Trabalho Prático Nº.2 – Streaming de áudio e vídeo a pedido e em tempo real

Duração: 2 aulas

Este trabalho deve ser realizado com recurso à máquina virtual **XubunCORE_7_5** que está disponibilizada em http://marco.uminho.pt/ferramentas/CORE/xubuncore.html (user: *core* password: *core*)

ATENÇÃO: *É preciso estar preparado a eventualidade de ter de desligar abruptamente a máquina virtual ("power off"), se o streaming consumir muito CPU, apesar dos cuidados na geração de vídeos. Exemplo: evitar a todo o custo o modo "full screen"!*

Objetivos

Neste trabalho pretende-se experimentar várias soluções de *streaming* a pedido e em tempo real usando o emulador CORE como bancada experimental. O objetivo é em primeiro lugar perceber as opções disponíveis em termos de pilha protocolar e as diferenças conceptuais entre elas. Por outro lado, espera-se também alguma familiarização com os formatos multimédia e como se faz o seu empacotamento (apenas de forma superficial, dada a imensidão de possibilidades), bem como com as ferramentas *open source* disponíveis para uso.

Ferramentas

- CORE Emulator – https://github.com/coreemu/core
- Wireshark – https://www.wireshark.org/
- FFmpeg – https://ffmpeg.org/
- VideoLAN VLC – https://www.videolan.org/
- OBS Studio – https://obsproject.com/

Descrição Geral

Esta proposta de trabalho está estruturada em 3 etapas. Inicialmente, antes de qualquer etapa, vamos criar manualmente um pequeno vídeo a partir da captura de ecrã, usando o *ffmpeg*. Esse vídeo será depois usado em todas as etapas. Na etapa 1, pretende-se fazer *streaming* por HTTP apenas, sem adaptação do débito. Nesta etapa o VLC é usado simultaneamente como servidor de *streaming* e como cliente. Depois são acrescentados mais dois clientes (*firefox* e *ffplay*) e estuda-se o impacto no tráfego da rede. Na etapa 2, o servidor de *streaming* é substituído pelo fantástico *ffmpeg* (que faz tudo e mais alguma coisa ☺), que serve o mesmo conteúdo por HTTP mas agora com débito adaptativo. Para simplificar, usa-se apenas duas *streams* com diferentes exigências em termos de recursos. Fazem-se umas manipulações da capacidade dos links, para ver como o broswer se tenta ajustar. Posteriormente, na etapa 3, vamos usar os históricos protocolos de streaming sobre UDP. Experimenta-se em primeiro lugar o uso do RTP/RTCP em unicast e anúncios SAP, com 3 clientes, mudando de *unicast* para *multicast* (ao nível da rede) para perceber o impacto em termos de tráfego na rede. Aqui é importante a análise protocolar.

Captura de um vídeo básico com gravação do ecrã

A experimentação no emulador deste tipo de aplicações, que consomem por vezes muitos recursos, pode ser problemática. O streaming vai por isso ser condicionado a uma amostra exemplificativa, que vai ser criada manualmente com auxílio da ferramenta **ffmpeg** (documentação extra no ANEXO 4).

Tarefas:

1. Instalar o *ffmpeg* com **`sudo apt install ffmpeg`**
2. Familiarização com algumas opções da linha de comando do *ffmpeg*: *-formats, -codecs, -devices e -protocols*.
3. Executar uma aplicação X11 gráfica (GUI) simples, como por exemplo o *xeyes*:
   core@xubunxore ~$ **`xeyes -geometry +200+300 &`**
   (a opção -geometry posiciona a janela nas coordenadas indicadas e o & manda para background – google "X11 command line opions"
4. Capturar uma janela de 180 por 120 (ajustar se necessário) daquela posição do ecrã com o *ffmpeg* gravando no ficheiro *video1.mp4*
   core@xubunxore ~$ **`ffmpeg -f x11grab -video_size 180x120 -framerate 20 -i :0.0+200,300 -pix_fmt yuv420p video1.mp4`**
   (0.0 é a identificação do *screen.display* do X11 e o +200,300 o deslocamento no X e no Y, para apanhar os olhos; o formato de saída é MP4)

5. Mexer o rato em círculos à volta dos olhos, que vão seguir o rato, durante alguns segundos, e depois parar a gravação com um **Ctrl+C**.
6. Para verificar se o vídeo ficou bem, usar o *ffmpeg* para ver informações do ficheiro e o *ffplay* para reproduzir:
   xubunxore ~$ **`ffmpeg -i video1.mp4`**
   xubunxore ~$ **`ffplay video1.mp4`**
   (usar o **Ctrl+C** para parar)
7. De seguida grave uma segunda versão *video2.mp4* com mais *frames* por segundo e tamanho maior:
   core@xubunxore ~$ **`ffmpeg -f x11grab -video_size 200x200 -framerate 30 -i :0.0+200,300 -pix_fmt yuv420p video2.mp4`**

8. Adicionalmente pode matar a aplicação X11, que ainda está em execução em background, com um **kill %1**

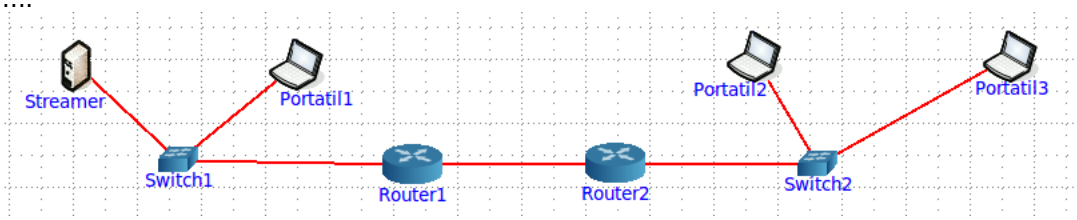
Os vídeos gerados, **video1.mp4** e **video2.mp4**, vão ser usados nas várias etapas do trabalho. Sem prejuízo de outras opções que queiram considerar. O objetivo não é fazer simples *copy paste* dos comandos, mas sim usar o enunciado como ponto de partida para a exploração dos conceitos e das ferramentas.

