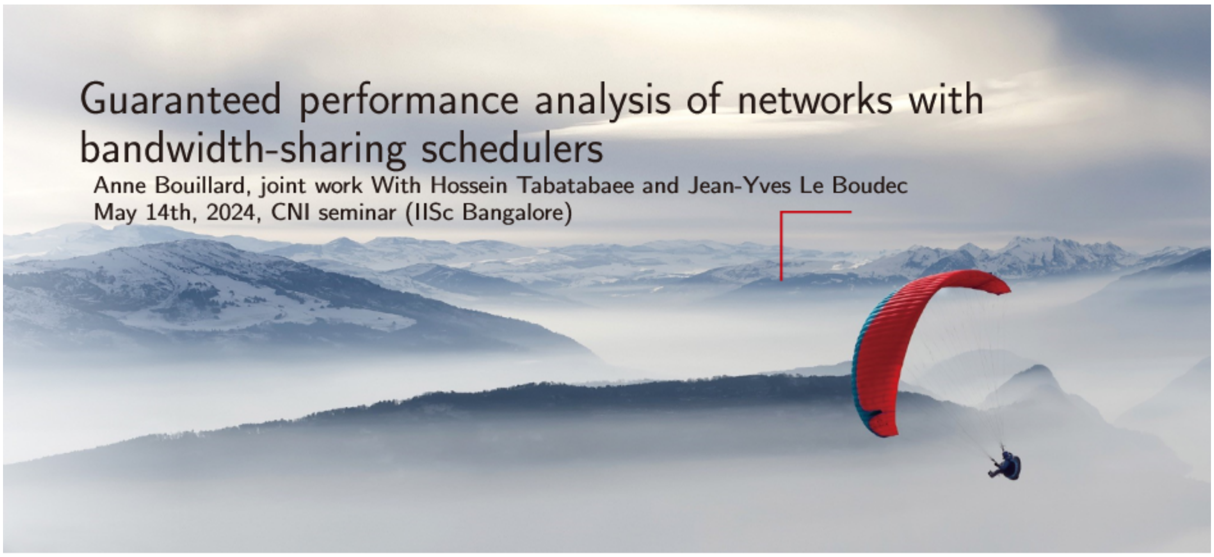
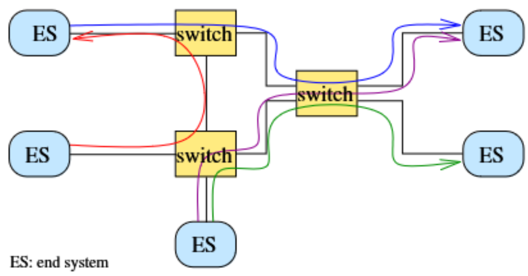


1 / 36 Huawei Public use HUAWEI

Time-sensitive networks

Time-sensitive networks need to satisfy timing constraints to ensure a correct behavior.



Examples of applications

- Embedded systems in avionics
- Industrial automation
- Campus networks...

Objective: performance guarantees upper bounds

Compute the maximum time it takes for a packet to cross the system (Worst-case delay)

2 / 36 Huawei Public use HUAWEI

Network calculus

- Theory developed in the 1990's by R.L. Cruz, then developed and popularized by C.S. Chang and J.-Y. Le Boudec.
- Filtering theory in the (min,plus) algebra.
- Applications:
  - Internet: video transmission (VoD),
  - Load-balancing in switches [Birkhoff-von Neumann switches, C.S. Chang]
  - Embedded systems: AFDX (Avionics Full Duplex) [Rockwell-Collins software used to certify A380], Networks-on-chip
- Extentions / variations:
  - Real-Time Calculus [L. Thiele, S. Chakraborty]
  - Extended to Stochastic network calculus [C.S. Chang, Y.M. Jiang, F. Ciucu, J. Schmitt]
- Recent trend to using this in 5G network that have strong latency and reliability requirements.

3 / 36 Huawei Public use HUAWEI

Network calculus *vs.* other models

Scheduling (Real-time analysis)

- Very precise analysis at the job level inside one scheduler
- Multi processor / multi-core
- Mostly deterministic
- Complex generalization to networks

Network Calculus

- Deterministic / probablistic analysis
- Models for scheduling policies
- Less precise bounds, room for improvements

Queuing theory

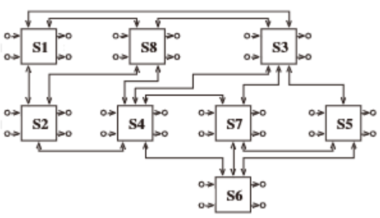
- Probablistic analysis and performance bounds
- Markovian assumptions for systems of queues (Jackson networks) + insensitivity properties
- Not so many service policies with networks

Network calculus premise

Real data		**(min,plus) functions**
Real input traffic	$\xrightarrow{\text{abstraction}}$	arrival curve
network element	$\xrightarrow{\text{abstraction}}$	service curve
$\Downarrow$		$\Downarrow$ (min,plus)-operators
Delay / backlog	$\longrightarrow$	Upper bound on the delay / backlog

Complex networks

Example of industrial topology



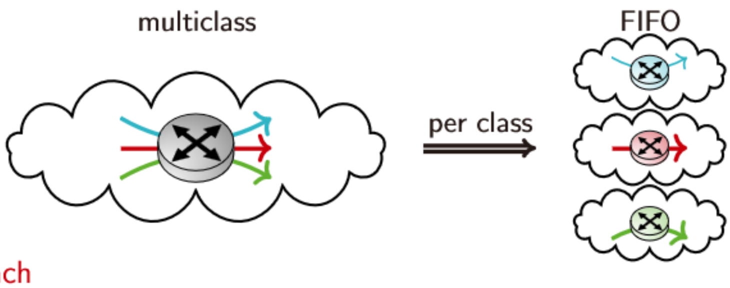
- 96 end-systems, 8 switches, 984 flows
- service rate: 1 Gb/s; switching latency: 16μs.
- Three classes of flows: (1) critical, (2) multimedia and (3) best-effort.

class	1	2	3
flows	128	500	266

Characteristics of the network

- **Multiclass:** each flow belongs to a traffic class, sharing the same bandwidth, but with differentiated services.
- **Cyclic dependencies:** flows create dependency loops, making the analysis more complex, and few works focus on general topologies.

