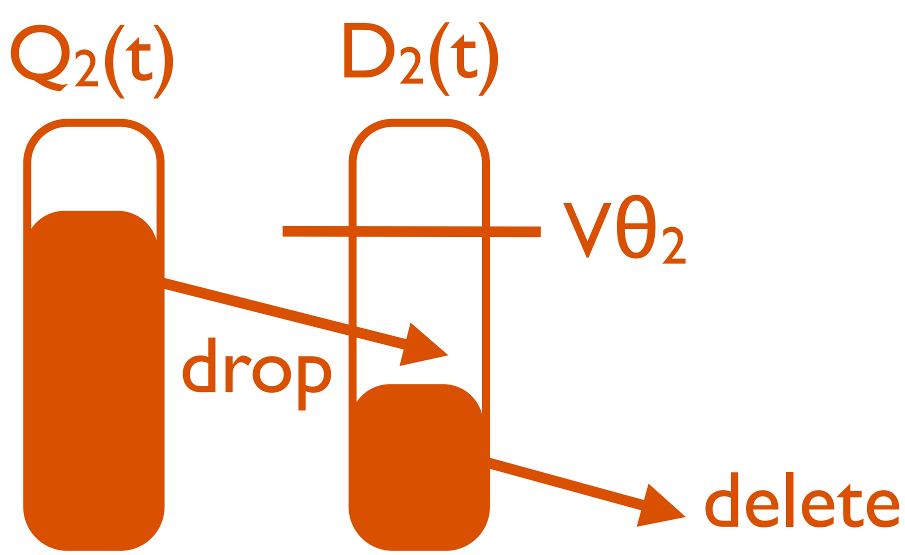


d_{max} : a finite constant guaranteeing the network can be stabilized by packet dropping

Overload Resilient Algorithm



Maximize: $\theta_1 r_1 + \theta_2 r_2 + \theta_3 r_3$



+

Back-pressure routing over queues $Q_c(t)$

Theorem:

$Q_c(t) \leq V\theta_c + 2d_{max}$ for all t

(using finite-size buffer $\propto \theta_c$)

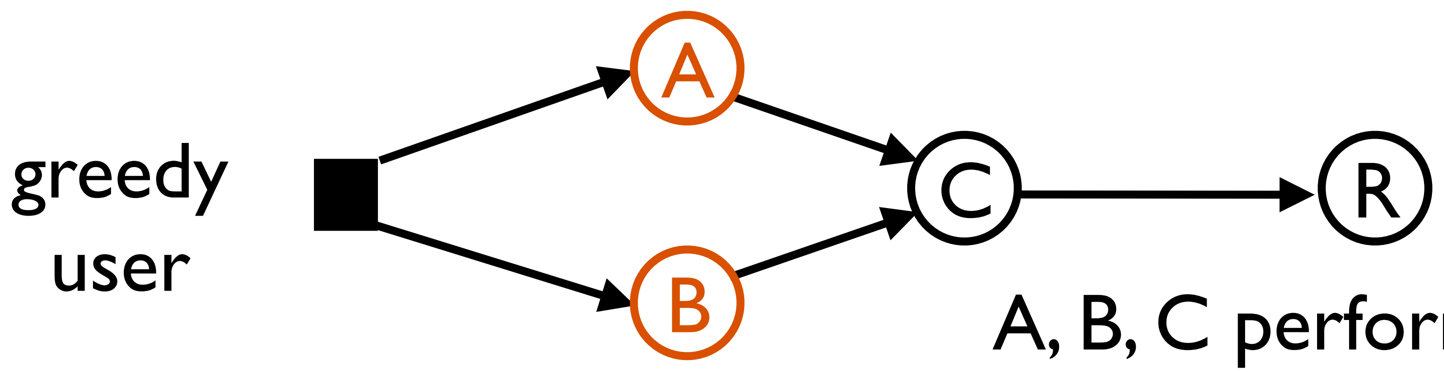
$\sum \theta_c r_c \geq$ optimal - $O(1/V)$

(performance gap diminishes as buffer size increases)

Independent of arrival rates

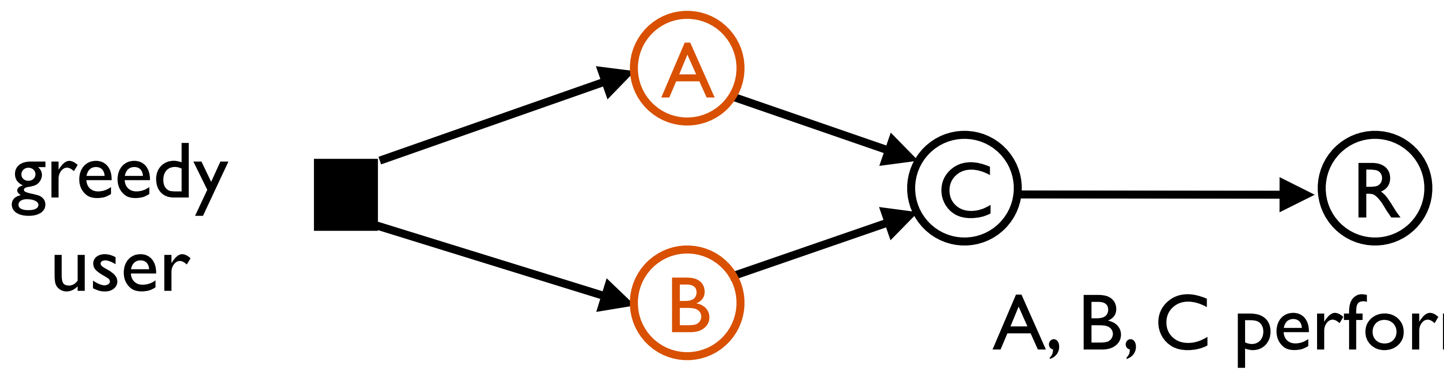
Distributed algorithm

In-Network Packet Dropping is Robust



A, B, C perform packet dropping
(A, B) is the defense perimeter protecting R

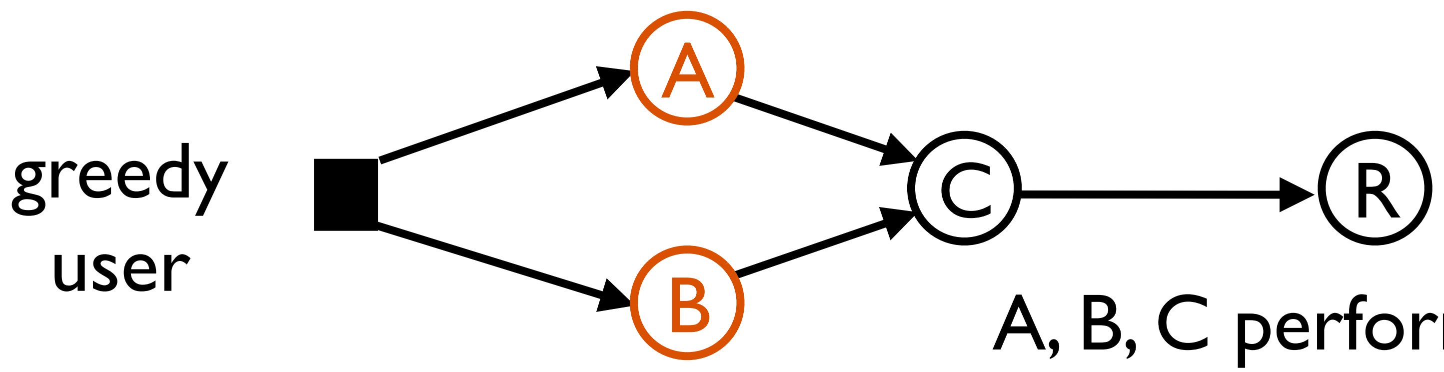
In-Network Packet Dropping is Robust



A, B, C perform packet dropping
(A, B) is the defense perimeter protecting R

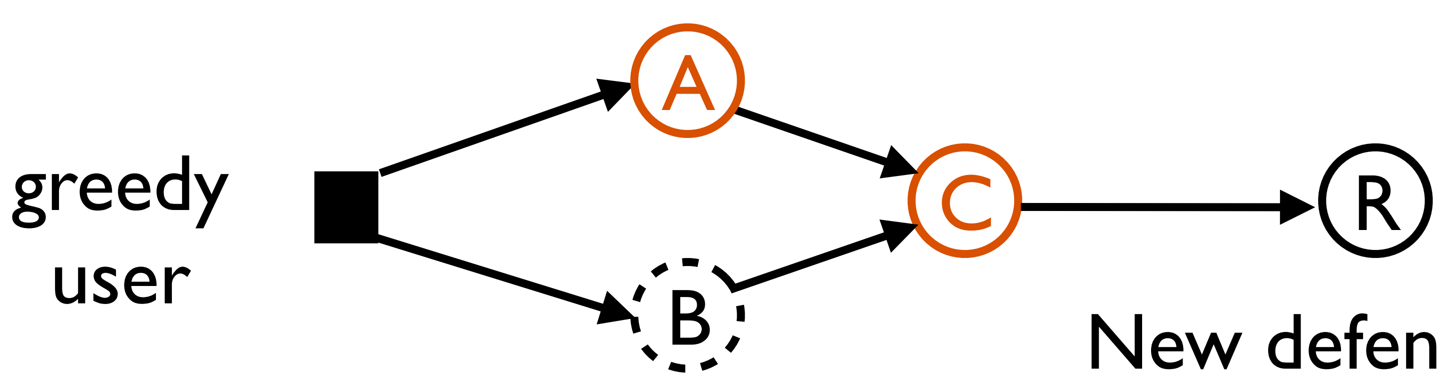
If node B is misconfigured and forwards everything