Etapa 1. Streaming HTTP simples sem adaptação dinâmica de débito

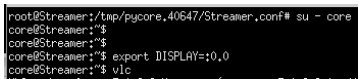
Nesta etapa, pretende-se testar o *streaming* sobre HTTP, uma opção pouco eficiente, mas muito popular na Internet. Sobretudo porque o protocolo HTTP está omnipresente e é "o" protocolo de aplicação dos dias de hoje.

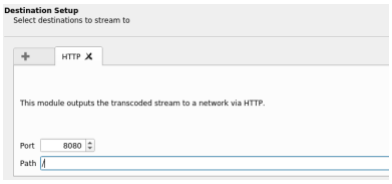
Tarefas:

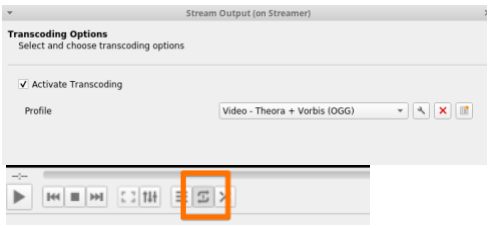
1. Construa uma topologia base no CORE, com pelo menos um servidor, 3 portáteis, 2 *switches* e um ou dois *routers*, mais ou menos como sugerido na figura.
   core@xubunxore ~$ **`sudo core-gui`**
   ....



2. Teste a conectividade da rede, com *ping*, *traceroute* ou outros comandos, para ter a certeza que funciona tudo bem.
3. Abra uma *bash* no servidor **Streamer** e, seguindo as indicações do ANEXO 2, coloque o VLC a fazer *streaming* por HTTP do ficheiro *video.mp4* com *transcoding* para "Video – Theora + Vorbis (Ogg)". VLC → **Media** → **Stream** → ...

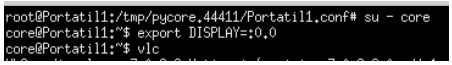


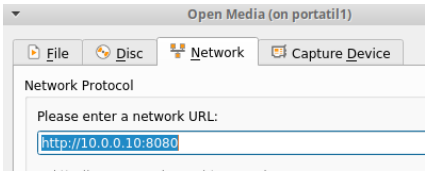




Será necessário garantir que o *streaming* é servido em ciclo, sem parar:

4. Abra uma nova *bash* no **Portátil1,** configure o DISPLAY e coloque um segundo VLC a funcionar agora como cliente: **Media → Open Network Stream →** http://<endereço-IP>:8080/





5. Recolha uma amostra de tráfego com o *Wireshark* na interface de saída do servidor.

6. Usando um editor de texto qualquer, construa uma página HTML, **video.html**, com um conteúdo simples usando a TAG **<vídeo>** conforme ilustrado abaixo. Nem todos os browsers suportam a tag. Nem todos os formatos são suportados. MP4 é o recomendado. Esta operação não necessita ser feita dentro CORE, pode estar previamente preparada na máquina nativa e copiada para a $HOME do utilizador core, pois o sistema de ficheiros é partilhado entre todos os nós.

core@xubunxore ~$ **cd**
core@xubunxore ~$ **pwd**
/home/core
core@xubunxore ~$ **vim video.html**
(.... se e só se conhece bem o vim senão escolha outro editor qualquer ☺ ESC:q!)

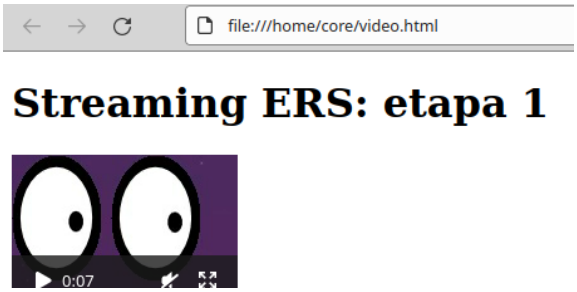
```
<!DOCTYPE html>
<html>
        <head>
                <title> Streaming ERS </title>
        </head>
        <body>
                <h1> Streaming ERS: etapa 1 </h1>
                <video controls autoplay>
                        <source src="http://10.0.0.10:8080">
                        A tag VIDEO não é suportada
                </video>
        </body>
</html>
```

```
<!DOCTYPE html>
<html>
    <head>
        <title> Streaming ERS </title>
    </head>
    <body>
        <h1> Streaming ERS: etapa 1 </h1>
        <video controls autoplay>
            <source src="http://10.0.0.10:8080">
            A tag VIDEO não é suportada
        </video>
    </body>
</html>
```

7. No **Portátil2**, abra uma *bash*, configure o DISPLAY, e execute o *firefox*. Abra a página *video.html* criada e verifique se o vídeo aparece corretamente. Ajuste o HTML se necessário.

```
root@Portatil2:/tmp/pycore.44411/Portatil2.conf# su - core
core@Portatil2:~$ export DISPLAY=:0.0
core@Portatil2:~$ cat video.html
<!DOCTYPE html>
<html>
        <head>
                <title> Streaming ERS </title>
        </head>
        <body>
                <h1> Streaming ERS: etapa 1 </h1>
                <video controls autoplay>
                        <source src="http://10.0.0.10:8080">
                        A tag VIDEO não é suportada
                </video>
        </body>
</html>

core@Portatil2:~$ firefox video.html
```



8. Recolha mais uma amostra de tráfego
9. No terceiro portátil, abra uma *bash*, configure o DISPLAY e use o comando *ffplay* http://<*endereço-IP*>:8080/ como terceiro cliente da stream de video.

