Advanced Computer Networking (ACN)

IN2097 – WiSe 2023–2024

Prof. Dr.-Ing. Georg Carle

Max Helm, Dr. Fabien Geyer

Chair of Network Architectures and Services
School of Computation, Information, and Technology
Technical University of Munich

Learning Objectives

At the end of this lecture, you will:

- Understand what Quality-of-Service and performance guarantees in computer networks are
- Express how to mathematically model the performance of a network (end-to-end latency)
- Know how to apply this model for computing worst-case latencies in a network

Quality-of-Service and Network Calculus

Introduction

Quality-of-Service

Performance guarantees

Deterministic Network Calculus

Basic elements: flows and servers

Performance bounds: delay and backlog

Min-plus algebra

Network analysis

Dealing with Packets

Conclusion

Tips on using network calculus

Recommended reading

Introduction

Quality-of-Service (QoS) is an important part of applications, services and network engineering.

QoS is used in various areas, with different level of criticality:

- Content delivery and streaming (ex: Youtube, Netflix, Spotify),
- Interactive applications (ex: gaming, VoIP, videoconferencing),
- Safety-critical applications (ex: aircraft, cars, remote surgery).

Performance guarantees and Service Level Agreement (SLA)

A **SLA** specifies bounds on metrics which need to be guaranteed in order for an application to work correctly.

Breaking a SLA can have different impacts depending on the application.

Examples of SLA

Similarly to the ISO layers, SLAs can be given at different layers:

Packet	• End-to-end latency and jitter • Packet loss
Flow	• Throughput • Transfer completion time
Application	• Time to complete a transaction (e.g. SQL, download, backup...)
User level	• Video or audio quality • Quality-of-Experience

Focus of today's talk

Packet level guarantees in wired networks (ex: Ethernet networks)

Quality-of-Service

To meet SLAs, the following tasks are needed:

- **Modeling** – Understand which part of the network have an impact on the SLA
- **Classification** – Identify which packet needs which service
- **Scheduling** – Give preferential service to packets
- **Monitoring** – Check that SLAs are met with passive or active measurements

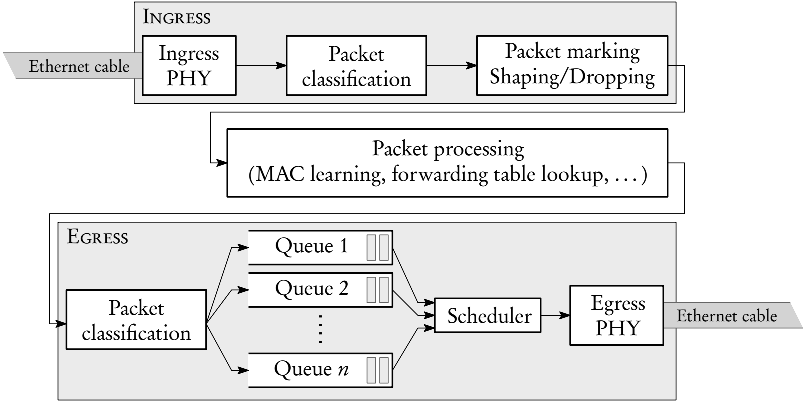


Figure 1: Packet processing pipeline of an Ethernet switch

Performance guarantees

Methods used in order to validate that SLAs will not be broken:

- Live testing in worst-case conditions,
- Emulations and simulations,
- Formal verification using analytical models.

Formal methods using mathematical modeling

Input Network topology and configuration, flow descriptions

Output Bound on metrics (ex: end-to-end latencies, buffer sizes)

If the bounds given by a formal method are below the requirements of a SLA, then it guarantees that the SLA will not be broken.

Question

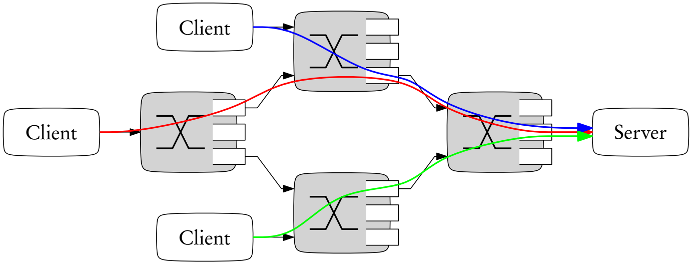
Question: Which method would you use for validating SLAs in a car or an aircraft?