In-Network Packet Dropping is Robust



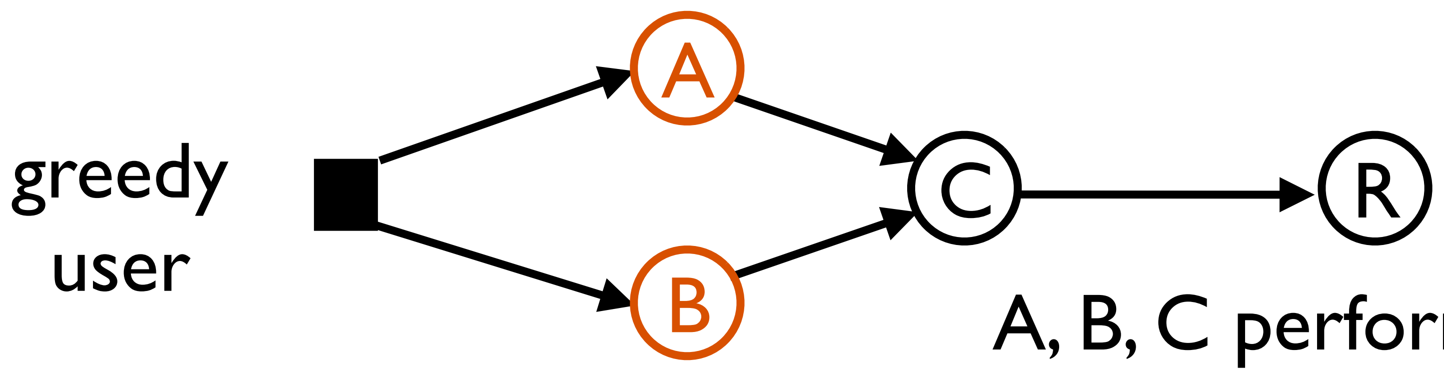
A, B, C perform packet dropping
(A, B) is the defense perimeter protecting R

If node B is misconfigured and forwards everything



New defense perimeter (A, C) is seamlessly formed without any changes in network operation

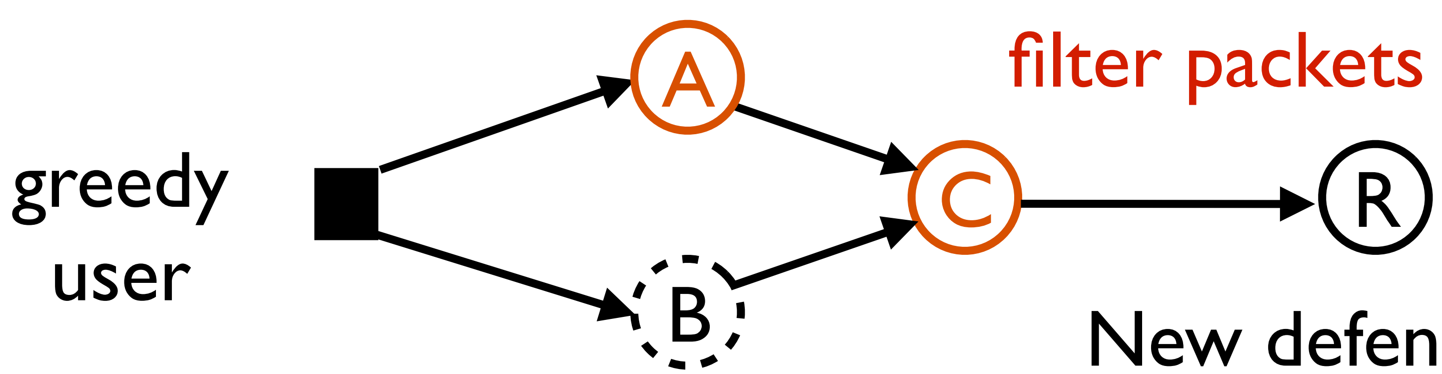
In-Network Packet Dropping is Robust



A, B, C perform packet dropping
(A, B) is the defense perimeter protecting R

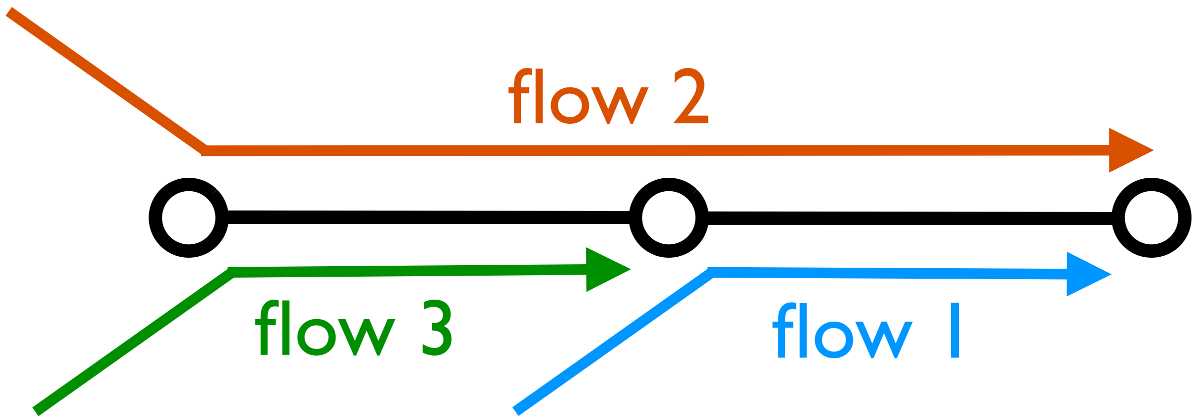
If node B is misconfigured and forwards everything

In source-based flow control, if B fails to filter packets then C & R are overloaded



New defense perimeter (A, C) is seamlessly formed without any changes in network operation

Simulation



Unit capacity links
Arrival rates (2, 2, 2)

As buffer size increases ↓

(a) Maximizing $3\,r_1 + 2\,r_2 + r_3$

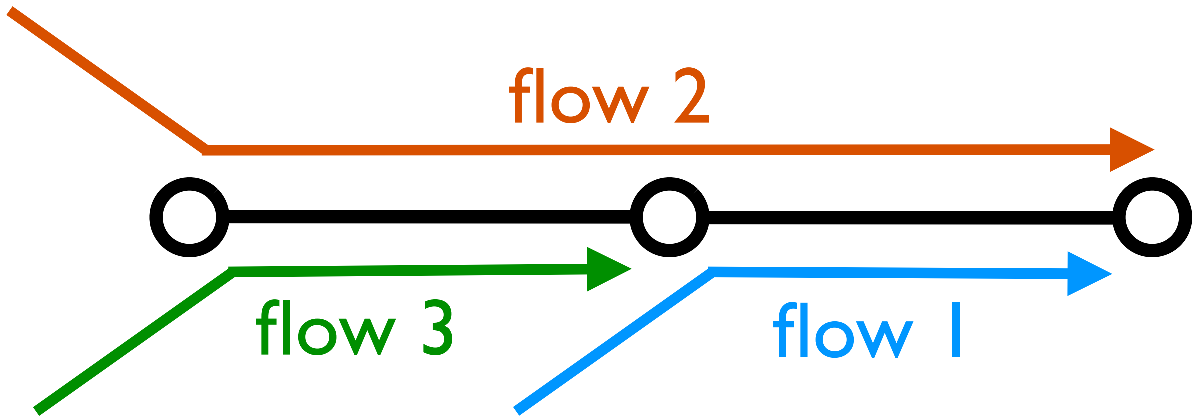
V	r_1	r_2	r_3
10	.787	.168	.099
20	.867	.133	.410
50	.992	.008	.967
100	.999	0	.999
opt	1	0	1

(b) Maximizing $3\,r_1 + 5\,r_2 + r_3$

V	r_1	r_2	r_3
10	.185	.815	.083
20	.107	.893	.095
50	.031	.969	.031
100	.002	.998	.001
opt	0	1	0

