ACM SIGCOMM 2019 Tutorial on Modeling and Analysis of Network Infrastructure in Cyber-Physical Systems

Dr. Liang Cheng * and Dr. Steffen Bondorf †

* Department of Computer Science and Engineering, Lehigh University
Bethlehem, Pennsylvania 18015, USA

† Department of Information Security and Communication Technology,
Norwegian University of Science and Technology, NO-7034, Trondheim, Norway

* cheng@lehigh.edu, † steffen.bondorf@ntnu.no

Cyber-Physical Systems

"engineered systems that are built from, and depend upon, the seamless integration of computation and physical components" *

- a wide assortment of distributed cyber-physical systems (CPS) built upon network infrastructure for mission-critical applications
  - industrial process control systems and avionics

* NSF Cyber-physical System Program, https://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503286.

CPS Network Infrastructure

- A convergence of CPS network infrastructure design
  - Fieldbus technologies, e.g. Profibus in factory automation
    - Increasing demands on data rates and interoperability
    - Ethernet-based solutions replacing bus technologies
  - Avionics
    - Airbus developed the Ethernet-based on-board communication system AFDX (Avionics Full-Duplex Ethernet) that has been adopted by its competitors Boeing and Bombardier.
  - IEEE AVB
    - Traffic shaping, scheduling and path reservation for time-synchronized low latency streaming services
    - IEEE TSN (Time-Sensitive Networking) working group
  - IETF
    - Deterministic Networking (DetNet) working group

Dr. Liang Cheng: http://liangcheng.info
Dr. Steffen Bondorf: http://steffenbondorf.com

Network Analysis

- **Systems theories for network analysis**
  - **Queueing theory**
    - Average: How long does a customer expect to wait in the queue before they are served? What is the average length of the queue?
    - Probability: Little's Law/Theorem, Kleinrock independence approximation, and Jackson's Theorem
  - **Network calculus**
    - Gives a theoretical framework for analyzing performance guarantees in computer networks based on the min-plus and max-plus algebras
      - A theory of deterministic queueing systems
    - Worst-case bounds on delay and buffer requirements in a network can be computed.

Min-Plus Algebra

- Conventional algebra works with the algebraic structure $(R, +, \times)$; min-plus algebra works with the algebraic structure $(R \cup \{+\infty\}, \wedge, +)$, where $\wedge$ is the infimum operator

- Closure of $\wedge$
  $\forall\, a,b \in R\cup\{+\infty\},\ a\wedge b \in R\cup\{+\infty\}$
- Associativity of $\wedge$
  $\forall\, a,b,c \in R\cup\{+\infty\},\ (a\wedge b)\wedge c = a\wedge(b\wedge c)$
- Neutral element for $\wedge$
  $\exists\, e \in R\cup\{+\infty\}: \forall\, a \in R\cup\{+\infty\},\ a\wedge e = a$
- Commutativity of $\wedge$
  $\forall\, a,b \in R\cup\{+\infty\},\ a\wedge b = b\wedge a$
- Idempotency of $\wedge$
  $\forall\, a \in R\cup\{+\infty\},\ a\wedge a = a$

- Closure of +
  $\forall\, a,b \in R\cup\{+\infty\},\ a+b \in R\cup\{+\infty\}$
- Associativity of +
  $\forall\, a,b,c \in R\cup\{+\infty\},\ (a+b)+c = a+(b+c)$
- Neutral element for +
  $\exists\, e \in R\cup\{+\infty\}: \forall\, a \in R\cup\{+\infty\},\ a+e = a$
- Commutativity of +
  $\forall\, a,b \in R\cup\{+\infty\},\ a+b = b+a$
- Distributivity of + w.r.t. $\wedge$
  $\forall a,b,c \in R\cup\{+\infty\},\ (a\wedge b)+c=(a+c)\wedge(b+c)$

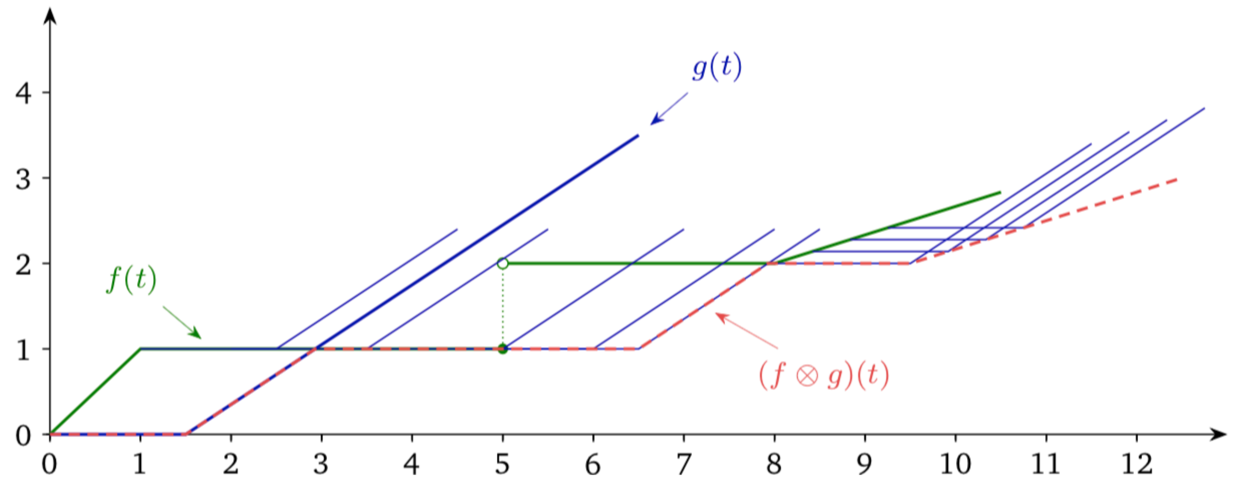
Min-Plus Convolution

- Conventional convolution

$$(f * g)(t) = \int_0^t f(t-s)g(s)\, ds$$

- Min-plus convolution

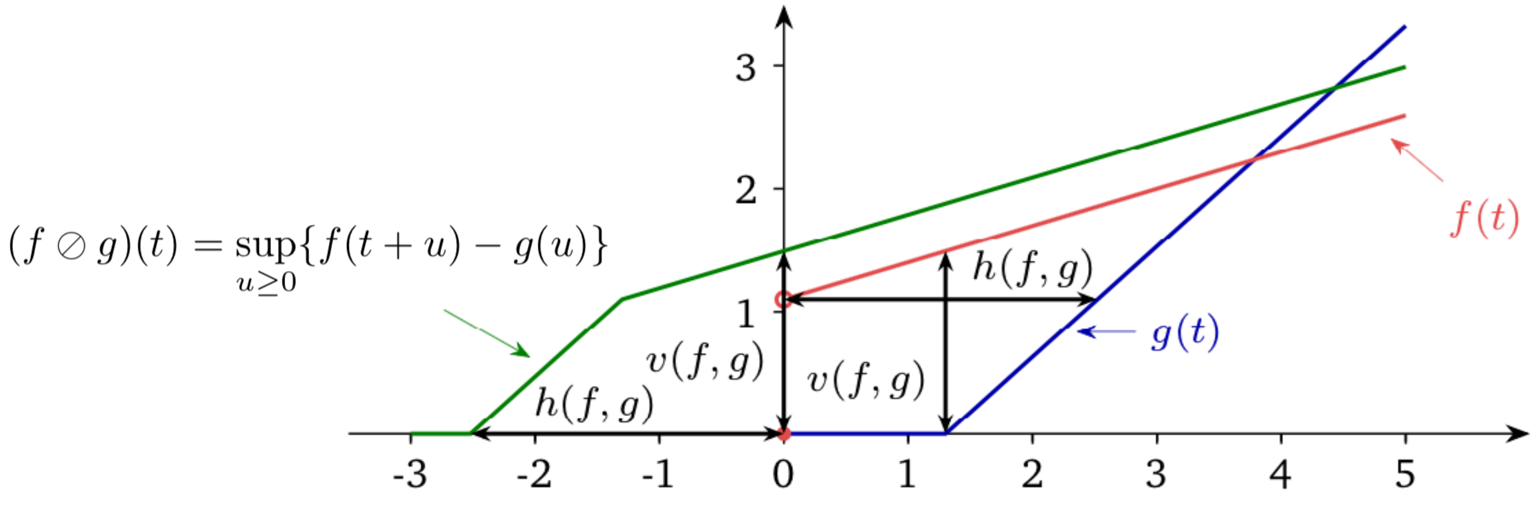
$$(f \otimes g)(t) = \inf_{0 \le s \le t} \{f(t-s) + g(s)\}$$



Figures by Amaury Van Beaten and Wolfgang Kellerer, Network Calculus: A Comprehensive Guide, Technical Report No. 201603, Technische Universitat Munchen, October 8, 2016

Min-Plus Deconvolution

- Min-plus deconvolution $(f \oslash g)(t) = \sup_{u \geq 0}\{f(t+u) - g(u)\}$



$$v(f,g) = (f \oslash g)(0)$$

$$h(f, g) = ?$$

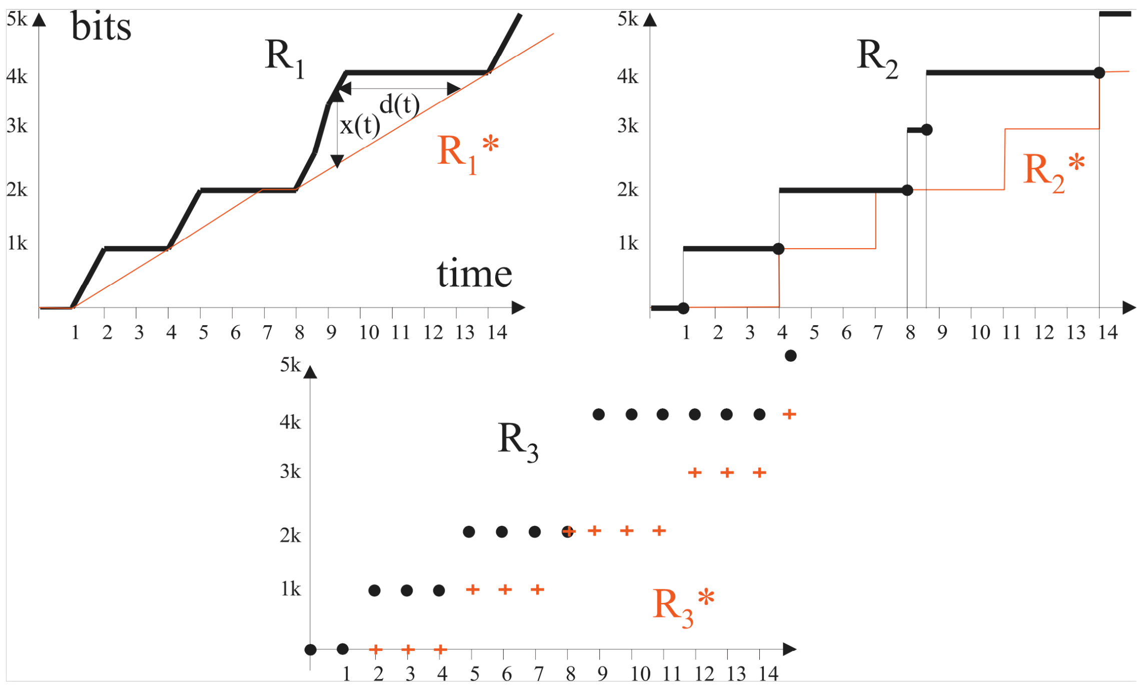
Figures by Amaury Van Bemten and Wolfgang Kellerer, Network Calculus: A Comprehensive Guide, Technical Report No. 201603, Technische Universität München, October 8, 2016

Network Calculus (1)

- **Data flow models**
  - Example: packets arrive at times 1, 4, 8, 8.6, and 14

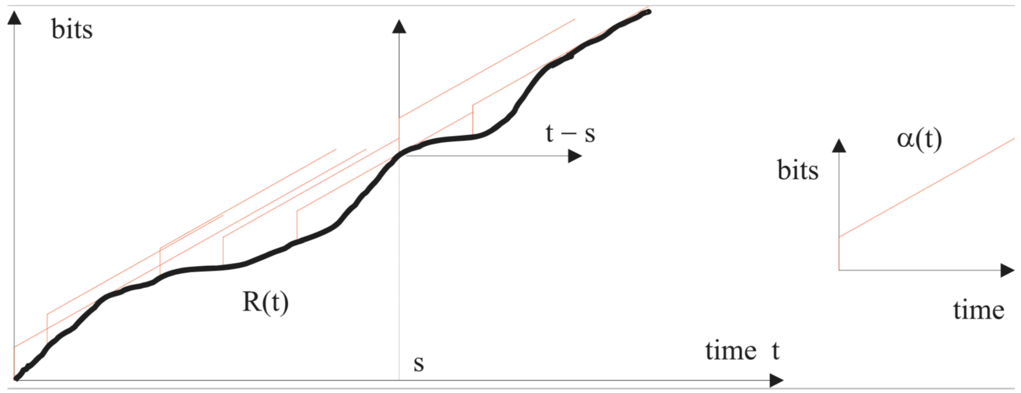
- **Continuous function of continuous time**
- **Discontinuous function of continuous time**
- **Discontinuous function of discontinuous time**

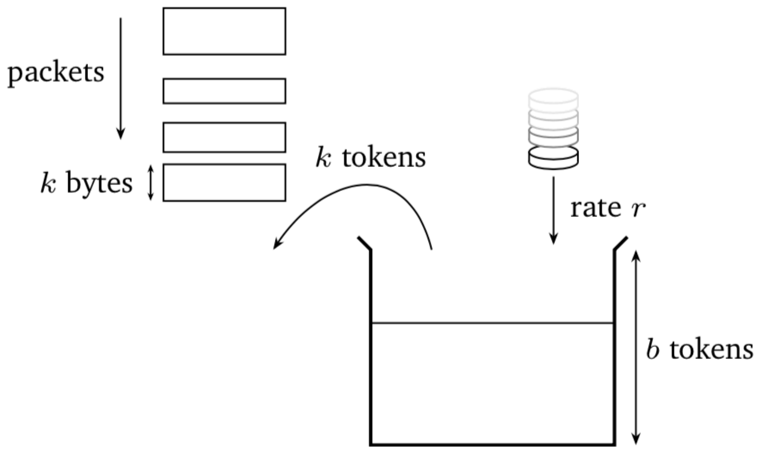
J.-Y. L. Boudec and P. Thiran, Network Calculus: a theory of deterministic queuing systems for the Internet, Lecture Notes in Computer Science, Vol. 2050, Springer, 2001.



Network Calculus (2)

- **Data flow models**
  - Arrival curve R(t)-R(s) <= $\alpha(t-s)$, for any t>=s>=0
  - Affine arrival curve $\alpha_{r,b}(t)=rt+b$ for $t>0$ and 0 otherwise

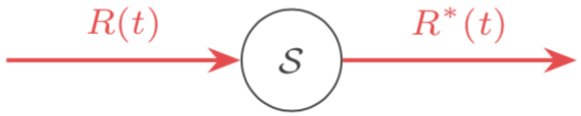




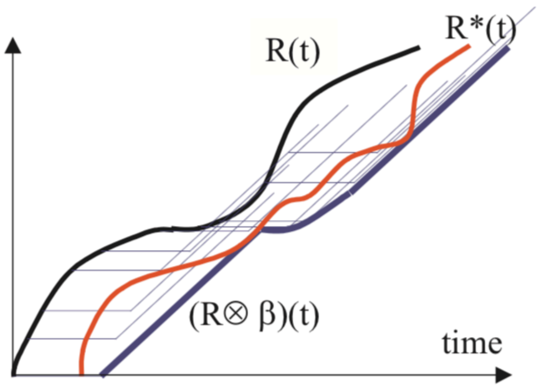
J.-Y. L. Boudec and P. Thiran, Network Calculus: a theory of deterministic queuing systems for the Internet, Lecture Notes in Computer Science, Vol. 2050, Springer, 2001.

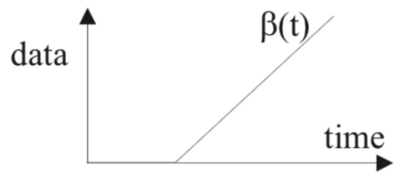
Network Calculus (3)

- **Data flow models**
  - Arrival curves: fluid model, general continuous time model
- **Node model**



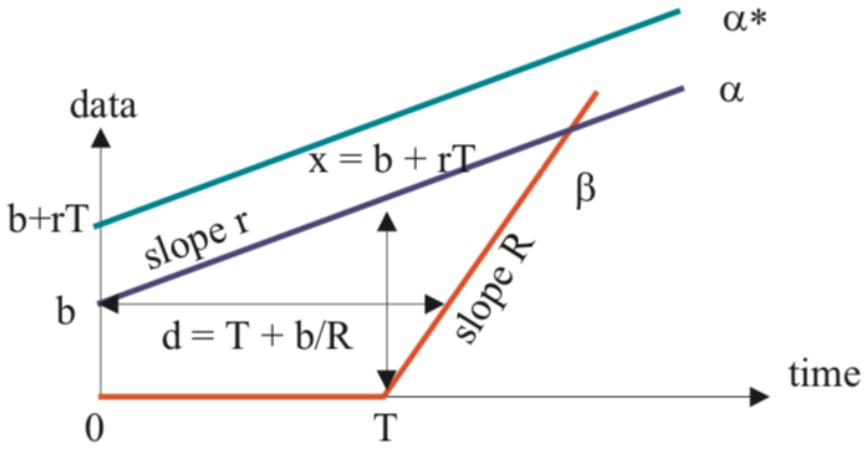
  - S offers to the flow a service curve β if and only if β is wide sense increasing and $\beta(0)=0$ and $R^* \geq R \otimes \beta$
  - S offers to the flow a maximum service curve γ if and only if γ is wide sense increasing and $R^* \leq R \otimes \gamma$
  - S offers to the flow a strict service curve δ if and only if during any backlogged period of duration u, the output of the flow is at least equal to $\delta(u)$.





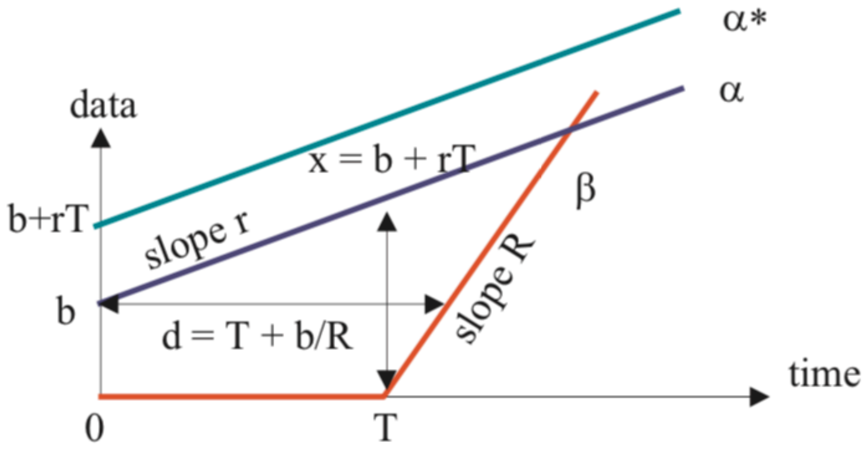
Network Calculus (4)

- **Data flow and node models**
  - Arrival curves (leaky-bucket)
  - Service curves (rate-latency)
- **Backlog**
  - R(t)-R*(t)
- **Virtual delay**
  - d(t) = inf{τ>=0: R(t)<=R*(t+τ)}
- **Backlog bound**
  - R(t)-R*(t) ≤ sup{α(s)-β(s)} for s>=0
- **Virtual delay bound**
  - *d*(t) <= h(α,β)



Network Calculus (5)

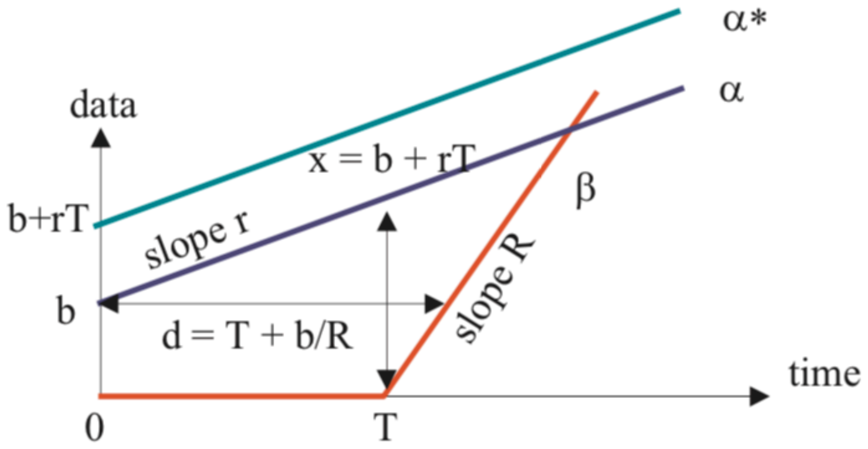
- **Data flow and node models**
  - Arrival curves (leaky-bucket)
  - Service curves (rate-latency)
- **Backlog**
  - R(t)-R*(t)
- **Virtual delay**
  - d(t) = inf{τ>=0: R(t)<=R*(t+τ)}



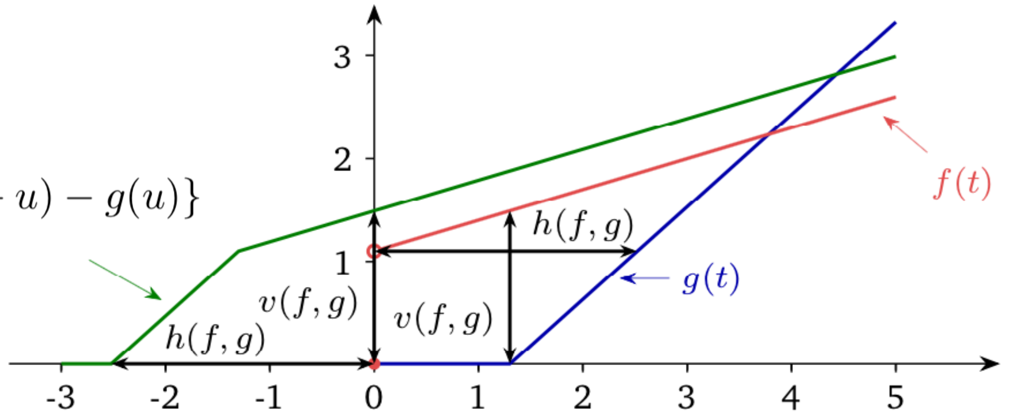
- What is the buffer bound, i.e. the maximum number of bytes stored in the buffer at any time?
- What is the delay bound, i.e. the maximum delay experience by any bit?
- What is the burstiness of the output flow?

