

Reverse Proxy

Relatório do trabalho prático

Grupo 35

Ana Paula Carvalho (A61855)

João Pires Barreira (A73831)

Simão Barbosa (AXXXXX)

Março 2018



Universidade do Minho
Escola de Engenharia

Comunicações por Computador
Mestrado Integrado em Engenharia Informática
Universidade do Minho

1 Introdução

Este projeto surge no âmbito da Unidade Curricular de Comunicações por Computador e tem como objetivo a implementação de um proxy reverso com balanceamento de carga.

O servidor *proxy* deverá guardar informação acerca dos servidores de *back-end* disponíveis e, sempre que chegar um pedido TCP vindo de um cliente, deve escolher o servidor mais apropriado para essa ligação (i.e. com menos carga a nível de RTT, CPU e RAM disponíveis, etc.).

2 Arquitetura da solução

A arquitetura da solução implementada encontra-se descrita na figura apresentada abaixo:

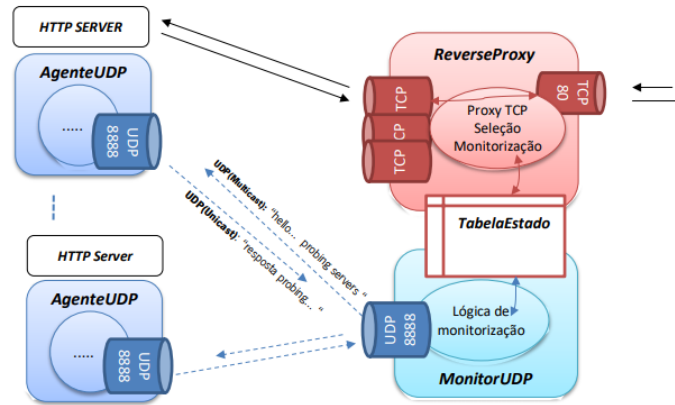


Figura 1: Arquitetura da solução implementada

O servidor proxy guarda informação relativa

Resumidamente, são enviados pedidos HTTP para a porta 80 do servidor de *proxy* que mantém informação acerca do estado dos servidores de *back-end* numa tabela de decisão. Ao receber cada um destes pedidos via *TCP*, o servidor *proxy* escolhe o melhor servidor de *back-end* disponível para o atender. Depois, o *proxy* atua como um mero intermediário, encaminhando a informação recebida do cliente para o servidor e vice-versa.

Paralelamente a isto, a comunicação entre servidor *proxy* e servidores de *back-end* é feita através de *UDP* através dos seguintes componentes:

- **MonitorUDP**: Sendo responsável pela descoberta de servidores, de t em t segundos envia uma mensagem de probing por UDP (encriptada usando *SHA-256*), dirigida ao grupo Multicast 239.8.8.8 para a porta 8888; depois, escuta as respostas UDP unicast na mesma porta, verificando a integridade

de todas as mensagens recebidas (encriptadas usando *SHA-256*) antes de atualizar a tabela de estado em conformidade

- **AgenteUDP**: Existindo um agente por cada servidor de *back-end*, estes ficam à escuta de pedidos de probing (encriptados) na porta 8888 enviados pelo MonitorUDP por multicast para o grupo com endereço IPv4 239.8.8.8 (juntando-se previamente a este grupo). Por cada mensagem recebida, o AgenteUDP deve responder com uma mensagem encriptada UDP dirigida em unicast ao MonitorUDP, incluindo na resposta a carga da máquina (RAM e CPU). As respostas são artificialmente atrasadas, por um pequeno intervalo de tempo aleatório entre 0 e 10 ms, de modo a não serem enviadas todas ao mesmo tempo.

3 Especificação do protocolo UDP

O protocolo desenvolvido para a comunicação entre o MonitorUDP (do *proxy*) e os AgentesUDP (dos servidores de *back-end*), foi implementado, a nível aplicacional, com recurso ao UDP.

3.1 Formato das mensagens protocolares

3.1.1 Formato da mensagem de probing (ProbeRequest)

- **timestamp** – campo do tipo *long* que serve para denotar a marca temporal relativa ao momento em que foi enviado a mensagem de ProbeRequest (será utilizado para cálculo do RTT).
- **hmac** – campo que possui o output gerado pela encriptação da mensagem de ProbeRequest com recurso ao SHA-256.