1. Live testing in worst-case conditions
2. Emulations and simulations
3. Formal verification using analytical models

Latencies in computer networks

Different latencies inside a network:

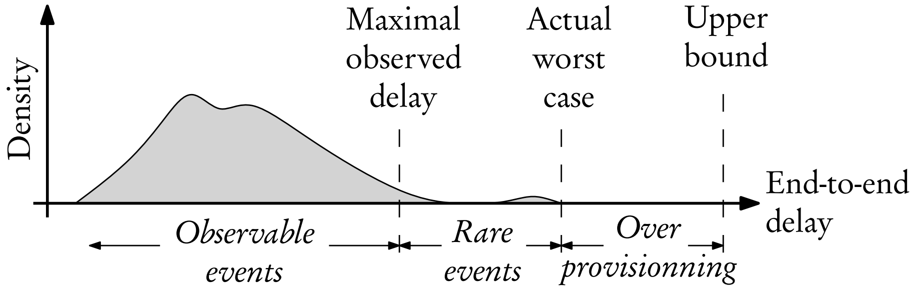
Propagation	depends on cable lengths and physical signal propagation,
Processing	depends on hardware,
Transmission	depends on packet sizes,
Queuing	**depends on the behavior of the other flows and the scheduling!**



End-to-end latency for the red flow

$$4 \cdot (D_{Propagation} + D_{Transmission}) + 3 \cdot D_{Processing} + D_{Queuing}$$

Vocabulary: performance bounds



Maximal Observed Delay Maximal delay which is measured on a real network during its normal operation, or via simulations.

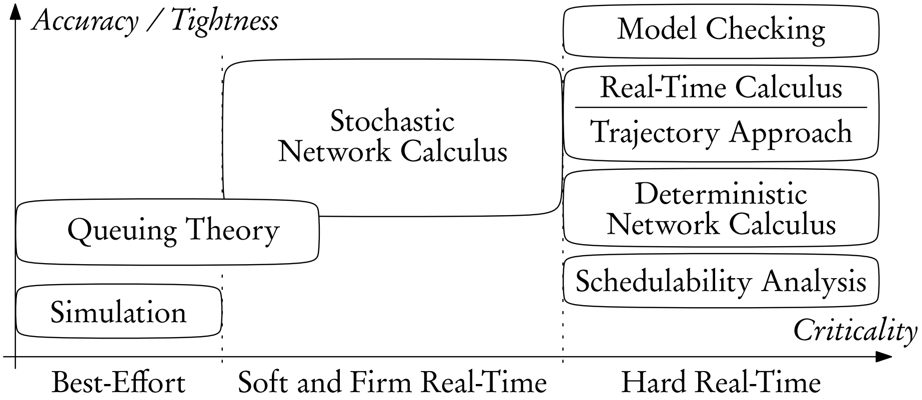
Actual Worst Case Theoretical worst case delay which can actually occur in case the elements of the network behave within their limits, but in a very specific pattern leading to this worst-case

Upper Bound Bound calculated by an analytical model, which is generally larger than the actual worst case due to approximations, simplifications or shortcomings of the formal method.

Vocabulary: requirements criticality

Hard real-time	Missing a deadline is a total system failure
Firm real-time	Infrequent deadline misses are tolerable. Late packets are discarded
Soft real-time	Infrequent deadline misses are tolerable. Late packets may be used
Best-effort	No deadline requirements

Performance guarantees
Different mathematical frameworks



- Better accuracy and tightness is often payed for with more complexity of the mathematical tools and more CPU time.
- Finding the exact worst-case is often an NP-hard problem.

Overview of network calculus

Network calculus is a theoretical framework developed for analyzing performance guarantees (latencies and buffer sizes) in networks of queues and schedulers.

It is mostly applied to communication networks and has some real-world industrial uses (ex: validation of embedded networks inside the Airbus A380 and A350).

