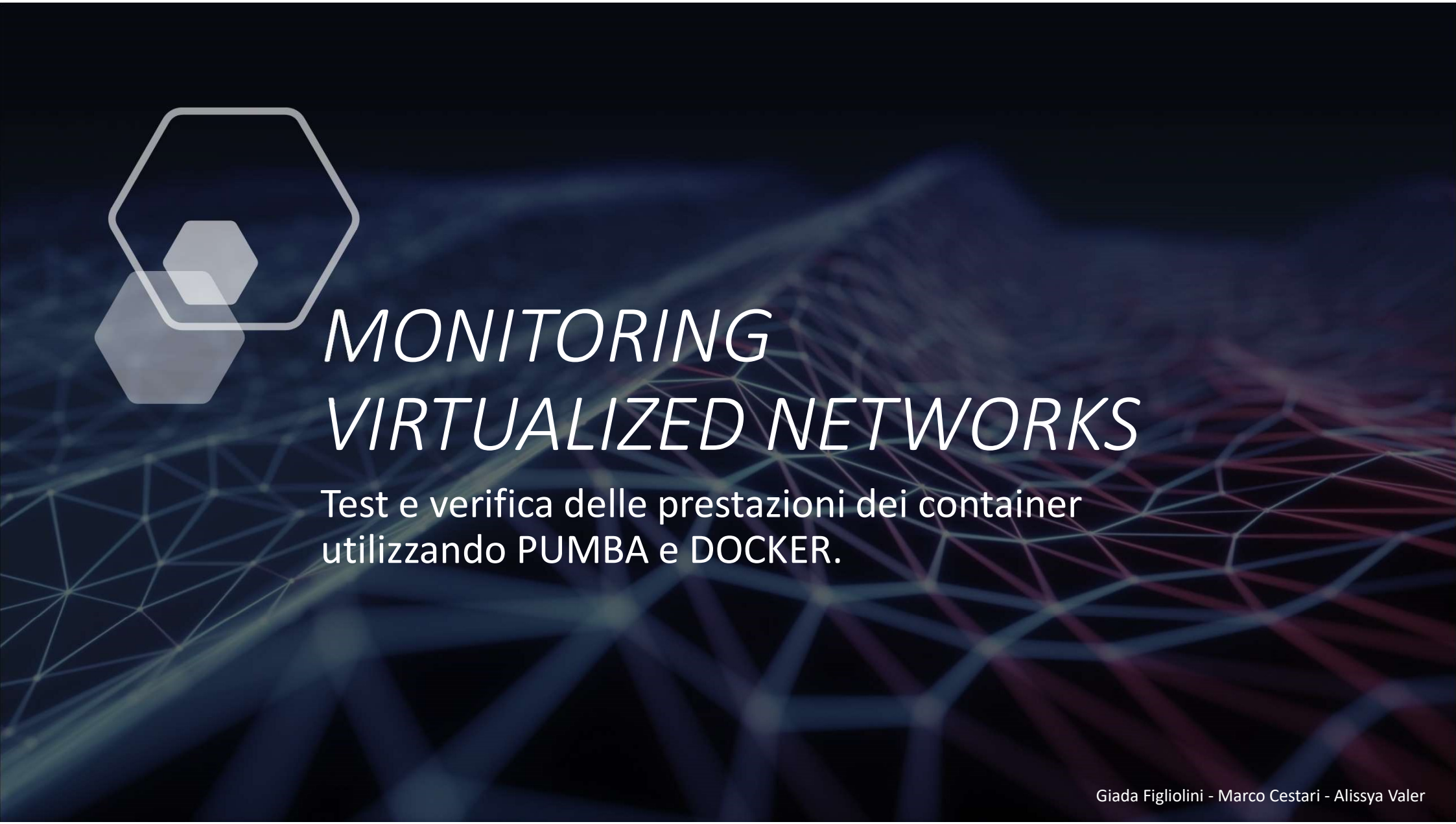

MONITORING
VIRTUALIZED NETWORKS
Test e verifica delle prestazioni dei container
utilizzando PUMBA e DOCKER.
Giada Figliolini - Marco Cestari - Alissya Valer

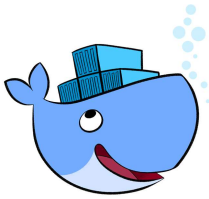
Che strumenti abbiamo utilizzato?

Per creare, gestire e visualizzare le risorse consumate abbiamo usato Docker Desktop.