Analysis of complex topologies with Network Calculus



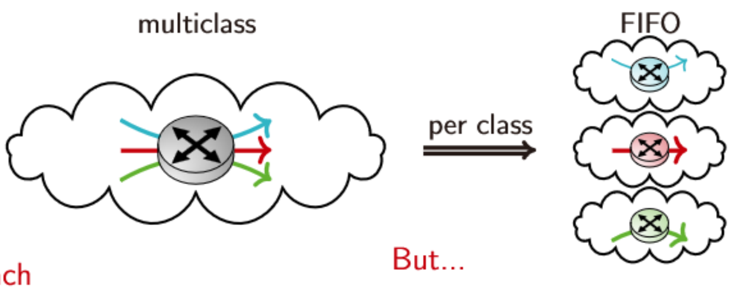
Typical approach

1. compute a service curve per class of traffic and network element (typically an output port of a switch)

⇒ independent per-class FIFO networks

2. compute the delay bounds in the FIFO networks

Analysis of complex topologies with Network Calculus



Typical approach

1. compute a service curve per class of traffic and network element (typically an output port of a switch)

⇒ independent per-class FIFO networks

2. compute the delay bounds in the FIFO networks

But...

- Recent works improve the per-class service curves, taking into account every class's characteristics

⇒ Inter-dependent per-class networks, a new analysis is required.

- This talk: **bandwidth-sharing and DRR networks**

Content

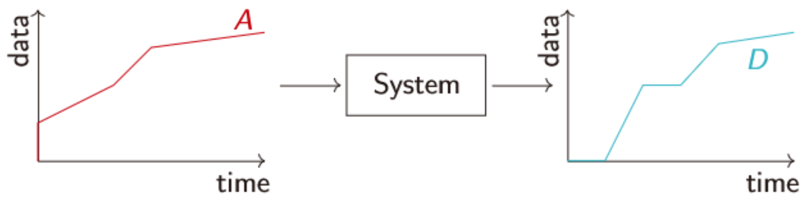
Network Calculus framework

Bandwidth-Sharing policies

Analysis of FIFO networks

Analysis of DRR network

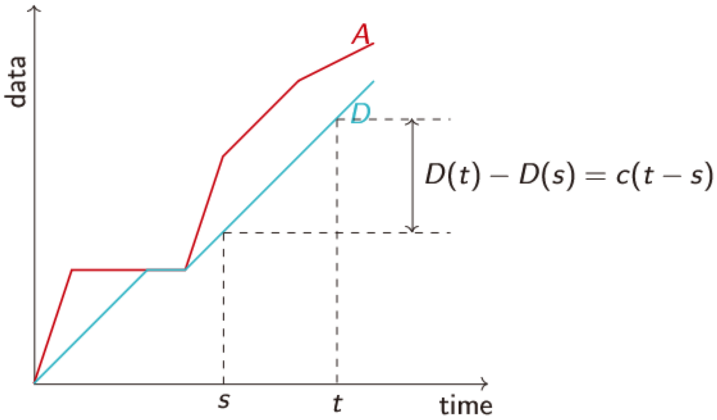
Cumulative processes



- $A : \mathbb{R}_+ \to \mathbb{R}_{\min +}$: process of the cumulative arrivals, non-decreasing function
- $D : \mathbb{R}_+ \to \mathbb{R}_{\min +}$: process of the cumulative departures, non-decreasing function
- Causality constraint: $A \geq D$

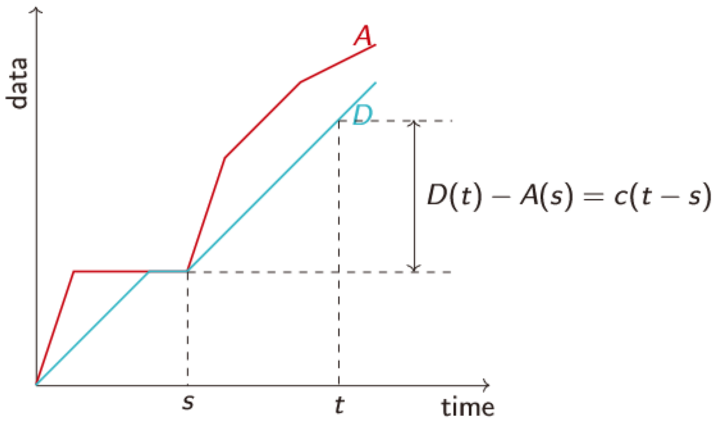
Example of a constant-rate server

Suppose that the system serves exactly c bits of data per unit of time.



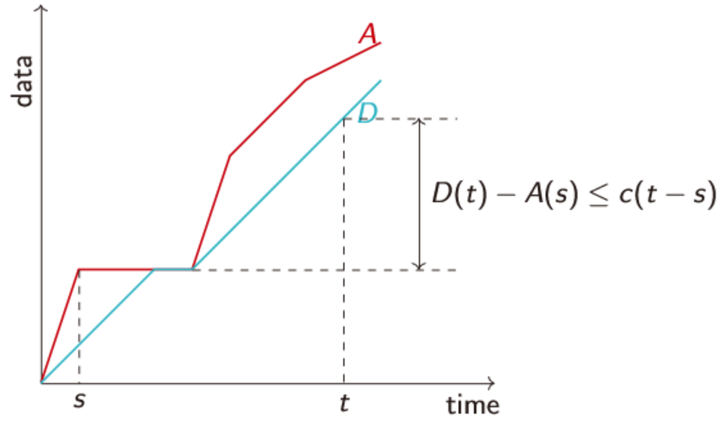
Example of a constant-rate server

Suppose that the system serves exactly c bits of data per unit of time.



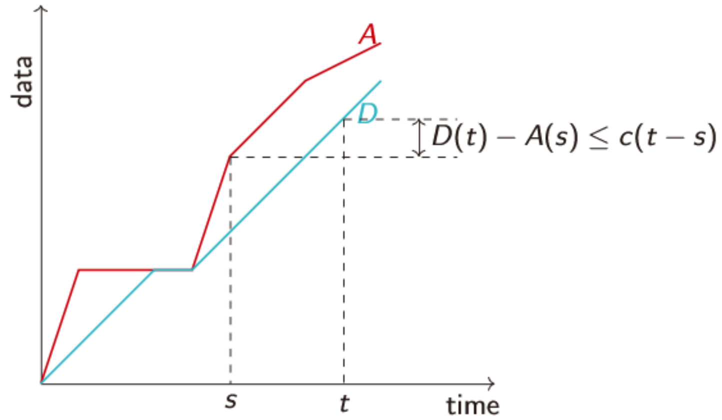
Example of a constant-rate server

Suppose that the system serves exactly c bits of data per unit of time.