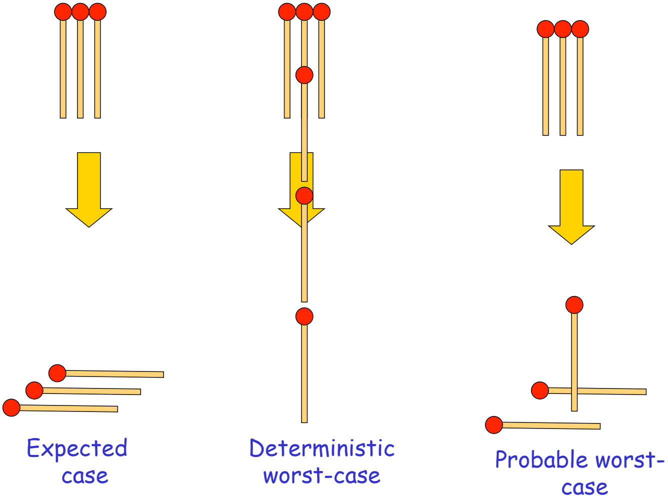
It has been developed since the early 1990's and it is still under active research.

Two variants of network calculus:

- **Deterministic** No randomness is involved, meaning that performances are guaranteed whatever happens on the network,
- **Stochastic** Randomness is involved, meaning that performances are characterized in probabilistic terms.

Deterministic Network Calculus

Illustration on worst-case analysis: dropping matches[1]



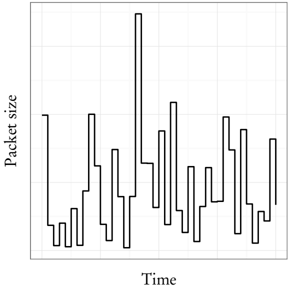
1 Source: Keynote from Prof. Jörg Liebeherr, Workshop on Network Calculus (WoNeCa), 2014

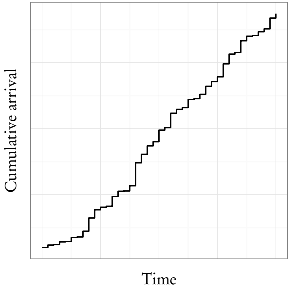
Flow description: cumulative arrival function

In network calculus, packets and network protocols are described as **flows**, meaning an unidirectional set of packets going from a sender to a receiver.

A flow is modeled by its **cumulative arrival function** A, where $A(t)$ represents the amount of data sent by the flow in the time[2] interval $[0, t)$. A is a non-decreasing strictly positive function, or more formally it is a member of the following set:

$$\mathcal{F} = \{f : \mathbb{R}^+ \rightarrow \mathbb{R}^+ | \ \forall 0 \leq t \leq s : f(t) \leq f(s), f(0) = 0\} \quad (1)$$





2 t can be discrete or continuous in DNC

Flow description: arrival curve

Arrival Curve

A flow is said to have a **deterministic arrival curve** $\alpha \in \mathcal{F}$ if its cumulative arrival function A satisfies:

$$A(t+s) - A(t) \leq \alpha(s), \forall (s,t) \geq 0 \quad (2)$$

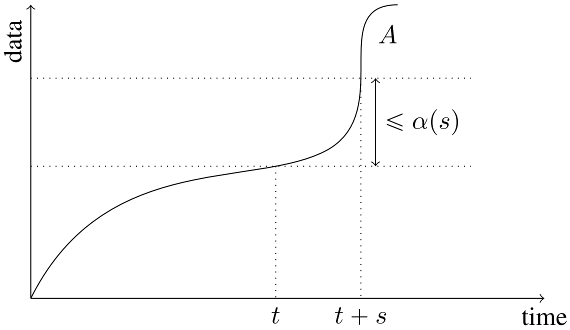
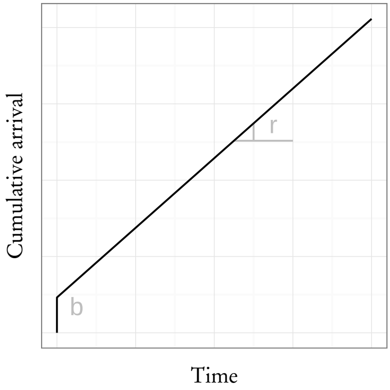