Questão 1: Capture três pequenas amostras de trágefo no link de saída do servidor, respetivamente com 1 cliente (VLC), com 2 clientes (VLC e Firefox) e com 3 clientes (VLC, Firefox e ffmeg). Identifique a taxa em *bps* necessária (usando o ffmpeg -i video1.mp4 e/ou o próprio wireshark), o encapsulamento usado e o número total de fluxos gerados. Comente a escalabilidade da solução. Ilustre com evidências da realização prática do exercício (ex: capturas de ecrã)

Etapa 2. Streaming adaptativo sobre HTTP (MPEG-DASH)

Nesta etapa pretende-se criar em primeiro lugar um formato MPEG-DASH usando o **ffmpeg**. O objetivo é pegar no **video2.mp4** como entrada e gerar pelo menos três variantes do video, com maior ou menor resolução, para que possa ser servido em *streaming* com débito adaptativo. No final é necessário produzir um ficheiro XML com a descrição do conteúdo (MPD).

Tarefas:

1. Produzir a primeira versão do **video2.mp4**, recodificado em **VP9**, formato **webm**:

xubunxore ~$ **ffmpeg -i video2.mp4 -s 180x120 -c:v libx264 -b:v 200k -bf 2 -g 50 -sc_threshold 0 -an -dash 1 -pix_fmt yuv420p video2_180_120_200k.mp4**

xubunxore ~$ **ffprobe video2_180_120_200k.mp4**
xubunxore ~$ **ffplay video2_180_120_200k.mp4**

2. Produzir a segunda versão do **video2.mp4**, da mesma forma, mudando apenas a dimensão:

xubunxore ~$ **ffmpeg -i video2.mp4 -s 360x240 -c:v libx264 -b:v 500k -bf 2 -g 50 -sc_threshold 0 -an -dash 1 -pix_fmt yuv420p video2_360_240_500k.mp4**

xubunxore ~$ **ffprobe video2_360_240_500k.mp4**
xubunxore ~$ **ffplay video2_360_240_500k.mp4**

3. Produzir a terceira versão do **video2.mp4**, da mesma forma, mudando apenas a dimensão:

xubunxore ~$ **ffmpeg -i video2.mp4 -s 540x360 -c:v libx264 -b:v 1000k -bf 2 -g 50 -sc_threshold 0 -an -dash 1 -pix_fmt yuv420p video2_540_360_1000k.mp4**

xubunxore ~$ **ffprobe video2_540_360_1000k.mp4**
xubunxore ~$ **ffplay video2_540_360_1000k.mp4**

4. Produzir o ficheiro MPD com a descrição das alternativas, usando o MP4Box:

(...Instalar o software GPAC com ***sudo apt install gpac*** *se necessário...)*

xubunxore ~$ **MP4Box -dash 500 -out video_manifest video2_180_120_200k.mp4 video2_360_240_500k.mp4 video2_540_360_1000k.mp4**

5. Preparar uma página HTML5 **video_dash.html** para visualizar o vídeo. Alguns browsers, como por exemplo o Firefox, já suportaram nativamente o DASH, mas agora já não suportam. A solução de ter uma extensão é atualmente a recomendada uma vez que pode ser implementada e mantida pela comunidade responsável, sendo tecnicamente mais fiável e flexível. As próprias normas DASH são testadas com o **Dash.js** (https://dashif.org). Dado que dentro do CORE não vai haver conectividade para a Internet, e não conseguimos chegar a uma CDN para descarregar nada, o objetivo é antes de mais obter as scripts necessárias, usando o comando **wget**:

wget http://cdn.dashjs.org/latest/dash.all.min.js
wget http://cdn.dashjs.org/latest/dash.all.debug.js

```
core@xubuncore:~$ wget http://cdn.dashjs.org/latest/dash.all.min.js
--2021-10-24 17:24:49--  http://cdn.dashjs.org/latest/dash.all.min.js
Resolving cdn.dashjs.org (cdn.dashjs.org)... 95.95.253.105, 95.95.253.83, 2a01:8:8000:20e::5f5f:fd69, ...
Connecting to cdn.dashjs.org (cdn.dashjs.org)|95.95.253.105|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 634509 (620K) [application/x-javascript]
Saving to: 'dash.all.min.js'

dash.all.min.js                 100%[=====================================================>] 619,64K  3,92MB/s    in 0,2s

2021-10-24 17:24:50 (3,92 MB/s) - 'dash.all.min.js' saved [634509/634509]

core@xubuncore:~$
core@xubuncore:~$ wget http://cdn.dashjs.org/latest/dash.all.debug.js
--2021-10-24 17:24:50--  http://cdn.dashjs.org/latest/dash.all.debug.js
Resolving cdn.dashjs.org (cdn.dashjs.org)... 95.95.253.105, 95.95.253.83, 2a01:8:8000:20e::5f5f:fd69, ...
Connecting to cdn.dashjs.org (cdn.dashjs.org)|95.95.253.105|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2592349 (2,5M) [application/x-javascript]
Saving to: 'dash.all.debug.js'

dash.all.debug.js               100%[=====================================================>]   2,47M  4,02MB/s    in 0,6s

2021-10-24 17:24:50 (4,02 MB/s) - 'dash.all.debug.js' saved [2592349/2592349]
```

6. A página HTML deverá pois simplesmente incluir referências aos dois módulos JavaScript e referenciar na tag *video* o ficheiro **video_manifest.mpd**:

core@xubunxore ~$ **cd**
core@xubunxore ~$ **pwd**
/home/core
core@xubunxore ~$ **vim video_dash.html**
(.... se e só se conhece bem o vim senão escolha outro editor qualquer ☺ ESC:q!)

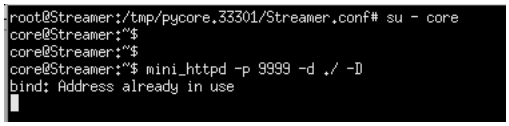
```
<!DOCTYPE html>
<html>
        <head>
                <title> Streaming ERS </title>
                <script src="dash.all.debug.js"></script>
        </head>
        <body>
                <h1> Streaming ERS: etapa 2 DASH </h1>
                <video data-dashjs-player autoplay src="video_manifest.mpd" controls="true">
                </video>
        </body>
</html>
```

```
<!DOCTYPE html>
<html>
    <head>
        <title> Streaming ERS </title>
        <script src="dash.all.debug.js"></script>
    </head>
    <body>
        <h1> Streaming ERS: etapa 2 DASH </h1>
        <video data-dashjs-player autoplay src="video_manifest.mpd"
controls="true">
        </video>
    </body>
</html>
```

7. Agora que está tudo preparado, podemos emular a topologia da Etapa 1 no CORE e testar a conectividade a ver se está tudo a funcionar bem.