Example of a constant-rate server

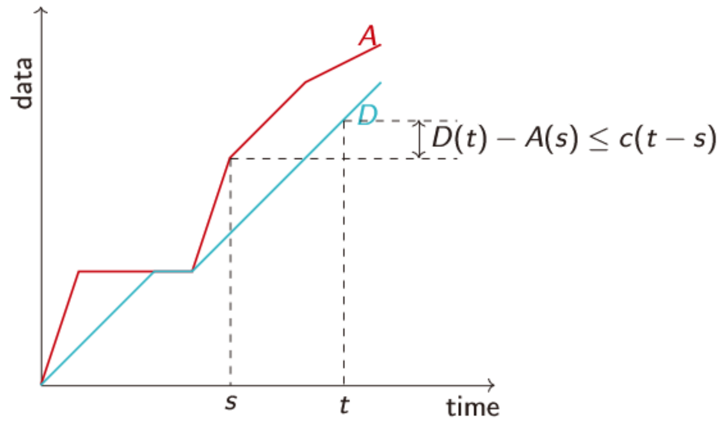
Suppose that the system serves exactly c bits of data per unit of time.



10 / 36 Huawei Public use

Example of a constant-rate server

Suppose that the system serves exactly c bits of data per unit of time.



$$D(t) = \inf_{0 \leq s \leq t} A(s) + c(t-s) = A * c(t)$$

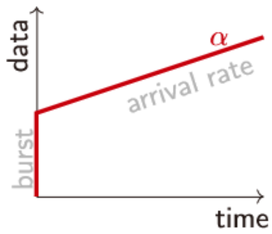
10 / 36 Huawei Public use

Arrival and service curves

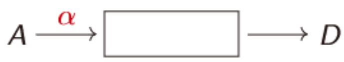
Arrival curve

A is constrained by the function α if
$\forall 0 \leq s \leq t$,

$$A(t) - A(s) \leq \alpha(t - s).$$



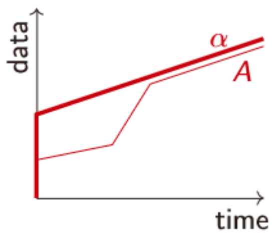
Arrival and service curves



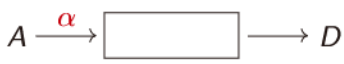
Arrival curve

A is constrained by the function α if
$\forall 0 \leq s \leq t$,

$$A(t) - A(s) \leq \alpha(t - s).$$



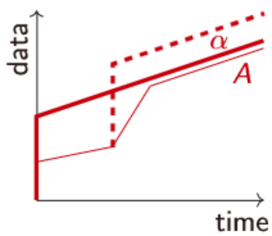
Arrival and service curves



Arrival curve

A is constrained by the function α if
$\forall 0 \leq s \leq t$,

$$A(t) - A(s) \leq \alpha(t - s).$$



Arrival and service curves

Arrival curve

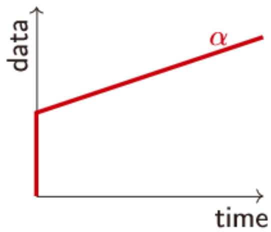
A is constrained by the function α if $\forall 0 \leq s \leq t$,

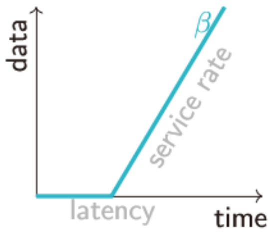
$$A(t) - A(s) \leq \alpha(t - s).$$

Strict service curve

A network element guarantees β for A if, while system not empty, D satisfies

$$D(t) \geq D(s) + \beta(t - s).$$





Arrival and service curves

Arrival curve

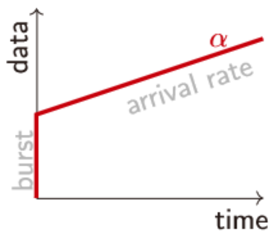
A is constrained by the function α if $\forall 0 \leq s \leq t$,

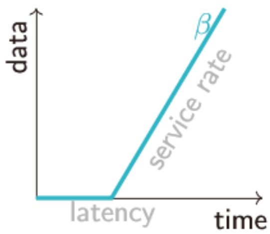
$$A(t) - A(s) \leq \alpha(t - s).$$

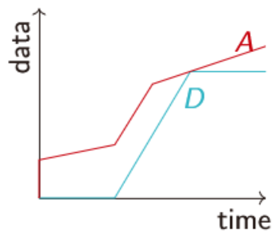
Strict service curve

A network element guarantees β for A if, while system not empty, D satisfies

$$D(t) \geq D(s) + \beta(t - s).$$



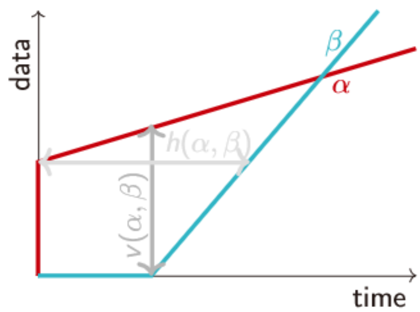




From constraints to performance bounds

Maximum backlog:
$b_{\max} = \sup_{t \geq 0} A(t) - D(t)$

Maximum delay:
$d_{\max} = \inf\{d \mid \forall t \in \mathbb{R}_+,\ D(t+d) \geq A(t)\}$



Performance bounds

- $b_{\max} \leq \alpha \oslash \beta(0) = v(\alpha, \beta) = \sup\{\alpha(t) - \beta(t) \mid t \geq 0\}$
- $d_{\max} \leq h(\alpha, \beta) = \inf\{\forall t \geq 0,\ d \geq 0 \mid \alpha(t) \leq \beta(t+d)\}$ (for FIFO per flow).

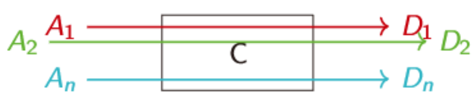
Content

Bandwidth-sharing policies

Assume n classes of traffic sharing the same network element.