Buffer size is $V\theta_c + 2d_{max}$

Simulation



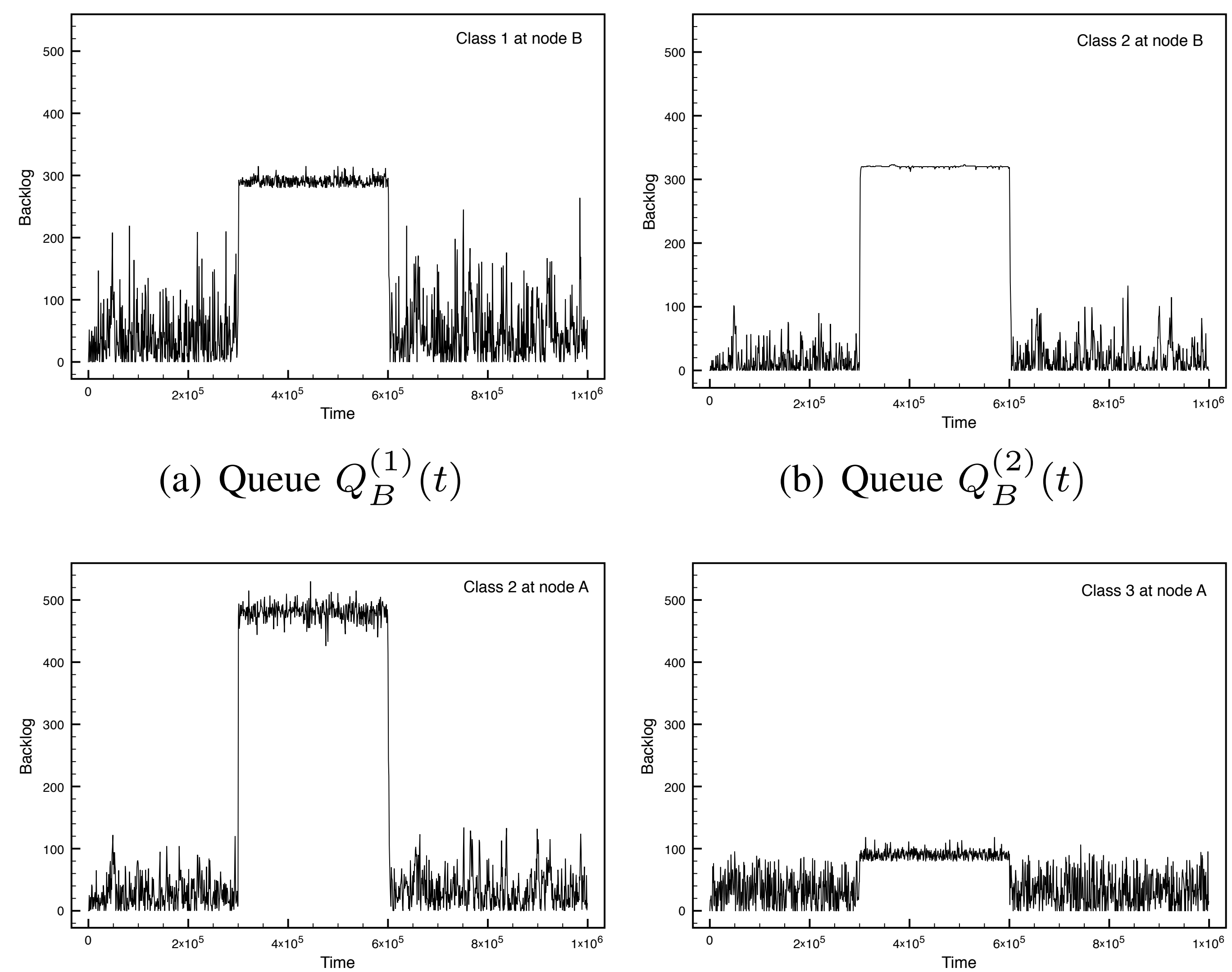
Time-varying arrival rates

(0.8, 0.1, 0.8) → (0.8, 2, 0.8) → (0.8, 0.1, 0.8)

stable — overload — stable

Maximize 3 r_1 + 5 r_2 + r_3

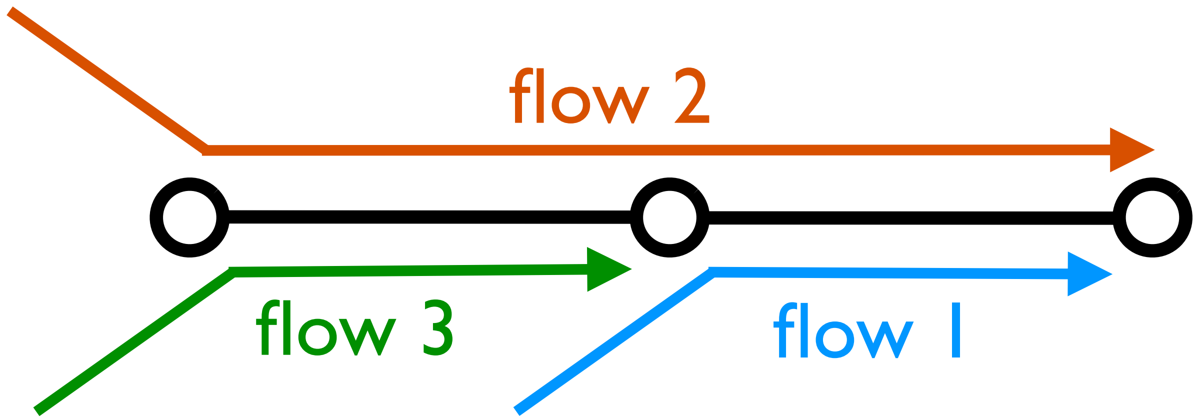
time interval	throughput in this interval	optimal value
$[0, 3 \cdot 10^5)$	$(.807, .104, .8)$	$(.8, .1, .8)$
$[3 \cdot 10^5, 6 \cdot 10^5)$	$(.002, .998, 0)$	$(0, 1, 0)$
$[6 \cdot 10^5, 10^6)$	$(.793, .106, .796)$	$(.8, .1, .8)$



(a) Queue $Q_B^{(1)}(t)$ (b) Queue $Q_B^{(2)}(t)$

(c) Queue $Q_A^{(2)}(t)$ (d) Queue $Q_A^{(3)}(t)$

Simulation



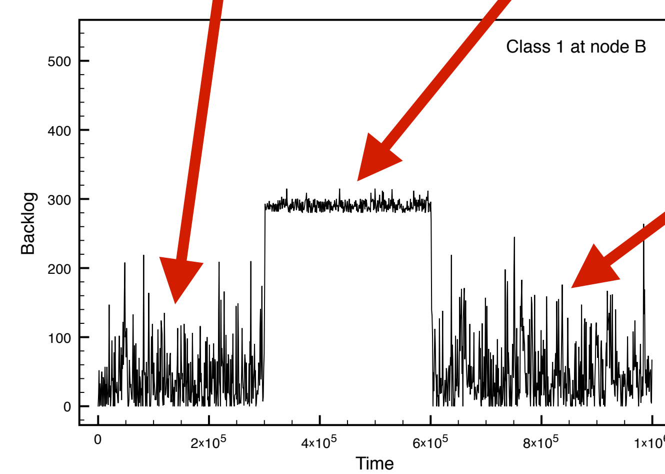
Time-varying arrival rates

(0.8, 0.1, 0.8) → (0.8, 2, 0.8) → (0.8, 0.1, 0.8)

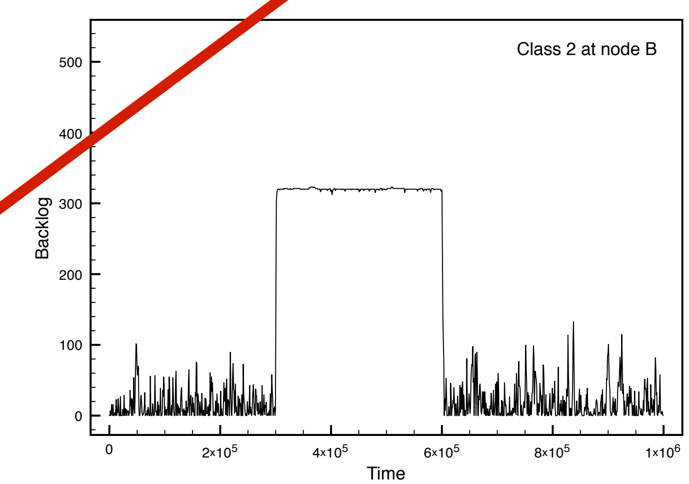
stable overload stable

Maximize 3 r_1 + 5 r_2 + r_3

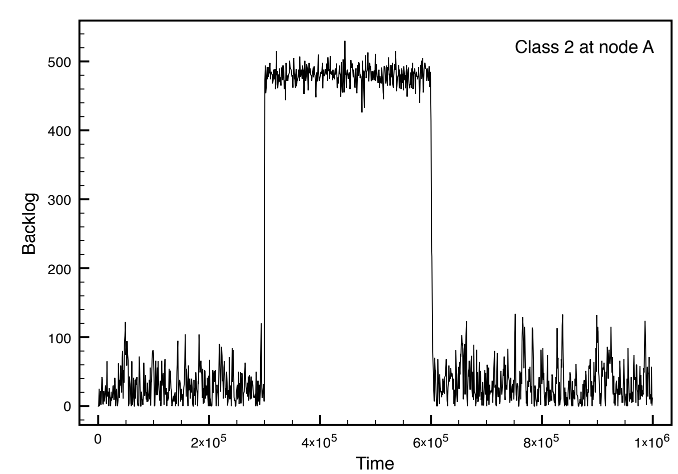
time interval	throughput in this interval	optimal value
$[0, 3 \cdot 10^5)$	$(.807, .104, .8)$	$(.8, .1, .8)$
$[3 \cdot 10^5, 6 \cdot 10^5)$	$(.002, .998, 0)$	$(0, 1, 0)$
$[6 \cdot 10^5, 10^6)$	$(.793, .106, .796)$	$(.8, .1, .8)$