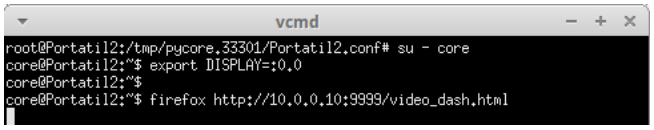
8. Abrir uma *bash* no servidor **Streamer** e servir o conteúdo com um servidor HTTP, neste caso o **mini_httpd**:

   `mini_httpd -p 9999 -d ./ -D`
   (nota: o mini_httpd fica à escuta na porta 9999 e serve a pasta local ./ que é a home onde estão todos os ficheiros)



9. Visualizar com o *firefox* no Portátil1

10. Visualizar com o firefox no Portátil2



    Se algo estiver a correr mal, podemos abrir a consola *JavaScript* do Firefox (More Tools → Web Devel. Tools) e recarregar a página para ver os erros gerados.

11. Capturar amostras de tráfego com Wireshark

12. Mexer na capacidade dos links de modo a forçar o Portátil 2 a mostrar o vídeo de menor dimensão.

Questão 2: Diga qual a largura de banda necessária, em bits por segundo, para que o cliente de *streaming* consiga receber o vídeo no *firefox* e qual a pilha protocolar usada neste cenário.
Questão 3: Ajuste o débito dos links da topologia de modo que o cliente no portátil 2 exiba o vídeo de menor resolução e o cliente no portátil 1 exiba o vídeo com mais resolução. Mostre evidências.
Questão 4: Descreva o funcionamento do DASH neste caso concreto, referindo o papel do ficheiro MPD criado.

Etapa 3. Streaming RTP/RTCP unicast sobre UDP e multicast com anúnicos SAP

Nesta última etapa o *streaming* será sempre feito sobre UDP, quer em *unicast*, quer em *multicast*. No primeiro caso, o cliente recebe dados sobre a sessão de *streaming* num ficheiro de descrição em formato SDP (*Session Description Protocol*). O servidor ao iniciar a *stream*, gera o ficheiro, que deveria ser depois fornecido ao cliente, mas que neste caso não é necessário, pois o sistema de ficheiros é partilhado. O cliente com base nos dados contidos no ficheiro, inicia a receção em UDP.

Já no caso multicast o modo de funcionamento é distinto. Também é gerada uma descrição da sessão, mas ela é enviada por SAP (*Session Annonucement Protocol*) para um grupo *multicast* (endereço de grupo) especial para o efeito que é o **224.2.127.254** (**sap.mcast.net**) para destinos IPv4 ou **ff0e::2:7ffe** para destinos IPv6. Tudo o que o cliente tem de fazer é estar à escuta nesse grupo e ouvir o anúncio com os dados SDP para poder iniciar a sessão como cliente. No caso *multicast*, vamos escolher também um endereço de envio especial, que é um endereço de grupo. Neste exemplo foi escolhido o **224.0.0.200** porta **5555**.

Tarefas:

1. Inicie a emulação CORE da topologia criada na etapa 1 e teste a conectividade.

2. No servidor **Streamer**, inicie uma sessão de streaming com RTP com o *ffmpeg*.
   `ffmpeg -stream_loop -1 -re -i video1.mp4 -f rtp -sdp_file video.sdp rtp://<endereço IP do Portatil 3>:5555`
   (a flag -re simula o envio em tempo real, a stream fica em loop permanente, é gerado um ficheiro com a descrição SDP da sessão: video.sdp)

```
root@Streamer:/tmp/pycore.46167/Streamer.conf# su - core
core@Streamer:~$
core@Streamer:~$ ffmpeg -stream_loop -1 -re -i video1.mp4 \
> -f rtp -sdp_file video.sdp rtp://10.0.2.21:5555
```

3. No cliente **Portátil3**, inicie um cliente *ffplay*:

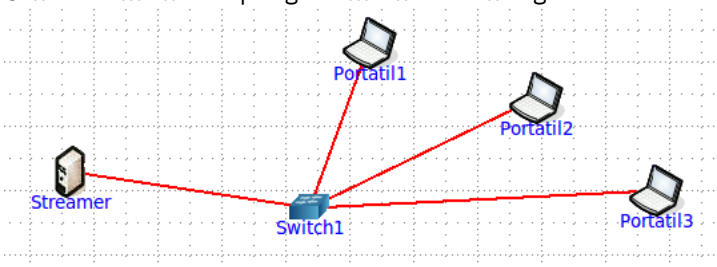
   `ffplay -protocol_whitelist file,rtp,udp  video.sdp`
   (tudo o que o cliente necessita para se ligar está guardado no ficheiro video.sdp, e o ffplay inicie o play do video)

```
root@Portatil3:/tmp/pycore.46167/Portatil3.conf# su - core
core@Portatil3:~$ export DISPLAY=:0.0
core@Portatil3:~$ cat video.sdp
SDP:
v=0
o=- 0 0 IN IP4 127.0.0.1
s=No Name
c=IN IP4 10.0.2.21
t=0 0
a=tool:libavformat 58.29.100
m=video 5555 RTP/AVP 96
b=AS:200
a=rtpmap:96 MP4V-ES/90000
a=fmtp:96 profile-level-id=1
core@Portatil3:~$ ffplay -protocol_whitelist file,udp,rtp video.sdp
```

4. Capture o tráfego com o Wireshark no link de saída do servidor.

5. Termine a emulação no CORE.

6. Para o exercício *multicast*, vamos usar apenas um *switch* com o servidor e os 3 portáteis ligados ao mesmo switch. Construa uma nova topologia como indicado na figura:



7. Inicie a emulação CORE e teste a conectividade entre os sistemas.

8. No servidor **Streamer**, inicie uma sessão de streaming multicast com o ffmpeg:

   `ffmpeg -stream_loop -1 -re -i video1.mp4 -f sap sap://224.0.0.100:5555`

```
vcmd
root@Streamer:/tmp/pycore.38749/Streamer.conf# su - core
core@Streamer:~$
core@Streamer:~$ ffmpeg -stream_loop -1 -re -i video1.mp4 -f sap sap://224.0.0.100:5555 []
```

9. Em cada um dos 3 portáteis (**Portátil1**, **Portátil2** e **Portátil3**) inicie um cliente da sessão diferente:

   `ffplay sap://`

```
vcmd
root@Portatil2:/tmp/pycore.38749/Portatil2.conf# su - core
core@Portatil2:~$ export DISPLAY=:0.0
core@Portatil2:~$
core@Portatil2:~$ ffplay sap://
```

(repetir nos 3 portáteis)

10. Capture o tráfego no link de saída do servidor com o Wireshark

Questão 5: Compare o cenário *unicast* aplicado com o cenário *multicast*. Mostre vantagens e desvantagens na solução *multicast* ao nível da rede, no que diz respeito a escalabilidade (aumento do nº de clientes) e tráfego na rede. Tire as suas conclusões.

Relatório

O relatório final do TP2 deve incluir:

- Uma secção de "Questões e Respostas" que dê resposta adequada às questões enumeradas no enunciado, incluindo para cada questão: a questão, a resposta e a prova da realização da mesma (se aplicável);
- Uma secção de "Conclusões" que autoavalie os resultados da aprendizagem decorrentes das várias vertentes estudadas no trabalho.

Submissão

O relatório em formato livre deve ser submetido na plataforma de ensino https://elearning.uminho.pt, usando a funcionalidade de transferência de ficheiros do grupo, com o nome ESR-TP2-PL<*Turno*>-G<*Grupo*>.pdf (por exemplo, ESR-TP2-PL1-G1.pdf para o grupo 1 do PL1) no final do dia da aula prevista para a conclusão do trabalho.

ANEXO 1. Execução de aplicações Gráficas: X11 no CORE

É relativamente fácil executar aplicações sem interface gráfico no CORE. Basta abrir um *shell* no nó respetivo, quando a emulação está ativa, como ilustrado na Figura 1.

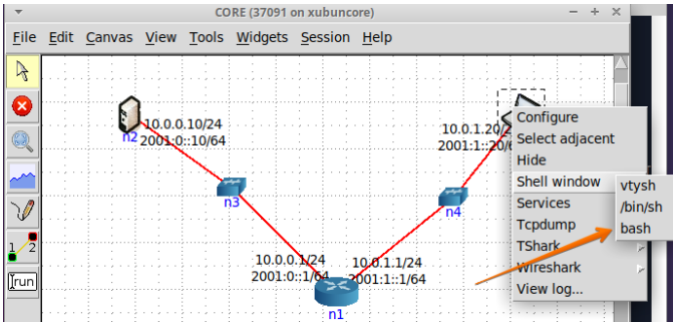
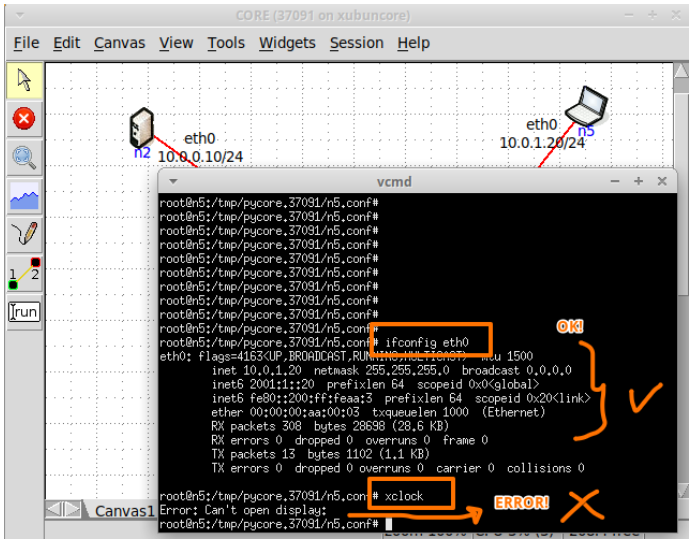


Figura 1: execução de aplicações GUI no CORE



Correr comandos como o "*ifconfig eth0*" não implica nenhuma configuração adicional, porque a saída produzida é apenas texto normal. Mas quando tentamos executar uma aplicação como o "*xclock*", que tem uma interface gráfica, origina um erro "*Can't open display*". Para compreender o erro, é preciso perceber que as interfaces gráficas em Linux são baseadas no X11 (X Window System, ou simplesmente X). O X é um protocolo que o usa o modelo cliente servidor para permitir que uma aplicação em execução num computador utilize um display remoto noutro computador. Em termos muito simplistas, o servidor X gere os recursos gráficos de um display, e aceita pedidos dos clientes remotos X para desenhar janelas e aceitar inputs dos utilizadores. As aplicações que usam interfaces gráficas são os clientes X.
No caso ilustrado na figura, o terminal é um processo em execução no nó n5 e o servidor X está em execução na máquina virtual que corre também o CORE. Para que a aplicação gráfica funcione são precisas duas ações:

1. Indicar ao cliente qual o display a usar (neste caso o display 0.0, na máquina local):
   .../n5.conf# **`export DISPLAY=:0.0`**

2. Dizer ao servidor no *xubuncore* que deve autorizar clientes remotos (qualquer um, para facilitar, embora isso traga problemas de segurança graves!):

   core@xubunxore ~$ **`xhost +`**

Estas duas ações permitem executar o *xclock* no nó n5 e mostrar um relógio gráfico no *xubuncore*, conforme ilustrado na Figura 2.