Network Calculus (6)

- **Data flow and node models**
  - Arrival curves (leaky-bucket)
  - Service curves (rate-latency)
- **Backlog**
  - R(t)-R*(t)
- **Virtual delay**
  - d(t) = inf{τ>=0: R(t)<=R*(t+τ)}

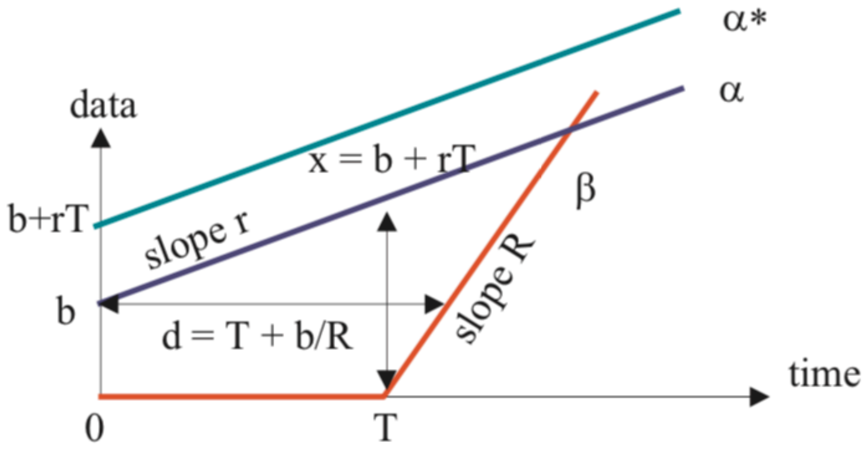


$$(f \oslash g)(t) = \sup_{u \geq 0}\{f(t+u) - g(u)\}$$



Network Calculus (7)

- **Data flow and node models**
  - Arrival curves (leaky-bucket)
  - Service curves (rate-latency)
- **Backlog**
  - R(t)-R*(t)
- **Virtual delay**
  - d(t) = inf{τ>=0: R(t)<=R*(t+τ)}

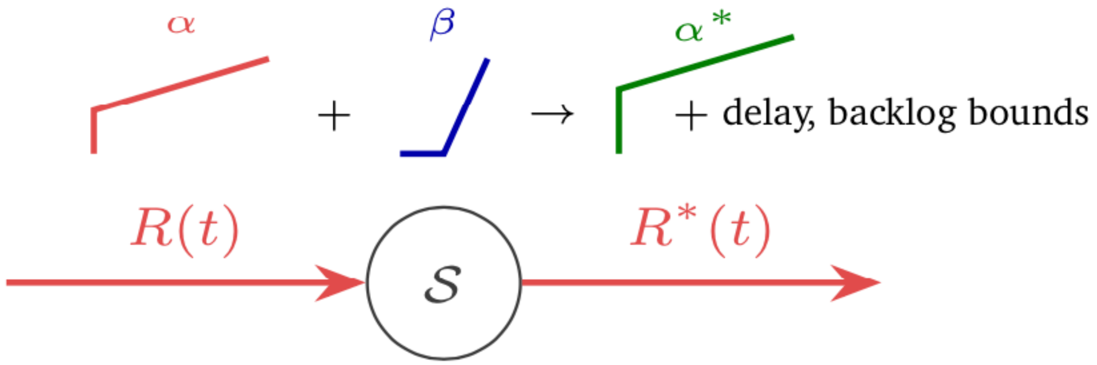


- **Output flow**
  - Assume a flow, constrained by an arrival curve α, travels a system that offers a service curve β. The output flow is constrained by the arrival curve $\alpha^* = \alpha \oslash \beta$.

Network Calculus (8)

From the arrival curve α of a flow and the service curve β of a node, network calculus theory allows to compute an upper bound of

- the backlog generated by the flow at this node
- the virtual delay the flow will experience at the node
- the new arrival curve α^* of the flow at the output of the node



An α-smooth flow traversing a node with service curve β and maximum service curve γ gets out of the node with an arrival curve α^* given by

$$\alpha^* = (\alpha \otimes \gamma) \oslash \beta$$

Case I. Sensor Networks

Sensor Network Calculus (SensorNC) for CPS:

CPS can be viewed as closed-loop systems, in terms of control theory, such that the sensing often becomes time-critical for timely actuation.

SensorNC is a framework continuously developed since 2005 to support the predictable design, control, and management of large-scale wireless sensor networks with timing constraints

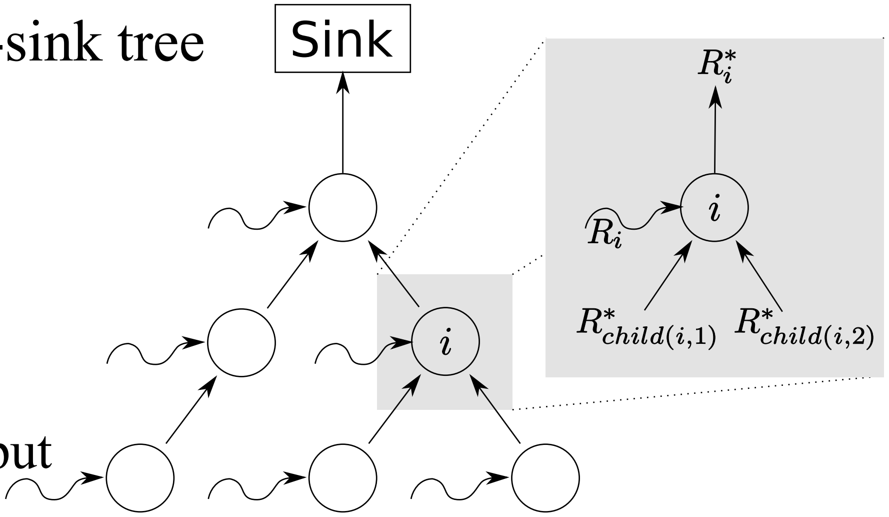
- Mathematically, it can be viewed as a special instance of the general network calculus presented before
- SensorNC was customized in several dimensions and also extended over the general network calculus to capture the special requirements of WSNs

Seminal Paper: Jens Schmitt and Utz Roedig: Sensor Network Calculus – A Framework for Worst Case Analysis. In the *Proceedings of The First IEEE International Conference Distributed Computing in Sensor Systems (DCOSS)*, 2005.

Recent Overview: Jens Schmitt, Steffen Bondorf and Wint Yi Poe: The Sensor Network Calculus as Key to the Design of Wireless Sensor Networks with Predictable Performance. *Journal of Sensor and Actuator Networks*. 2017; 6(3):21.

SensorNC Model

System Model: single-sink tree
(more complex topologies can be treated as overlapping sink-trees)

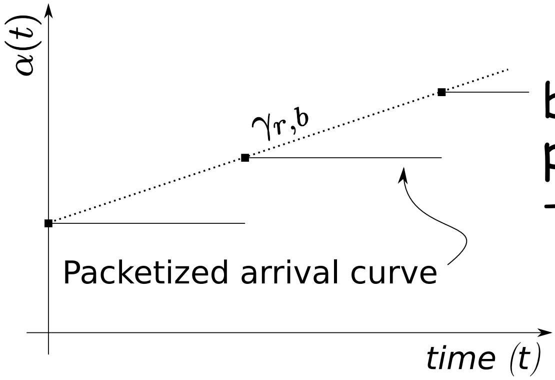


sensed input

node *output* towards sink

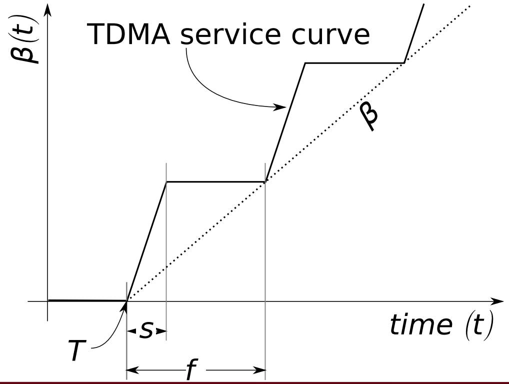
node *input* from sensing and child nodes

Periodic data generation



b = packet size
p = period
-> worst-case:
initial burst b
rate r = b/p

TDMA medium access



C = Capacity
f = frame duration
s = time per node
-> worst-case:
initial delay T=f-s
rate R = (s/f)*C

End-to-End Delay Bounds (1)

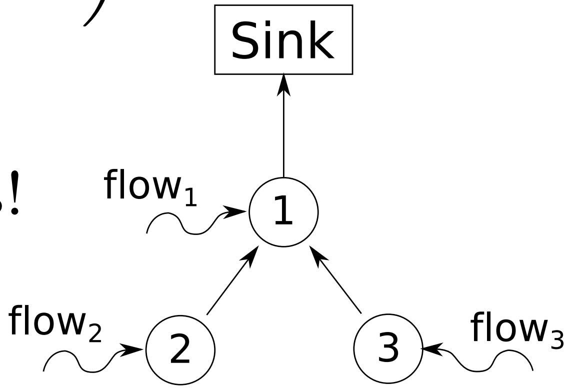
Deconvolution computes node-local delay bounds, we add them up to a flow's end-to-end bound:

$$D^i_{\text{end-to-end}} = \sum_{j \in P(i)} D_j$$

where *i* denotes a the flow, *j* is a node on its path *P(i)* and D_j the delay at node *j*:

$$D_j = h\left(\alpha^{\text{flow}_j} + \sum_{k \in \text{child}(j)} \alpha^*_k,\ \ \beta_j\right)$$

Internal arrival curves are required at non-leaves!
In this example:
an arrival curve for all flows at node 1



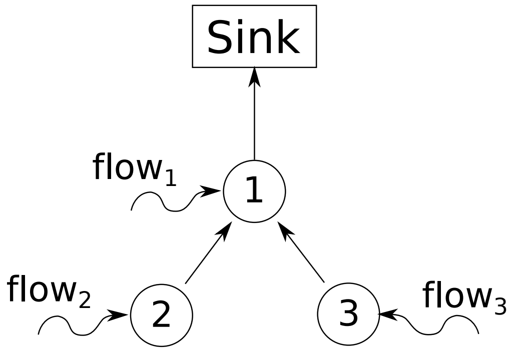
End-to-End Delay Bounds (2)

Internal arrival curves

an algorithm based on the output bound computation:

$$\alpha_k^* = \alpha_k \oslash \beta_k = \left(\alpha^{\text{flow}_k} + \sum_{l \in \text{child}(k)} \alpha_l^*\right) \oslash \beta_k \qquad (2)$$

Example: arrivals at node 1



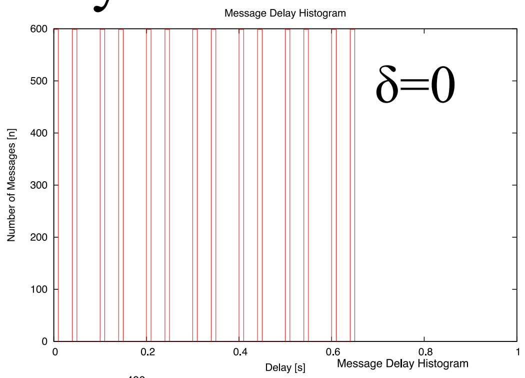
Algorithm 1 Calculating the internal flows of a network.

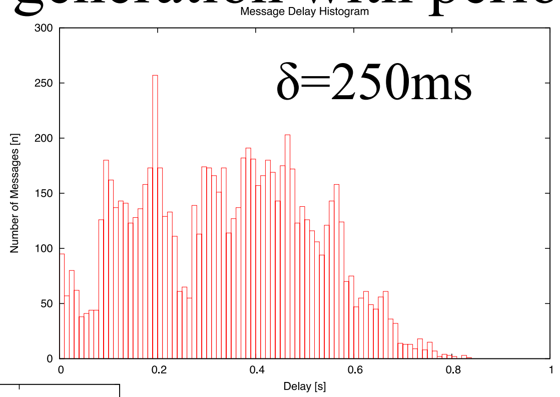
1. Let us assume that arrival curves for the sensed input α_i and service curves β_i for sensor node i, $i = 1, \ldots, n$, are given.
2. For all leaf nodes the output bound α_i^* can be calculated as $\alpha_i^* = \alpha_i \oslash \beta_i$. Each leaf node is now marked as "calculated".
3. For all nodes only having children which are marked "calculated" the output bound α_i^* can be calculated according to (2) and they can again be marked "calculated".
4. If node 1 is marked "calculated" the algorithm terminates, otherwise go to step 3.

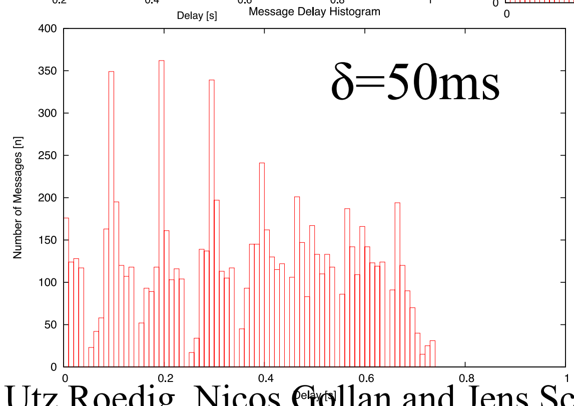
$$\alpha_1 = \alpha^{\text{flow}_1} + \alpha_2^* + \alpha_3^* = \alpha^{\text{flow}_1} + (\alpha^{\text{flow}_2} \oslash \beta_2) + (\alpha^{\text{flow}_3} \oslash \beta_3)$$

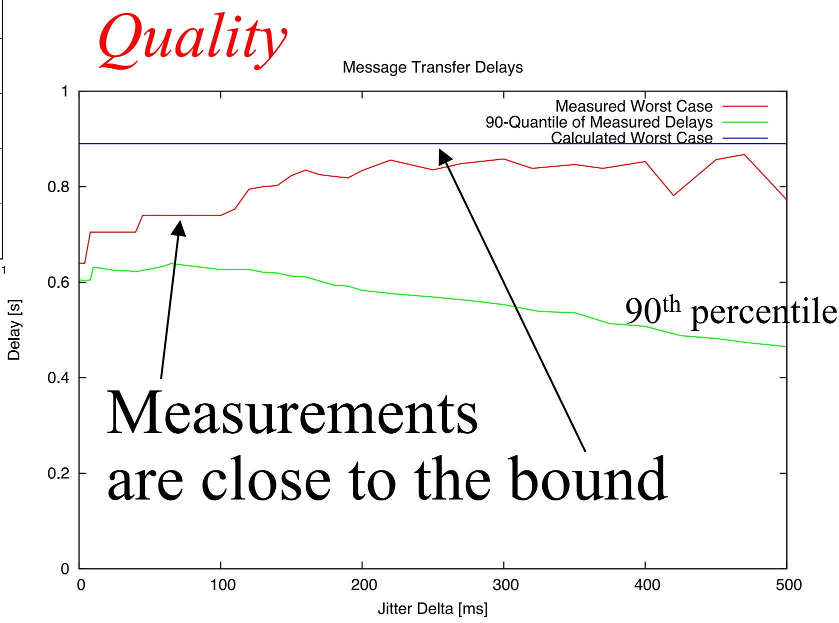
Quality of Delay Bounds

Validation by Experiments ZigBee implementation of TDMA service, 15-node tree network of depth three (details are in *), synchronized data generation with period 0.1s and jitter $0 \leq \delta \leq 0.5$s.









* Utz Roedig, Nicos Gollan and Jens Schmitt: Validating the Sensor Network Calculus by Simulations. In the *Proceedings of The Third International Conference on Wireless Internet (WICON), 2007.*

Computational Demand (1)

Computing *internal arrival curves* is based on the output bound

$$\alpha_k^* = \left(\alpha^{\text{flow}_k} + \sum_{l \in \text{child}(k)} \alpha_l^*\right) \oslash \beta_k$$

and ¤ shows that for simple SensorNC curves, it is distributive:

$$\left(\alpha^{f_1} + \alpha^{f_2}\right) \oslash \beta = \left(\alpha^{f_1} \oslash \beta\right) + \left(\alpha^{f_2} \oslash \beta\right)$$

Unfolding the output bound recursively and applying the above distributivity rule and $(\alpha \oslash \beta_1) \oslash \beta_2 = \alpha \oslash (\beta_1 \otimes \beta_2)$,

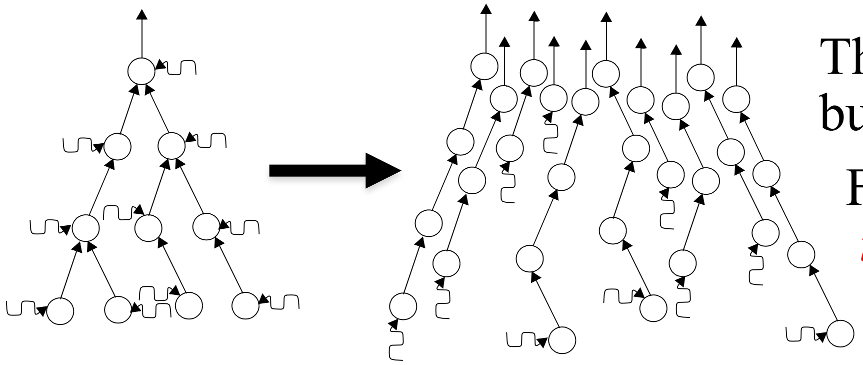
we obtain a simple, yet complete equation:

$$\alpha_k = \sum_{\text{flow}_i \in k} \left(\alpha^{\text{flow}_i} \oslash \Big(\bigotimes_{j \in P(i)} \beta_j \Big)\right)$$

¤ Steffen Bondorf and Jens Schmitt: Boosting Sensor Network Calculus by Thoroughly Bounding Cross-Traffic. In the *Proceedings of The 34th IEEE International Conference on Computer Communications* (INFOCOM), 2015.