Assim sendo, cada mensagem de ProbeRequest tem um **tamanho fixo de 40 bytes**, correspondentes aos 8 bytes do timestamp e 32 bytes (256 bits) do HMAC gerado pelo SHA-256.

3.1.2 Formato da mensagem de resposta ao probing (ProbeResponse)

- **cpuUsage** – byte que corresponde à percentagem de CPU que está ser utilizada pelo servidor de *back-end*
- **freeRAM** – 4 bytes que correspondem à quantidade de RAM disponível no servidor de *back-end* (em megabytes)
- **timestamp** – campo do tipo *long* que serve para denotar a marca temporal relativa ao momento em que foi enviado a mensagem de ProbeRequest (será utilizado para cálculo do RTT).

- **hmac** – campo que possui o output gerado pela encriptação da mensagem de ProbeRequest com recurso ao SHA-256.

Assim sendo, cada mensagem de ProbeResponse tem um **tamanho fixo de 45 bytes**, correspondentes aos 4 bytes de freeRam, 1 bytes de cpuUsage, 8 bytes do timestamp e 32 bytes (256 bits) do HMAC gerado pelo SHA-256.

3.2 Interações

As interações entre o MonitorUDP e os AgentesUDP encontram-se explicadas nos dois pontos da parte inferior da secção **Arquitetura da solução**.

No entanto, é importante explicitar melhor a encriptação das mensagens enviadas e recebidas. Para isto, foi utilizado o algoritmo de hash chamado *SHA-256* por ser um algoritmo seguro e robusto. Desta forma, tanto o MonitorUDP como os AgentesUDP possuem uma chave privada partilhada entre ambos. Antes de enviarem uma mensagem, pegam nos conteúdos dessa mensagem e, como eles, geram um byte array correspondente ao HMAC gerado pela chamada do algoritmo SHA-256 com a combinação desses mesmo conteúdos e da chave privada. Assim, o recetor ao receber a mensagem, vai fazer o mesmo analogamente, gerando um código que depois terá de ser comparado ao recebido. Como, ambos partilham a mesma chave privada e os conteúdos são os mesmo, irão obter o mesmo código HMAC, pelo que fica assegurada a autenticidade da origem visto que sem aquela chave privada não seria possível gerar o mesmo output, pela definição da função de hash.

4 Implementação

Para a implementação destes conceitos, foi utilizada a linguagem de programação orientada a objetivos *Java*, e definidas as seguintes classes:

- **HMAC** - Implementa as funcionalidade referidas na subsecção **Interações**.
- **ProbeRequest** - Implementa as funcionalidade referidas na subsecção **Formato da mensagem de probing**.
- **ProbeResponse** - Implementa as funcionalidade referidas na subsecção **Formato da mensagem de resposta ao probing**.
- **UDPMonitor** - Implementa as funcionalidade referidas na descrição deste componente na secção **Arquitetura da solução**. Para isso, foi criada uma função main que lança uma thread de probing que envia pedidos de probing encriptados através de Multicast UDP. Paralelamente, para cada resposta recebida em unicast, é lançada outra thread responsável por verificar a integridade da resposta e atualizar a tabela de estado. É feito *logging* do envio e receção de mensagens para o standard output.