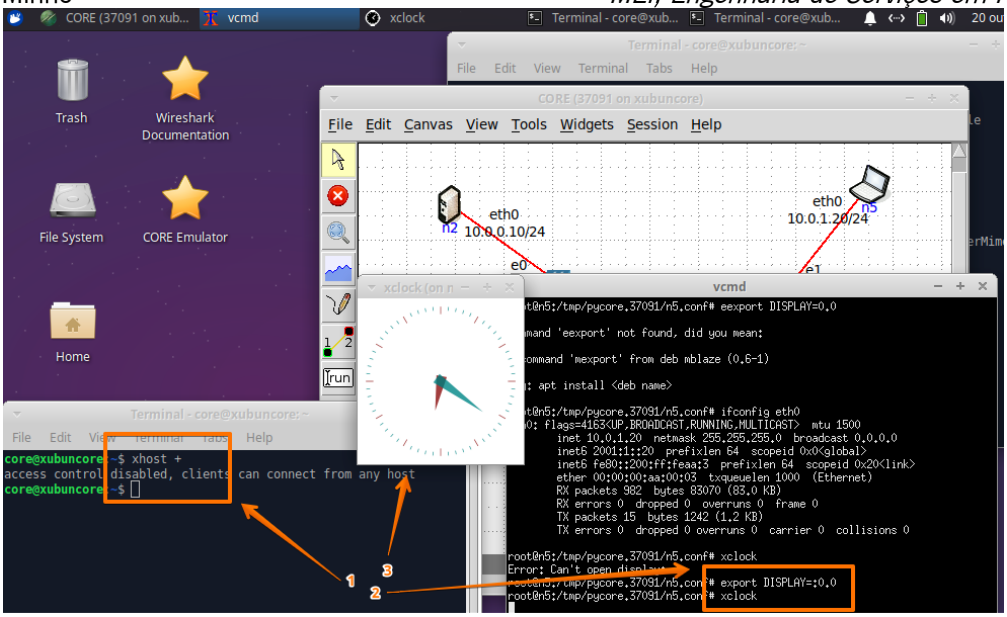


Figura 2: Controlo de acesso X11

Alternativamente, pode encapsular o tráfego do protocolo X11 nas conexões SSH e a configuração ocorre de forma mais simples e segura. Essa solução é mais difícil de aplicar no CORE, mas deve ser a preferida em todos os cenários reais. Num cenário real nunca usar **xhost +** em nenhuma circunstância ☺. Ver links abaixo para mais informação.

Links:

- https://docstore.mik.ua/orelly/networking_2ndEd/ssh/ch09_03.htm

ANEXO 2. Streaming no CORE usando o VLC

O VLC é uma aplicação versátil de multimédia que além de ser um bom player, permite também fazer streaming. No XubunCORE, ode ser instalado com os comandos:

- `sudo apt install vlc`
- `sudo apt install vlc-plugin-access-extra`

Além de ser uma aplicação com interface gráfico (GUI), que exige as configurações referidas no ANEXO 1, acresce que não pode ser executada como *root* ☹ Logo, para funcionar bem no CORE, onde precisamos de ser *root* para capturar os pacotes dos interfaces em modo promíscuo usando o Wireshark, é preciso i) mudar de utilizador; ii) definir o display; e iii) executar o VLC:

I. *.../n5.conf#* **`su - core`**
II. *core@n5$* **`export DISPLAY=:0.0`**
III. *core@n5$* **`vlc`**

O processo está ilustrado na Figura 3.

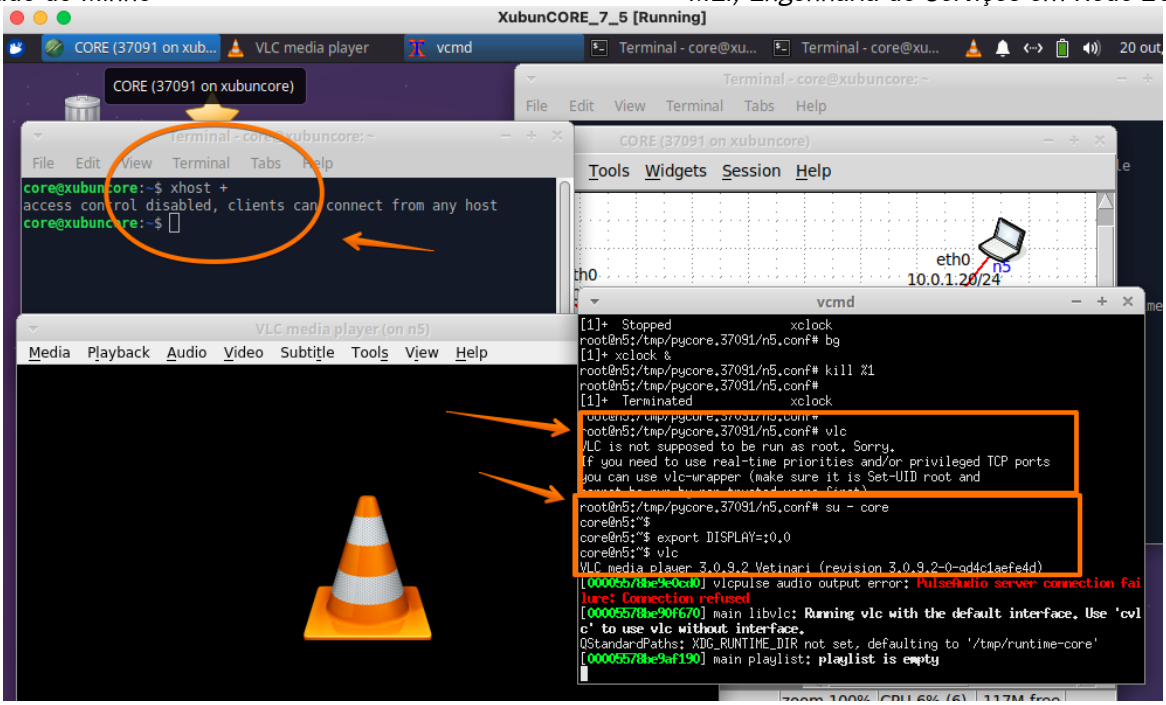
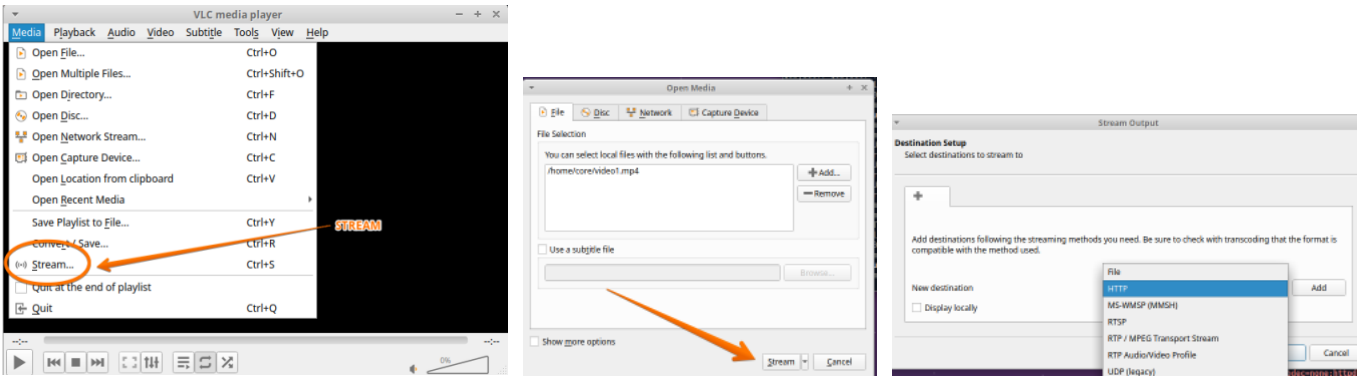