- **Generalized processor sharing (GPS):** the bandwidth is perfectly shared among the flows (fluid model).
- **Round-robin (RR) scheduling:** packets of each class are served in rounds
- **Weighted Round-Robin (WRR) scheduling:** w_i packets of class i are served in each round.
  - WRR: no order is assumed within a round: worst-case when all packets of a class are served consecutively
  - Interleaved WRR (IWRR): one packet of each class is served at each time into sub-rounds.
- **Deficit Round-Robin (DRR):** the sharing is based on the quantity of data (a quantum at each round) rather than on the number of packets

NC objective: finding a per-class service curve



- $A_i(s,t) \le b_i + r_i(t-s) = \alpha_i(t-s)$
- $C(s,t) \ge R(t-s-T)_+ = \beta(t-s)$

$(x)_+ = \max(0,x)$.
$X(s,t) = X(t) - X(s)$.

- $A_i(s,t)$: amount of data of class i arrived during $[s,t)$;
- $D_i(s,t)$: amount of data of class i departed during $[s,t)$;
- $C(s,t)$: guarantee on the total amount of data offered during $[s,t)$.
- α_i are arrival curves, β a strict service curve

If the system is not empty during $(s,t]$, then $\sum_i D_i(s,t) \ge C(s,t)$.

Objective

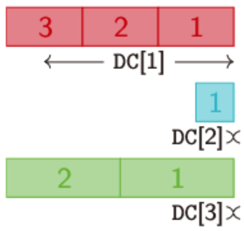
Find C_i and β_i such that for all $s \le t$, whenever there is always data of class i during $(s,t]$, $D_i(s,t) \ge C_i(s,t) \ge \beta_i(t-s)$.

Contents

DRR implementation

Given Q_c a quantum for each class c

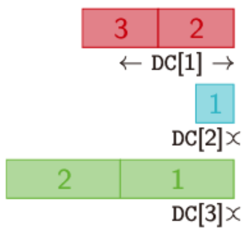
```
for c = 1 to n do DC[c] ← 0 ;
while True do
    for c = 1 to n do
        if not empty(c) then
            DC[c] ← DC[c] + Q_c ;
            while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
                send(head(c)) ;
                DC[c] ← DC[c] − size(head(c)) ;
                removeHead(c) ;
            if empty(c) then DC[c] ← 0 ;
```



DRR implementation

Given Q_c a quantum for each class c

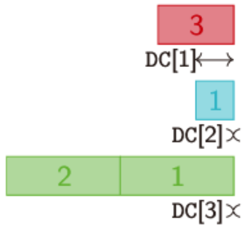
```
for c = 1 to n do DC[c] ← 0 ;
while True do
    for c = 1 to n do
        if not empty(c) then
            DC[c] ← DC[c] + Q_c ;
            while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
                send(head(c)) ;
                DC[c] ← DC[c] − size(head(c)) ;
                removeHead(c) ;
            if empty(c) then DC[c] ← 0 ;
```



DRR implementation

Given Q_c a quantum for each class c

```
for c = 1 to n do DC[c] ← 0 ;
while True do
    for c = 1 to n do
        if not empty(c) then
            DC[c] ← DC[c] + Q_c ;
            while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
                send(head(c)) ;
                DC[c] ← DC[c] − size(head(c)) ;
                removeHead(c) ;
            if empty(c) then DC[c] ← 0 ;
```

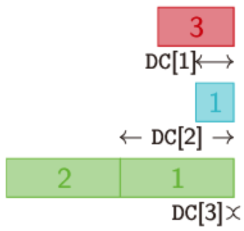


HUAWEI

DRR implementation

Given Q_c a quantum for each class c

```
for c = 1 to n do DC[c] ← 0 ;
while True do
    for c = 1 to n do
        if not empty(c) then
            DC[c] ← DC[c] + Q_c ;
            while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
                send(head(c)) ;
                DC[c] ← DC[c] − size(head(c)) ;
                removeHead(c) ;
            if empty(c) then DC[c] ← 0 ;
```

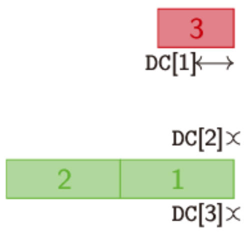


HUAWEI

DRR implementation

Given Q_c a quantum for each class c

```
for c = 1 to n do DC[c] ← 0 ;
while True do
    for c = 1 to n do
        if not empty(c) then
            DC[c] ← DC[c] + Q_c ;
            while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
                send(head(c)) ;
                DC[c] ← DC[c] − size(head(c)) ;
                removeHead(c) ;
            if empty(c) then DC[c] ← 0 ;
```

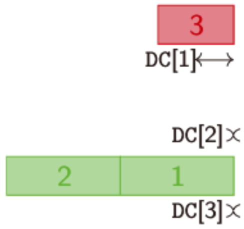


HUAWEI

DRR implementation

Given Q_c a quantum for each class c

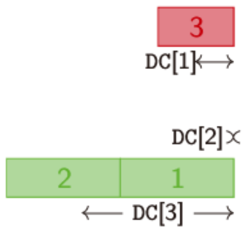
```
for c = 1 to n do DC[c] ← 0 ;
while True do
    for c = 1 to n do
        if not empty(c) then
            DC[c] ← DC[c] + Q_c ;
            while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
                send(head(c)) ;
                DC[c] ← DC[c] − size(head(c)) ;
                removeHead(c) ;
            if empty(c) then DC[c] ← 0 ;
```



DRR implementation

Given Q_c a quantum for each class c

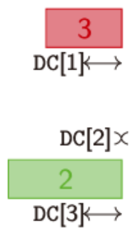
```
for c = 1 to n do DC[c] ← 0 ;
while True do
    for c = 1 to n do
        if not empty(c) then
            DC[c] ← DC[c] + Q_c ;
            while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
                send(head(c)) ;
                DC[c] ← DC[c] − size(head(c)) ;
                removeHead(c) ;
            if empty(c) then DC[c] ← 0 ;
```



DRR implementation

Given Q_c a quantum for each class c

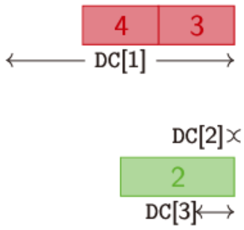
```
for c = 1 to n do DC[c] ← 0 ;
while True do
    for c = 1 to n do
        if not empty(c) then
            DC[c] ← DC[c] + Q_c ;
            while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
                send(head(c)) ;
                DC[c] ← DC[c] − size(head(c)) ;
                removeHead(c) ;
            if empty(c) then DC[c] ← 0 ;
```