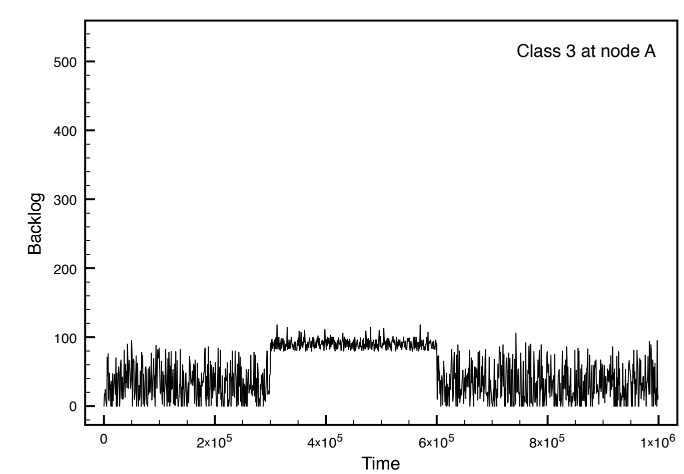
(a) Queue $Q_B^{(1)}(t)$



(b) Queue $Q_B^{(2)}(t)$



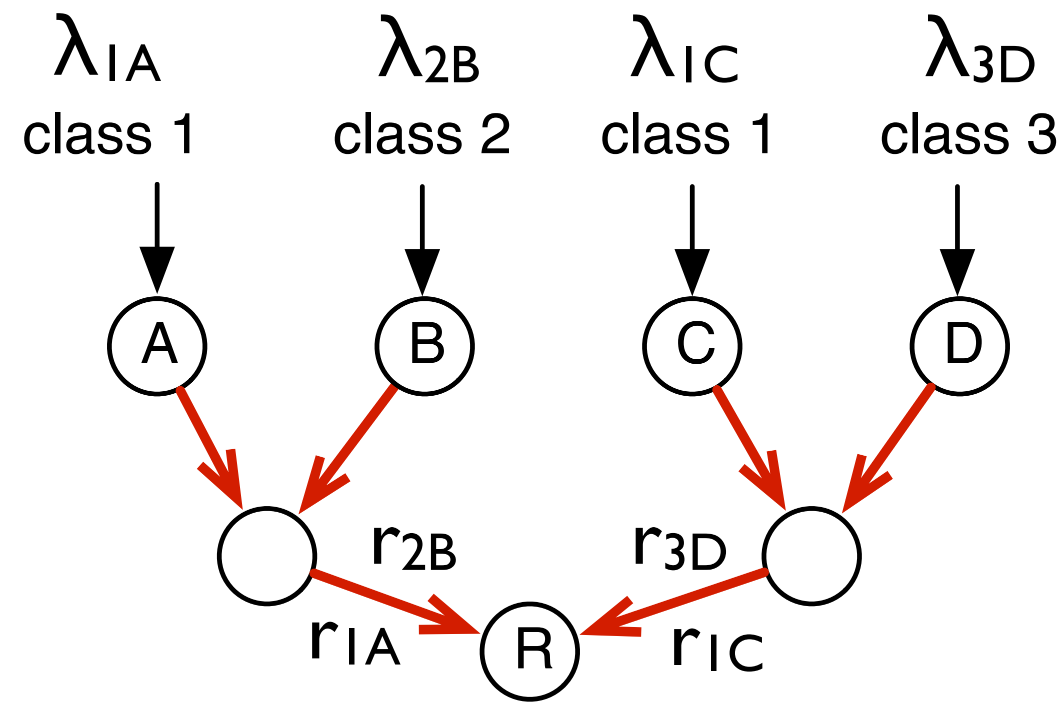
(c) Queue $Q_A^{(2)}(t)$



(d) Queue $Q_A^{(3)}(t)$

Task 2: Controlling the aggregate rate of flows by receiver-based flow control

Class-Based Utility Maximization

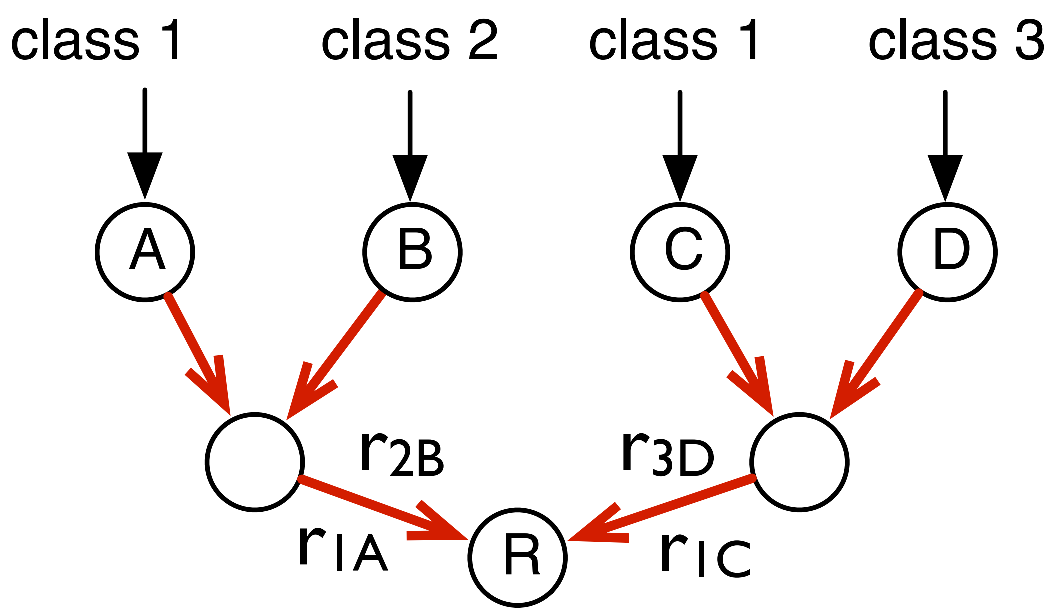


Assign a utility to a class of flows to control its aggregate throughput

$$\begin{array}{ll} \text{maximize} & g_1(r_{1A} + r_{1C}) + g_2(r_{2B}) + g_3(r_{3D}) \\ \text{subject to} & r_{ij} \leq \lambda_{ij}, \ (r_{1A}, r_{1C}, r_{2B}, r_{3D}) \text{ feasible} \end{array}$$