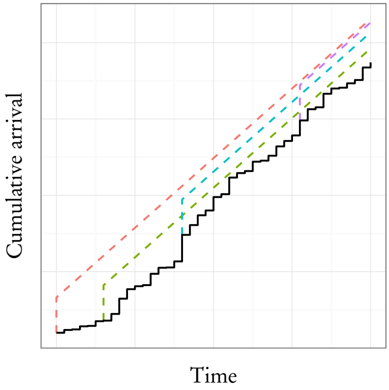
Figure 2: Illustration of Equation 2 (source: Bouillard et al., 2018)

Arrival curve: Example

A simple example of deterministic arrival curve is the **token bucket**, which is defined by its long term average rate r and its burstiness parameter b (often noted $\gamma_{r,b}$):

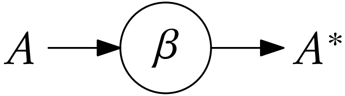
$$\gamma_{r,b}(s) = r \cdot s + b, \forall s \geq 0 \tag{3}$$





Server description

In network calculus, queues and schedulers are described as **servers**.



Service Curve

A server S offers a strict service curve β if, during any period of duration Δ where there is data waiting to be served by S, the output of S is at least $\beta(\Delta)$:

$$A^*(t + \Delta) - A^*(t) \geq \beta(\Delta) \tag{4}$$

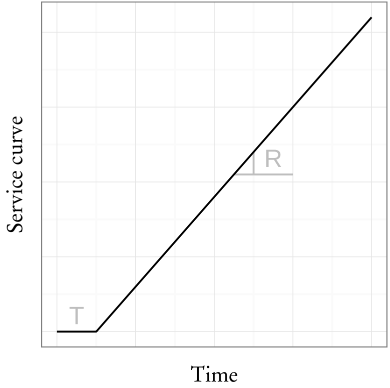
A^* can also be defined as:

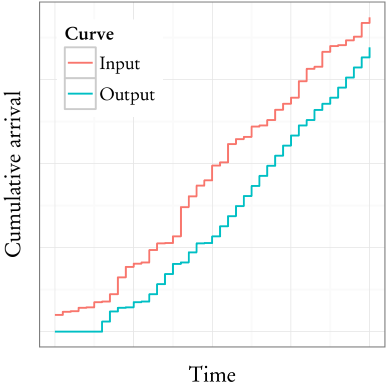
$$A^*(t) \geq A(s) + \beta(t - s) \tag{5}$$

Server description: Example

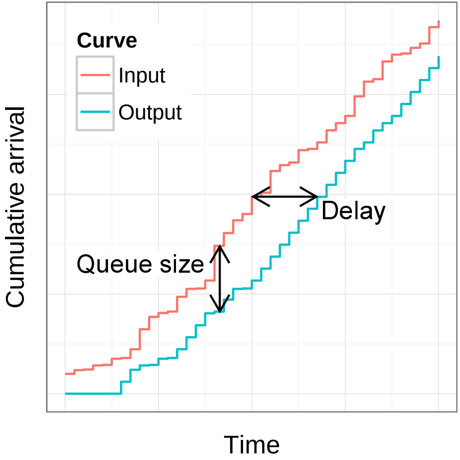
A simple example of deterministic service curve is the **rate-latency**, defined by its rate R and processing delay T (often noted $\beta_{R,T}$):

$$\beta_{R,T}(t) = R[t - T]^{+}, \text{ where } [x]^{+} = \max(0, x) \tag{6}$$





Intuition behind performance bounds



Delay time it takes for a packet to traverse the queue

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

$\Rightarrow$ **Horizontal deviation**

Queue size backlog at the server

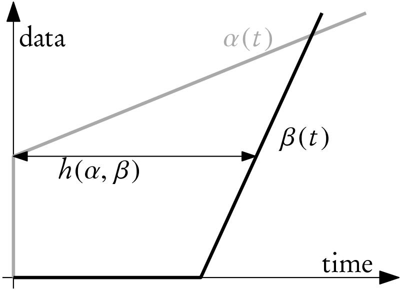
$B(t) = A(t) - A^*(t)$

$\Rightarrow$ **Vertical deviation**

Delay bound

The **delay bound** corresponds to the maximum time a given data has to wait before being processed by the server.