DRR implementation

Given Q_c a quantum for each class c

```
for c = 1 to n do DC[c] ← 0 ;
while True do
  for c = 1 to n do
    if not empty(c) then
      DC[c] ← DC[c] + Q_c ;
      while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
        send(head(c)) ;
        DC[c] ← DC[c] − size(head(c)) ;
        removeHead(c) ;
      if empty(c) then DC[c] ← 0 ;
```

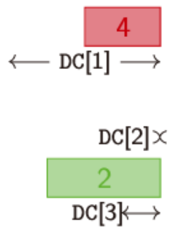


HUAWEI

DRR implementation

Given Q_c a quantum for each class c

```
for c = 1 to n do DC[c] ← 0 ;
while True do
  for c = 1 to n do
    if not empty(c) then
      DC[c] ← DC[c] + Q_c ;
      while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
        send(head(c)) ;
        DC[c] ← DC[c] − size(head(c)) ;
        removeHead(c) ;
      if empty(c) then DC[c] ← 0 ;
```

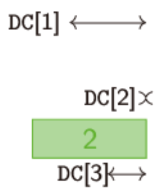


HUAWEI

DRR implementation

Given Q_c a quantum for each class c

```
for c = 1 to n do DC[c] ← 0 ;
while True do
  for c = 1 to n do
    if not empty(c) then
      DC[c] ← DC[c] + Q_c ;
      while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
        send(head(c)) ;
        DC[c] ← DC[c] − size(head(c)) ;
        removeHead(c) ;
      if empty(c) then DC[c] ← 0 ;
```

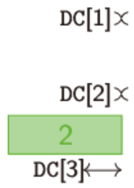


HUAWEI

DRR implementation

Given Q_c a quantum for each class c

```
for c = 1 to n do  DC[c] ← 0 ;
while True do
    for c = 1 to n do
        if not empty(c) then
            DC[c] ← DC[c] + Q_c ;
            while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
                send(head(c)) ;
                DC[c] ← DC[c] − size(head(c)) ;
                removeHead(c) ;
            if empty(c) then  DC[c] ← 0 ;
```



DRR implementation

Given Q_c a quantum for each class c

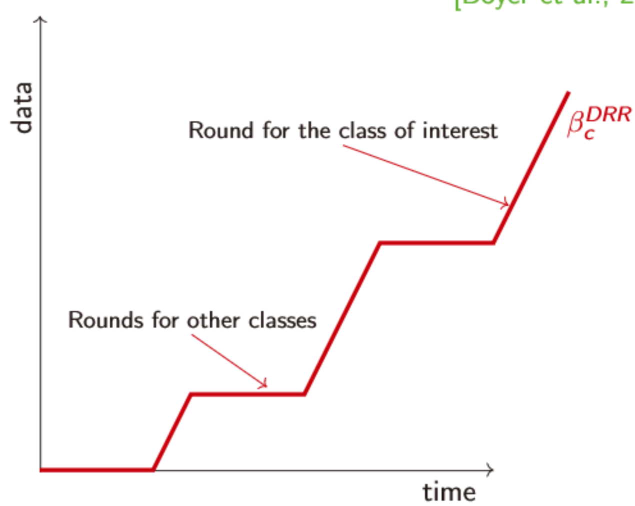
```
for c = 1 to n do  DC[c] ← 0 ;
while True do
    for c = 1 to n do
        if not empty(c) then
            DC[c] ← DC[c] + Q_c ;
            while (not empty(c)) and (size(head(c)) ≤ DC[c]) do
                send(head(c)) ;
                DC[c] ← DC[c] − size(head(c)) ;
                removeHead(c) ;
            if empty(c) then  DC[c] ← 0 ;
```

Property

If ℓ_c is the maximum length of a packet of class c, $0 \leq \mathtt{DC}[c] \leq Q_c + \ell_c$.

A strict service curve for DRR - traffic-agnostic

[Boyer et al., 2012] [Tabatabaee, Le Boudec, 2021]

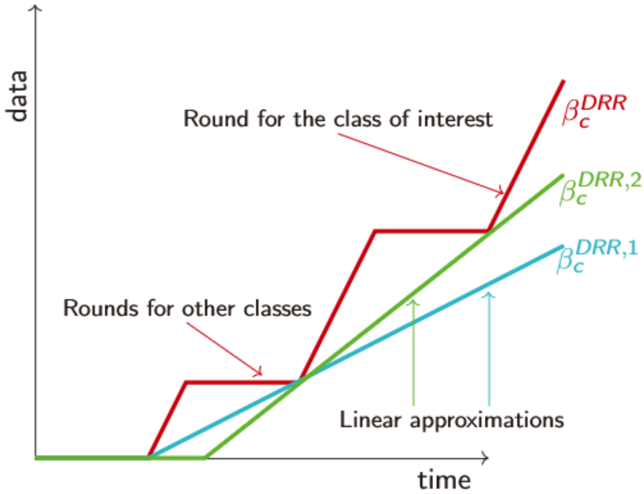


For a given class of traffic c:

- [Tabatabaee, Le Boudec, 2021]
  1. β_c^{DRR} is a strict service curve

A strict service curve for DRR - traffic-agnostic

[Boyer et al., 2012] [Tabatabaee, Le Boudec, 2021]



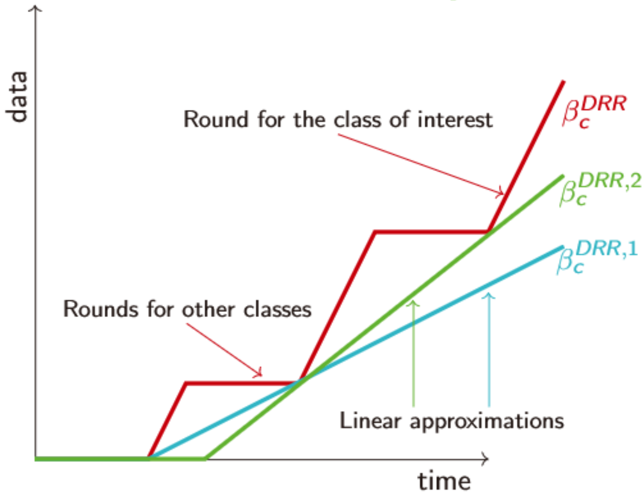
For a given class of traffic c:

- [Tabatabaee, Le Boudec, 2021]
  1. β_c^{DRR} is a strict service curve
  2. linear approximations: $\beta_c^{DRR,1} \vee \beta_c^{DRR,2}$ is a strict service curve

HUAWEI

A strict service curve for DRR - traffic-agnostic

[Boyer et al., 2012] [Tabatabaee, Le Boudec, 2021]



For a given class of traffic c:

- [Tabatabaee, Le Boudec, 2021]
  1. β_c^{DRR} is a strict service curve
  2. linear approximations: $\beta_c^{DRR,1} \vee \beta_c^{DRR,2}$ is a strict service curve
- [Boyer et al. 2012]
  - β_2 is a strict service curve

$$\beta_c^{DRR,2} = \frac{Q_c}{\sum_{c'} Q_{c'}}(\beta - L_c)_+$$

HUAWEI

Traffic-aware service curves

If a queue receive more bandwidth than its demand, the unused bandwidth can be shared by other classes.

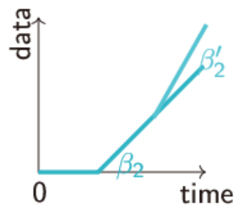
Perfect Bandwidth-sharing policies

For all backlogged period of class i, $\phi_j D_i(s,t) \geq \phi_i(D_j(s,t) - H_{i,j})_+$.

Theorem

There exist $(H_{i,M})_{i,M\subseteq\mathbb{N}_n}$ depending on $(\phi_j, H_{i,j})_{i,j}$ only such that a strict service curve for class i is

$$\beta_i = \sup_{M\subseteq\mathbb{N}_n\setminus\{i\}} \frac{\phi_i}{\Phi_{\overline{M}}}\Big(\beta - \sum_{j\in M}\alpha_j - H_{i,M}\Big)_+$$



This applies to DRR or WRR with fixed packet lengths:

- [Tabatabaee, Le Boudec 2021] is based on computing a fixed point using departure characterizations
- [B. 2021], GPS[Liebeherr, Burchart 2018], are based on computing times from which a flow does not use its allocated bandwidth.

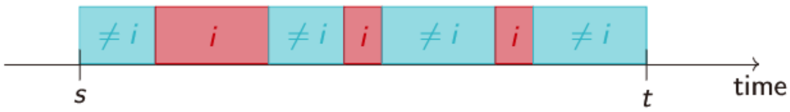
HUAWEI

Contents

Imperfect bandwidth-sharing: WRR

Class i servers w_i packets of length in $[\ell_i^{\min}, \ell_i^{\max}]$ in each round for each class i:

$$D_i(s,t) \geq \frac{w_i\ell_i^{\min}}{w_i\ell_i^{\min} + \sum_{j\notin i} w_j\ell_j^{\max}}(C(s,t) - \sum_{j\neq i} w_j\ell_j^{\max})_+.$$



When the queue is saturated, the share of the bandwidth is not exactly known and is in the interval

$$[\frac{w_i\ell_i^{\min}}{w_i\ell_i^{\min} + \sum_{j\notin i} w_j\ell_j^{\max}}, \frac{w_i\ell_i^{\max}}{w_i\ell_i^{\max} + \sum_{j\notin i} w_j\ell_j^{\min}}]$$

Using only packet sizes may not be enough to compute finite performance bounds even if the system is stable.

Imperfect bandwidth-sharing policy

Definition (Imperfect bandwidth-sharing policy (slightly simplified))

The server has an imperfect bandwidth-sharing policy if there exist non-negative numbers $\phi_i^{\min}$, $\phi_i^{\max}$, $H_{i,j}$, such that for all classes i, for all backlogged period $(s,t]$ of class i, for all classes j,

$$\phi_j^{\max} D_i(s,t) \geq \phi_i^{\min}(D_j(s,t) - H_{i,j})_+.$$

Example

- WRR: $w_j\ell_j^{\max} D_i(s,t) \geq w_i\ell_i^{\min}(D_j(s,t) - w_j\ell_j^{\max})_+$
- IWRR: $w_j\ell_j^{\max} D_i(s,t) \geq w_i\ell_i^{\min}(D_j(s,t) - h_{i,j}\ell_j^{\max})_+$,
  with $h_{i,j} = w_j - w_i$ if $w_j > w_i$ and $h_{i,j} = w_j\big(1 - \frac{w_j - 1}{w_i}\big)$ if $w_i \geq w_j$.

Per-class service guarantees

Lemma

If β is a strict service curve for the server, then for all i,

$$\beta_i = \frac{\phi_i^{\min}}{\phi_i^{\min} + \sum_{j \notin i} \phi_j^{\max}} (\beta - \sum_{j \neq i} H_{i,j})_+.$$

is strict service curve for class i.

Stability issue

If $(\phi_i^{\max}) \gg (\phi_i^{\min})$, we can have $r_i > \frac{\phi_i^{\min}}{\phi_i^{\min} + \sum_{j \notin i} \phi_j^{\max}} R$ for all i.
We can obtain infinite delays for all classes, even if the system is stable ($R > \sum_i r_i$).

R: service rate of the server
r_i: arrival rate of class i.

Computation of traffic-aware per-class service guarantees

1. Per-class guarantees: Imperfect bandwidth-sharing is also valid for any subset of classes if the server: if β_M^{ag} is a ssc for classes in M,

$$\beta_i = \frac{\phi_i^{\min}}{\phi_i^{\min} + \sum_{j \in M \setminus \{i\}} \phi_j^{\max}} (\beta_M^{ag} - \sum_{j \in M \setminus \{i\}} H_{i,j})_+.$$

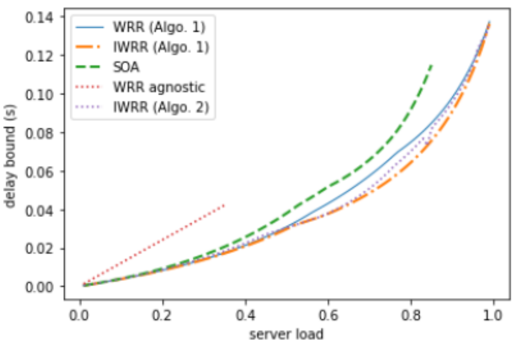
2. Residual service for a set of flows: If per-class guarantees are knows for a subset M of classes, an aggregate service for the other classes can be computed, there exists $q_M \in \mathbb{R}_+ \cup \{+\infty\}$ such that