- **UDPAgent** - Implementa as funcionalidade referidas na descrição deste componente na secção **Arquitetura da solução**. Para isso, foi criada uma função main que, em primeiro lugar, se junta ao grupo Multicast (UDP), ficando à espera de um pedido de probing enviado pelo MonitorUDP. Assim que recebe uma resposta, verifica a integridade da mensagem recebida e, caso esta seja verificada com sucesso, lança uma thread responsável por retirar as medidas necessárias do servidor *back-end* associado (CPU e RAM), encriptar a resposta e, de seguida, enviar a mesma por unicast UDP ao MonitorUDP, atrasando artificialmente este envio por um período aleatório entre 1 e 10 segundos. É feito *logging* do envio e receção de mensagens para o standard output.
- **ServeStatus** - Possui as informações relativas ao estado de um servidor de *back-end*, nomeadamente: o endereço IP, a porta, o timestamp do última resposta de probing recebida, a RAM disponível, o CPU a ser utilizado, o RTT atual, o somatório de todos os RTT até ao momento e o número de respostas recebidas. Estes dois últimos campos servem para calcular o RTT atual através da média.
- **StatusTable** - Tabela de estado que é atualizada pelo MonitorUDP do ReverseProxy e que contém a informação acerca dos estados (ServerStatus) de todos os servidores de *back-end*. A função de update, verifica se a informação recebida é mais recente do que a que já constava da tabela e, em caso afirmativo, calcula o RTT do percurso (através do timestamp da PDU e do tempo atual). É necessário fazer-se um controlo de concorrência ao nível desta tabela, por causa das diferentes threads que estão a correr (MonitorUDP) que podem despoletar updates simultaneos. A função chooseServer é a responsável por escolher o servidor com melhores condições atuais, através de uma heurística simples que atribui um peso definido pelo grupo para cada uma das medidas de estado do servidor.
- **ReverseProxy** - É a classe principal do programa. A sua função main inicia a estrutura de dados relativa à tabela de estado e coloca o MonitorUDP a correr. Paralelamente, cria os sockets para a comunicação com o cliente e com o servidor de *back-end* que será escolhido através da função chooseServer da StatusTable sempre que é recebido um pedido HTTP de um cliente. A comunicação entre servidor e cliente é reencaminhada pelo proxy de forma transparente e direta.

5 Testes e resultados

Para efeitos de teste foi utilizado o software *CORE* com a topologia fornecida durante as aulas que se encontra representada na imagem seguinte:

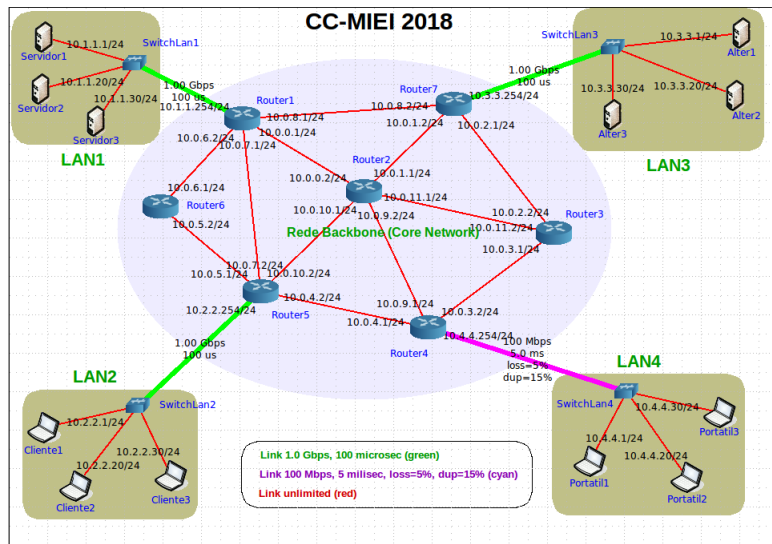


Figura 2: Topologia CORE utilizada

Desta forma, foi posto a correr no Servidor 1 o ReverseProxy; em cada um dos Servidores 2 e 3 foi posto a correr um servidor HTTP simples feito em Python 3 (SimpleHTTPServer) e um AgenteUDP. Esta situação está descrita na seguinte imagem:

```
root@Server1:/tmp/pscore.45913/Server1.conf
root@Server1:/tmp/pscore.45913/Server1.conf# java -cp "/cc-1718/src/ ReverseProxy
Sent: ProbeRequest(Timestamp: 1527787414788, HHC: 1V67L4zoz3dP1G0JVE6qaz0Qp2pJkDh9k=)
Received: ProbeResponse(CPU: 4%, RMT: 236 MB, Timestamp: 1527787414788, HHC: E01X5z1h73wHgKauNovyIdUkzL57FrUJ9ShoDys=)
StatusTable(
  IP: 10.1.1.1.30:60623, SentTimestamp: 1527787414788, FreeRMT: 296MB, CPU: 4%, RTT: 2844ns, :
)
Sent: ProbeRequest(Timestamp: 1527787420065, HHC: eMEY5a1D4uPhwq0Dy1NRN6TqR0dH/AqIwW20U=)
Received: ProbeResponse(CPU: 0%, RMT: 236 MB, Timestamp: 1527787420065, HHC: yF9Vz3SsQ1006hMUHLvC0HMMvUQ210kUd30VvU2qg=)
StatusTable(
  IP: 10.1.1.1.20:38958, SentTimestamp: 1527787420065, FreeRMT: 296MB, CPU: 0%, RTT: 1003ns, :
  IP: 10.1.1.1.30:60623, SentTimestamp: 1527787414788, FreeRMT: 296MB, CPU: 4%, RTT: 2844ns, :
)
Received: ProbeResponse(CPU: 4%, RMT: 236 MB, Timestamp: 1527787414788, HHC: E01X5z1h73wHgKauNovyIdUkzL57FrUJ9ShoDys=)
StatusTable(
  IP: 10.1.1.1.20:38958, SentTimestamp: 1527787420065, FreeRMT: 296MB, CPU: 0%, RTT: 5422ns, :
  IP: 10.1.1.1.30:60623, SentTimestamp: 1527787414788, FreeRMT: 296MB, CPU: 4%, RTT: 2844ns, :
)
[no update on server load]
Sent: ProbeRequest(Timestamp: 1527787425066, HHC: oKSe2u6CNo0t3M5U82BC0M3UK5cIugz21v69ot4=)
Received: ProbeResponse(CPU: 0%, RMT: 236 MB, Timestamp: 1527787420065, HHC: yF9Vz3SsQ1006hMUHLvC0HMMvUQ210kUd30VvU2qg=)
StatusTable(
  IP: 10.1.1.1.20:38958, SentTimestamp: 1527787420065, FreeRMT: 296MB, CPU: 0%, RTT: 5422ns, :
  IP: 10.1.1.1.30:60623, SentTimestamp: 1527787420065, FreeRMT: 296MB, CPU: 0%, RTT: 4423ns, :
)

root@Server2:/tmp/pscore.45913/Server2.conf
root@Server2:/tmp/pscore.45913/Server2.conf# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
2
[1]- Stopped python -m SimpleHTTPServer 80
[1]+ python -m SimpleHTTPServer 80 &
root@Server2:/tmp/pscore.45913/Server2.conf# bg
root@Server2:/tmp/pscore.45913/Server2.conf# java -cp "/cc-1718/src/ UDPAgent
Received: ProbeRequest(Timestamp: 1527787414788, HHC: 1V67L4zoz3dP1G0JVE6qaz0Qp2pJkDh9k=)
Sent: ProbeResponse(CPU: 0%, RMT: 236 MB, Timestamp: 1527787420065, HHC: yF9Vz3SsQ1006hMUHLvC0HMMvUQ210kUd30VvU2qg=)
Received: ProbeRequest(Timestamp: 1527787414788, HHC: E01X5z1h73wHgKauNovyIdUkzL57FrUJ9ShoDys=)
Sent: ProbeResponse(CPU: 4%, RMT: 236 MB, Timestamp: 1527787425066, HHC: oKSe2u6CNo0t3M5U82BC0M3UK5cIugz21v69ot4=)
Received: ProbeRequest(Timestamp: 1527787425066, HHC: oKSe2u6CNo0t3M5U82BC0M3UK5cIugz21v69ot4=)

root@Server3:/tmp/pscore.45913/Server3.conf
root@Server3:/tmp/pscore.45913/Server3.conf# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
2
[1]- Stopped python -m SimpleHTTPServer 80
[1]+ python -m SimpleHTTPServer 80 &
root@Server3:/tmp/pscore.45913/Server3.conf# bg
root@Server3:/tmp/pscore.45913/Server3.conf# java -cp "/cc-1718/src/ UDPAgent
Received: ProbeRequest(Timestamp: 1527787414788, HHC: 1V67L4zoz3dP1G0JVE6qaz0Qp2pJkDh9k=)
Sent: ProbeResponse(CPU: 4%, RMT: 236 MB, Timestamp: 1527787420065, HHC: eMEY5a1D4uPhwq0Dy1NRN6TqR0dH/AqIwW20U=)
Received: ProbeRequest(Timestamp: 1527787425066, HHC: oKSe2u6CNo0t3M5U82BC0M3UK5cIugz21v69ot4=)
Sent: ProbeResponse(CPU: 0%, RMT: 236 MB, Timestamp: 1527787420065, HHC: yF9Vz3SsQ1006hMUHLvC0HMMvUQ210kUd30VvU2qg=)
```