Computational Demand (2)

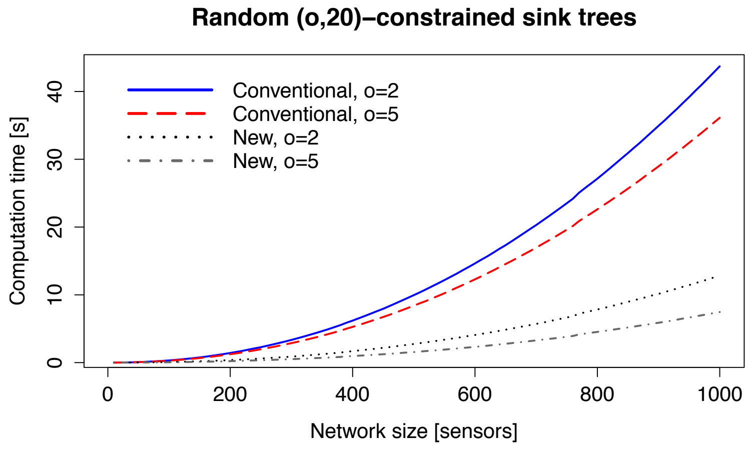
$$\alpha_k = \sum_{\text{flow}_i \in k} \left(\alpha^{\text{flow}_i} \oslash \left(\bigotimes_{j \in P(i)} \beta_j \right) \right)$$ has a graphical interpretation:

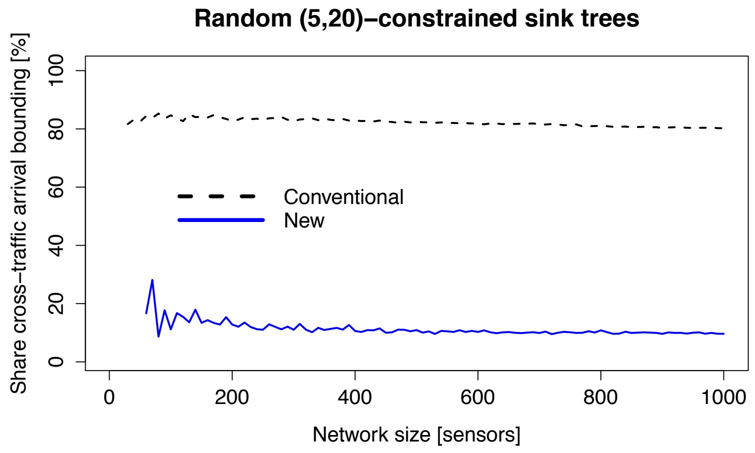


The computation does not focus on single nodes but on separate flows (*neglecting multiplexing!*)

Flows can even carry their *internal arrival curves* for self-modeling tasks

Computational demand shrinks



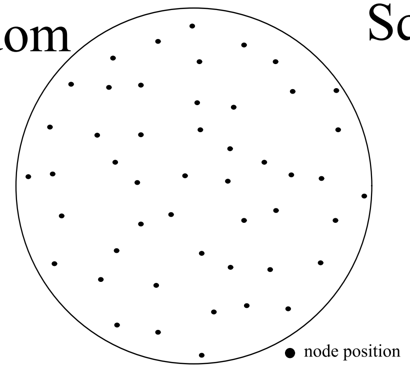


Dr. Liang Cheng: http://liangcheng.info
Dr. Steffen Bondorf: http://steffenbondorf.com

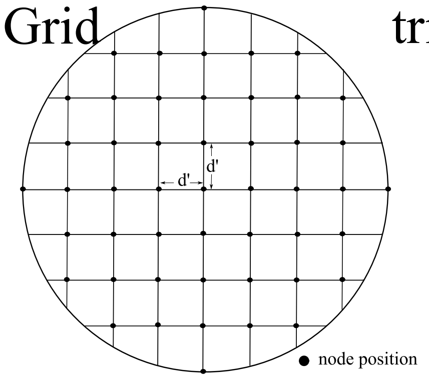
Application: Node Placement

Ranking of different node placement strategies

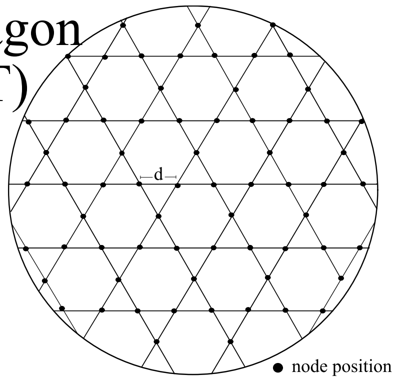
Random



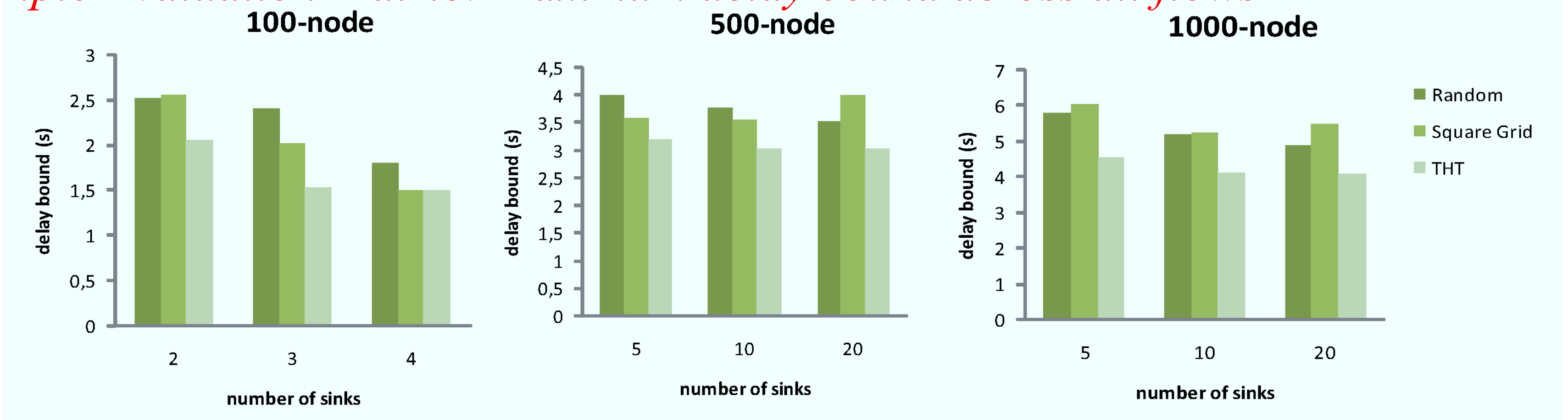
Square Grid



tri-hexagon (THT)



Simple Evaluation Matric: Maximum delay bound across all flows



Dr. Liang Cheng: http://liangcheng.info
Dr. Steffen Bondorf: http://steffenbondorf.com

Case II. Substation Networks

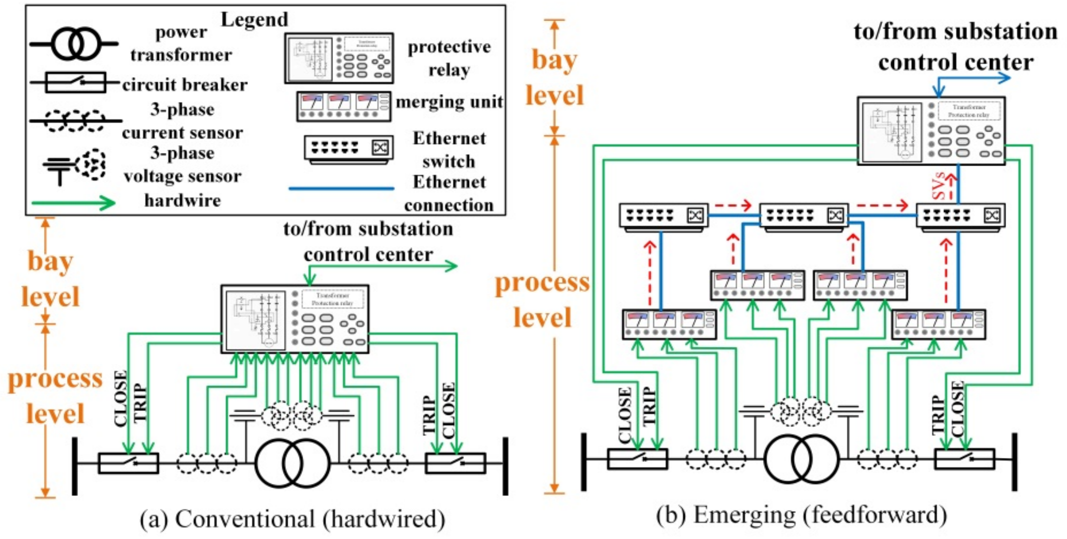
Deterministic performance analysis of service provisioning systems with *random* inbound flows

- discrete-event simulations
- testbed measurements

Limitations

- simulation tool availability
- testbed scalability

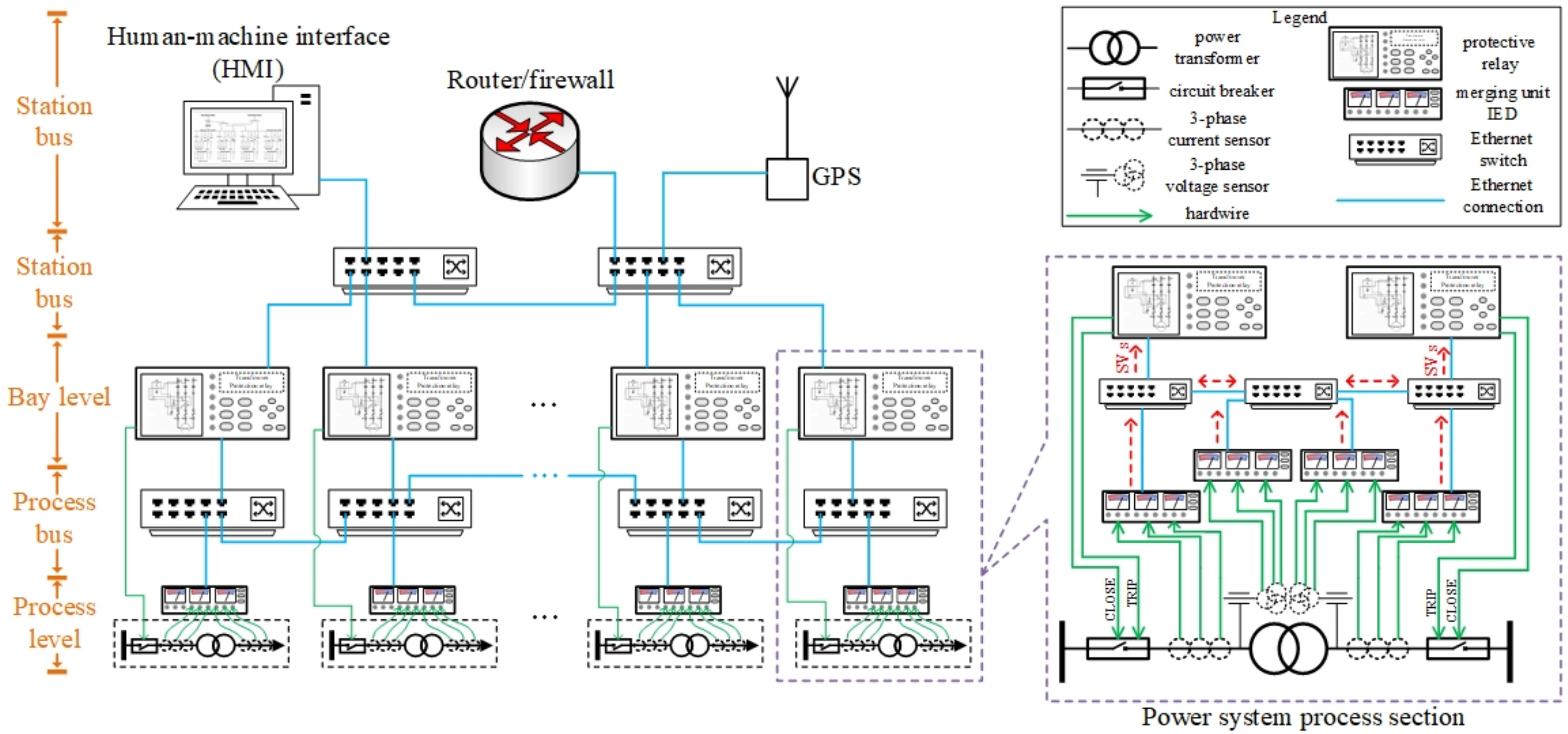
A *network-calculus*-based framework to analyze worst-case network delay in substation automation system



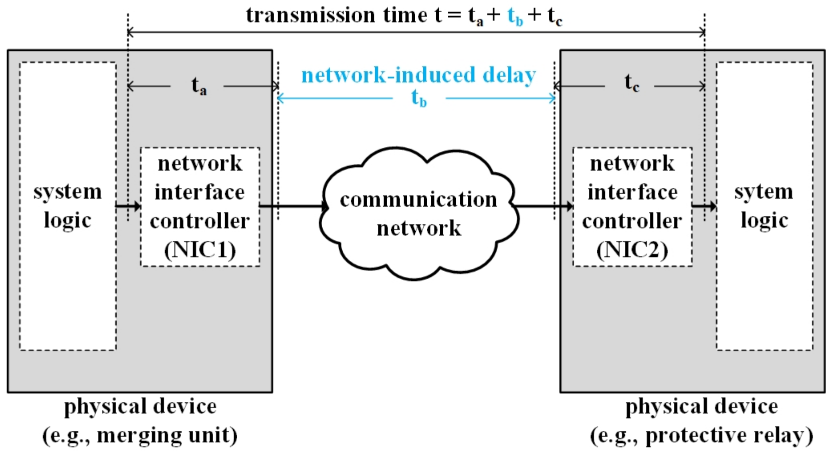
Substation Automation Systems *

* Huan Yang, Liang Cheng, and Xiaoguang Ma, Analyzing worst-case delay performance of IEC 61850-9-2 process bus networks using measurements and network calculus, in the *Proceeding of The Eighth International Conference on Future Energy Systems* (ACM e-Energy), Hong Kong, May 17-19, 2017.

Substation Automation Systems (SAS) Based on IEC 61850



Deterministic Delay Performance Requirements

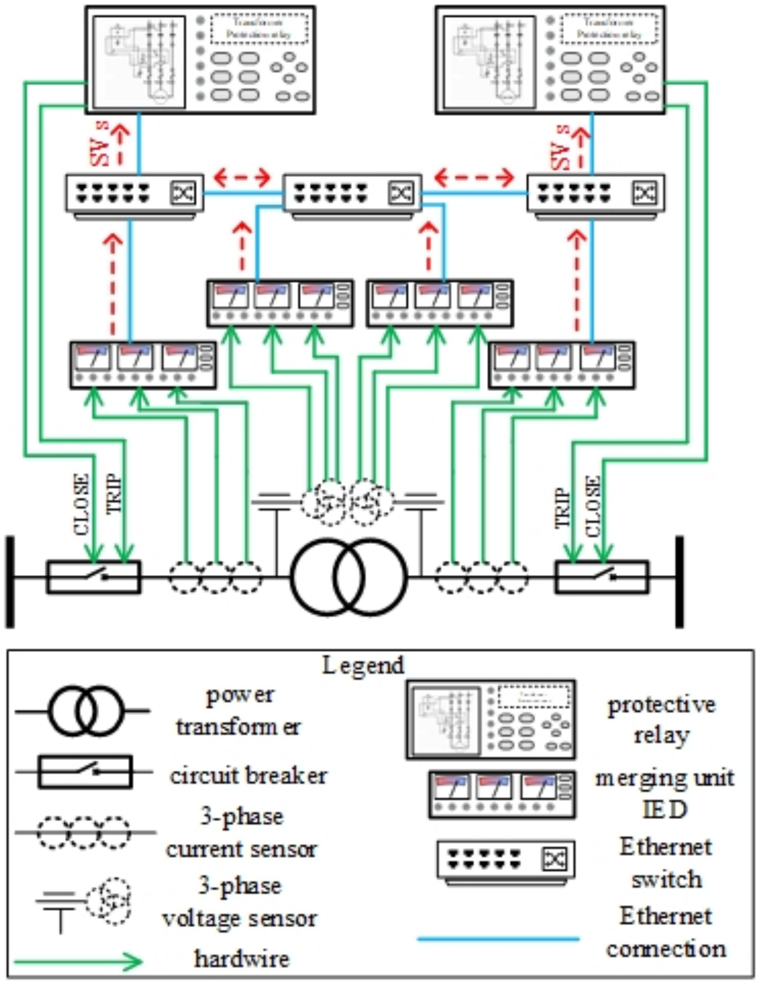

transmission time t = $t_a + t_b + t_c$
t_a
network-induced delay t_b
t_c
system logic
network interface controller (NIC1)
communication network
network interface controller (NIC2)
sytem logic
physical device (e.g., merging unit)
physical device (e.g., protective relay)

Message type	Performance class	Max. delay
Type 1A "Trip"	P1	10 ms
	P2/3	3 ms
Type 1B "Others"	P1	100 ms
	P2/3	20 ms
Type 2 (medium speed messages)	-	100 ms
Type 3 (low speed messages)	-	500 ms
Type 4 (raw data messages)	**P1**	**10 ms**
	P2/3	**3 ms**

Reference:
"Communication Networks and Systems for Power Utility Automation - Part 5: Communication Requirements for Functions and Device Models," International Electrotechnical Commission (IEC), IEC 61850-5:2013.

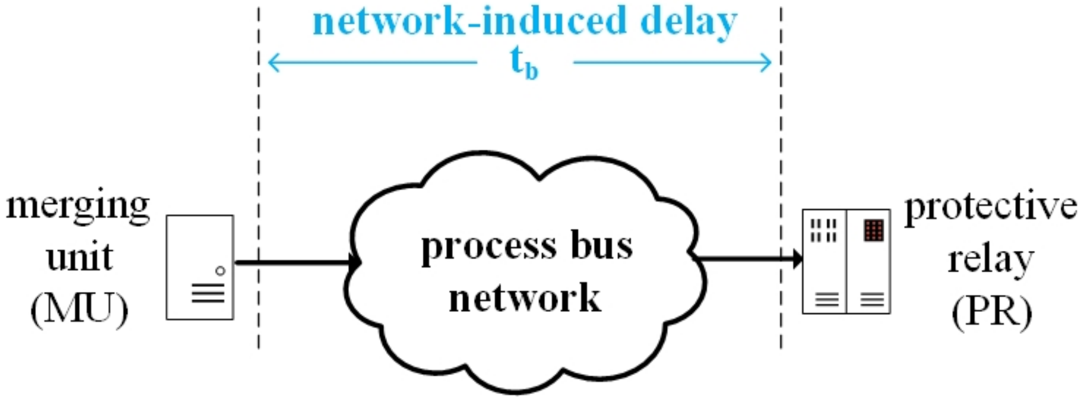
IEC 61850-9-2 Process Bus Network

- **Merging unit (MU) collects voltage and current readings and encapsulates them into sampled value messages (SVMs)**
- **Protection and control functions (e.g., system state estimation, fault protection) are implemented by protective relays (PRs)**
- **Process bus network transmits SVMs from multiple MUs to one or multiple PRs**
  - ➤ Commands for actuators (e.g., circuit breakers) can be transmitted out-of-band (e.g., via hardwired connections)



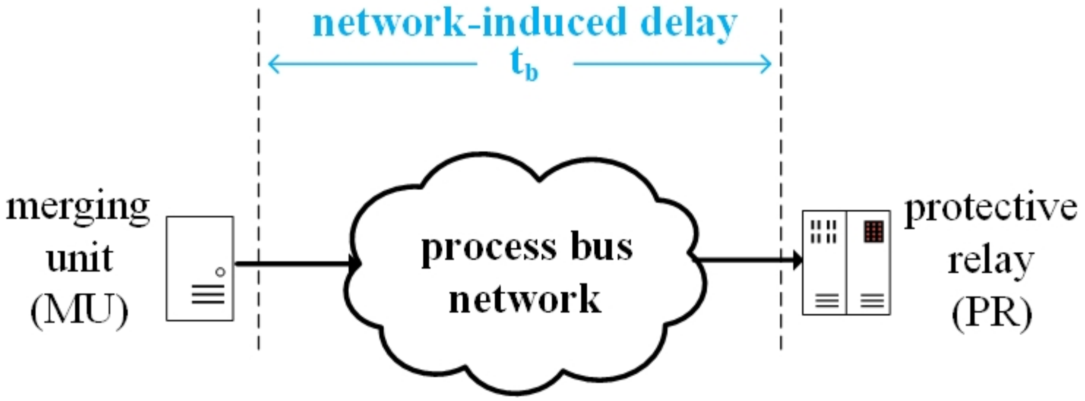
Related Work: Measurement-Based Delay Analyses

- **Accurate delay measurements can be taken via specialized hardware**
  - Network interface card with hardware time-stamping capability
  - Synchronization device (e.g., IEEE 1588 master clock)



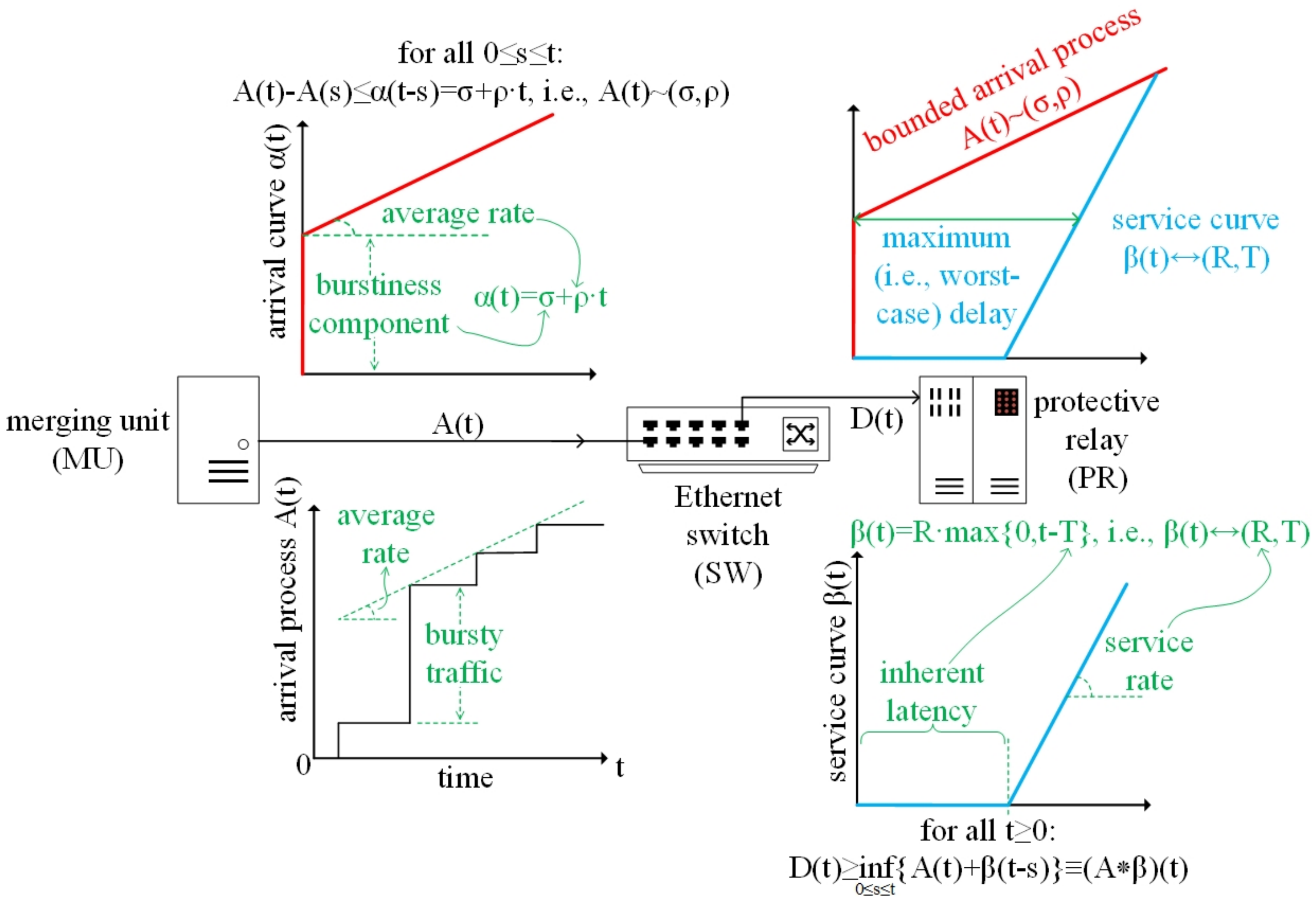
Related Work: Measurement-Based Delay Analyses