In graphical terms it corresponds to the maximum horizontal deviation between the arrival and service curve.



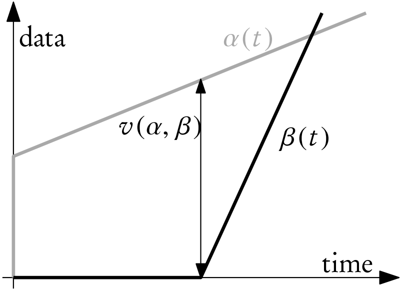
In mathematical terms:

$$A^*(t) - A(t-s) \leq \sup_{t \geq 0} \left\{ \inf_{s \geq 0} \{\alpha(t) \leq \beta(t+s)\} \right\} \tag{7}$$

Backlog bound

The **backlog bound** corresponds to the maximum amount of data that will have to wait before being processed by the server.

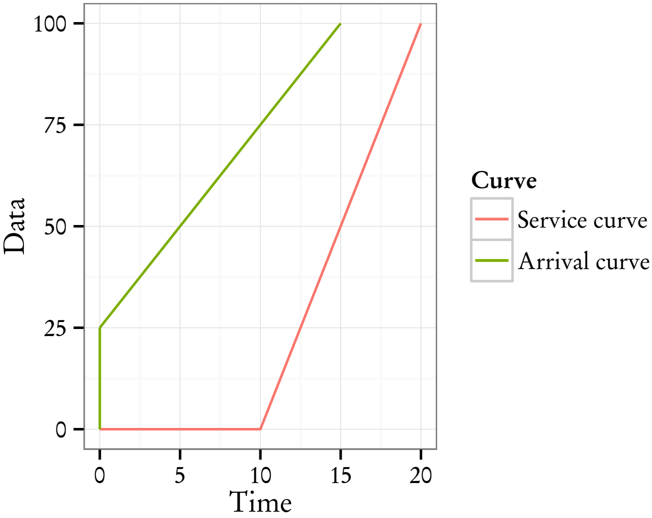
In graphical terms it corresponds to the maximum vertical deviation between the arrival and service curve.



In mathematical terms:

$$A(t) - A^*(t) \leq \sup_{s \geq 0}\{\alpha(s) - \beta(s)\} \quad (8)$$

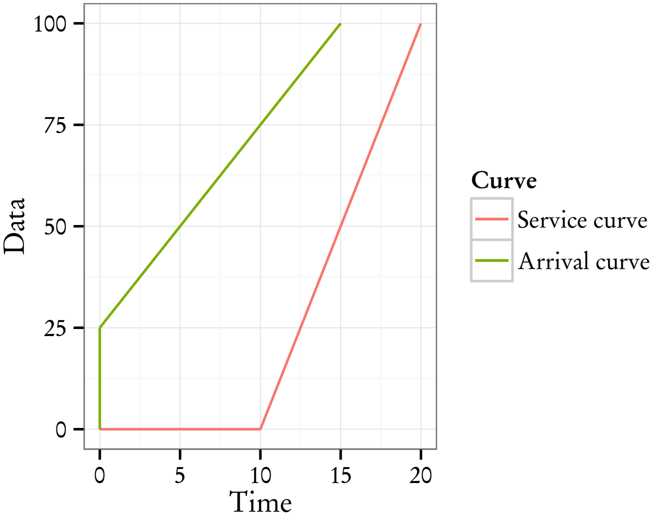
Performance bounds: delay and backlog
Question



Question: What is the latency bound?

1. 10
2. 12.5
3. 25

Performance bounds: delay and backlog
Question



Question: What is the backlog bound?

1. 25
2. 50
3. 75

Min-plus algebra

The mathematical operations presented earlier can be abstracted in the **min-plus algebra**.