Figura 3: Servidor 1 (ReverseProxy) e Servidor 2 e Servidor 3 (AgentesUDP com servidor HTTP)

Como podemos verificar, o ReverseProxy e os AgentesUDP encontram-se a correr corretamente e a trocar mensagens (pedidos e respostas de probing entre si). Além disso, é imprimida também a tabela de estado e caso não haja atualização da tabela após uma resposta recebida por esta conter informação mais antiga do que a que já constava na tabela, aparece uma marca "no update on server load".

No portátil 1, pusemos a correr um cliente que faz a conexão ao Reverse-Proxy, sendo-lhe enviada a resposta contendo um ficheiro index.html. De notar, que no exemplo abaixo foi escolhido o servidor 2 para efetuar a conexão com o cliente por ter os melhores valores, nomeadamente o RTT (pode-se verificar isso abaixo, no log do Servidor 1, o proxy).

```
root@Server1:/tmp/pycore.45913/Server1.conf
root@Server1:/tmp/pycore.45913/Server1.conf# java -cp "/cc-1718/src/ ReverseProxy
Sent: ProbeRequest(Timestamp: 1527787414788, HMAC: 1V67FL4zoz391PtoJUVeGwuz0Q2pPjVQdH9k=)
Received: ProbeResponse(CPU: 4%, RM: 236 MB, Timestamp: 1527787414788, HMAC: E01X5z17h73wHgaNovy1dUk-zL57FrUJ9ShoDys=)
StatusTable(
  IP: 10.1.1.1.30:60623, SentTimestamp: 1527787414788, FreeRM: 296MB, CPU: 4%, RTT: 2844ns, :
)
Sent: ProbeRequest(Timestamp: 1527787420065, HMAC: eMIEY5a1D4Phwq0Dy1NRN6TqR0dH/AqIwW20U=)
Received: ProbeResponse(CPU: 0%, RM: 236 MB, Timestamp: 1527787420065, HMAC: yF9Vz35a1Q06oHMH5LVC0HMMVUQZ1QkUld30VU2qg=)
StatusTable(
  IP: 10.1.1.1.20:38958, SentTimestamp: 1527787420065, FreeRM: 296MB, CPU: 0%, RTT: 1003ns, :
  IP: 10.1.1.1.30:60623, SentTimestamp: 1527787414788, FreeRM: 296MB, CPU: 4%, RTT: 2844ns, :
)
Received: ProbeResponse(Timestamp: 1527787414788, HMAC: E01X5z17h73wHgaNovy1dUk-zL57FrUJ9ShoDys=)
StatusTable(
  IP: 10.1.1.1.20:38958, SentTimestamp: 1527787420065, FreeRM: 296MB, CPU: 0%, RTT: 5422ns, :
  IP: 10.1.1.1.30:60623, SentTimestamp: 1527787414788, FreeRM: 296MB, CPU: 4%, RTT: 2844ns, :
)
[no update on server load]
Sent: ProbeRequest(Timestamp: 1527787425066, HMAC: oKSe2v8N0x3M61sB2BC10M3UK5Lugz21v69ot4=)
Received: ProbeResponse(CPU: 0%, RM: 236 MB, Timestamp: 1527787420065, HMAC: yF9Vz35a1Q06oHMH5LVC0HMMVUQZ1QkUld30VU2qg=)
StatusTable(
  IP: 10.1.1.1.20:38958, SentTimestamp: 1527787420065, FreeRM: 296MB, CPU: 0%, RTT: 5422ns, :
  IP: 10.1.1.1.30:60623, SentTimestamp: 1527787420065, FreeRM: 296MB, CPU: 0%, RTT: 4423ns, :
)

root@Server2:/tmp/pycore.45913/Server2.conf
root@Server2:/tmp/pycore.45913/Server2.conf# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
2
[1]* Stopped python -m SimpleHTTPServer 80
[1]* python -m SimpleHTTPServer 80 &
root@Server2:/tmp/pycore.45913/Server2.conf# java -cp "/cc-1718/src/ WPAgent
Received: ProbeRequest(Timestamp: 1527787414788, HMAC: 1V67FL4zoz391PtoJUVeGwuz0Q2pPjVQdH9k=)
Sent: ProbeResponse(CPU: 0%, RM: 236 MB, Timestamp: 1527787420065, HMAC: yF9Vz35a1Q06oHMH5LVC0HMMVUQZ1QkUld30VU2qg=)
Received: ProbeRequest(Timestamp: 1527787414788, HMAC: E01X5z17h73wHgaNovy1dUk-zL57FrUJ9ShoDys=)
Received: ProbeResponse(CPU: 4%, RM: 236 MB, Timestamp: 1527787425066, HMAC: oKSe2v8N0x3M61sB2BC10M3UK5Lugz21v69ot4=)
Received: ProbeRequest(Timestamp: 1527787425066, HMAC: oKSe2v8N0x3M61sB2BC10M3UK5Lugz21v69ot4=)

root@Server3:/tmp/pycore.45913/Server3.conf# python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
2
[1]* Stopped python -m SimpleHTTPServer 80
[1]* python -m SimpleHTTPServer 80 &
root@Server3:/tmp/pycore.45913/Server3.conf# java -cp "/cc-1718/src/ WPAgent
Received: ProbeRequest(Timestamp: 1527787414788, HMAC: 1V67FL4zoz391PtoJUVeGwuz0Q2pPjVQdH9k=)
Sent: ProbeResponse(CPU: 4%, RM: 236 MB, Timestamp: 1527787414788, HMAC: E01X5z17h73wHgaNovy1dUk-zL57FrUJ9ShoDys=)
Received: ProbeRequest(Timestamp: 1527787420065, HMAC: eMIEY5a1D4Phwq0Dy1NRN6TqR0dH/AqIwW20U=)
Received: ProbeRequest(Timestamp: 1527787425066, HMAC: oKSe2v8N0x3M61sB2BC10M3UK5Lugz21v69ot4=)
Sent: ProbeResponse(CPU: 0%, RM: 236 MB, Timestamp: 1527787420065, HMAC: yF9Vz35a1Q06oHMH5LVC0HMMVUQZ1QkUld30VU2qg=)
[1]
```