- **Accurate delay measurements can be taken via specialized hardware**
  - ➢ Network interface card with hardware time-stamping capability
  - ➢ Synchronization device (e.g., IEEE 1588 master clock)



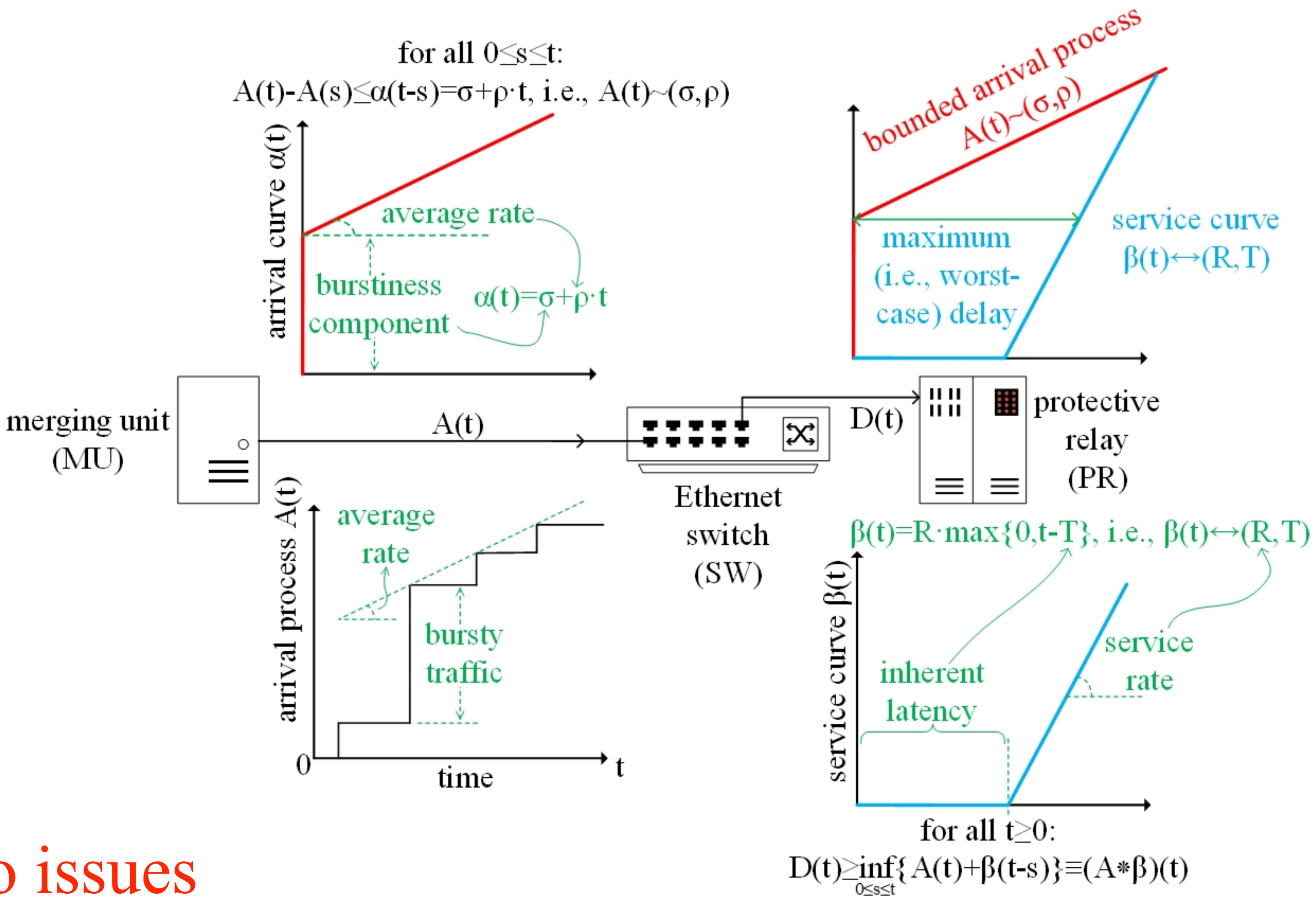
- Limitations
  1. Case-specific
     ➢New measurements need to be taken if changes (e.g., topology, connectivity pattern, number of MUs) are made
  2. Boundary scenarios may not be covered even after an extensive period of measurement

Related Work: Worst-Case Delay Analyses Using Network Calculus



Dr. Liang Cheng: http://liangcheng.info
Dr. Steffen Bondorf: http://steffenbondorf.com

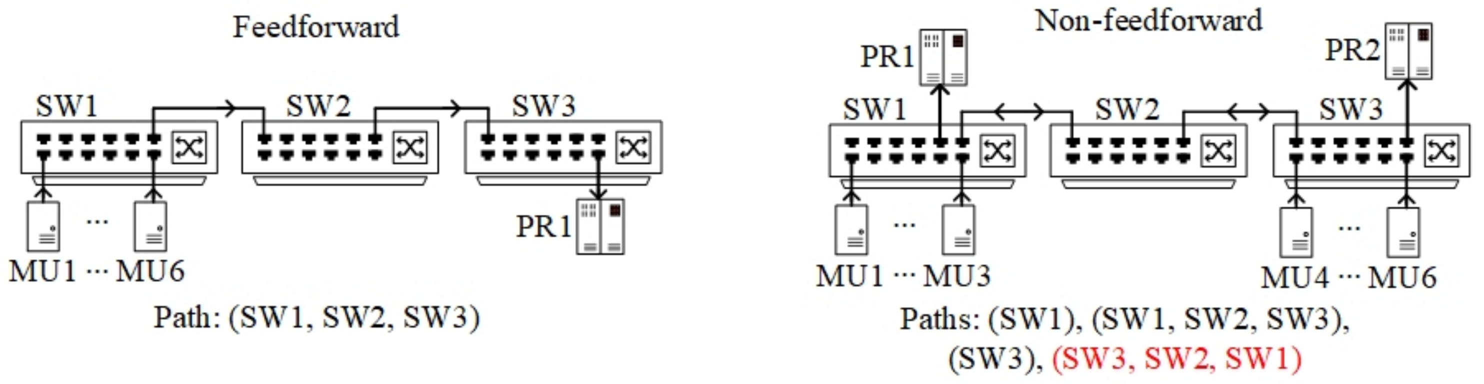
Related Work: Worst-Case Delay Analyses Using Network Calculus



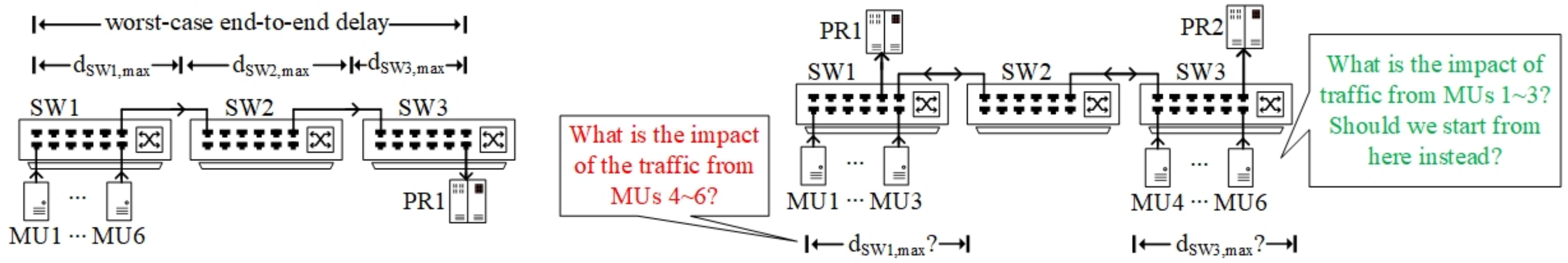
- Two issues

Feedforward vs. Non-Feedforward Traffic Patterns

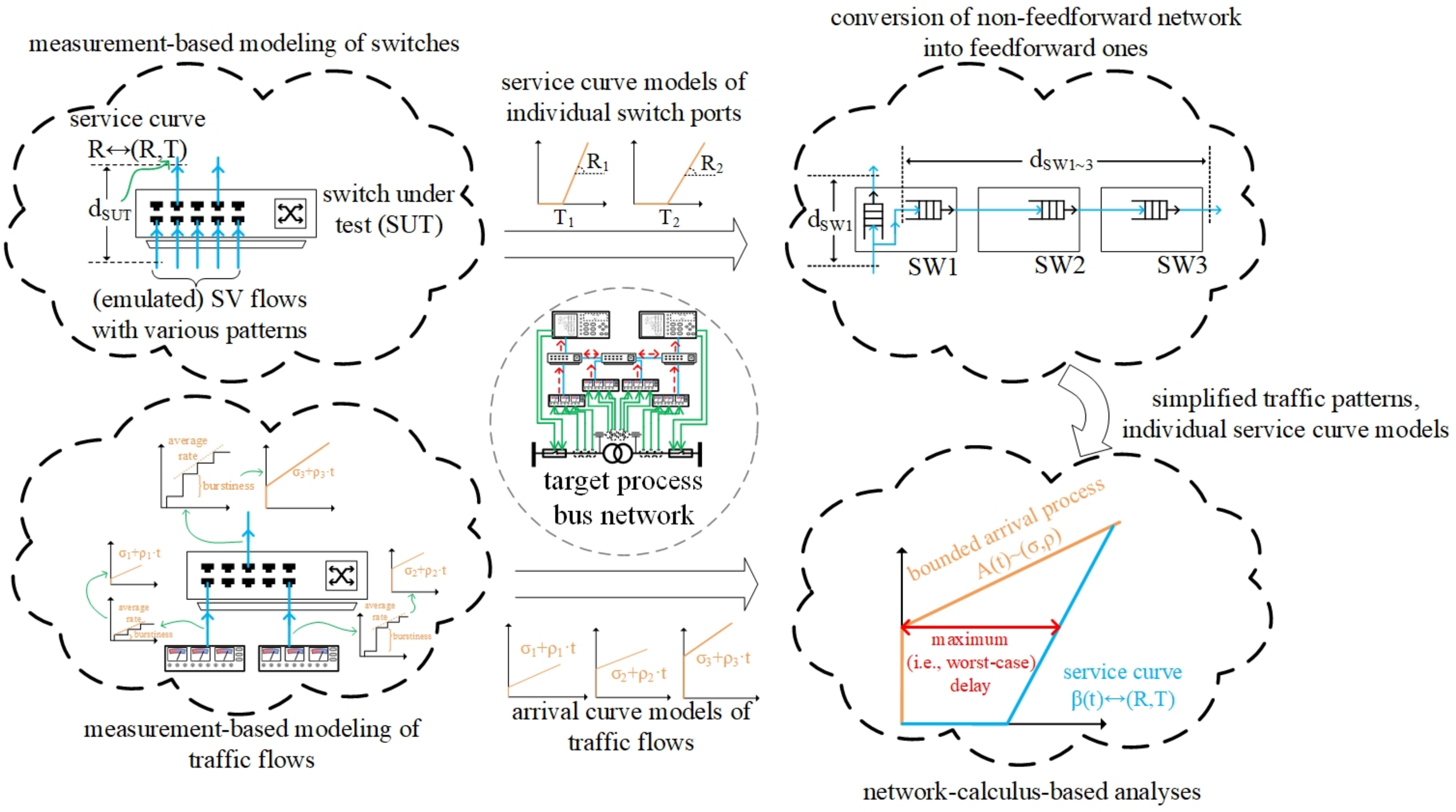
- Sampled value messages (SVMs) are typically transmitted in broadcast or multicast fashion over a process bus network



- Non-feedforward traffic patterns are intricate to analyze

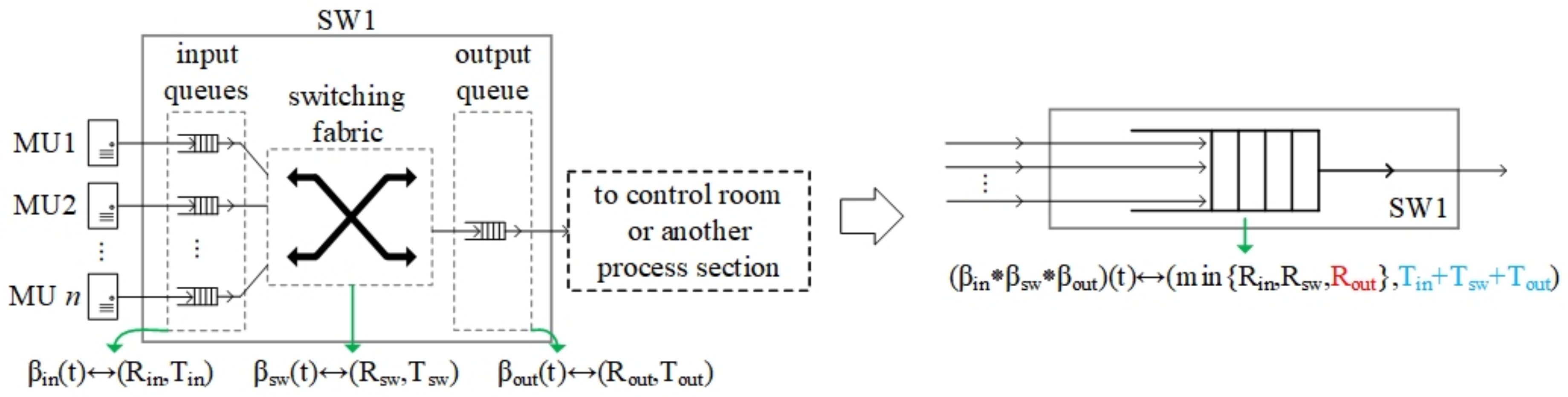

worst-case end-to-end delay
$d_{SW1,max}$
$d_{SW2,max}$
$d_{SW3,max}$
SW1
SW2
SW3
PR1
MU1 ··· MU6
What is the impact of the traffic from MUs 4~6?
PR1
PR2
SW1
SW2
SW3
MU1 ··· MU3
MU4 ··· MU6
$d_{SW1,max}$?
$d_{SW3,max}$?
What is the impact of traffic from MUs 1~3? Should we start from here instead?

Combining Network Calculus with Measurement

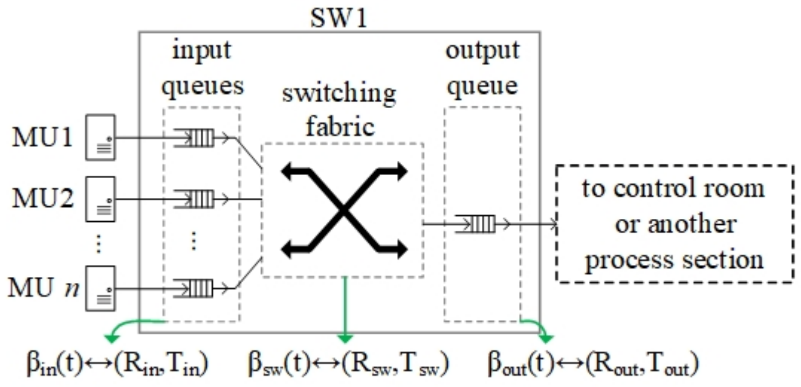

measurement-based modeling of switches
service curve
R↔(R,T)
d_{SUT}
switch under test (SUT)
(emulated) SV flows with various patterns
service curve models of individual switch ports
R_1
T_1
R_2
T_2
conversion of non-feedforward network into feedforward ones
$d_{SW1\sim3}$
d_{SW1}
SW1
SW2
SW3
target process bus network
simplified traffic patterns, individual service curve models
average rate
burstiness
$\sigma_3+\rho_3\cdot t$
$\sigma_1+\rho_1\cdot t$
$\sigma_2+\rho_2\cdot t$
measurement-based modeling of traffic flows
arrival curve models of traffic flows
bounded arrival process
A(t)~(σ,ρ)
maximum (i.e., worst-case) delay
service curve
β(t)↔(R,T)
network-calculus-based analyses

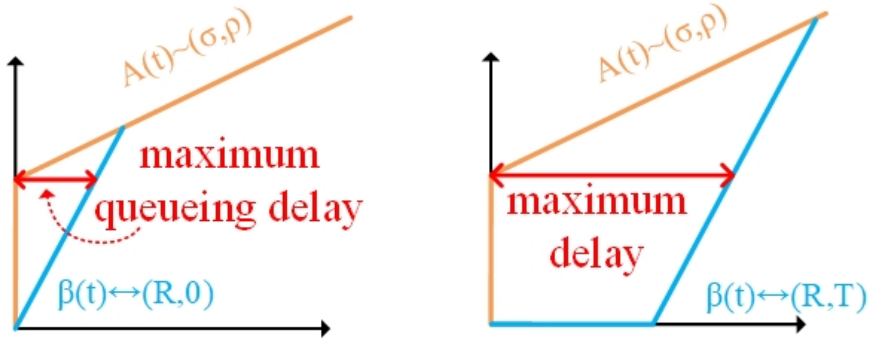
Network-Calculus Model for Ethernet Switches

- **Each output interface can be modeled with a rate-latency service curve**
  - ✓ Rate component
    - ➢ Output interface will be the bottleneck provided that the switching fabric offers sufficient processing capacity
  - ? Latency component
    - ➢ Measurements need to be taken


SW1
input queues
switching fabric
output queue
MU1
MU2
MU n
to control room or another process section
SW1
$(\beta_{in}*\beta_{sw}*\beta_{out})(t) \leftrightarrow (\min\{R_{in},R_{sw},R_{out}\}, T_{in}+T_{sw}+T_{out})$
$\beta_{in}(t) \leftrightarrow (R_{in}, T_{in})$
$\beta_{sw}(t) \leftrightarrow (R_{sw}, T_{sw})$
$\beta_{out}(t) \leftrightarrow (R_{out}, T_{out})$

Queueing vs. Non-Queueing Delay Components

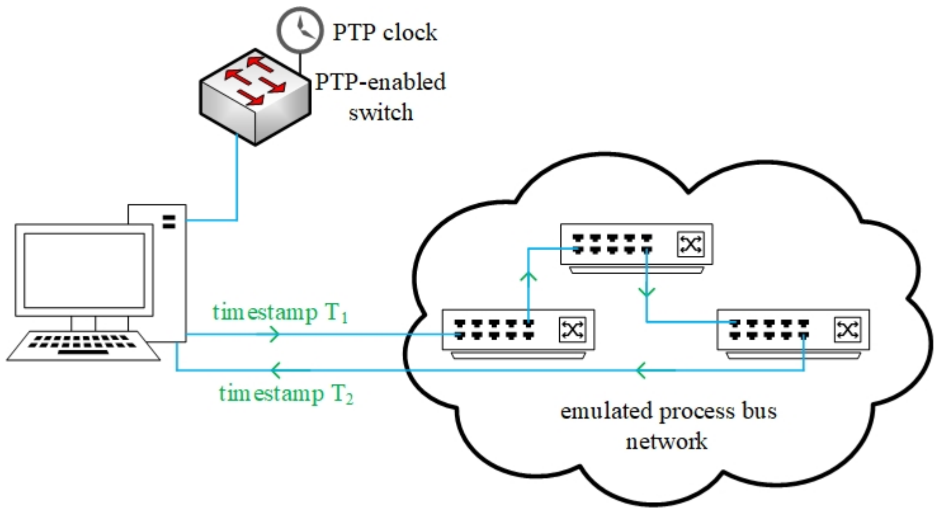

SW1
input queues
switching fabric
output queue
MU1
MU2
MU n
to control room or another process section
$\beta_{in}(t)\leftrightarrow(R_{in},T_{in})$
$\beta_{sw}(t)\leftrightarrow(R_{sw},T_{sw})$
$\beta_{out}(t)\leftrightarrow(R_{out},T_{out})$