Generalizing the traditional approach of optimizing per-flow utilities

Class-Based Utility Maximization

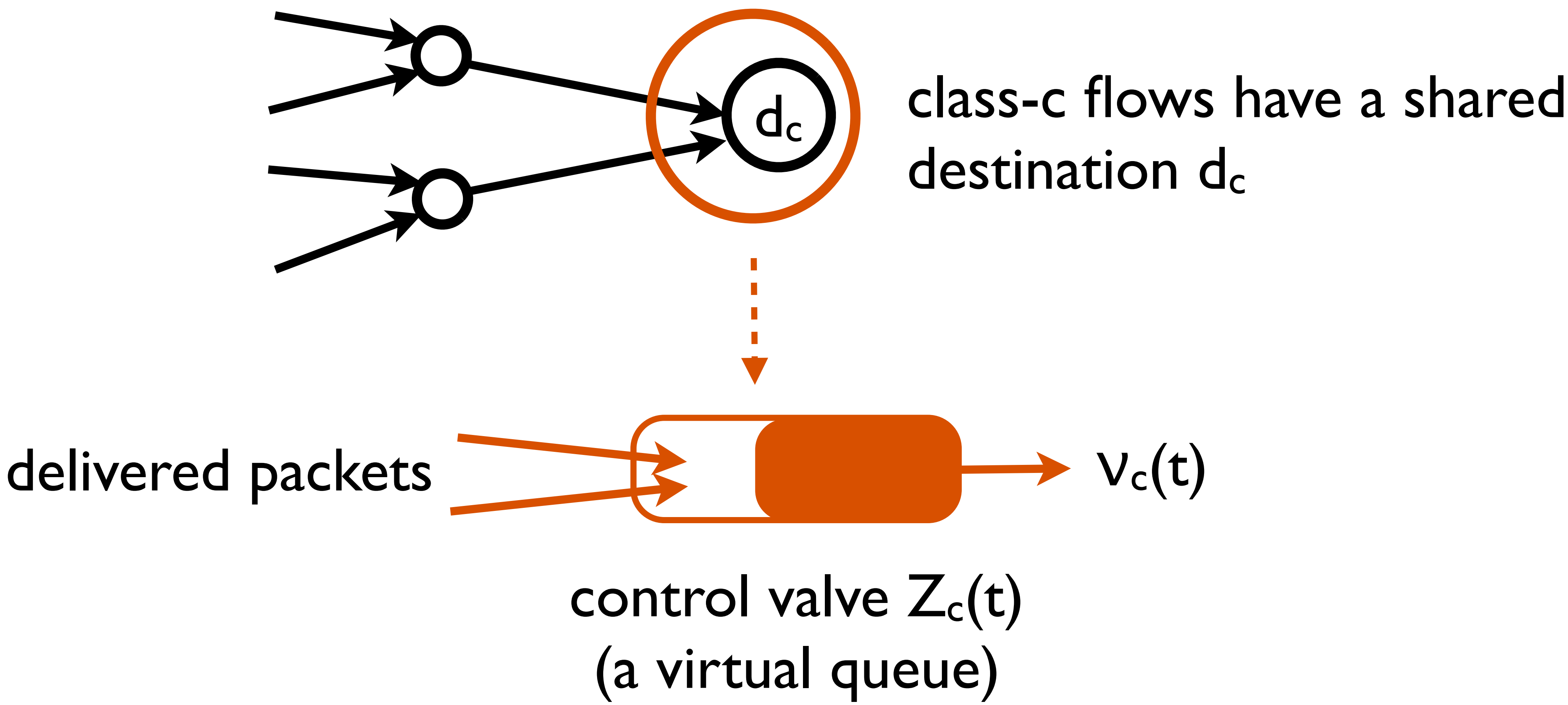


$$\begin{array}{ll} \text{maximize} & g_1(r_{1A}+r_{1C}) + g_2(r_{2B}) + g_3(r_{3D}) \\ \text{subject to} & r_{ij} \leq \lambda_{ij},\ \ (r_{1A}, r_{1C}, r_{2B}, r_{3D}) \text{ feasible} \end{array}$$

Can we design a distributed control policy?

- non-separable objective function
- how to distributively drop packets, without knowing arrival rates, to optimize aggregate throughput of each class?

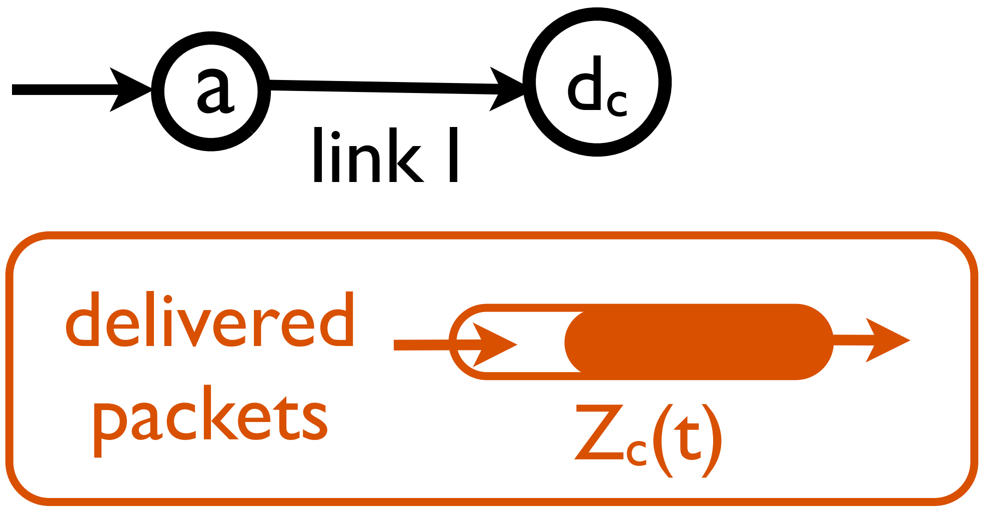
Receiver-End Flow Control



Control valve $Z_c(t)$ interacts with back-pressure routing to optimize class-c aggregate throughput

Utility-Optimal Overload Resilient Algorithm

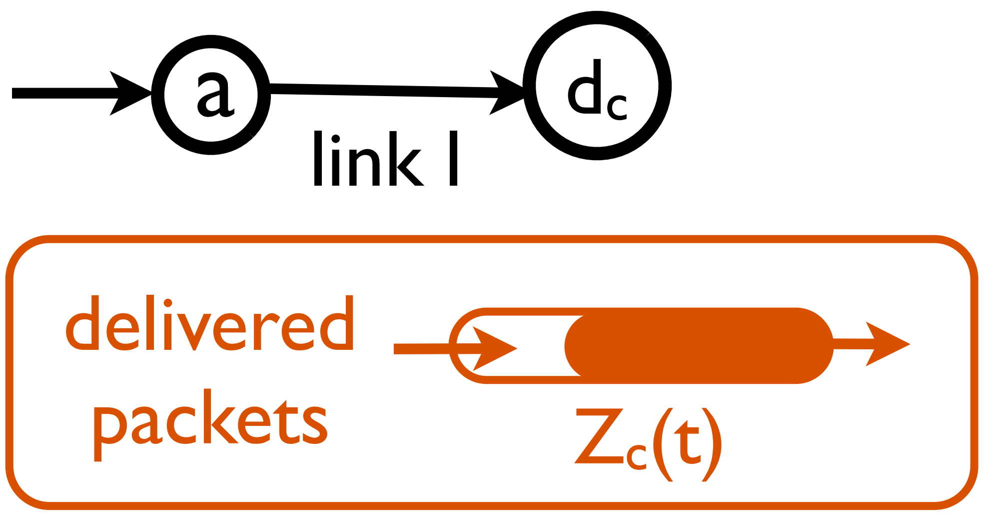
(1) Back-pressure routing, where each node maintains per-class queues.
For flow c over link (a, d_c), use the differential backlog



$$W_l^{(c)}(t) = Q_a^{(c)}(t) - Q_{d_c}^{(c)}(t)$$

Utility-Optimal Overload Resilient Algorithm

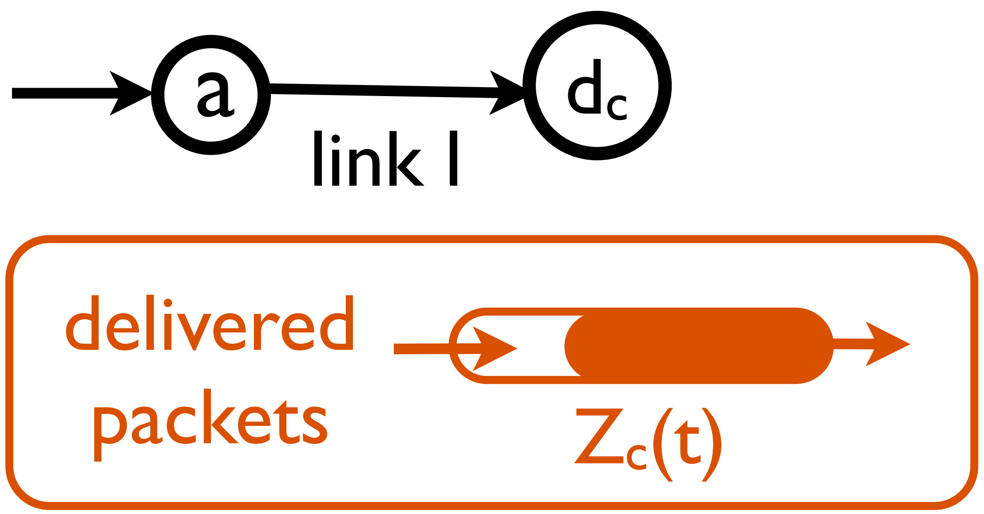
(1) Back-pressure routing, where each node maintains per-class queues.
For flow c over link (a, d_c), use the differential backlog



$$W_l^{(c)}(t) = Q_a^{(c)}(t) - \underbrace{Q_{d_c}^{(c)}(t)}_{\substack{=0 \text{ in normal} \\ \text{back-pressure}}}$$

Utility-Optimal Overload Resilient Algorithm

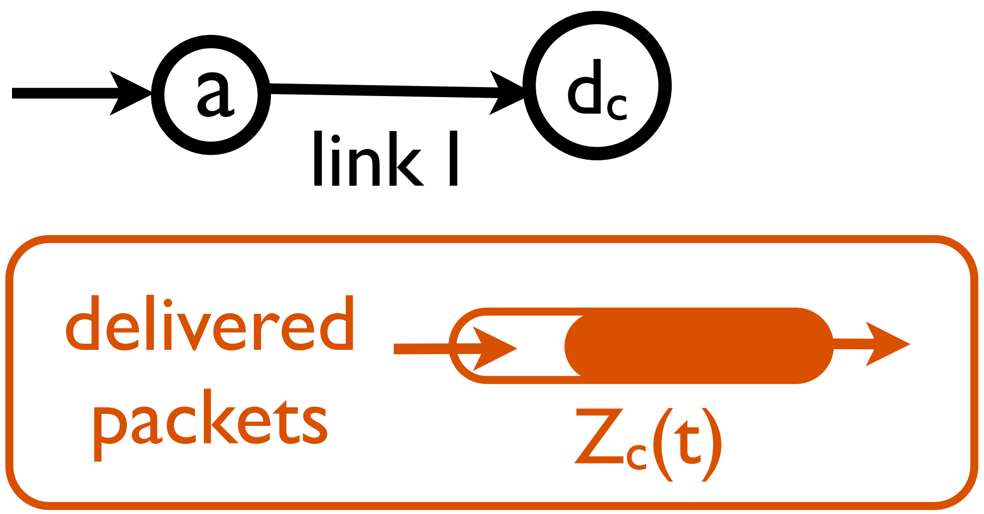
(1) Back-pressure routing, where each node maintains per-class queues. For flow c over link (a, d_c), use the differential backlog



$$W_l^{(c)}(t) = Q_a^{(c)}(t) - Q_{d_c}^{(c)}(t)$$

Utility-Optimal Overload Resilient Algorithm

(1) Back-pressure routing, where each node maintains per-class queues.
For flow c over link (a, d_c), use the differential backlog