Min-plus algebra operators

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

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

The previous expression can then be expressed as:

Output curve $A^*(t) \geq (A \otimes \beta)(t)$

Output envelope $\alpha^*(t) = (\alpha \oslash \beta)(t)$

Delay bound $\inf\{s : (\alpha \oslash \beta)(-s) \leq 0\}$

Backlog bound $(\alpha \oslash \beta)(0)$

Application of deconvolution operation

Once a flow has traversed a server, it's envelope will change and is characterized by

$$\alpha^*(t) = (\alpha \oslash \beta)(t)$$

Application: a token bucket $\gamma_{r,b}$ traversing a rate-latency server $\beta_{R,T}$

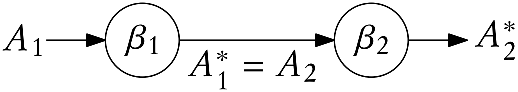
$$(\gamma_{r,b} \oslash \beta_{R,T})(t) = \gamma_{r,b+rT}(t) \tag{9}$$

Demonstration:

$$\begin{aligned}
(\gamma_{r,b} \oslash \beta_{R,T})(t) &= \sup_{s \geq 0} \{\gamma_{r,b}(t+s) - \beta_{R,T}(s)\} \\
&= \sup_{s \geq 0} \left\{\gamma_{r,b}(t+s) - R[s-T]^+\right\} \\
&= \sup_{0 \leq s \leq T} \{\gamma_{r,b}(t+s)\} \vee \sup_{s > T} \{\gamma_{r,b}(t+s) - R(s-T)\} \\
&= \{\gamma_{r,b}(t+T)\} \vee \{\gamma_{r,b}(t+T)\} \\
&= \gamma_{r,b+rT}(t)
\end{aligned}$$

Network analysis
Multiple servers: concatenation property

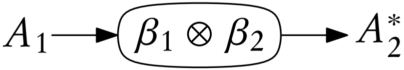
Let us consider the following example of a flow traversing two servers:



From the previous results we have:

$$
\begin{aligned}
A_2^*(t) &\geq (A_2 \otimes \beta_2)(t) && \\
&\geq (A_1^* \otimes \beta_2)(t) && \text{using } A_2 = A_1^* \\
&\geq ((A_1 \otimes \beta_1) \otimes \beta_2)(t) && \text{using } A_1^*(t) \geq (A_1 \otimes \beta_1)(t) \\
&\geq (A_1 \otimes (\beta_1 \otimes \beta_2))(t) && \text{associativity of } \otimes
\end{aligned}
$$

Hence, the two servers can be concatenated into one:

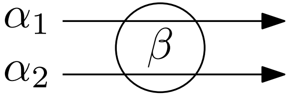


This property is also called **Pay Burst Only Once**

Example: $(\beta_{R_1,T_1} \otimes \beta_{R_2,T_2})(t) = \beta_{\min(R_1,R_2),T_1+T_2}(t)$

Multiple flows: left-over property

Let us consider the following example of two flows traversing a server:



The performance of one flow depends on the influence of the other flow. If we don't know anything about it, we assume the worst case. This is called **blind multiplexing** or **arbitrary multiplexing**.

To compute the bounds of flow 1, we use the service which left after flow 2 has been serviced. This is called the **residual** or **left-over service curve**:

$$\beta^{l.o.} = [\beta - \alpha_2]^+ \tag{10}$$

If $\beta = \beta_{R,T}$ and $\alpha_2 = \gamma_{r,b}$, the left-over service curve for α_1 is:

$$\beta^{l.o.} = \beta_{R-r,\frac{b+RT}{R-r}} \tag{11}$$

Strict Priority Queuing – Reminder

Principle

- Queues are polled in their priority order, until a non-empty queue is found
- A queue can be served only if all higher priority queues are empty

This packet scheduling algorithm can be found in the majority of Ethernet switches from the market

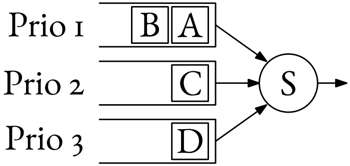
Pros

- Easy implementation
- Zero configuration
- Simple formal verification

Cons

- Starvation problem

Strict Priority Queuing – Question



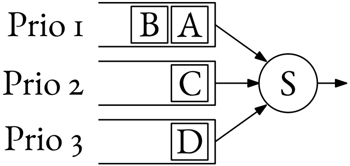
- Prio 1: highest priority
- Prio 3: lowest priority

Question: In which order will the packets be processed?