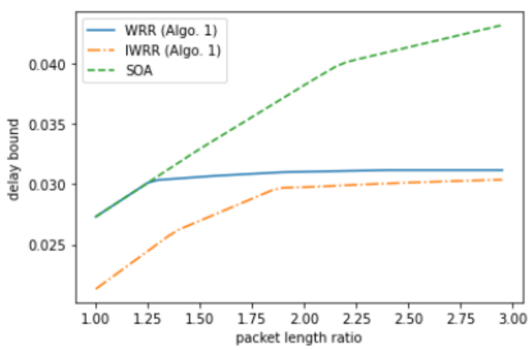
$$\beta_{\overline{M}}^{ag}(s,t) \geq \left(\beta - \sum_{j \in M} \alpha_j + q_M\right)_+.$$

3. Solving the stability issue: Bounding the maximum backlog at the start of a backlogged period of a class stabilizes the computed bounds: one can replace q_M by $\min(B_M, q_M) \in \mathbb{R}_+$

Numerical evaluation

class	1	2	3	4
b_i (b)	30208	19968	24576	27648
r_i ($\mathrm{Mb.s}^{-1}$)	0.65	0.85	0.95	0.55
$\ell_i^{\min}$ (b)	4096	3072	4608	3072
$\ell_i^{\max}$ (b)	8704	5632	6656	8192
w_i	4	6	7	10





Content

Analysis of FIFO networks

- DRR analysis ⟶ service curves for each server and each class of traffic
- FIFO inside each DRR class ⟶ analysis of FIFO networks

Modular analysis

Analyze the network node-per-node

- Total Flow Analysis (TFA)
- Separated Flow Analysis (SFA)

Advantages:

- Fast computations

Limits:

- Lack of accuracy, especially in cyclic networks

Global analysis

Analyze the network as a global system

- Least upper delay bound (LUDB)
- Linear programming (LP, PLP...)

Advantages:

- Improved accuracy, especially in cyclic networks

Limits:

- Computation time

Analysis of FIFO networks

- DRR analysis ⟶ service curves for each server and each class of traffic
- FIFO inside each DRR class ⟶ analysis of FIFO networks

Modular analysis

Analyze the network node-per-node

- **Total Flow Analysis (TFA)**
- Separated Flow Analysis (SFA)

Advantages:

- Fast computations

Limits:

- Lack of accuracy, especially in cyclic networks

Global analysis

Analyze the network as a global system

- Least upper delay bound (LUDB)
- **Linear programming (LP, PLP...)**

Advantages:

- Improved accuracy, especially in cyclic networks

Limits:

- Computation time

Modular analysis: Total flow analysis (TFA)

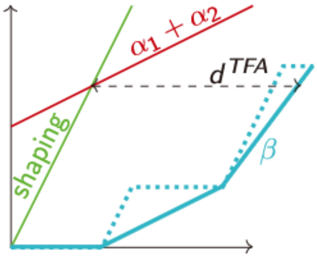
[Grieux 2004], [Mifdaoui, Leydier 2017], [Thomas et al. 2019]

Ideas

1. The worst-case delay in a FIFO server is the same for all flows crossing it
2. The delay can be used to propagate the burstiness of the traffic

$\alpha_1(t) = b_1 + r_1 t$

$\alpha_2(t) = b_2 + r_2 t$



$\alpha_1^d = \alpha_1 + d^{TFA} r_1$

$\alpha_2^d = \alpha_2 + d^{TFA} r_2$

HUAWEI

Modular analysis: Total flow analysis (TFA)

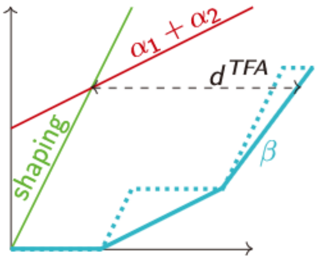
[Grieux 2004], [Mifdaoui, Leydier 2017], [Thomas et al. 2019]

Ideas

1. The worst-case delay in a FIFO server is the same for all flows crossing it
2. The delay can be used to propagate the burstiness of the traffic

$\alpha_1(t) = b_1 + r_1 t$

$\alpha_2(t) = b_2 + r_2 t$



$\alpha_1^d = \alpha_1 + d^{TFA} r_1$

$\alpha_2^d = \alpha_2 + d^{TFA} r_2$

3. End-to-end delay: sum of the delays on the path
4. Cyclic dependencies: compute a fixed point on the delays.

HUAWEI

Global analysis: polynomial-size linear programming (PLP)

Main idea

[Bouillard, Stea 2012], [Bouillard 2021]

- Write a linear program where
  - There is a finite set of times $\mathbf{t}$ of interests, regarding the processes of data on the network $\mathbf{A}_i^{(j)}\mathbf{t}$, $\mathbf{D}_i^{(j)}\mathbf{t}$
  - Write the network calculus constraints at those dates as linear constraints:
    1. Arrival constraints of flow i: $\mathbf{A}_i^{(0)}\mathbf{t} - \mathbf{A}_i^{(0)}\mathbf{t}' \leq b_i + r_i(\mathbf{t} - \mathbf{t}')$
    2. Service constraints at server j: $\sum_i \mathbf{D}_i^{(j)}\mathbf{t} - \mathbf{A}_i^{(j)}\mathbf{t}' \geq R_j(\mathbf{t} - \mathbf{t}' - L_j)_+$
    3. FIFO constraints, non-decreasing constraints...
  - **Objective:** maximizing the sojourn time of a bit of data in the system.

Can also compute

- The burstiness of a departure process

HUAWEI

Global analysis: polynomial-size linear programming (PLP)

[Bouillard, Stea 2012], [Bouillard 2021]

Main idea

- Write a linear program where
  - There is a finite set of times $\mathbf{t}$ of interests, regarding the processes of data on the network $\mathbf{A}_i^{(j)}\mathbf{t}$, $\mathbf{D}_i^{(j)}\mathbf{t}$
  - Write the network calculus constraints at those dates as linear constraints:
    1. Arrival constraints of flow i: $\mathbf{A}_i^{(0)}\mathbf{t} - \mathbf{A}_i^{(0)}\mathbf{t}' \leq b_i + r_i(\mathbf{t} - \mathbf{t}')$
    2. Service constraints at server j: $\sum_i \mathbf{D}_i^{(j)}\mathbf{t} - \mathbf{A}_i^{(j)}\mathbf{t}' \geq R_j(\mathbf{t} - \mathbf{t}' - L_j)_+$
    3. FIFO constraints, non-decreasing constraints...