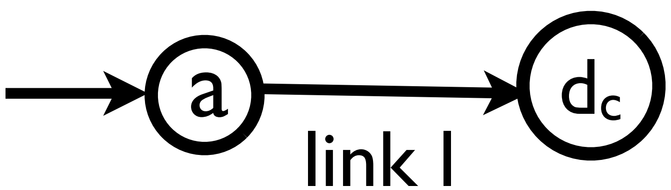
$$W_l^{(c)}(t) = Q_a^{(c)}(t) - Q_{d_c}^{(c)}(t)$$

if $Z_c(t) \geq Q$ $\longrightarrow$ $Q_{d_c}^{(c)}(t) = we^{w(Z_c(t)-Q)} > 0$

(receiving too many)

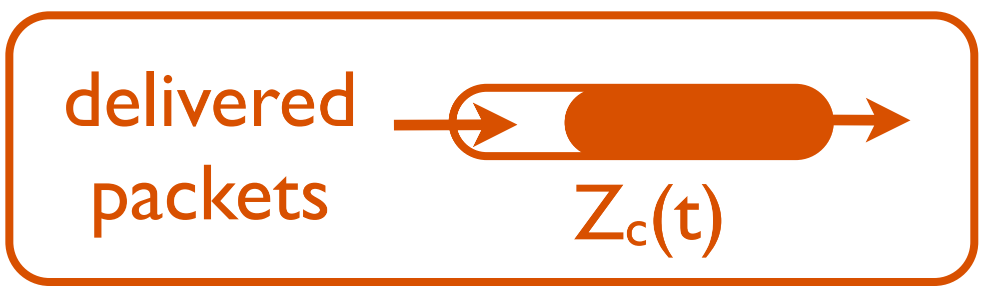
Utility-Optimal Overload Resilient Algorithm

(1) Back-pressure routing, where each node maintains per-class queues.
For flow c over link (a, d_c), use the differential backlog



$$W_l^{(c)}(t) = Q_a^{(c)}(t) - Q_{d_c}^{(c)}(t)$$

push-back mechanism (slow down)

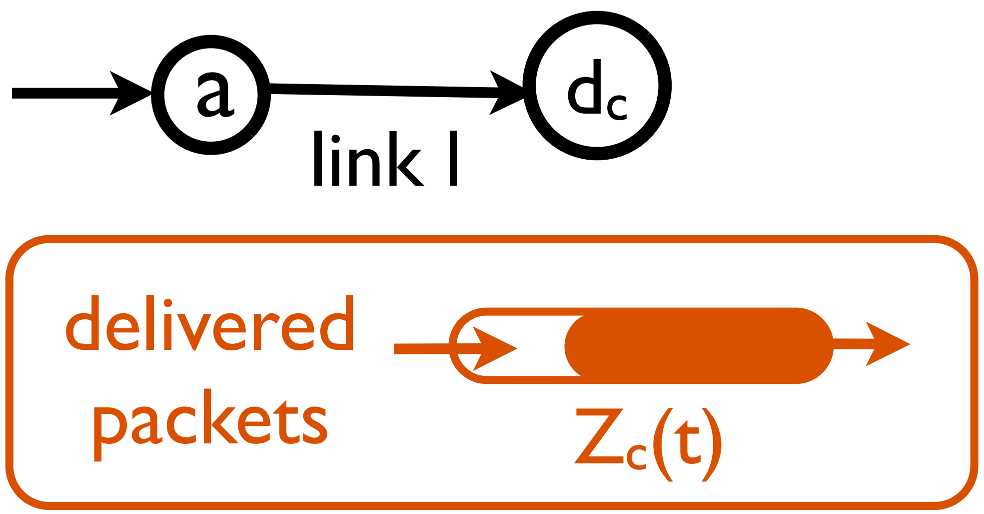


if $Z_c(t) \geq Q$ $\longrightarrow$ $Q_{d_c}^{(c)}(t) = we^{w(Z_c(t)-Q)} > 0$

(receiving too many)

Utility-Optimal Overload Resilient Algorithm

(1) Back-pressure routing, where each node maintains per-class queues. For flow c over link (a, d_c), use the differential backlog



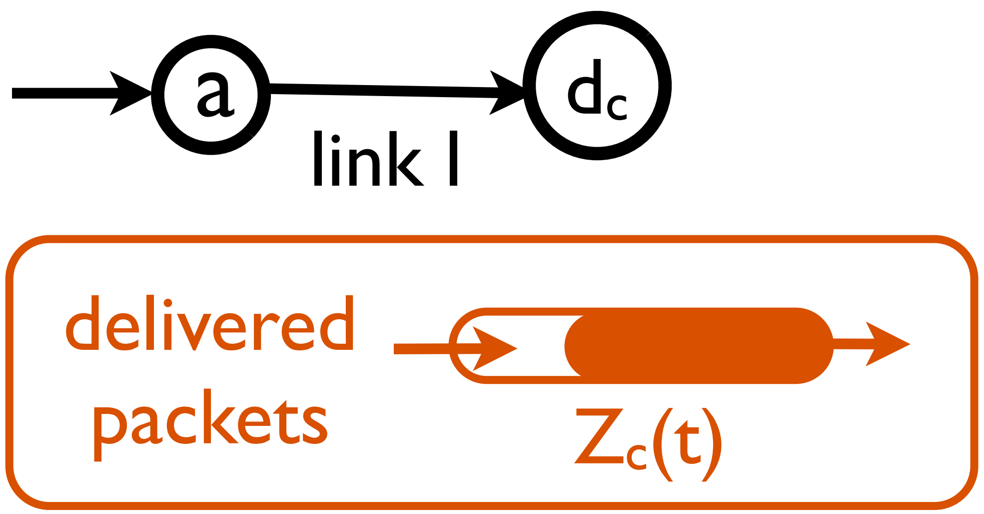
$$W_l^{(c)}(t) = Q_a^{(c)}(t) - Q_{d_c}^{(c)}(t)$$

if $Z_c(t) \geq Q$ $\longrightarrow$ $Q_{d_c}^{(c)}(t) = we^{w(Z_c(t)-Q)} > 0$

(receiving too many)

Utility-Optimal Overload Resilient Algorithm

(1) Back-pressure routing, where each node maintains per-class queues. For flow c over link (a, d_c), use the differential backlog



$$W_l^{(c)}(t) = Q_a^{(c)}(t) - Q_{d_c}^{(c)}(t)$$

$$\text{if } Z_c(t) \geq Q \longrightarrow Q_{d_c}^{(c)}(t) = we^{w(Z_c(t)-Q)} > 0$$

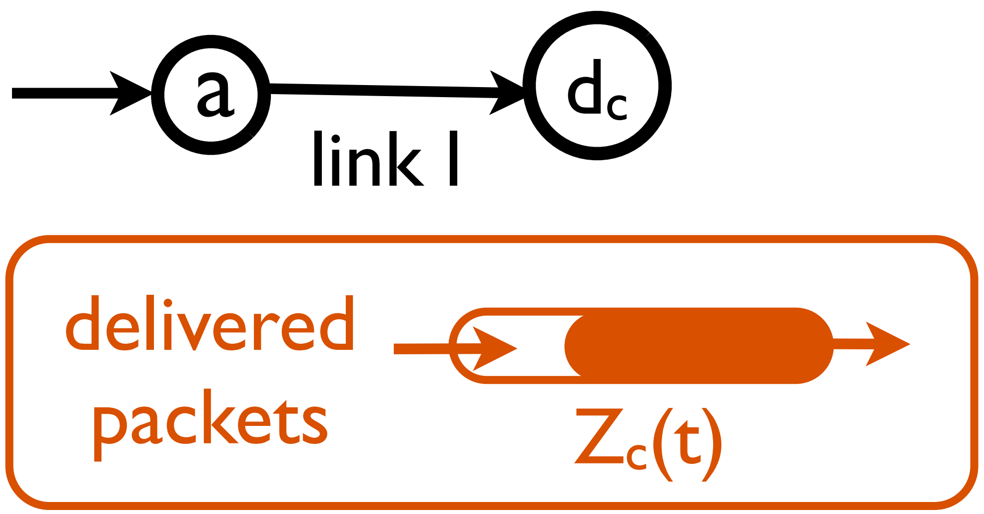
(receiving too many)

$$\text{if } Z_c(t) < Q \longrightarrow Q_{d_c}^{(c)}(t) = -we^{w(Q-Z_c(t))} < 0$$

(receiving too few)