1. D - C - A - B
2. A - B - C - D
3. A - C - D - B
4. B - A - C - D

Strict Priority Queuing - Service curves

- Let flow i be a flow with arrival curve α_i
- If $i < j$, then flow i has a higher priority than flow j
- The flows traverse a server S offering a service curve β



Left-over service curve for Strict Priority

$\beta^{l.o.i}$ is a strict service curve offered to flow i, with:

$$\beta^{l.o.i} = \left[\beta - \sum_{k=1}^{i-1} \alpha_k\right]^+ \tag{12}$$

$$\beta^{l.o.1} = \beta \tag{13}$$

$$\beta^{l.o.2} = [\beta - \alpha_1]^+ \tag{14}$$

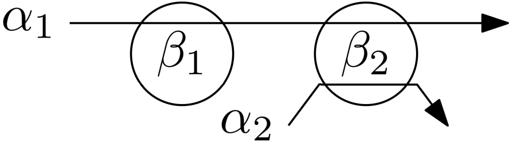
$$\beta^{l.o.3} = [\beta - (\alpha_1 + \alpha_2)]^+ \tag{15}$$

Note: this result considers that the scheduler is preemptive

Bounds in network of servers and flows

To compute bounds in a network, the operations previously presented can be applied in different orders. The simplest method – called **Separate Flow Analysis** – is:

1. Compute the left-over service curve for each server traversed by the flow
2. Concatenate the left-over service curves
3. Compute the bounds



If we analyze flow 1 under blind multiplexing, we have for step 1:

$$\beta_1^{l.o.1} = \beta_1$$

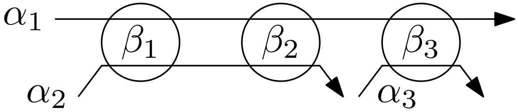
$$\beta_2^{l.o.1} = [\beta_2 - \alpha_2]^+$$

Then, we compute the bounds using: $\beta_1^{l.o.1} \otimes \beta_2^{l.o.1}$

Bounds in network of servers and flows

To compute bounds in a network, the operations previously presented can be applied in different orders. The simplest method – called **Separate Flow Analysis** – is:

1. Compute the left-over service curve for each server traversed by the flow
2. Concatenate the left-over service curves
3. Compute the bounds



If we analyze flow 1 under blind multiplexing, we have for step 1:

$$\beta_1^{l.o.1} = [\beta_1 - \alpha_2]^+$$
$$\beta_2^{l.o.1} = [\beta_2 - \alpha_2^*]^+ = [\beta_2 - (\alpha_2 \oslash \beta_1^{l.o.2})]^+ = [\beta_2 - (\alpha_2 \oslash [\beta_1 - \alpha_1]^+)]^+$$
$$\beta_3^{l.o.1} = [\beta_3 - \alpha_3]^+$$

Then, we compute the bounds using: $\beta_1^{l.o.1} \otimes \beta_2^{l.o.1} \otimes \beta_3^{l.o.1}$

Packetizer

We looked at a **fluid model**, i.e. data can be indefinitely divided into small parts! **What about packets?**

A **packetizer** is a server that groups the data of a flow into its packets: it stores the bits of data of a packet until the whole packet has arrived. When the last bit of the packet arrives, it serves all the bits of the packet simultaneously.

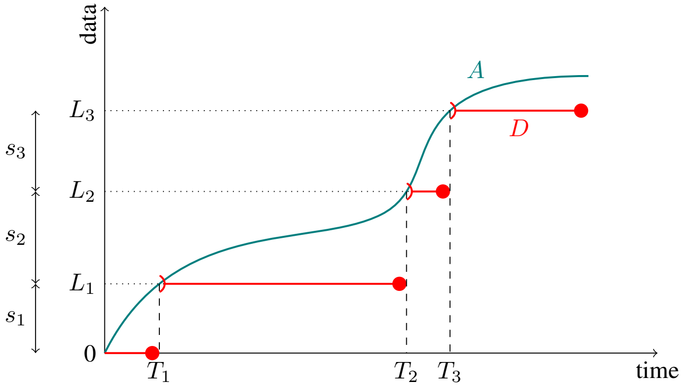


Figure 3: Illustration of a packetizer (source: Bouillard et al., 2018)

Introduction

Deterministic Network Calculus

Conclusion

- Tips on using network calculus
- Recommended reading

Deploying network calculus

Verification using network calculus is only one step towards guarantees!