Figura 3: Execução do VLC num nó de uma topologia emulada

O VLC dispõe de um menu de ajuda (uma espécie de *Wizard*) para iniciar o modo de streaming.

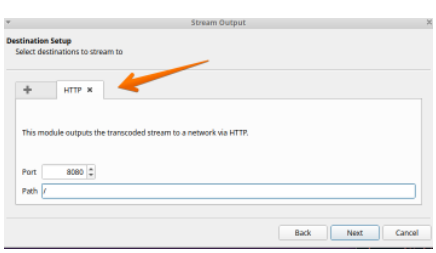


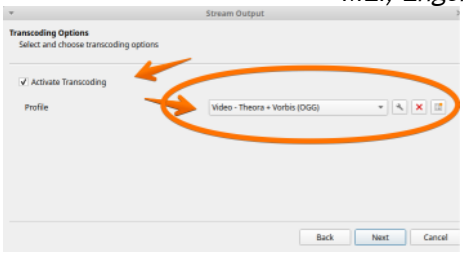
O *wizard* permite em primeiro lugar escolher a fonte, que tanto pode ser captura de um dispositivo, gravação do ecrã, ficheiros multimédia, etc. Escolhida a fonte (input) é necessário definir a saída (output). Para streaming o VLC disponibiliza um conjunto de alternativas péconfiguradas: *display locally* (que mostra a stream no monitor apenas), *File* (que grava a stream num ficheiro), *HTTP* (que disponibiliza a stream em HTTP e carece de um endereço IP e de uma porta), *MS-WMSP* (stream para Microsoft Windows Media Player), *UDP* (stream em unicast ou em multicast, apenas funciona com encapsulamento TS), *RTP* (que usa o protocolo RTP sobre UDP, também em unicast ou multicast) e *IceCast* (stream para um servidor IceCast).

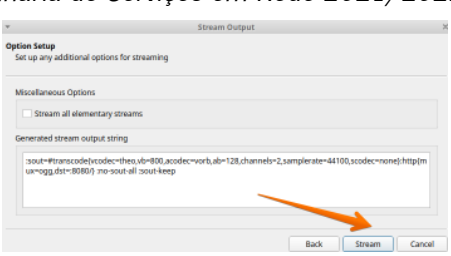
Não há problemas com o streaming se for de VLC para VLC, ou de VLC para *FFmpeg*. Mas para que o streaming funcione com a etiqueta html *<video>* do HTML5, é necessário forçar o *transcoding* do video com o codec "Theora", que é gratuito e aberto, empacotando-o de seguido num contentor do tipo "ogg". Este método consome bastante CPU mas garante a compatibilidade dos principais browsers. O resultado final será uma linha deste género, que pode ser executada na linha de comando:

```
:sout=#transcode{vcodec=theo,vb=800,scale=Auto,acodec=none,scodec=none}:http{mux=ogg,dst=:8080/} :no-sout-all :sout-keep
```

Que pode ser conseguida no interface gráfico da seguinte forma, no menu de transcoding, simplesmente escolhendo como destino o HTTP e como transcoding o perfil "" como ilustrado abaixo:







Links:

- https://wiki.videolan.org/Documentation:Documentation
- https://wiki.videolan.org/Documentation:Streaming_HowTo_New/

ANEXO 3. Formatos de dados para *Streaming*

Nesta UC o foco não são os múltiplos formatos de áudio, vídeo e de empacotamento de áudio e vídeo que existem. E que são mais que muitos. Há que distinguir em primeiro lugar *codecs* áudio e video de formatos de empacotamento. Os Vamos apenas olhar um pouco para o MP4 e o MPD/DASH, que são formatos de empacotamento de vídeo/áudio (definem formatos para meta-dados e os segmentos em que os dados têm de ser divididos para serem enviados) e não formatos de representação, codificação e compressão de vídeo/áudio. Ou seja, os *codecs* (módulos de software que implementam os descodificadores) têm de suportar esses formatos de representação, codificação e compressão mas não os formatos de empacotamento. Os módulos de interpretação e desempacotamento de MP4 ou MPD têm apenas que ser entendidos pelas aplicações clientes para obterem os dados e depois utilizarão os codecs necessários para reconstruir a stream de vídeo para visualização.