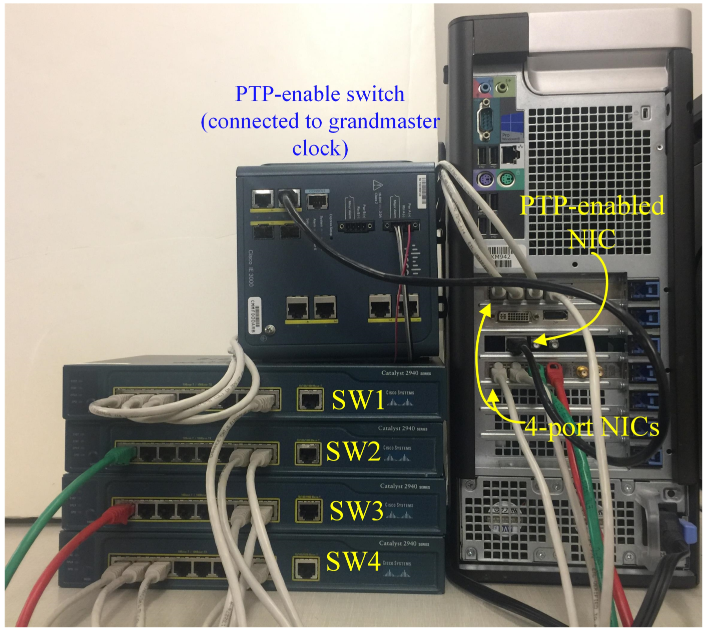


Latency component	Description
T_{in}	Propagation delay of input connection, processing delay of input interface
T_{sw}	Processing delay of switching fabric
T_{out}	Propagation delay of output connection, processing delay of output interface

- $T=T_{in}+T_{sw}+T_{out}$
  - Nearly constant (hardware processing)
  - Measurements must be taken under light workloads

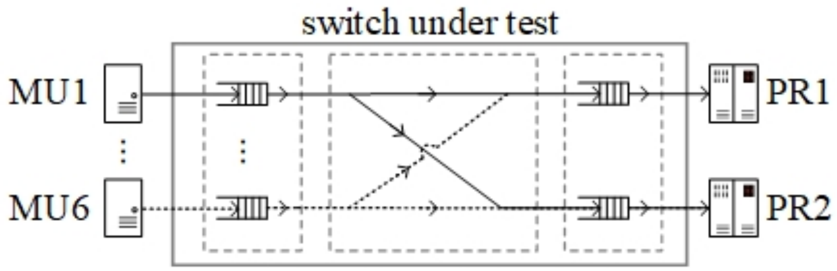
Test Bed for Delay Performance Analysis





- **End-to-end delay is computed using the two timestamps T_1 and T_2**
  - ➢ Modeling of Ethernet switches
  - ➢ Evaluation of network-calculus-based delay performance analysis

Identifying the Latency Value from Measurements

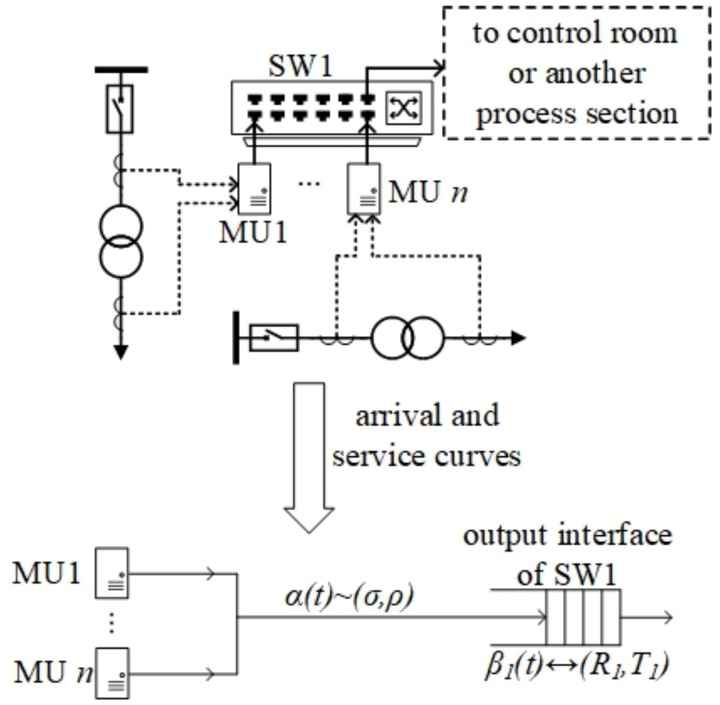


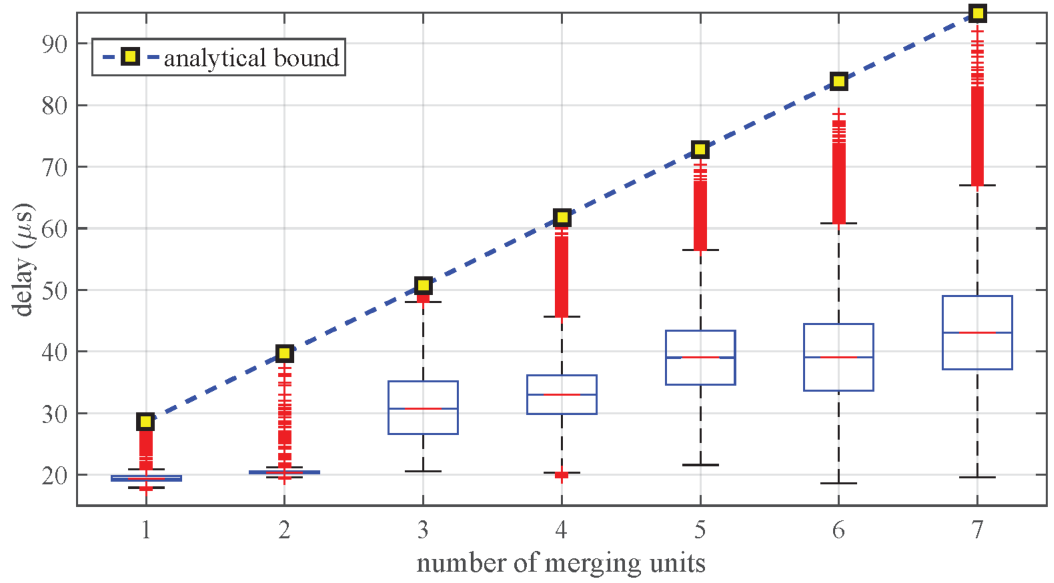
- **Delays experienced by SVMs injected from different ports fall within the range of 15~17.6 μs**
  - ➢ Non-queueing delays observed from switches of the same model are similar

SVMs per second	MU1			MU2		
	Minimum	Average	Maximum	Minimum	Average	Maximum
1	15.2	16.3	17.2	15.3	16.1	17.1
10	15.1	15.9	16.7	14.6	16.0	17.3
100	16.0	16.6	16.9	15.5	16.1	17.4
1000	15.7	16.2	17.0	14.9	15.8	16.9
2000	15.4	16.3	17.1	15.4	16.6	**17.6**
3000	15.2	16.3	17.5	15.8	16.2	17.0
4000	14.9	16.1	17.4	15.3	15.9	17.4
4800	15.6	16.5	**17.6**	15.1	16.4	16.8

latency component

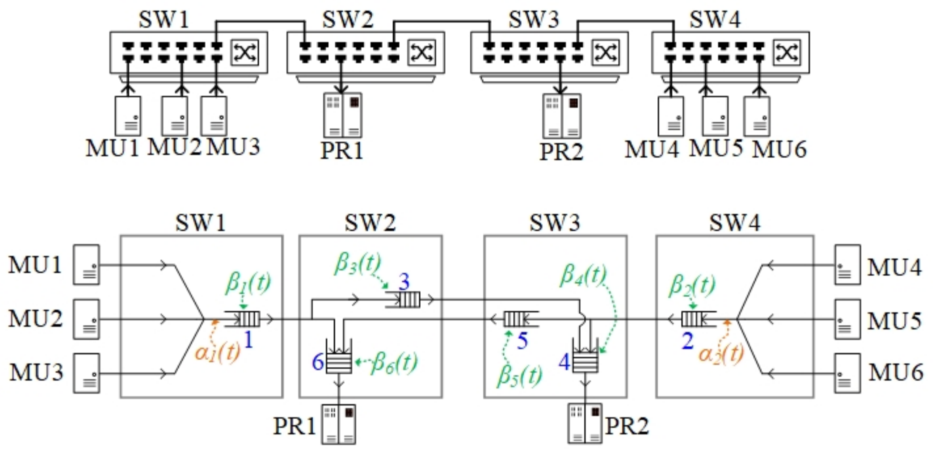
Single-Switch Process Bus Network

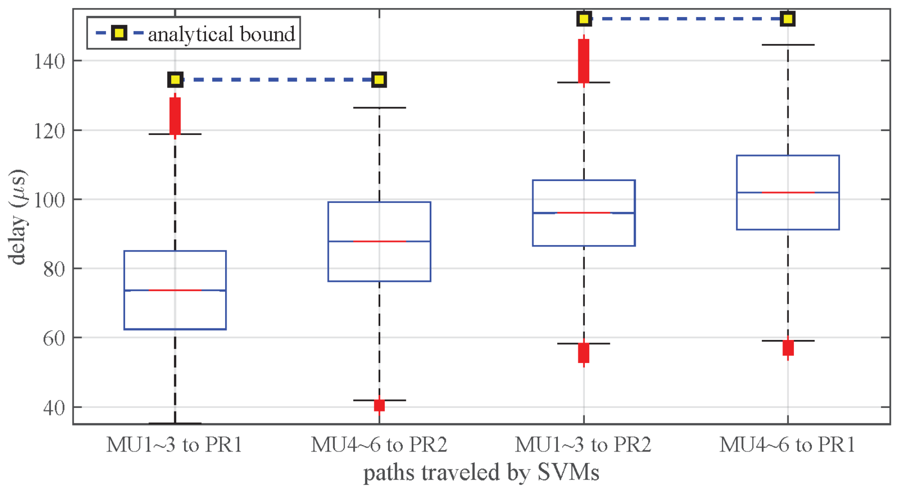

SW1
to control room or another process section
MU1
MU n
arrival and service curves
output interface of SW1
MU1
MU n
$\alpha(t)$~(σ,ρ)
$\beta_1(t)\leftrightarrow(R_1,T_1)$



- **The single-switch network is always feedforward**
- **Delay bounds computed are sufficiently tight**
  - At most 3.5% greater than the maximum delay observed

Four-Switch Process Bus Network





- Derived delay bounds are sufficiently tight (at most 6.3% greater than maximum delays observed)
- Both network-calculus-based analysis and measurements suggest that an extra 17.6-μs latency is introduced when the number of switches in the path increased by one

Review Questions (1)

Compute the min-plus convolution of the two functions/curves described in linear piece-wise form below (they are left-continuous).

curve f:
for $0 <= x < 1$, $f(x) = 3*x$
$1 <= x$, $f(x) = 1*x + 2$

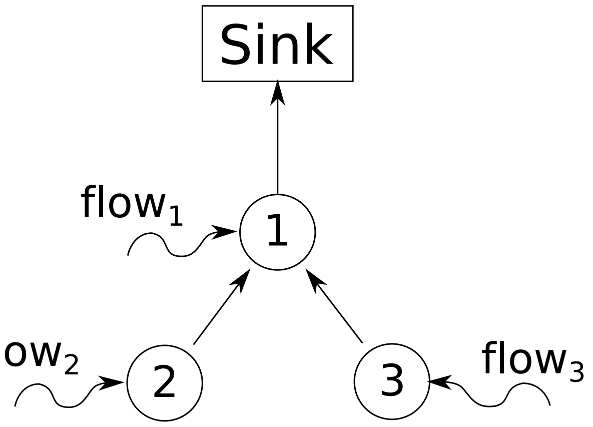
curve g:
for $0 <= x < 2$, $g(x) = 0$
$2 <= x$, $g(x) = 2*x - 4$

The solution is the piece-wise function h:
for: $0 <= x < 2$, $h(x) = 0$
$2 <= x < 4$, $h(x) = 2*x - 4$
$4 <= x$, $h(x) = x*1$

Dr. Liang Cheng: http://liangcheng.info
Dr. Steffen Bondorf: http://steffenbondorf.com

Review Questions (2)

Derive the equation(s) bounding the end-to-end delay of flow$_3$



Solution: $D^{\text{flow}_3}_{\text{end-to-end}} = D_3 + D_1$

$$D_3 = h(\alpha_3, \beta_3) = h(\alpha^{\text{flow}_3}, \beta_3)$$

$$D_1 = h(\alpha_1, \beta_1) = h(\alpha^{\text{flow}_1} + (\alpha^{\text{flow}_2} \oslash \beta_2) + (\alpha^{\text{flow}_3} \oslash \beta_3),\ \beta_3)$$

(distributivity does not change this equation because there is no multiplexing at nodes 2 and 3)

Tutorial on Modeling and Analysis of Network Infrastructure in Cyber-Physical Systems

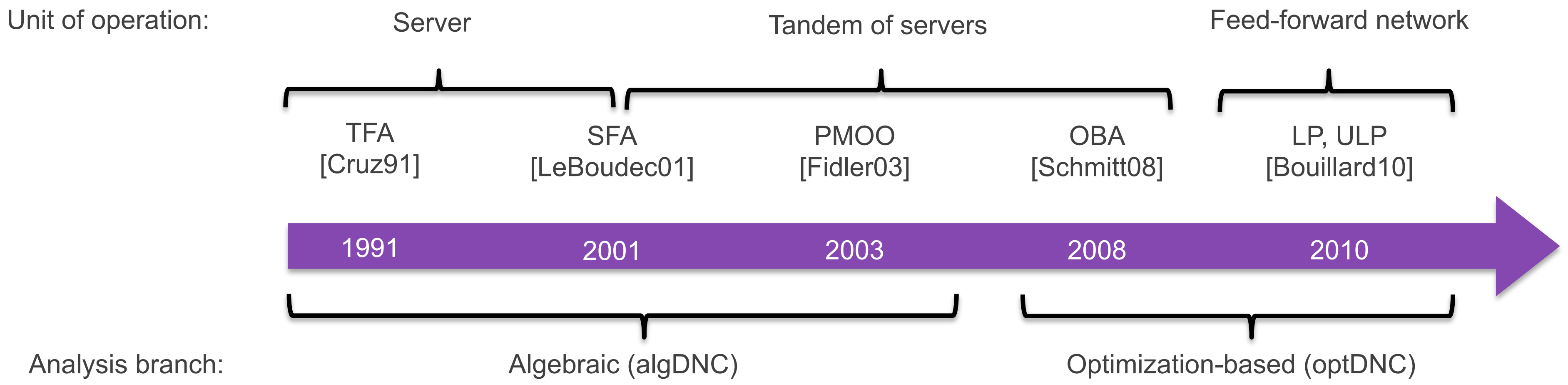
Liang Cheng [*] and Steffen Bondorf [†]

[*] Department of Computer Science and Engineering, Lehigh University
Bethlehem, Pennsylvania 18015, USA

[†] Department of Information Security and Communication Technology,
Norwegian University of Science and Technology, NO-7034, Trondheim, Norway

[*] cheng@lehigh.edu, [†] steffen.bondorf@ntnu.no

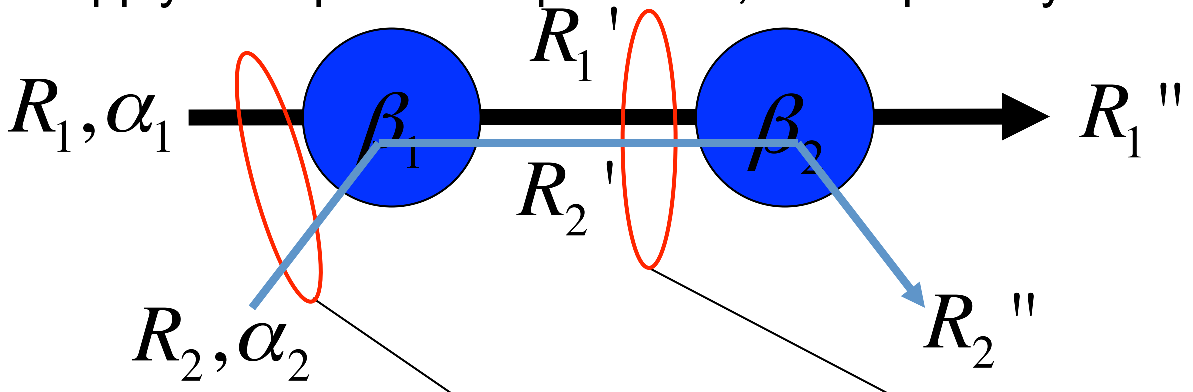
Deterministic Network Calculus Analysis



- A lot of new results until 2010 (and even thereafter)
  - More well known authors are Cheng-Shang Chang, Jörg Liebeherr, …
- We will cover the most important ones next
  - TFA, SFA, PMOO, OBA

Total Flow Analysis (TFA)

- First Idea: Total Flow Analysis [Cruz91]
  - work your way up from sources to sinks to compute output bounds
  - apply NC operations per node, sum up delay bounds



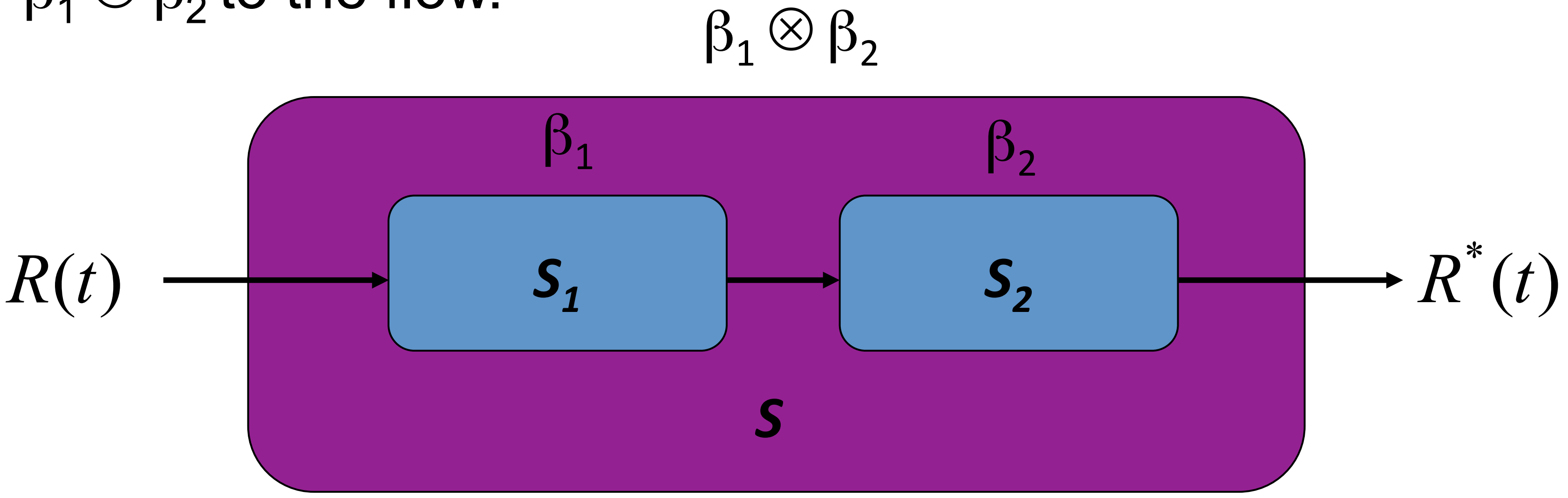
Calculations: $d^{\text{TFA}} = h(\alpha_1 + \alpha_2, \beta_1) + h((\alpha_1 + \alpha_2) \oslash \beta_1, \beta_2)$

- Note:
  - FIFO assumption implicit, otherwise the horizontal deviation is not a delay bound
  - Bad scaling of performance bounds because the analyzed flow's burstiness is paid at every server

Convolution in a Tandem Analysis

Theorem (**Concatenation of Nodes**):

- Assume a flow *R(t)* traverses system S_1 and S_2 in sequence which offer service curves β_1 and β_2, respectively. Then the concatenation of the two systems offers a service curve of $\beta_1 \otimes \beta_2$ to the flow.