- **But what happens if a computer sends more data than what was expected?**

Enforcement: switches and routers need to check that each flow is conform to its arrival curve.

Some solutions are already available:

- In commercial switches and routers:
  - Mechanisms: rate shaping/limiting and flow filtering
  - Protocols: IntServ and RSVP, DiffServ, MPLS-TE
- For critical applications (ex: aircrafts, cars, . . .):
  - Commercial devices usually have limitations or unwanted functionalities
  - Custom-designed and/or industry-specific devices and protocols
- For Ethernet-based networks: new protocols are currently being standardized:
  - IEEE Time Sensitive Networking (TSN)

Not presented today

This was only a brief introduction to network calculus.

Not presented today:

- Models of packet scheduling,
- Different schedulers: FIFO, Generalized Processor Sharing, Deficit Round Robin, Fair Queuing, . . . ,
- Stochastic variant of network calculus,
- Computational aspects:
  - Some open-source and commercial tools available (eg: NetworkCalculus.org DNC[3], Nancy[4])
- More complex network protocols, ex: TCP

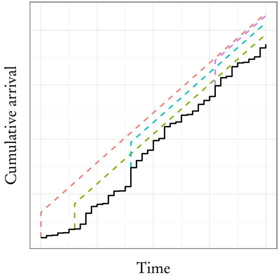
3 https://github.com/NetCal/DNC

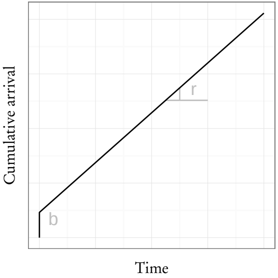
4 https://rzippo.github.io/nancy

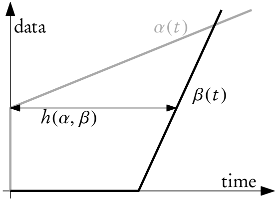
Key concepts of network calculus

The key concepts of network calculus are:

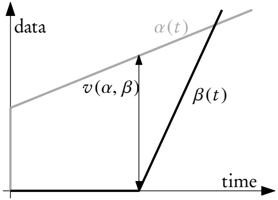
- Study the **cummulative arrival** of traffic,
- Characterize the behavior of servers as **function of the cummulative arrival**,
- Define a (simple) **bounding function** of the traffic and server behavior,
- Work with the bounding functions – not the traffic itself – to compute bounds.







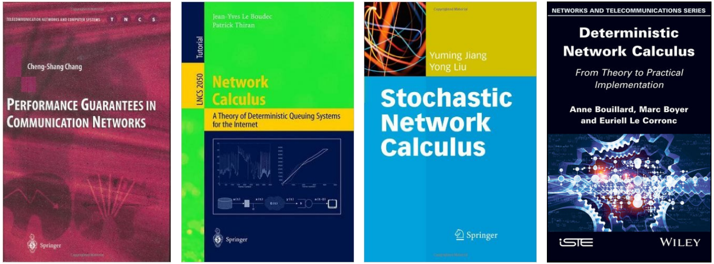
Delay bound



Backlog bound

Recommended reading

- **(DNC+SNC)** Performance Guarantees in Communication Networks, Chang, 2000
- **(DNC)** Network Calculus – A Theory of Deterministic Queuing Systems for the Internet, Le Boudec and Thiran, 2001 *(also available for free online[5])*
- **(SNC)** Stochastic Network Calculus, Jiang and Liu, 2008
- **(DNC)** Deterministic Network Calculus: From Theory to Practical Implementation, Bouillard, Boyer, Le Corronc, 2018



Current research:

- Bi-yearly workshop: `https://plassart.github.io/WoNeCa/2022/` (includes video recordings of talks)

5 http://ica1www.epfl.ch/PS_files/NetCal.htm