O MPEG-4 (MP4) é a versão mais recente do formato MPEG, um formato de empacotamento (contentor) usado no Web e com suporte bastante alargado.

O MPEG-DASH é uma técnica normalizada de streaming com bitrate adaptativo para uso sobre protocolo HTTP. Tem bastantes semelhanças com o protocolo proprietário da Apple designado por HLS (*HTTP Live Streaming*). Os servidores HTTP (*Web*) convencionais podem usar esse processo para servir conteúdos multimédia na Internet. Para tal faz-se uso de um ficheiro com a descrição dos conteúdos em formato XML, que se designa por MPD (*Media Presentation Description*). Esta informação é útil para que o browser saiba todas as informações sobre as várias streams e a largura de banda associada a cada uma delas, bem como outros metadados como o tipo MIME e os codecs a usar. Na página HTML o browser encontra uma referência a este ficheiro MPD e não aos ficheiros multimédia como acontece no caso de não se usarem técnicas adaptativas. Há dois tipos de perfil MPD, um para VOD (*Video on demand*) e outro para *streaming* ao vivo (LIVE).

Além do FFmpeg, existem outras ferramentas capazes de empacotar e criar a descrição MPD:

- **edash-packager** https://github.com/google/edash-packager/
- **MP4Box** http://gpac.wp.mines-telecom.fr/mp4box/
- **DASHEncoder** https://github.com/slederer/DASHEncoder

Links úteis:

- https://developer.mozilla.org/en-US/docs/Web/Media/Formats
- https://developer.mozilla.org/en-US/docs/Web/Media/Formats/Containers
- https://developer.mozilla.org/en-US/docs/Web/Media/DASH_Adaptive_Streaming_for_HTML_5_Video
- https://gpac.wp.imt.fr/
- https://github.com/slederer/DASHEncoder

ANEXO 4. *Streaming* com *FFmpeg*

O **FFmpeg** é uma das mais completas *frameworks* multimédia, "a" Framework, um autêntico canivete suíço capaz de codificar, descodificar, transcoding, mud, demux, streaming, reprodução, filtragem, etc, etc. A quantidade de formatos de media suportados é muita vasta, desde os mais comuns aos mais obscuros. Outra enorme vantagem desta plataforma é que é multiplataforma, correndo em todos os sistemas operativos. É ainda usado por outras ferramentas e programas de uso mais amigável.

Para instalar no Xubuntu, basta usar o gestor de software apt:

- `sudo apt install ffmpeg`

São muitas as opções de parâmetros para o ffmpeg, que exige uma consulta à secção 5 da página https://ffmpeg.org/ffmpeg.html
Podemos por exemplo verificar a lista de formatos suportados, os codecs, os dispositivos e os protocolos:

core@xubunxore ~$ **ffmpeg -formats**
core@xubunxore ~$ **ffmpeg -codecs**
core@xubunxore ~$ **ffmpeg -devices**
core@xubunxore ~$ **ffmpeg -protocols**

A flag **-re** faz com que o input seja lido com a *frame* rate nativa (*read input at native frame rate*). A ideia é evitar que o output seja enviado à velocidade máxima possível e que a leitura se faça como se fosse de um dispositivo real.

A estrutura base dos parâmetros é

ffmpeg -i input_file […parameter list…] output_file

Input_file e *output_file* podem ser ficheiros, mas também podem ser objetos referenciados pelos protocolos que a ferramenta suporta, que são muitos, por exemplo: **file, http, pipe, rtp/rtsp, raw udp, rtmp**. A ferramenta pode ser usada para conversão de formatos, filtragem, junção e separação de streams, etc. São literalmente centenas de opções disponíveis.

Links úteis:
- https://ffmpeg.org/
- https://ffmpeg.org/ffmpeg.html
- https://ffmpeg.org/documentation.html
- https://trac.ffmpeg.org/wiki/StreamingGuide

ANEXO 5. *Streaming* com *OBS*

Outra ferramenta "Open Source" que permite fazer streaming de forma muito simples é o OBS (Open Broadcaster Software). No entanto, esta ferramenta revelou-se demasiado pesada em termos de consumos de recursos (RAM e CPU) para poder ser usada convenientemente na máquina virtual XubunCORE_7_5. Está preparada para fazer streaming com meia dúzia de cliques para os principais serviços disponíveis na Internet, mas qualquer uso mais personalizado acaba por recair no uso externo do FFmpeg. Daí que seja preferível usarmos no trabalho diretamente o FFmpeeg. Fica aqui apenas uma breve referência de como pode ser usado em ambiente de testes adequado.

O interface gráfico parece complexo, mas mal se começa a usar percebe-se que até está bem organizado e facilita o uso. Na parte inferior da janela principal, tem 5 itens: *Scenes, Sources, Audio Mixer, Scene Tansitions e Controls*. O primeiro (*Scenes*) permite definir um conjunto de cenas que se podem usar em sequência com uma transição entre elas. Por exemplo uma tela inicial com o título da sessão é uma cena, depois a transmissão ao vivo é uma segunda cena. Para cada cena podemos depois definir a fonte (*Sources*), que pode ser a entrada de uma camara de vídeo, de um microfone, uma gravação de uma janela ou área do ecrã, ou simples textos, como ilustrado na figura. Depois pode-se misturar áudio no *Audio Mixer*, quer do micro quer do desktop, definir uma transição entre cenas (se aplicacável) no *Scene Transition* e finalmente iniciar o streaming. Antes de fazer "Start Streaming" convém ir às definições e escolher o serviço de streaming desejado. Aparece uma lista de serviços predefinidos à escolha.
O OBS socorre-se das bibliotecas de funções do FFmpeg. Privilegia protocolos como o SRT sobre UDP ou RTMP sobre TCP. Nos settings pode-se definir manualmente usando por exemplo o url: rtmp://[endereço_ip]:[porta]/[path]

Links:

- https://obsproject.com/
- https://obsproject.com/docs/