Figura 4: Cliente (Portatil1) conectou-se com sucesso

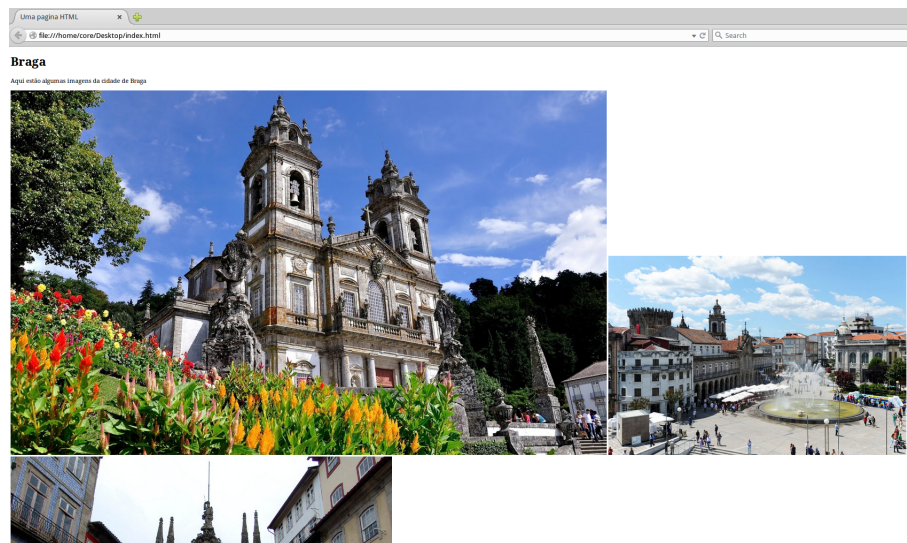
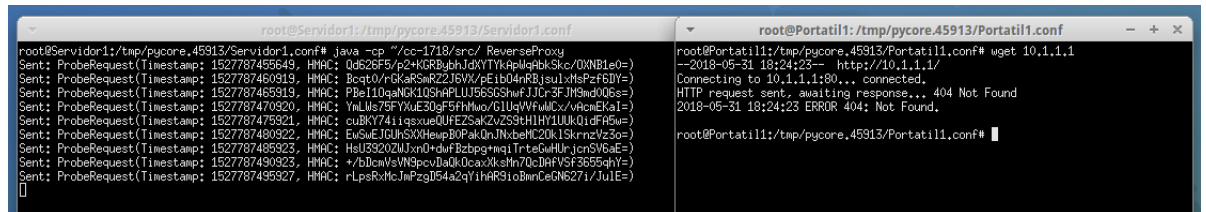