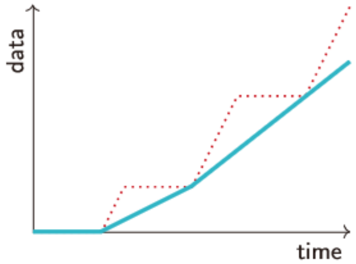
    PLP Tradeoff between accuracy and tractability (i.e. obtain a polynomial-size algorithm): replace some service curve constraints by TFA delay constraints.
  - **Objective:** maximizing the sojourn time of a bit of data in the system.

Can also compute

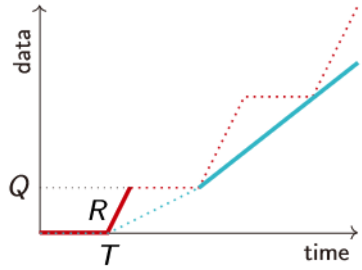
- The burstiness of a departure process

iPLP: PLP for non-convex functions

PLP requires piece-wise linear convex service curves



Introducing an integer variable: we can model the first round



$$\mathbf{Dt} - \mathbf{At}' \geq R(\mathbf{t} - \mathbf{t}' - T) - M\mathbf{b}$$
$$\mathbf{Dt} - \mathbf{At}' \leq Q + M\mathbf{b}$$
$$\mathbf{Dt} - \mathbf{At}' \geq Q - M(1 - \mathbf{b})$$

Content

Analyses in the state ot the art

- Use TFA analysis only

Agnostic analysis:

use only $(Q_c)_c$, $(\ell_c)_c$ to compute the per-class service curves

1. Compute the strict service curve for each class of traffic
2. Analyze each class independently

Analyses in the state ot the art

- Use TFA analysis only

Agnostic analysis:

use only $(Q_c)_c$, $(\ell_c)_c$ to compute the per-class service curves

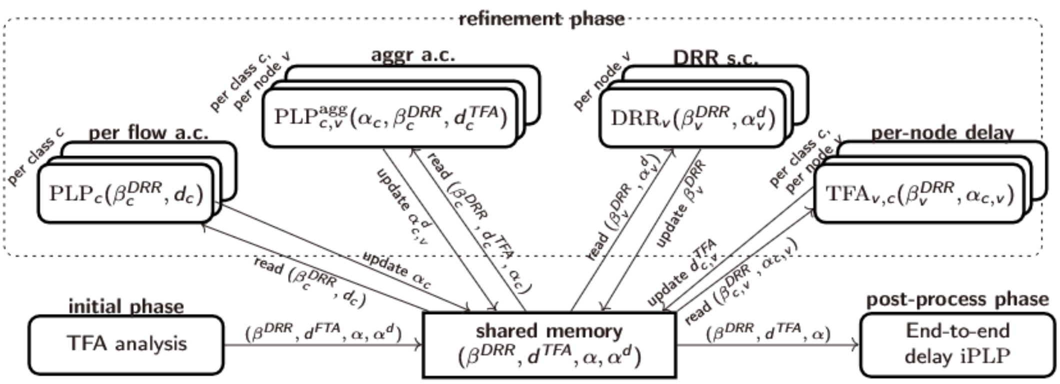
1. Compute the strict service curve for each class of traffic
2. Analyze each class independently

Feed-forward networks:

Use the arrival curves of the cross traffic, and proceed in topological order on the nodes

1. Compute the arrival curves for each class in each server (depends on the preceding servers)
2. Use the iterative scheme to compute the residual service curves for each curves
3. Compute the delay bounds, and propagate the output arrival curves for each flow

The iterative scheme



The main result

Theorem

Init. *Valid bounds and curves are computed at the initial phase*

Ref. *Given these valid bounds:*

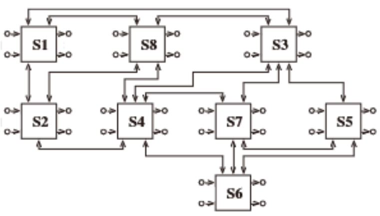
- *After each update of a function of the refinement phase, the bounds are valid and improved*
- *The memory converges to some fixed point, independent of the order of the operations*
- *any stopping criterion permits to obtain valid bounds*

Fin. *Given valid bounds of the refinement phase, the final phase computes delay upper bounds for all flows of the network*

Strategies for fast convergence

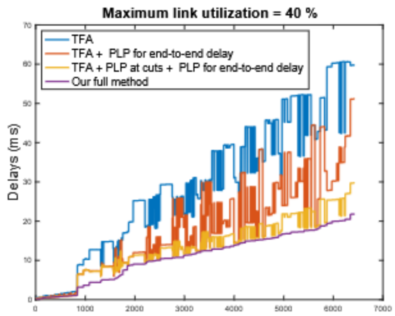
- Alternate PLP with the other (faster) functions in parallel

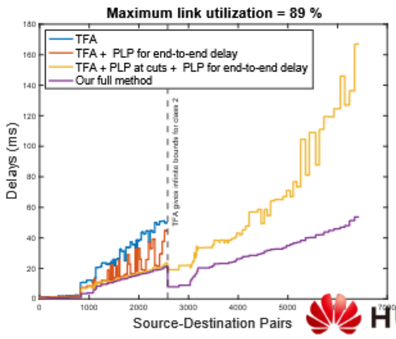
Experimental result: industrial-size network



- 96 end-systems, 8 switches, 984 flows
- service rate: 1 Gb/s; switching latency: 16μs.
- Three classes of flows: (1) critical, (2) multimedia and (3) best-effort.

class	1	2	3
flows	128	500	266
quantum	3070	1535	1535





Conclusion

Summary

- A NC service curve model for the bandwidth-sharing scheduling policies
- An iterative scheme to improve the computation of delay bounds in DRR networks
  - Allow parallel computations
  - Improves delay bounds by up to 66% for medium link utilization
  - Improves stability region for high load utilization
- Some improvements and generalization of linear-programming techniques

Future work

- Measure the quality of the bounds vs. lower bounds or simulation

Thank you.

Bring digital to every person, home and organization for a fully, connected, intelligent world.