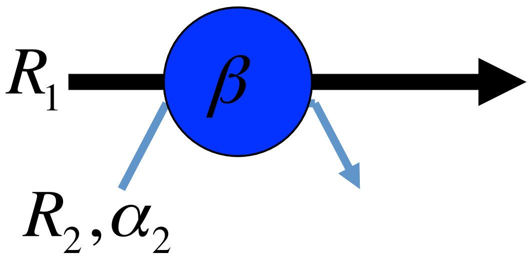


- Note that $\beta_1 \otimes \beta_2$ is not strict, even if β_1 and β_2 were

Arbitrary Multiplexing

- Definition: (**Arbitrary Multiplexing**)
  At multiplexing nodes, the merging of flows may happen in an arbitrary order.
- Theorem: (**Arbitrary Multiplexing at Single Node**)
  Consider a node arbitrarily multiplexing two flows 1 and 2. Assume that the node guarantees a (strict) service curve β to the aggregate of the two flows. Assume that flow 2 has α_2 as an arrival curve. Then

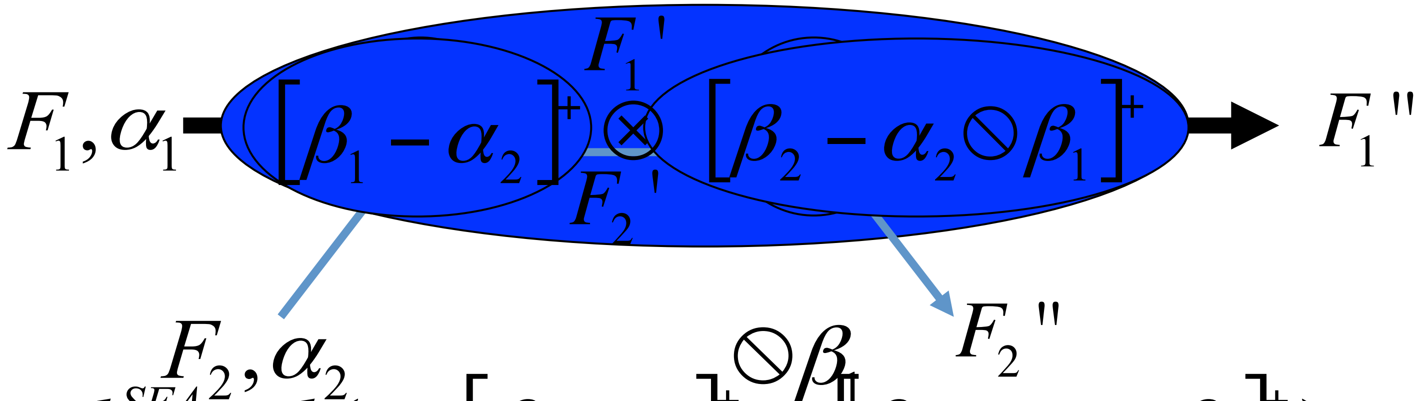
$$\beta^1 = [\beta - \alpha_2]^+ = \max\{\beta - \alpha_2, 0\}$$



  is a left-over service curve for flow 1.
  - Strictness of service curve is necessary
  - The left-over service curve is no longer strict
  - It lower bounds the guarantee in FIFO multiplexing, the FIFO left-over β is complex (introduces a free parameter)

Separated Flow Analysis (SFA)

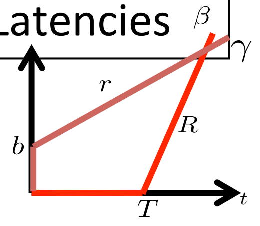
- Second Idea: Separated Flow Analysis
  - use arbitrary multiplexing nodal service curve
  - convolve to end-to-end service curve for flow of interest



Calcs: $d^{SFA} = h(\alpha_1, [\beta_1 - \alpha_2]^+ \otimes [\beta_2 - \alpha_2 \oslash \beta_1]^+)$

Token Buckets
Rate-Latencies

$$= T_1 + T_2 + \frac{b_1}{\min(R_1, R_2) - r_2} + \frac{b_2 + r_2 T_1}{R_1 - r_2} + \frac{b_2 + r_2 (T_1 + T_2)}{R_2 - r_2}$$



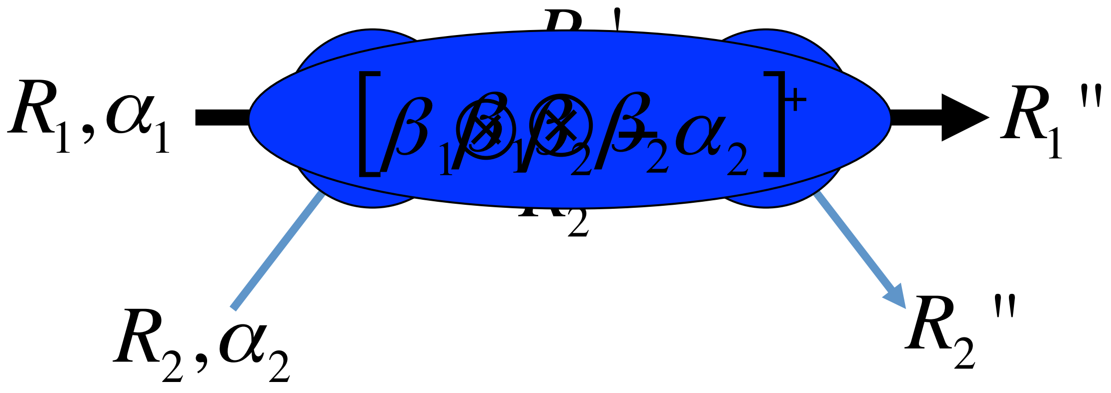
- Concatenation achieved, PBOO exploited,
  but *multiplexing* is *paid multiple times* for the cross-flow F_2

Multiplexing with Cross-traffic

- Is it necessary to multiplex multiple times?
- Usually, order of data is retained after multiplexing
  - Foremost example: FIFO networks
  - But also when we do not know the multiplexing (arbitrary)
  - Flows exceed their overall “burst quota” given by their α
- This is called the *Pay Multiplexing Only Once* principle (PMOO)
  - Separate Flow Analysis does not exploit the PMOO principle
- Objective: Compute a left-over service curve of a tandem (sequence of servers) that considers the *burstiness of cross flows* only once.

PMOO Analysis

- Third Idea: Pay Multiplexing Only Once Principle
  - exploit degrees of freedom in order of concatenation and left-over service curve derivation (→ consider sub-path sharing) ?

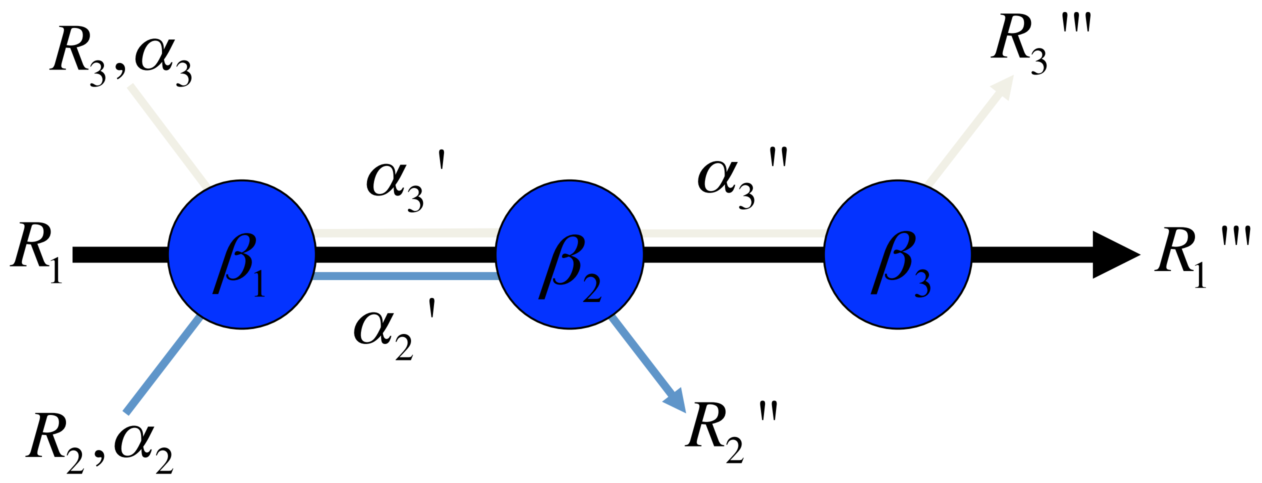


Calcs: $d^{PMOO} = h(\alpha_1, [\beta_1 \otimes \beta_2 - \alpha_2]^+)$

- So: „Convolution before Subtraction!“
  - PBOO and PMOO achieved
  - However, there is a problem with strictness of service curves: $(\beta_1 \otimes \beta_2)$ is not strict → subtracting α_2 is not allowed !

PMOO for Piece-wise Linear Curves

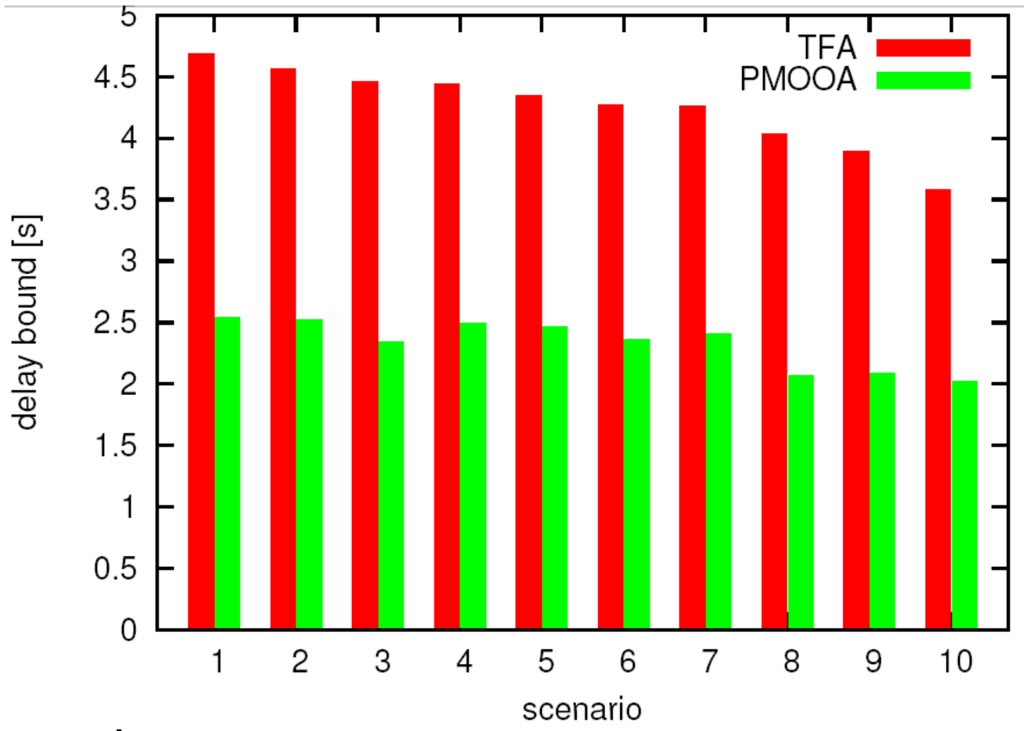
- Yet, there is a direct proof for a PMOO left-over service curve
- It works in even for more complex tandems:



- Theorem: (**PMOO E2E Service Curve**) [Schmitt et al. 2008] "Given *PWL concave* arrival and *general* service curves, PMOO can still be achieved, that means each arrival curve is only subtracted once during the service curve construction."
  - fairly complex (inelegant) proof on the level of I/O relationships

End-to-End vs. Hop-by-Hop Analysis

- TFA vs. PMOO in a sink-tree sensor network



- Each scenario:
  simulate random deployment of 100 nodes in a 100m^2 plane, transmission range of 20m, sink in the center, shortest path routing

When Network Calculus Leaves You in the Lurch …

- Separated Flow Analysis can outperform PMOO Analysis set $T_1 = 0,\ b_2 = 0, R_2 > R_1, r_2 > 0$, then

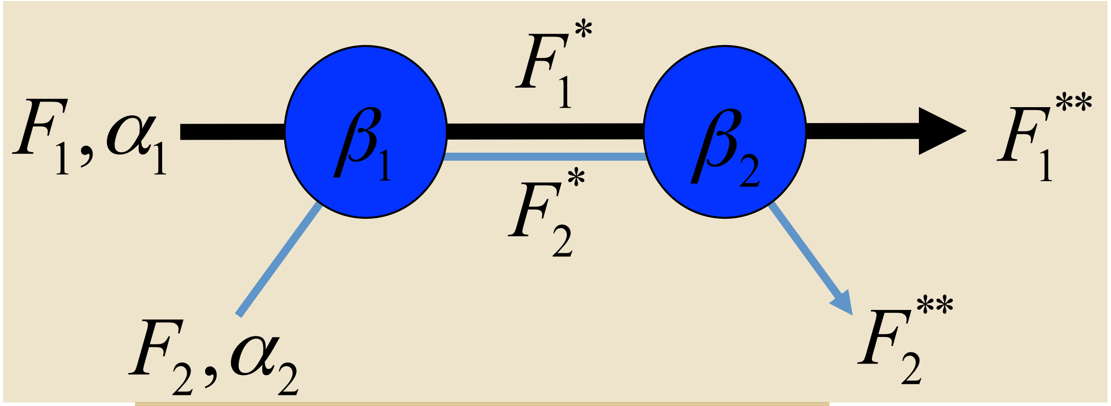
$$d^{SFA} = T_2 + \frac{b_1}{R_1 - r_2} + \frac{r_2 T_2}{R_2 - r_2}$$

$$d^{PMOO} = T_2 + \frac{b_1 + r_2 T_2}{R_1 - r_2}$$

$$\Rightarrow d^{SFA} < d^{PMOO}$$

- SFA correctly accounts for the burstiness and the increase at the right server (indices of R and T), PMOO does not
- Min-plus convolution of nodes „swallows“ necessary topological information
- Commutativity is lost, dramatically speaking „Algebra Broken“

Optimization-based Bounding Method



$$\min. \left(\frac{1}{R_1 - r_2} - \frac{1}{R_2 - r_2} \right) s_2^{(1)}$$

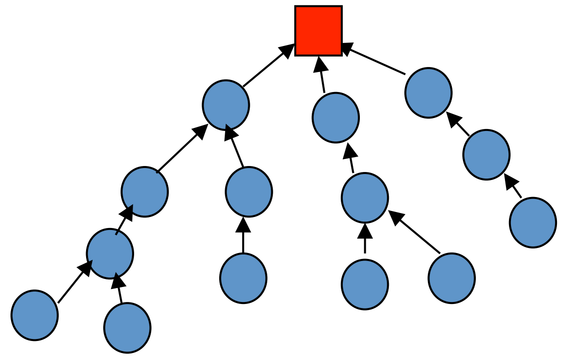
$$\text{s.t.} \quad 0 \le s_2^{(1)} \le b_2 + r_2 T_1$$

$$\beta_{opt}^1 = \beta_{\min(R_1,R_2)-r_2,\, T_1+T_2+\frac{b_2+r_2T_1}{\min(R_1,R_2)-r_2}+\frac{r_2T_2}{R_2-r_2}}$$

$$d^{TIGHT} = h\left(\alpha_1, \beta_{opt}^1\right) = T_1 + T_2 + \frac{b_1 + b_2 + r_2 T_1}{\min(R_1, R_2) - r_2} + \frac{r_2 T_2}{R_2 - r_2}$$

Burstiness inc. accounted for correctly

Numerical Example: Sink Trees

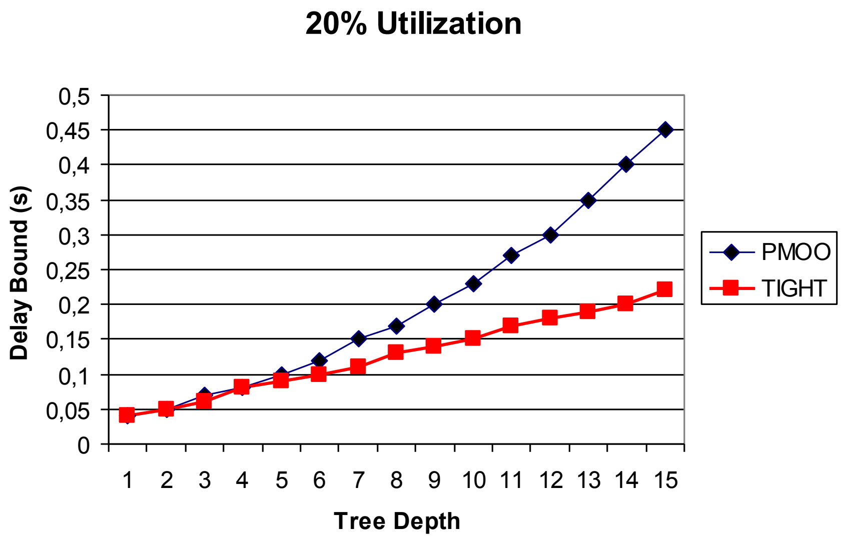


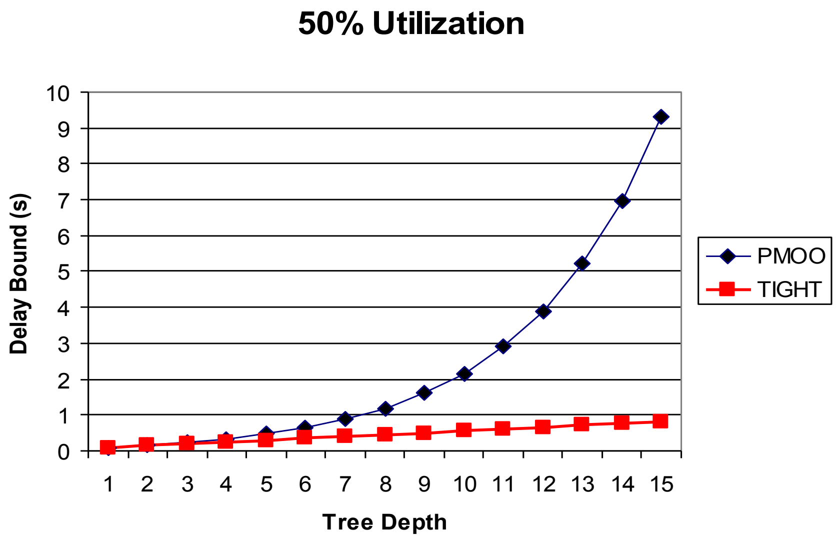
- Sink tree graph
  - Simplest feed-forward network
  - Explicit solution of optimization problem feasible

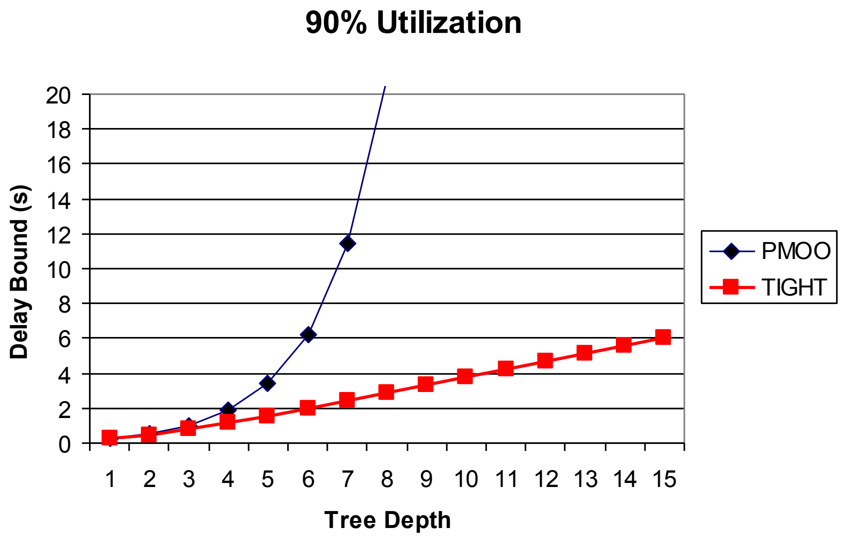
- Competitors: PMOO vs. TIGHT (based on optimization)
- Primary factors
  - tree depth
  - server utilization
- Secondary factors
  - arrival curves: token buckets with r=10Mb/s and b=1 Mb
  - service curves: rate-latency functions with latency 0.1ms and rate dimensioned to meet target utilization

Numerical Example: Sink Trees

- Results for Worst-Case Delay Bounds
  - Utilization at server directly connected to the sink



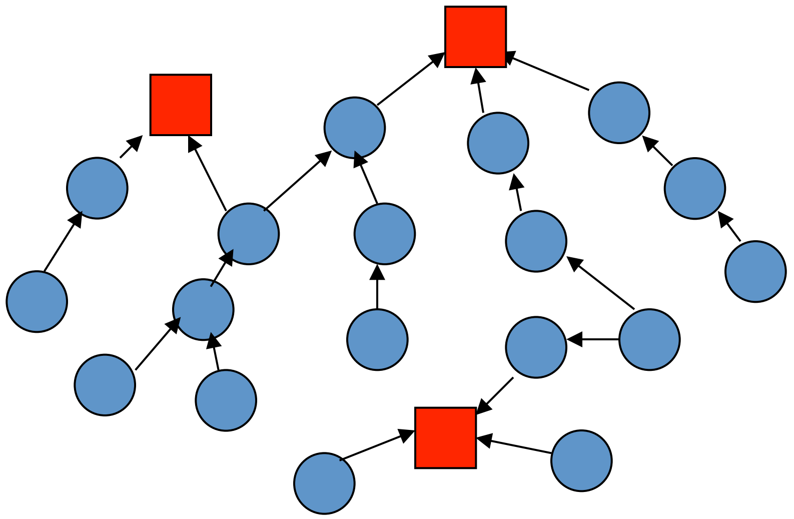




Network Analysis

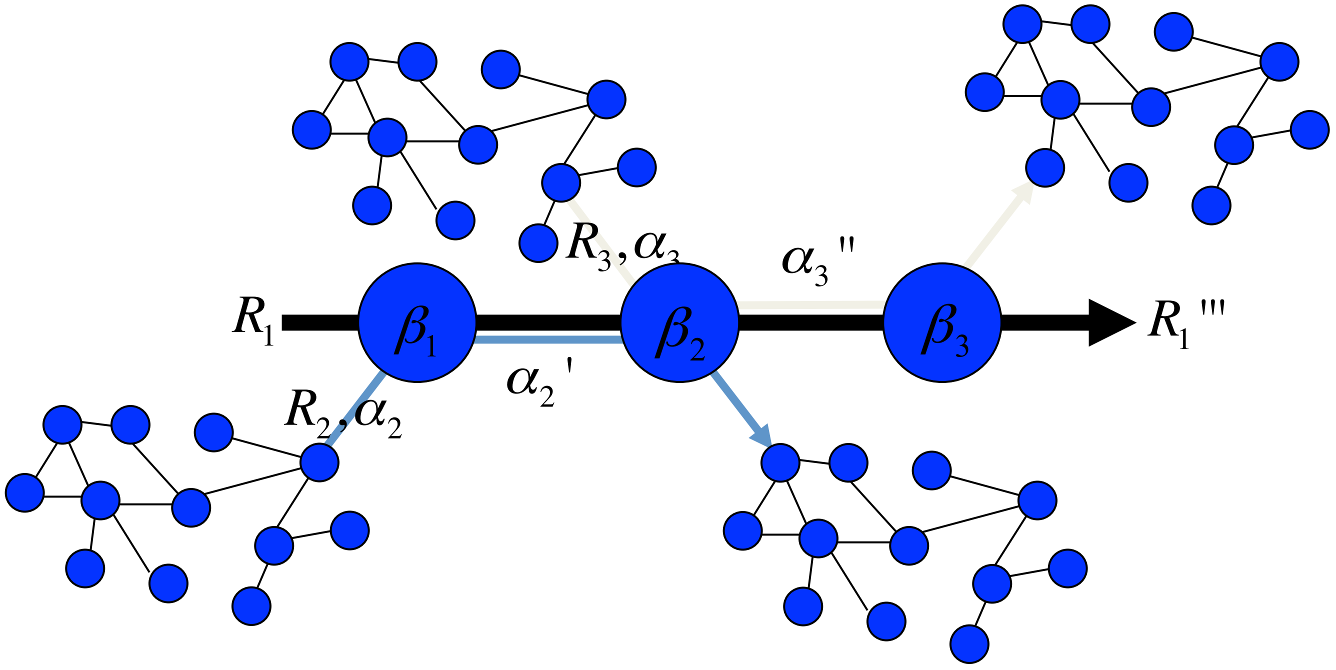
- So far: only tandem network analysis for single flow possible
  - accommodated by "scheduling away" cross-traffic
- General network, multiple flow case
  - very involved, [Charny00] and many others did not really succeed

