Advanced Computer Networking (ACN)

IN2097 – WiSe 2019-2020

Prof. Dr.-Ing. Georg Carle

Fabien Geyer

Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich
Chapter 11: 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
Scheduling
Dealing with Packets

Conclusion

Tips on using network calculus
Active research field
Recommended reading
Chapter 11: Quality-of-Service and Network Calculus

Introduction

Quality-of-Service

Performance guarantees

Deterministic Network Calculus

Conclusion
Quality-of-Service

Introduction

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: aircrafts, 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.
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

![Packet processing pipeline of an Ethernet switch](image)

*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**

<table>
<thead>
<tr>
<th>Input</th>
<th>Network topology and configuration, flow descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>Bound on metrics (ex: end-to-end latencies, buffer sizes)</td>
</tr>
</tbody>
</table>

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

**Exact 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 delay.

**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.
### Performance guarantees

**Vocabulary: requirements criticality**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard real-time</td>
<td>Missing a deadline is a total system failure</td>
</tr>
<tr>
<td>Firm real-time</td>
<td>Infrequent deadline misses are tolerable. Late packets are discarded</td>
</tr>
<tr>
<td>Soft real-time</td>
<td>Infrequent deadline misses are tolerable. Late packets may be used</td>
</tr>
<tr>
<td>Best-effort</td>
<td>No deadline requirements</td>
</tr>
</tbody>
</table>
Better accuracy and tightness is often paid with more complexity of the mathematical tools and more CPU time.

Finding the exact worst-case is often a NP-hard problem.
Introduction

Deterministic Network Calculus

- Basic elements: flows and servers
- Performance bounds: delay and backlog
- Min-plus algebra
- Network analysis
- Scheduling
- Dealing with Packets

Conclusion
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

Expected case

Deterministic worst-case

Probable worst-case

---

1 Source: Keynote from Prof. Jörg Liebeherr, Workshop on Network Calculus (WoNeCa), 2014
Basic elements: flows and servers

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 interval $[0, t]$. $A$ is a non-decreasing strictly positive function, or more formally it is a member of the following set:

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

---

\(^2\) $t$ can be discrete or continuous in DNC
Arrival Curve
A flow is said to have a **deterministic arrival curve** $\alpha \in \mathcal{F}$ if its cumulative arrival function $A$ satisfies:

$$A(t + s) - A(t) \leq \alpha(s), \forall (s, t) \geq 0$$  \hspace{1cm} (2)

**Figure 2**: Illustration of Equation 2 (source: Bouillard et al., 2018)
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 $\gamma_{r,b}$):

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

(3)
Basic elements: flows and servers
Server description

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

\[ A \rightarrow \beta \rightarrow A^* \]

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

\[ A^*(t + s) - A^*(t) \geq \beta(s) \]  \hspace{1cm} (4)

\( A^* \) can also be defined as:

\[ A^*(t) \geq A(s) + \beta(t - s) \]  \hspace{1cm} (5)
A simple example of deterministic service curve is the **rate-latency**, defined by its rate $R$ and processing delay $T$ (often noted $\beta_{R,T}$):

$$\beta_{R,T}(t) = R[t - T]^+, \text{ where } [x]^+ = \max(0, x)$$ (6)
Performance bounds: delay and backlog

Intuition behind performance bounds

Delay time it takes for a packet to traverse the queue

\[ A^*(t) - A(t - s) \]

⇒ Horizontal deviation

Queue size backlog at the server

\[ A(t) - A^*(t) \]

⇒ Vertical deviation
Performance bounds: delay and backlog

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:

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

(7)
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:

\[
A(t) - A^*(t) \leq \sup_{s \geq 0} \{ \alpha(s) - \beta(s) \}
\]  

(8)
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**
  \[ (\alpha \oslash \beta)(0) \]
Min-plus algebra
Application of deconvolution operation

Once a flow has traversed a server, its envelope will change and is characterized by
\[ \alpha^*(t) = (\alpha \ominus \beta)(t) \]

**Application:** a token bucket \( \gamma_{r,b} \) traversing a rate-latency server \( \beta_{R,T} \)
\[
(\gamma_{r,b} \ominus \beta_{R,T})(t) = \gamma_{r,b+rT}(t) \tag{9}
\]

**Demonstration:**

\[
(\gamma_{r,b} \ominus \beta_{R,T})(t) = \sup_{s \geq 0} \{ \gamma_{r,b}(t + s) - \beta_{R,T}(s) \} \\
= \sup_{s \geq 0} \left\{ \gamma_{r,b}(t + s) - R[s - T]^+ \right\} \\
= \sup_{0 \leq s \leq T} \{ \gamma_{r,b}(t + s) \} \lor \sup_{s > T} \{ \gamma_{r,b}(t + s) - R(s - T) \} \\
= \{ \gamma_{r,b}(t + T) \} \lor \{ \gamma_{r,b}(t + T) \} \\
= \gamma_{r,b+rT}(t)
\]
Network analysis

Multiple servers: concatenation property

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

\[
A_1 \xrightarrow{\beta_1} A_1^* = A_2 \xrightarrow{\beta_2} A_2^*
\]

From the previous results we have:

\[
A_2^*(t) \geq (A_2 \otimes \beta_2)(t)
\]

\[
\geq (A_1^* \otimes \beta_2)(t)
\]

\[
\geq ((A_1 \otimes \beta_1) \otimes \beta_2)(t)
\]

\[
\geq (A_1 \otimes (\beta_1 \otimes \beta_2))(t)
\]

Hence, the two servers can be concatenated into one:

\[
A_1 \xrightarrow{\beta_1 \otimes \beta_2} A_2^*
\]

This property is also called Pay Burst Only Once

Example: \((\beta_{R_1,T_1} \otimes \beta_{R_2,T_2})(t) = \beta_{\min(R_1,R_2),T_1+T_2}(t)\)
Scheduling
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
Scheduling
Strict Priority Queuing - Service curves

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

Residual service curve for Strict Priority

$\beta_i$ is a strict service curve offered to flow $i$, with:

$$\beta_i = \left[ \beta - \sum_{k=1}^{i-1} \alpha_k \right]^+$$  \hspace{1cm} (10)

Note: this result considers that the scheduler is preemptive
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)
Chapter 11: Quality-of-Service and Network Calculus

Introduction

Deterministic Network Calculus

Conclusion

Tips on using network calculus

Active research field

Recommended reading
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 **bounding function** of the traffic and server behavior, either deterministic or stochastic,
- Work with the bounding functions, not the traffic itself, to give bounds.
This was only a brief introduction to network calculus.

Not presented today:

- Models of packet scheduling,
- Stochastic variant of network calculus,
- Computational aspects:
  - Few open-source and commercial tools available (eg: DiscoDNC\(^3\))

\(^3\)http://disco.informatik.uni-kl.de/index.php/projects/disco-dnc
Tips on using network calculus

When should DNC be used?

Network calculus shines in the following use-cases:

- Multi-hop real-time communications,
- No simple analysis exists (due to state-space explosion),
- No cyclic dependencies or feedback loops (eg. TCP).
Verification using network calculus is only the first 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
  - OpenFlow meters
Active research field

Network calculus is still under active research

- Computational aspects
- Practical real-world uses
- Pessimism of the method
- Not all network protocols easily fit into network calculus (ex: TCP)
- Application layer performance
- Network calculus as a network design tool
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)*
- **(DNC)** Deterministic Network Calculus: From Theory to Practical Implementation, Bouillard, Boyer, Le Corronc, 2018

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