Figura 5: Index.html recebido

Num segundo exemplo, pusemos o cliente a tentar conectar-se não havendo ainda qualquer servidor de *back-end* disponível. Assim, o cliente recebeu uma mensagem de erro HTTP 404 enviada pelo ReverseProxy (MonitorUDP), como se pode verificar pela imagem abaixo:



```
root@Servidor1:/tmp/pycore.45913/Servidor1.conf# java -cp "/cc-1718/src/ ReverseProxy
Sent: ProbeRequest(Timestamp: 152778745643, HMAC: Qd526F5/p2+KGRBubhJdKYTYkApMqAbkSkc/0XNB1e0=)
Sent: ProbeRequest(Timestamp: 1527787460319, HMAC: Bcqt0/rGKaRSwRZ2J6VX/pE1b04nRBjsuLxMsPzF6DY=)
Sent: ProbeRequest(Timestamp: 1527787465919, HMAC: PBeI10qaNGK1QShAPLUJ566SGShwFJJCr3FJN3wd006s=)
Sent: ProbeRequest(Timestamp: 1527787470920, HMAC: YnLUs79FY6uE30gF5PHMwo/GIUqVWFWuLc/vRacEKaI=)
Sent: ProbeRequest(Timestamp: 1527787475921, HMAC: ou8KY74i1q8ue8URFZSaZoz26SH1H1UUK0Idf96w=)
Sent: ProbeRequest(Timestamp: 1527787480922, HMAC: Ev6uEJGU8XWwepD0Fak0nJNdbetC20k1SkrmzVz3o=)
Sent: ProbeRequest(Timestamp: 1527787485923, HMAC: HsU39202UJnn0+dufBzbpq+qaiTTeGwHLrjcnSV6aE=)
Sent: ProbeRequest(Timestamp: 1527787490923, HMAC: +/bIcmVzW9pcvDa0k0ca0KsM70cDaFVSf3655ghY=)
Sent: ProbeRequest(Timestamp: 1527787495927, HMAC: rLpsRvMcJmPzgD54a2QYih4R9ioBwnCeGN6271/JulE=)
[]

root@Portatil1:/tmp/pycore.45913/Portatil1.conf# uget 10.1.1.1
--2018-05-31 18:24:23-- http://10.1.1.1/
Connecting to 10.1.1.1:80... connected.
HTTP request sent, awaiting response... 404 Not Found
2018-05-31 18:24:23 ERROR 404: Not Found.

root@Portatil1:/tmp/pycore.45913/Portatil1.conf#
```

Figura 6: Index.html recebido

6 Conclusões e trabalho futuro

Após a conclusão deste projeto prático o grupo conseguiu implementar com sucesso o *reverse proxy* proposto. Incluíram-se os componentes exigidos e as funcionalidades como os envios de mensagens de *probing* e receção das respostas às mesmas, distinção entre envios em *Multicast* e *Unicast*, respostas quanto à carga da máquina, etc.

Para trabalho futuro, visto que não houve a oportunidade de incorporar essa funcionalidade na nossa solução por escassez de tempo, poderia-se estruturar um tratamento para as quebras de conexões de servidores do *back-end*.

Em suma, a realização deste trabalho proporcionou-nos uma destreza e um conhecimento mais completo quando aos conteúdos lecionados aos longo das aulas teóricas e práticas, revelando-se uma atividade vantajosa para alcançar um bom resultado na Unidade Curricular de Comunicações por Computador.