- Thus turn attention to feed-forward (FF) networks with multiple flows
  - less restrictive than tandem, yet still tractable
  - Examples
    - wireless sensor networks
    - MPLS networks with sink trees
    - "feed-forwardized" networks



Output Bounds

- In order to obtain scenarios as before, the network must be trimmed first



- P{B,M}OO result can also be used for output bounds
  - recursive application along sub-paths shared by interfering flows
  - can become complex → tool support needed

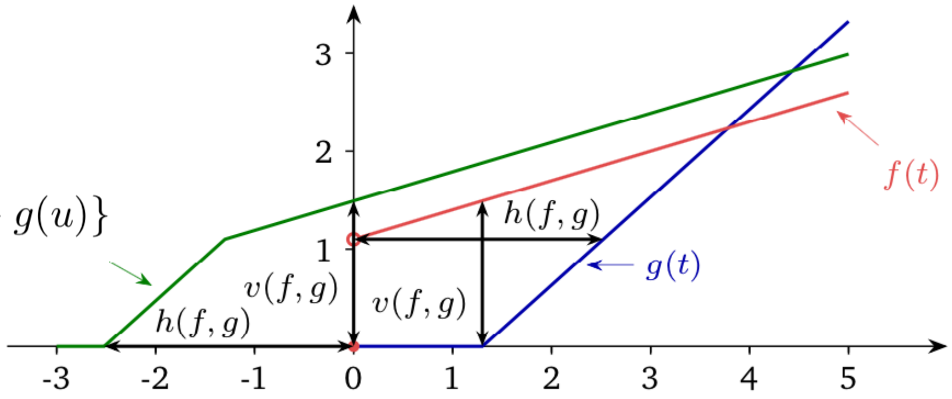
How to trim? Output Flow (from before)

- Consider system *S*

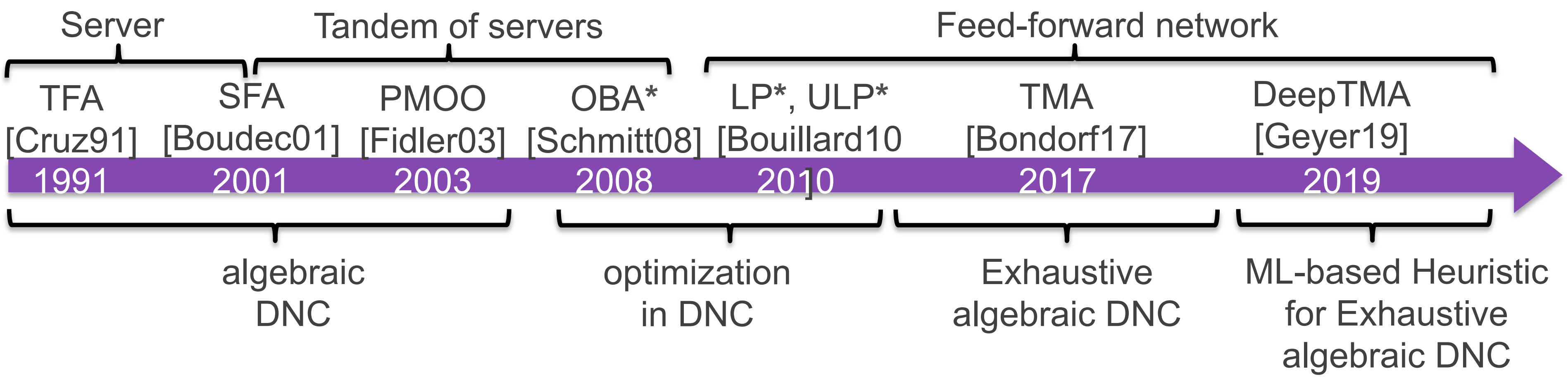
$$\begin{tikzcd} R(t) \arrow[r] & \boxed{S} \arrow[r] & R^*(t) \end{tikzcd}$$

- $R^*(t)$ is the cumulative output of *S* at time *t*
- *S* might be a **single buffer, system or a complete network of systems**
- R*(t)-R*(s) ≤ (α⊘β)(*t*-*s*), for any t≥s≥0

$$(f \oslash g)(t) = \sup_{u \geq 0}\{f(t+u) - g(u)\}$$



Feed-forward Network Analysis



*** Becomes computationally infeasible**

- Do not trim at all? LP and ULP analyses:
  Convert the entire network model to an optimization formulation. Becomes infeasible to solve.
- Therefore: Further improve the algebraic calculus

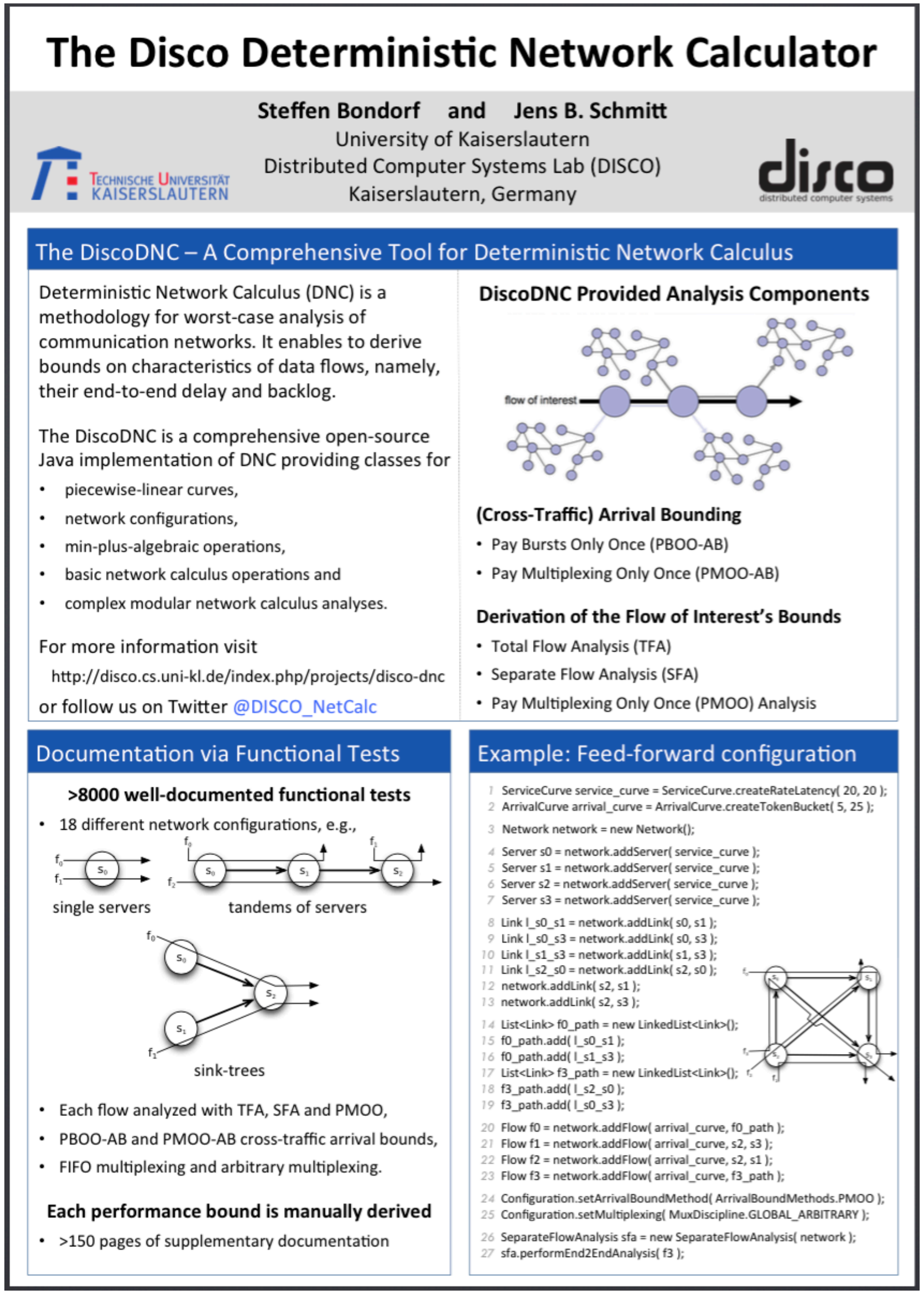
→ Complex “trimming” of networks with tool support

Network Calculus Tool Support

Most prominent examples:

- Real-Time Calculus (RTC) Toolbox
  - ETH Zürich, Switzerland
  - Matlab + Java
  - Manual formula derivation
- Delay Bound Rating Algorithm (DEBORAH)
  - University of Pisa, Italy
  - FIFO Multiplexing, C++
  - only line topologies
- RealTime-at-Work (RTaW) Pegase
  - Commercial, closed source
- Network Calculus.org Deterministic Network Calculator (DNC)

- Many others have coded as well:
  - COINC, NC-maude, CyNC, CATS, WOPANets, DIMTOOL, MinMaxGD, ContainerMinMaxGD, Network Calculus Engine, DelayLyzer, ConfGen, NCBounds



The Disco Deterministic Network Calculator

Steffen Bondorf and Jens B. Schmitt
University of Kaiserslautern
Distributed Computer Systems Lab (DISCO)
Kaiserslautern, Germany

The DiscoDNC – A Comprehensive Tool for Deterministic Network Calculus

Deterministic Network Calculus (DNC) is a methodology for worst-case analysis of communication networks. It enables to derive bounds on characteristics of data flows, namely, their end-to-end delay and backlog.

The DiscoDNC is a comprehensive open-source Java implementation of DNC providing classes for

- piecewise-linear curves,
- network configurations,
- min-plus-algebraic operations,
- basic network calculus operations and
- complex modular network calculus analyses.

For more information visit

http://disco.cs.uni-kl.de/index.php/projects/disco-dnc

or follow us on Twitter @DISCO_NetCalc

DiscoDNC Provided Analysis Components

flow of interest

(Cross-Traffic) Arrival Bounding

- Pay Bursts Only Once (PBOO-AB)
- Pay Multiplexing Only Once (PMOO-AB)

Derivation of the Flow of Interest's Bounds

- Total Flow Analysis (TFA)
- Separate Flow Analysis (SFA)
- Pay Multiplexing Only Once (PMOO) Analysis

Documentation via Functional Tests

>8000 well-documented functional tests

- 18 different network configurations, e.g.,

single servers   tandems of servers

sink-trees

- Each flow analyzed with TFA, SFA and PMOO,
- PBOO-AB and PMOO-AB cross-traffic arrival bounds,
- FIFO multiplexing and arbitrary multiplexing.

Each performance bound is manually derived

- >150 pages of supplementary documentation

Example: Feed-forward configuration

```
ServiceCurve service_curve = ServiceCurve.createRateLatency( 20, 20 );
ArrivalCurve arrival_curve = ArrivalCurve.createTokenBucket( 5, 25 );

Network network = new Network();

Server s0 = network.addServer( service_curve );
Server s1 = network.addServer( service_curve );
Server s2 = network.addServer( service_curve );
Server s3 = network.addServer( service_curve );

Link l_s0_s1 = network.addLink( s0, s1 );
Link l_s0_s3 = network.addLink( s0, s3 );
Link l_s1_s3 = network.addLink( s1, s3 );
Link l_s2_s0 = network.addLink( s2, s0 );
network.addLink( s2, s1 );
network.addLink( s2, s3 );

List<Link> f0_path = new LinkedList<Link>();
f0_path.add( l_s0_s1 );
f0_path.add( l_s1_s3 );
List<Link> f3_path = new LinkedList<Link>();
f3_path.add( l_s2_s0 );
f3_path.add( l_s0_s3 );

Flow f0 = network.addFlow( arrival_curve, f0_path );
Flow f1 = network.addFlow( arrival_curve, s2, s3 );
Flow f2 = network.addFlow( arrival_curve, s2, s1 );
Flow f3 = network.addFlow( arrival_curve, f3_path );

Configuration.setArrivalBoundMethod( ArrivalBoundMethods.PMOO );
Configuration.setMultiplexing( MuxDiscipline.GLOBAL_ARBITRARY );

SeparateFlowAnalysis sfa = new SeparateFlowAnalysis( network );
sfa.performEnd2EndAnalysis( f3 );
```

NetworkCalculus.org Deterministic Network Calculator

- Java library
  - Implements most of the analyses presented before (TFA, SFA, PMOO, ULP, TMA)
- Started as (Disco)DNClib, first paper in 2006
  - Initially in the DISCO group at TU Kaiserslautern, Germany
  - Maintained by Steffen Bondorf
- Open source, see
  - dnc.networkcalculus.org
  - github.com/NetCal/DNC

The DNC library: Design Decisions

- Piece-Wise Linear (PWL) curves
  - Ultimately affine curves,
  - finite amount of pieces,
  - there's always an infinite last segment.
- Explicit solutions of min-plus-algebraic operations on PWL curves.
- Methods to carry out an entire network analysis
  - Network configurations: More complex than tandems.
  - Result: Bounds on output, delay and backlog.
  - Various approaches to benefit from
    - PBOO effect or
    - PMOO effect.

Piecewise-Linear Curves (1)

- Catalog of useful functions …
  - burst-delay function $\delta_T(t) = \begin{cases} +\infty & t > T \\ 0 & t \le T \end{cases}$

    note: $\forall f \in F : \left(f \otimes \delta_T\right)(t) = f(t-T)$,

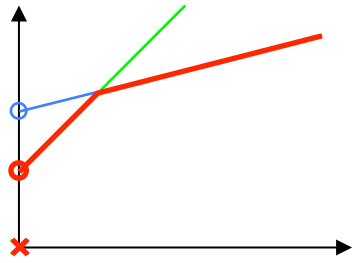
    $\forall f \in F : f = f \otimes \delta_T \otimes \delta_{-T}$
  - token-bucket function $\gamma_{r,b}(t) = rt + b$
  - rate-latency function $\beta_{R,T}(t) = R[t-T]^+$

Piecewise-Linear Curves (2)

- … and their applications

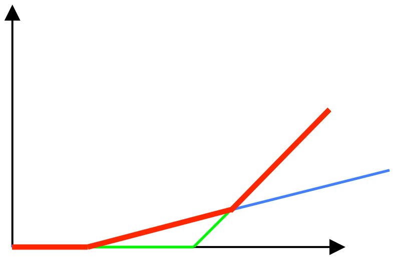
  - *Concave* arrival curve

$$\alpha = \bigwedge_{i=1}^{n} \gamma_{r_i, b_i}$$



  - *Convex* service curve

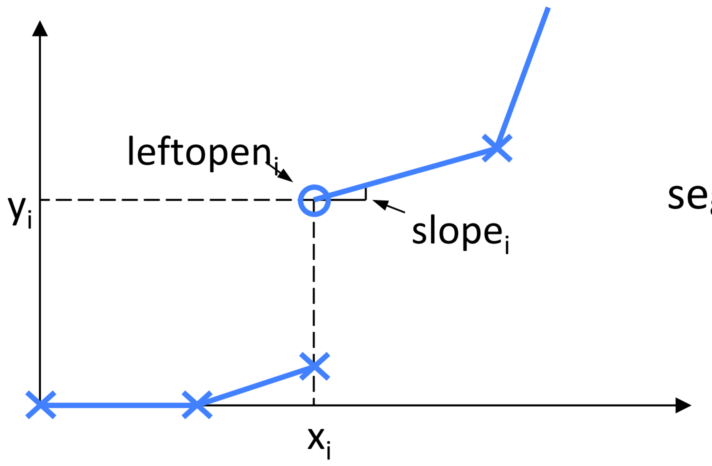
$$\beta = \bigvee_{j=1}^{m} \beta_{R_j, T_j}$$



Network Calculus with PWL Curves (1)

Operations:

- Exploit the knowledge about curves
  - Location of inflection points (x_i,y_i) defined by
    - Change of slope and/or
    - Jump (left- or right-continuity)



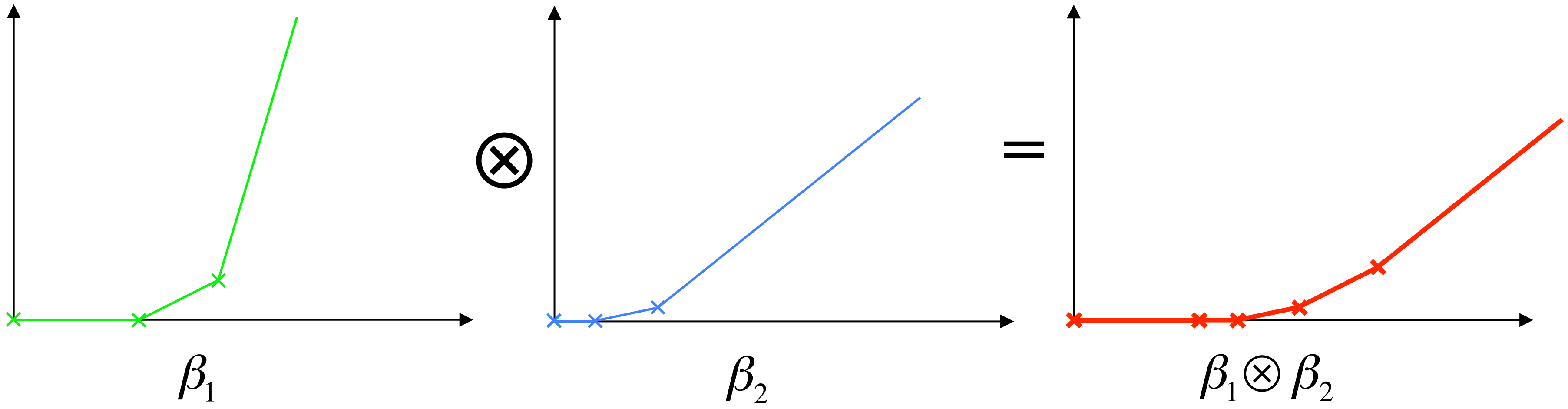
$\text{segment}_i := < (x_i,y_i), \text{slope}_i, \text{leftopen}_i >$

$\text{curve} := < \text{segment}_1, \ldots, \text{segment}_n >$

  - Shape (concave, convex)

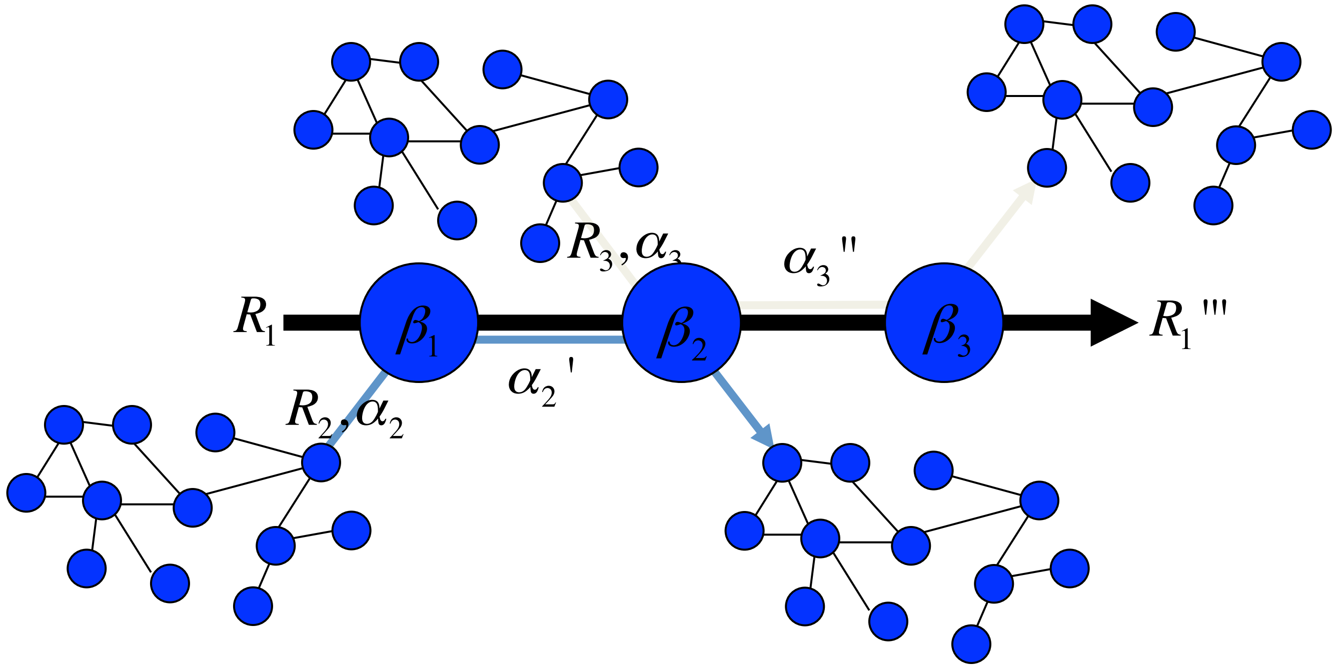
Network Calculus with PWL Curves (3)

- Convolution of service curves
  - β_1, β_2 are piecewise linear convex curves.
  - $\beta_1 \otimes \beta_2$ is obtained by putting the different linear pieces of β_1, β_2 one after another, sorted by increasing slopes.
  - There can be only one last segment.