Utility-Optimal Overload Resilient Algorithm

(1) Back-pressure routing, where each node maintains per-class queues.
For flow c over link (a, d_c), use the differential backlog



$$W_l^{(c)}(t) = Q_a^{(c)}(t) - Q_{d_c}^{(c)}(t)$$

"receiver asks for data" (speed up)

if $Z_c(t) \geq Q \longrightarrow Q_{d_c}^{(c)}(t) = we^{w(Z_c(t)-Q)} > 0$

(receiving too many)

if $Z_c(t) < Q \longrightarrow Q_{d_c}^{(c)}(t) = -we^{w(Q-Z_c(t))} < 0$

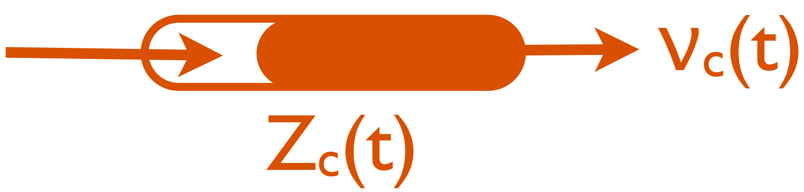
(receiving too few)

Utility-Optimal Overload Resilient Algorithm

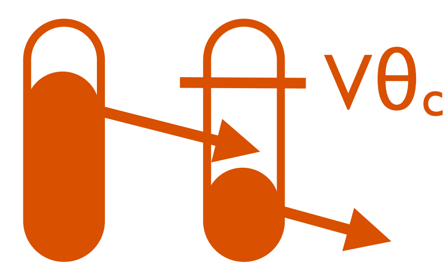
(2) Choose $\nu_c(t)$ that solves

$$\begin{aligned} &\text{maximize} \quad V h_c\big(\nu_c(t)\big) + \nu_c(t)\, Q_{d_c}^{(c)}(t) \\ &\text{subject to} \quad 0 \le \nu_c(t) \le \nu_{\max} \end{aligned}$$

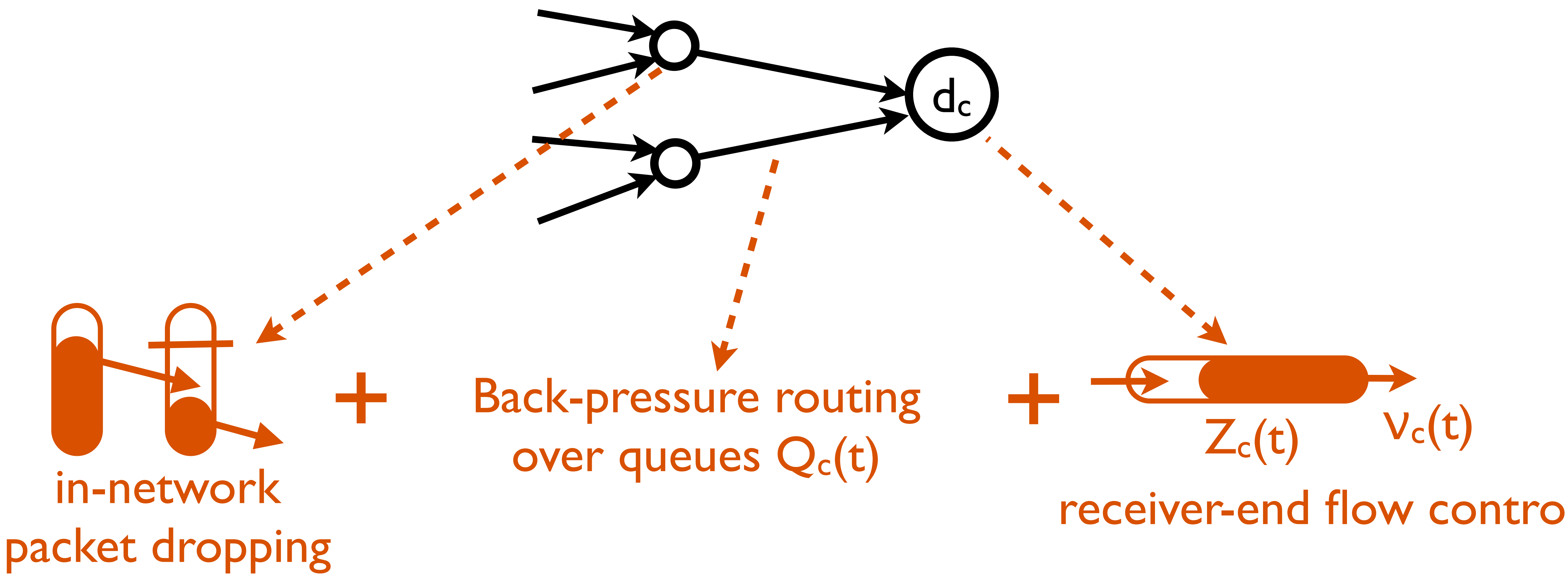
$h_c(x) = g_c(x) - \theta_c x$



(3) The same packet dropping policy for per-class queues at each node



Utility-Optimal Overload Resilient Algorithm



Theorem:

$Q_c(t) \leq V\theta_c + 2d_{max}$ for all t (using finite buffer)

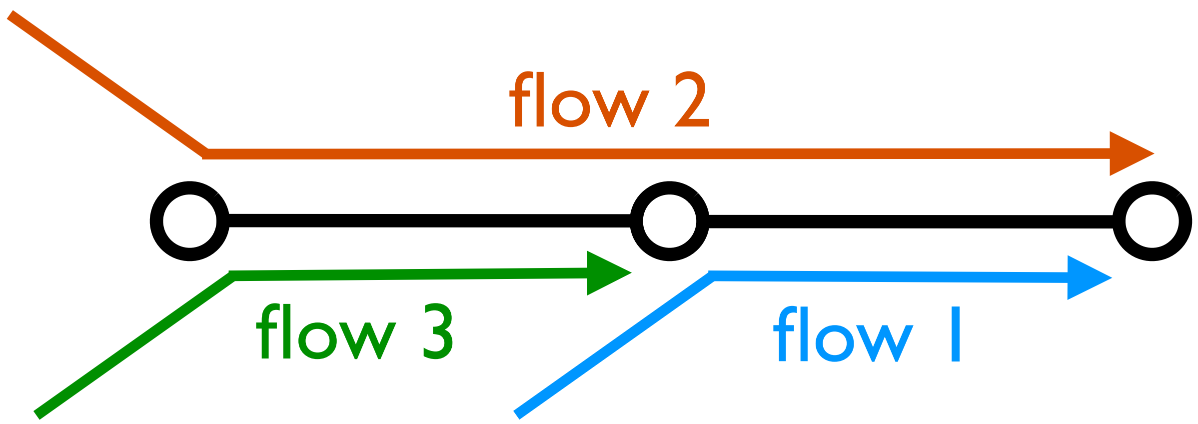
$\sum g_c(r_c) \geq$ optimal utility - $O(1/V + \varepsilon)$

(large buffer reduces performance gap)

Sketch of analysis

- Given a fixed arrival rate vector, characterizing the set of achievable throughput vectors in terms of queue overflow rates
  - optimal queue overflow rate = optimal packet dropping rate
- Decoupling a utility function into two components
  - new utility functions for receiver-end flow control
  - distributed packet dropping
  - optimizing original utility = max. new utility + min. packet dropping
- Transforming utility optimization problems using auxiliary variables
- Using exponential Lyapunov functions for drift analysis

Utility-Optimal Algorithm / Simulations



Unit capacity links
Arrival rates (2, 2, 2)

Objective: proportional fairness

Optimal throughput: (2/3, 1/3, 2/3)

Throughput

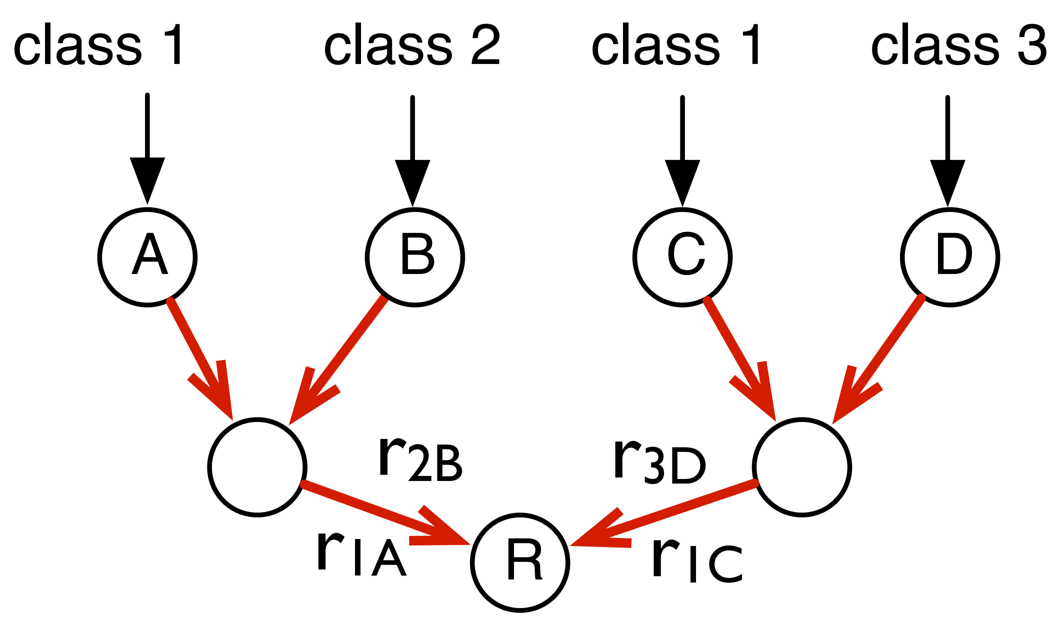
V	r_1	r_2	r_3	$\sum \log(r_c)$
10	.522	.478	.522	-2.038
20	.585	.415	.585	-1.952
50	.631	.369	.631	-1.918
100	.648	.352	.647	-1.912
optimal	.667	.333	.667	-1.910