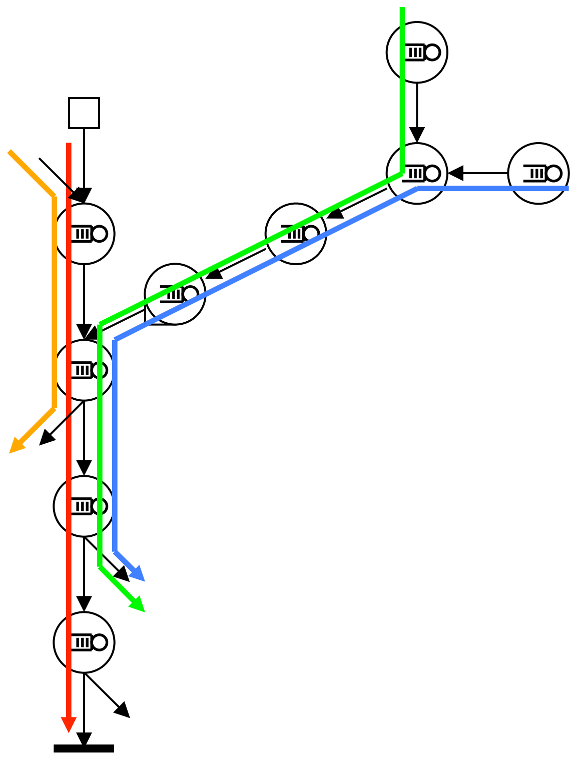
Remember this slide:
Output Bounds

- In order to obtain scenarios as before, the network must be trimmed first



- P{B,M}OO result can also be used for output bounds
  - recursive application along sub-paths shared by interfering flows
  - can become complex → tool support needed

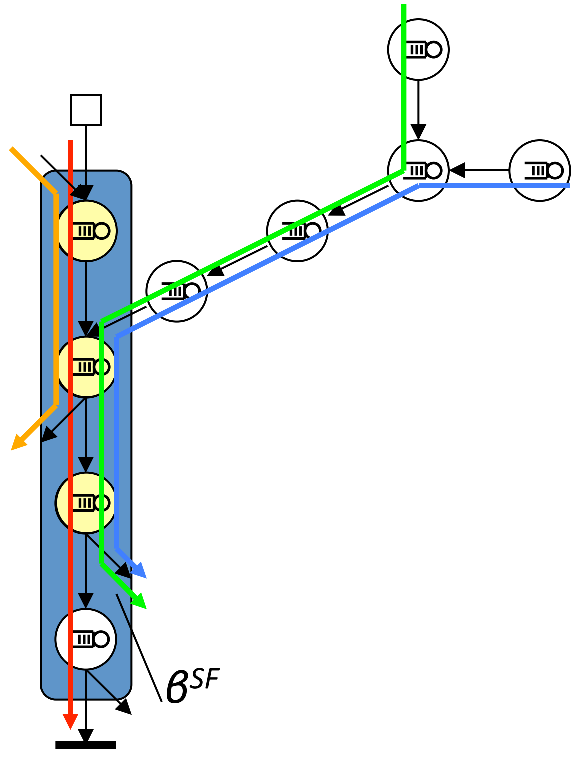
Total Flow Analysis



```
computeDelayBound( Flow f ) {
    computeOutputBound( sink(f),
            {flows from pred(sink(f)) to sink(f)} )
    total_delay = 0
    forall Node i∈path(f) {
        total_delay += $h\left(\alpha_{pred}, \beta_i^{eff}\right)$
    }
    return total_delay
}
```

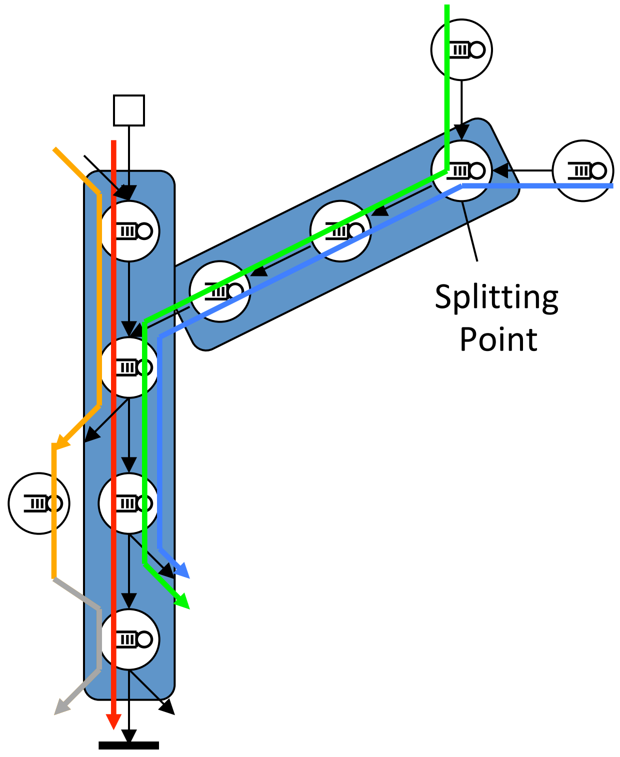
```
computeOutputBound( Node i, FlowSet $f_{out}$ ) {
    forall Node j∈predecessors(i) {
        $\alpha_{pred}$ = 0;    $\alpha_{excl}$ = 0
        $f_{in}$ = {flows from node j to node i};
        $\alpha_{pred}$ += computeOutputBound(j, $f_{in} \cap f_{out}$)
        $\alpha_{excl}$ += computeOutputBound(j, $f_{in} \setminus f_{out}$)
    }
    $\beta_i^{eff} = \left[\beta_i - \alpha_{excl}\right]^+$
    store( $\alpha_{pred}$, $\beta_i^{eff}$ )
    return $\alpha_{pred} \oslash \beta_i^{eff}$
}
```

Separated Flow Analysis



```
computeDelayBound( Flow $f$ ) {
    forall Node $i \in$ path($f$) {
        $\alpha_{pred} = 0$
        forall Node $j \in$ predecessors($i$) {
            $f_{in}$ = {flows from node $j$ to node $i$};
            $\alpha_{pred}$ += computeOutputBound($j$, $f_{in} \setminus \{f\}$)
        }
        $\beta_i^{eff} = [\beta_i - \alpha_{pred}]^+$
    }
    $\beta^{SF} = \otimes_{i=1}^{n} \beta_i^{eff}$
    return $h(\alpha_f, \beta^{SF})$
}
computeOutputBound( Node $i$, FlowSet $f_{out}$ ) {
    forall Node $j \in$ predecessors($i$) {
        $\alpha_{pred} = 0$;    $\alpha_{excl} = 0$
        $f_{in}$ = {flows from node $j$ to node $i$};
        $\alpha_{pred}$ += computeOutputBound($j$, $f_{in} \cap f_{out}$)
        $\alpha_{excl}$ += computeOutputBound($j$, $f_{in} \setminus f_{out}$)
    }
    $\beta_i^{eff} = [\beta_i - \alpha_{excl}]^+$
    store( $\alpha_{pred}$, $\beta_i^{eff}$ )
    return $\alpha_{pred} \oslash \beta_i^{eff}$
}
```

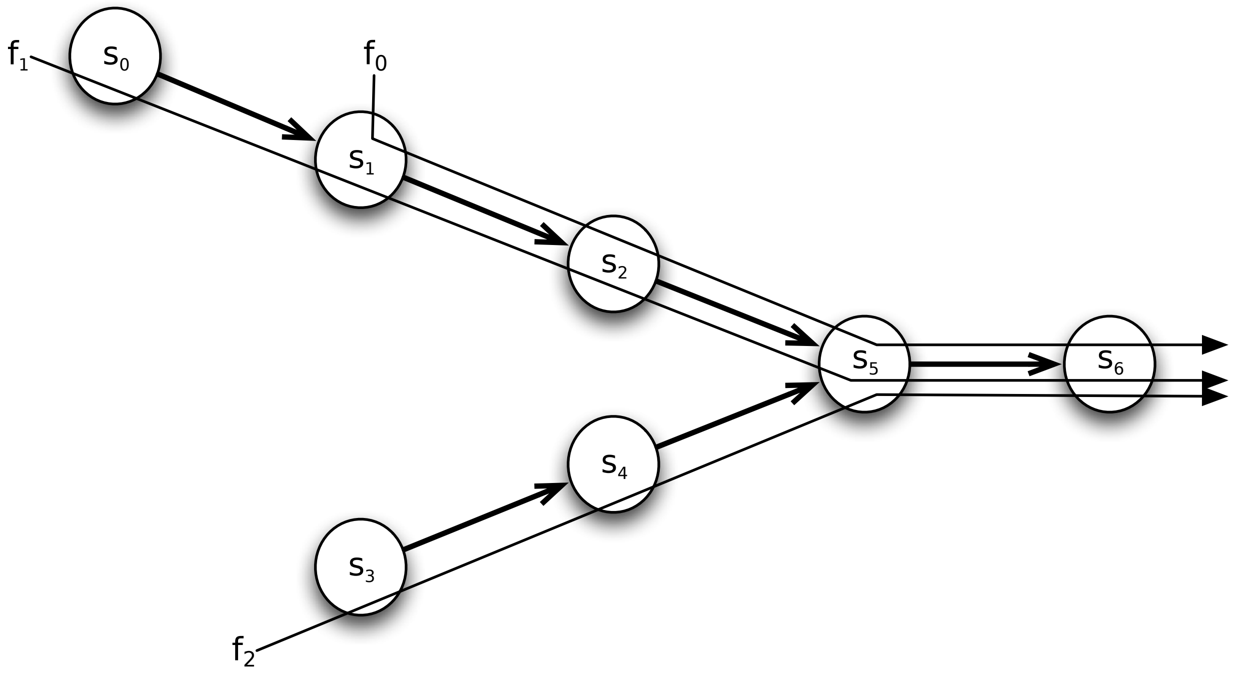
PMOO Analysis



```
computeDelayBound( Flow $f$ ) {
    $\beta^{PMOO}$ = computePMOOServiceCurve(path($f$), {$f$})
    return $h\left(\alpha_f, \beta^{PMOO}\right)$
}
computePMOOServiceCurve( Path $p$, FlowSet $f_{out}$ ) {
    eliminateRejoiningFlows()
    forall Flow $f_i \in$ interferingFlows() {
        Node $n_{fi}$ = ingressNode($f_i$)
        forall Node $j \in$ predecessors($n_{fi}$) {
            $\alpha_{fi}$ += computeOutputBound($j$, {$f_i$})
        }
    }
    return getPMOOServiceCurve($p$, $\alpha_{fi}$)
}
computeOutputBound( Node $i$, FlowSet $f_{out}$ ) {
    Node $s$ = getSplittingPoint()
    Path $p$ = getPath($i$, $s$)
    forall Node $j \in$ predecessors($s$) {
        $f_{in}$ = {flows from node $j$ to node $i$};
        $\alpha_s$ += computeOutputBound($j$, $f_{in} \cap f_{out}$)
    }
    $\beta^{PMOO}$ = computePMOOServiceCurve($p$, $f_{out}$)
    return $\alpha_s \oslash \beta^{PMOO}$
}
```

DNC Tool: Unexpected Observations (1)

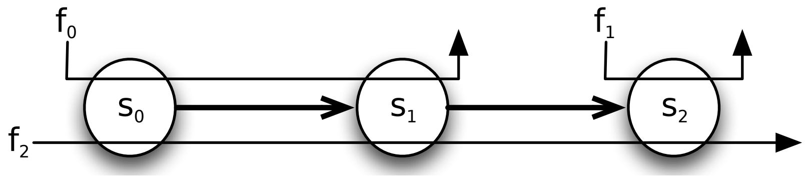
- TFA has a good scaling behavior w.r.t. backlog
- Derive a bound on f_1's data in the network, B

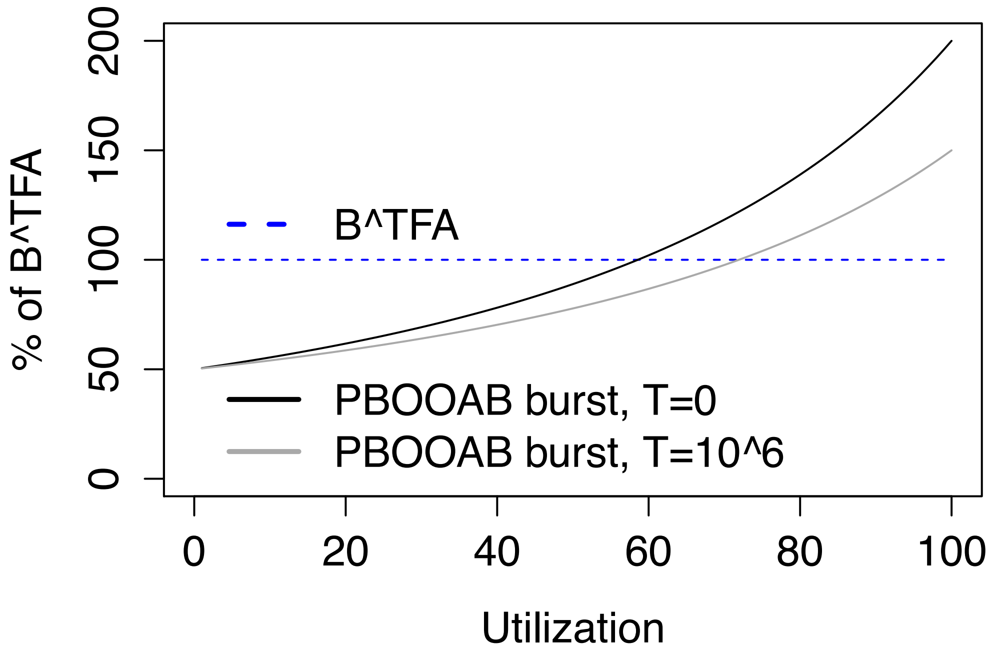


- $B^{TFA} = 1375$
- $B^{SFA} = 1391\ 2/3$

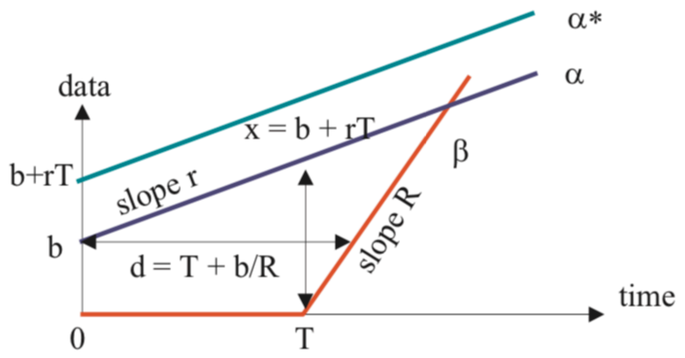
DNC Tool: Unexpected Observations (2)

- System backlog is an output bound!
- Analyze f_1's cross-traffic





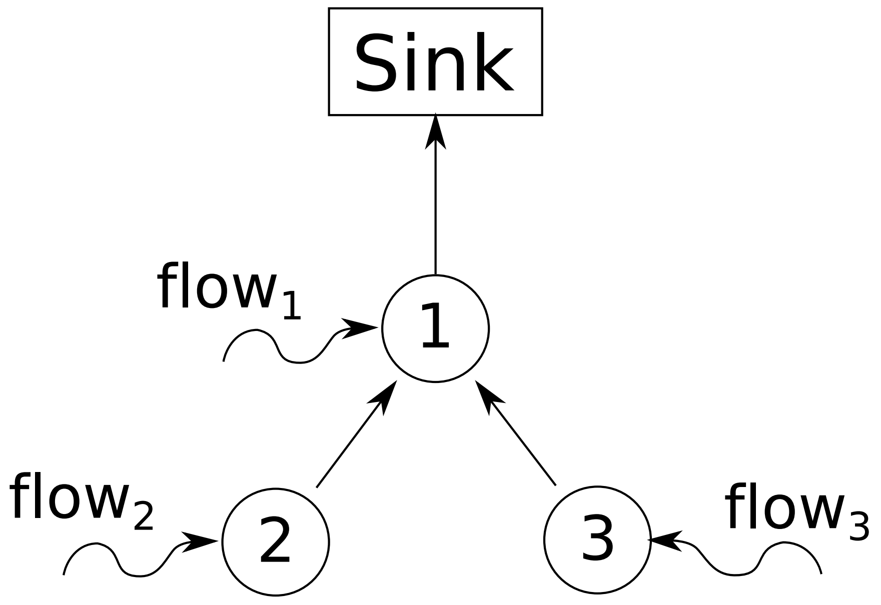
$$(f \oslash g)(t) = \sup_{u \geq 0}\{f(t+u) - g(u)\}$$



- How can f_2's output burstiness at s_2 possibly exceed the backlog bound at server s_1? Subtraction is overly pessimistic!

Short Tool Demo

- Arbitrary multiplexing delay bound in



References

[Bondorf17] S. Bondorf, P. Nikolaus, and J. Schmitt. Quality and Cost of Deterministic Network Calculus – Design and Evaluation of an Accurate and Fast Analysis. In Proc. of ACM SIGMETRICS 2017, with full version in Proceedings of the ACM on Measurement and Analysis of Computing Systems, volume 1, 2017.

[Bouillard10] A. Bouillard, L. Jouhet, and E. Thierry. Tight Performance Bounds in the Worst- Case Analysis of Feed-Forward Networks. In Proc. of IEEE INFOCOM, 2010.

[Bouillard14] A. Bouillard. Algorithms and Efficiency of Network Calculus. Habilitation thesis, ENS, 2014.

[Cruz91] R. L. Cruz. A Calculus for Network Delay, Part I: Network Elements in Isolation. In IEEE Transactions on Information Theory, 1991, and R. L. Cruz. A Calculus for Network Delay, Part II: Network Analysis. In IEEE Transactions on Information Theory, 1991.

[Fidler03] M. Fidler. Extending the Network Calculus Pay Bursts Only Once Principle to Aggregate Scheduling. In Proc. of QoS-IP, 2003.

[Kiefer10] A. Kiefer, N. Gollan, and J. Schmitt. Searching for Tight Performance Bounds in Feed-Forward Networks. In Proc. of GI/ITG MMB and DFT, 2010.

[LeBoudec01] J.-Y. Le Boudec and P. Thiran. Network Calculus: A Theory of Deterministic Queuing Systems for the Internet. Springer, 2001.

[Bondorf14] S. Bondorf and J. Schmitt. The DiscoDNC v2 – A Comprehensive Tool for Deterministic Network Calculus. In Proc. of ValueTools, 2014.

[Bondorf15] S. Bondorf and J. Schmitt. Calculating Accurate End-to-End Delay Bounds – You Better Know Your Cross-Traffic. In Proc. of ValueTools, 2015.

[Schmitt08] J. Schmitt, F. A. Zdarsky, and M. Fidler. Delay Bounds under Arbitrary Multiplexing: When Network Calculus Leaves You in the Lurch ... In Proc. of IEEE INFOCOM, 2008.

[Tomasik10] J. Tomasik and M.-A. Weisser. Internet Topology on AS-level: Model, Generation Methods and Tool. In Proc. IEEE IPCCC, 2010.

[Lampka17] K. Lampka, S. Bondorf, J. Schmitt, N. Guan and W. Yi. Generalized Finitary Real-Time Calculus. In Proc. of IEEE INFOCOM, 2014.

[Geyer19] F. Geyer and S. Bondorf. DeepTMA: Predicting Effective Contention Models for Network Calculus using Graph Neural Networks. In Proc. of INFOCOM, 2019.

[Bondorf16] S. Bondorf and J. Schmitt, *Improving Cross-Traffic Bounds in Feed-Forward Networks – There is a Job for Everyone*, In Proc. of GI/ITG MMB & DFT, 2016.

[Bondorf18] S. Bondorf, P. Nikolaus and J. Schmitt, *Catching Corner Cases in Network Calculus – Flow Segregation Can Improve Accuracy*, In Proc. of GI/ITG MMB, 2018.

[Scheffler18] A. Scheffler, M. Fögen and S. Bondorf, *The Deterministic Network Calculus Analysis: Reliability Insights and Performance Improvements*, In Proc of IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2018.