Maximum backlog

V	$Q_B^{(1)}(t)$	$Q_B^{(2)}(t)$	$Q_A^{(2)}(t)$	$Q_A^{(3)}(t)$	$Q_{\max}^{(c)}$
10	140	97	137	137	142
20	237	187	240	236	242
50	539	441	538	540	542
100	1036	865	1039	1039	1042

Finite buffer size: $V\theta_c+2d_{max}$

Utility-Optimal Algorithm / Simulations



Unit capacity links
Arrival rates (2, 2, 2, 2)

Objective: max-min fairness over classes

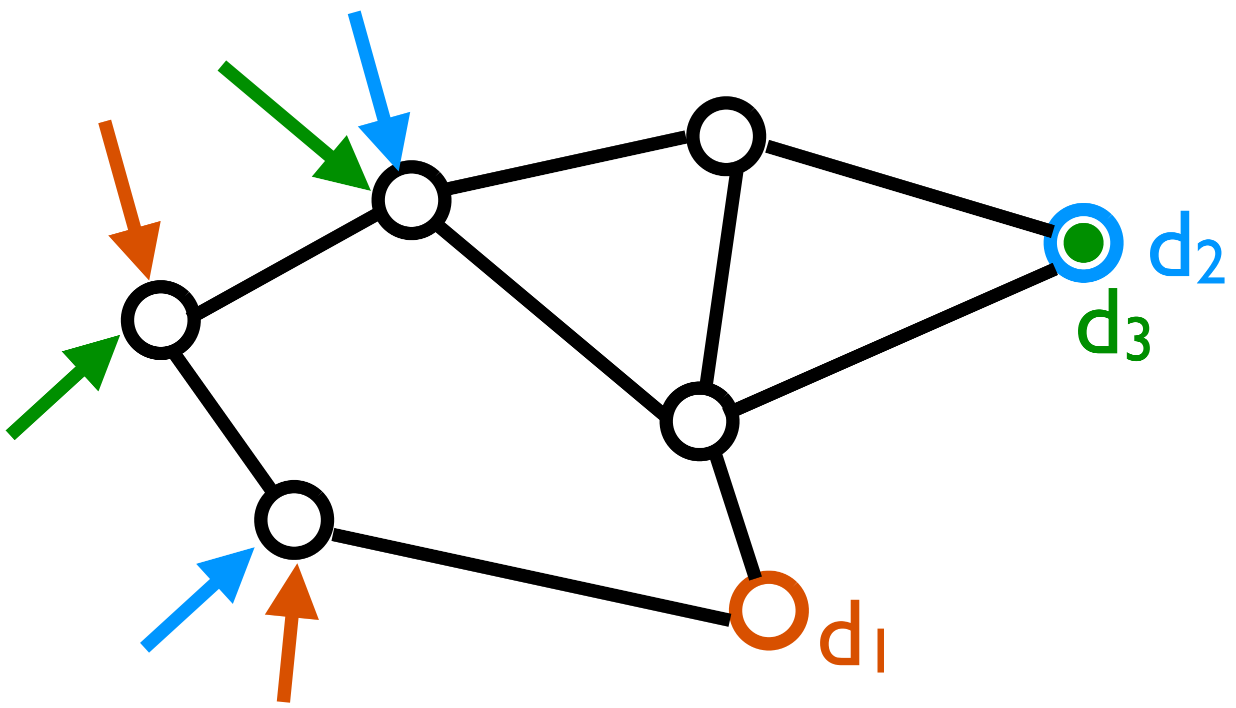
Per-class throughput

V	$r_{1A} + r_{1C}$	r_{2B}	r_{3D}
10	.200	.100	.100
20	.364	.206	.205
30	.661	.650	.651
50	.667	.667	.667
optimal	.667	.667	.667

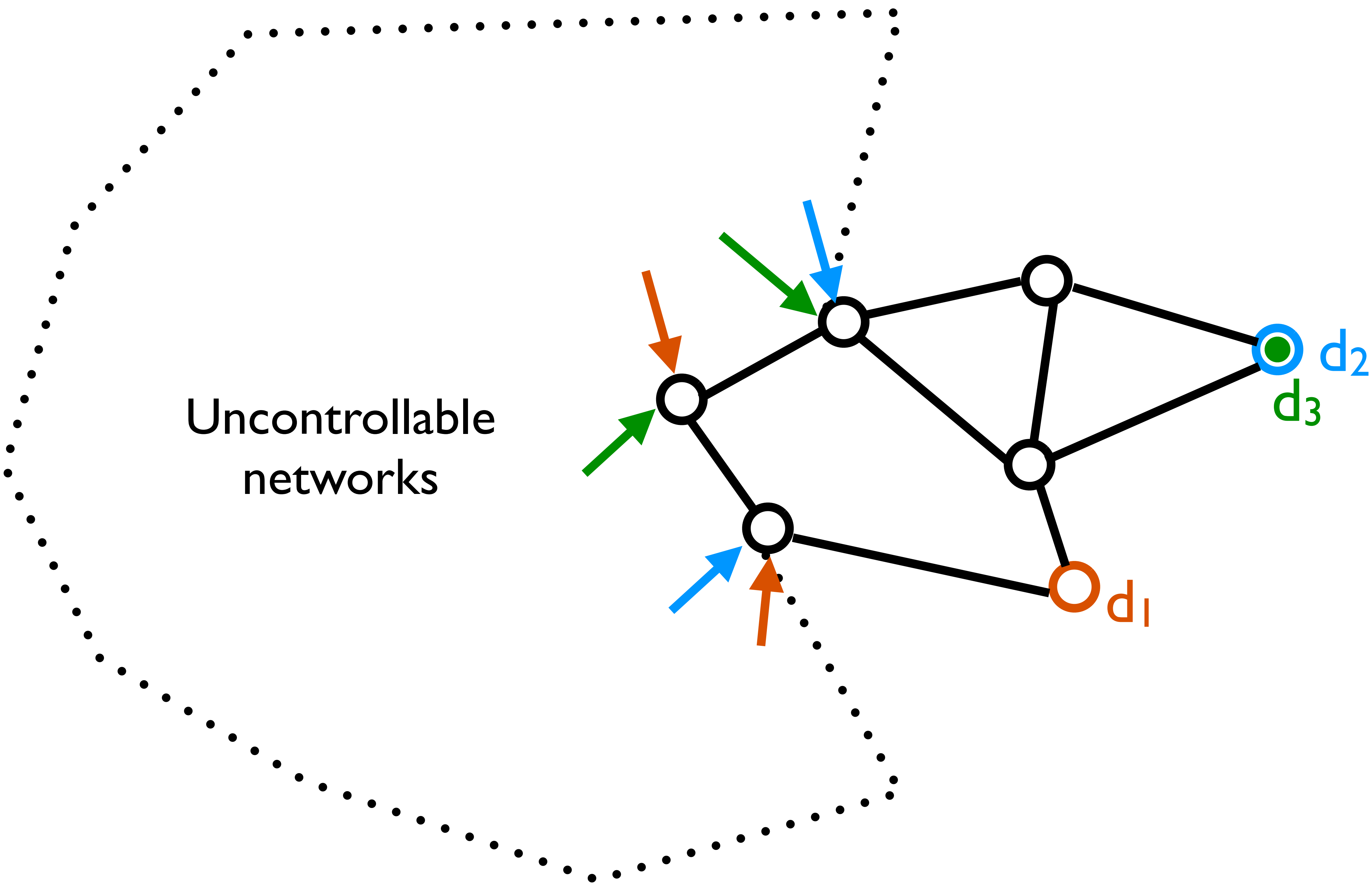
As buffer size increases

Finite buffer size: $V\theta_c + 2d_{max}$

Remarks

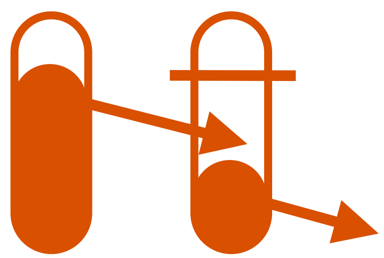


Remarks



Summary

- Key problem: optimizing network performance without source cooperation
- Can’t rely on the sources to resolve network overload
  - greedy users, congestion-unaware traffic, flash crowd, DDoS attacks
- We propose a robust, distributed, utility-optimal algorithm:



threshold-based packet dropping

+ Back-pressure routing over queues $Q_c(t)$ +

$Z_c(t)$ $\nu_c(t)$

receiver-end flow control

- This algorithm controls the aggregate rate of flows, providing differentiated services to different classes of flows