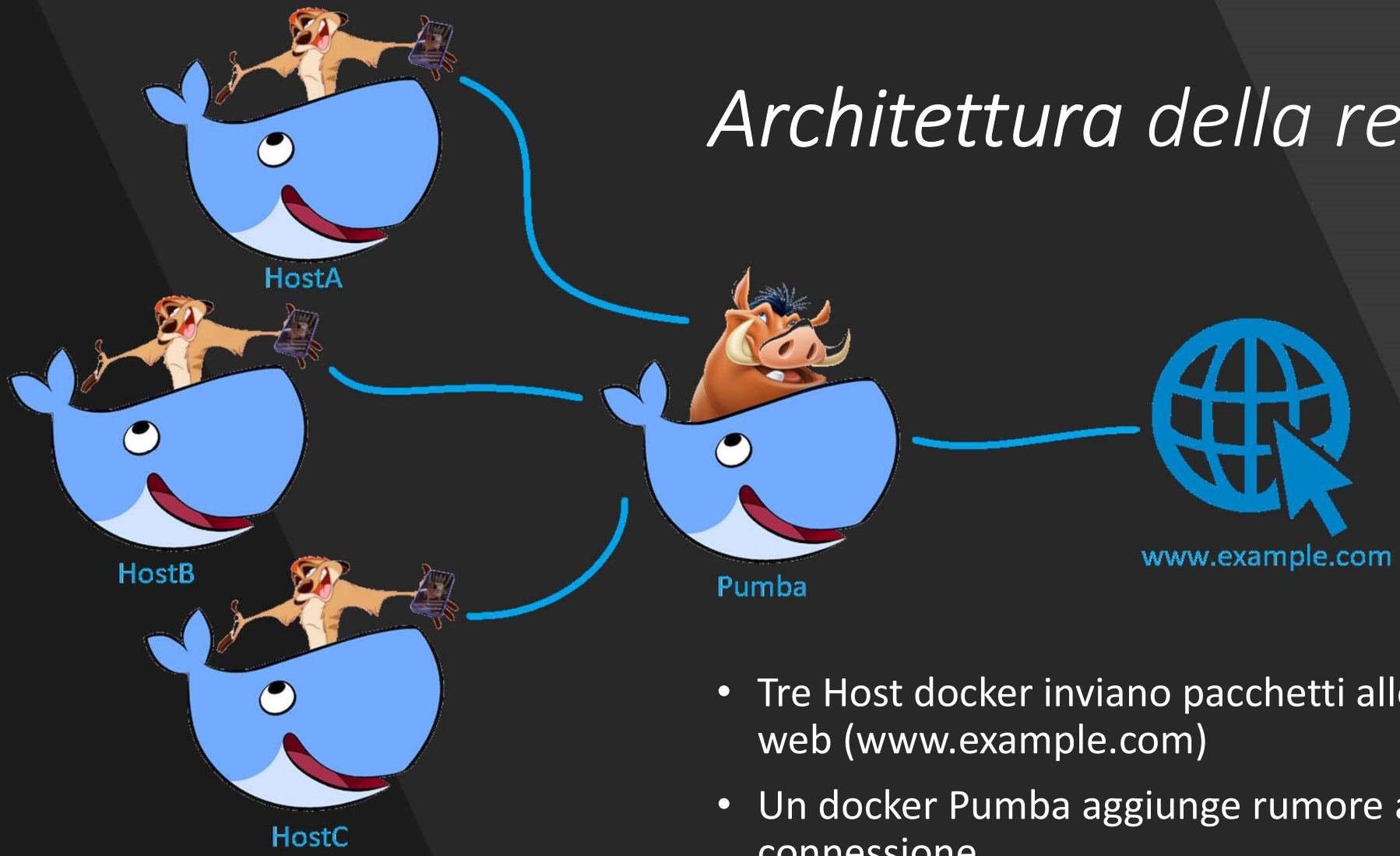
Per simulare ritardi, perdite e stress sui Containers abbiamo usato Pumba.

Ambiente: Windows 10

Docker images: Alpine, Pumba



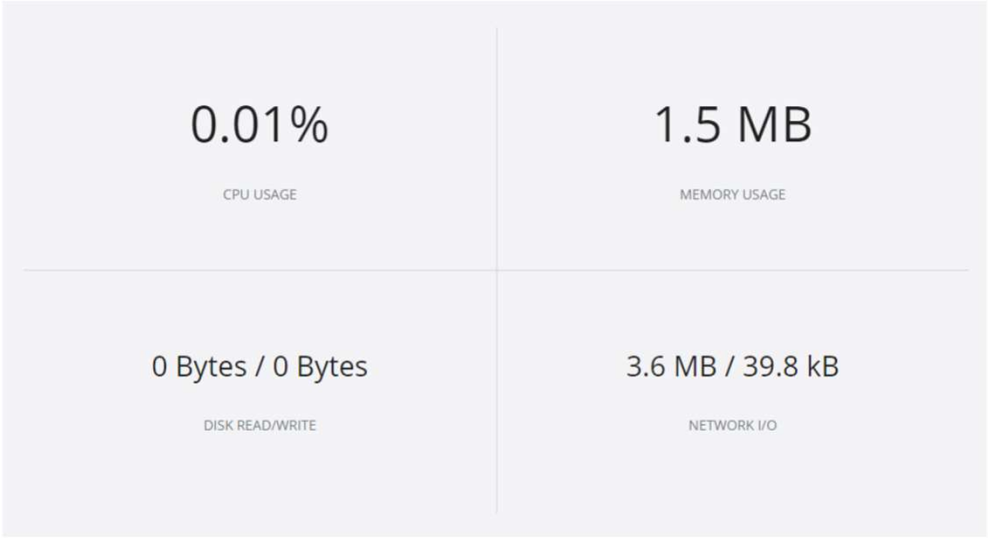
Architettura della rete

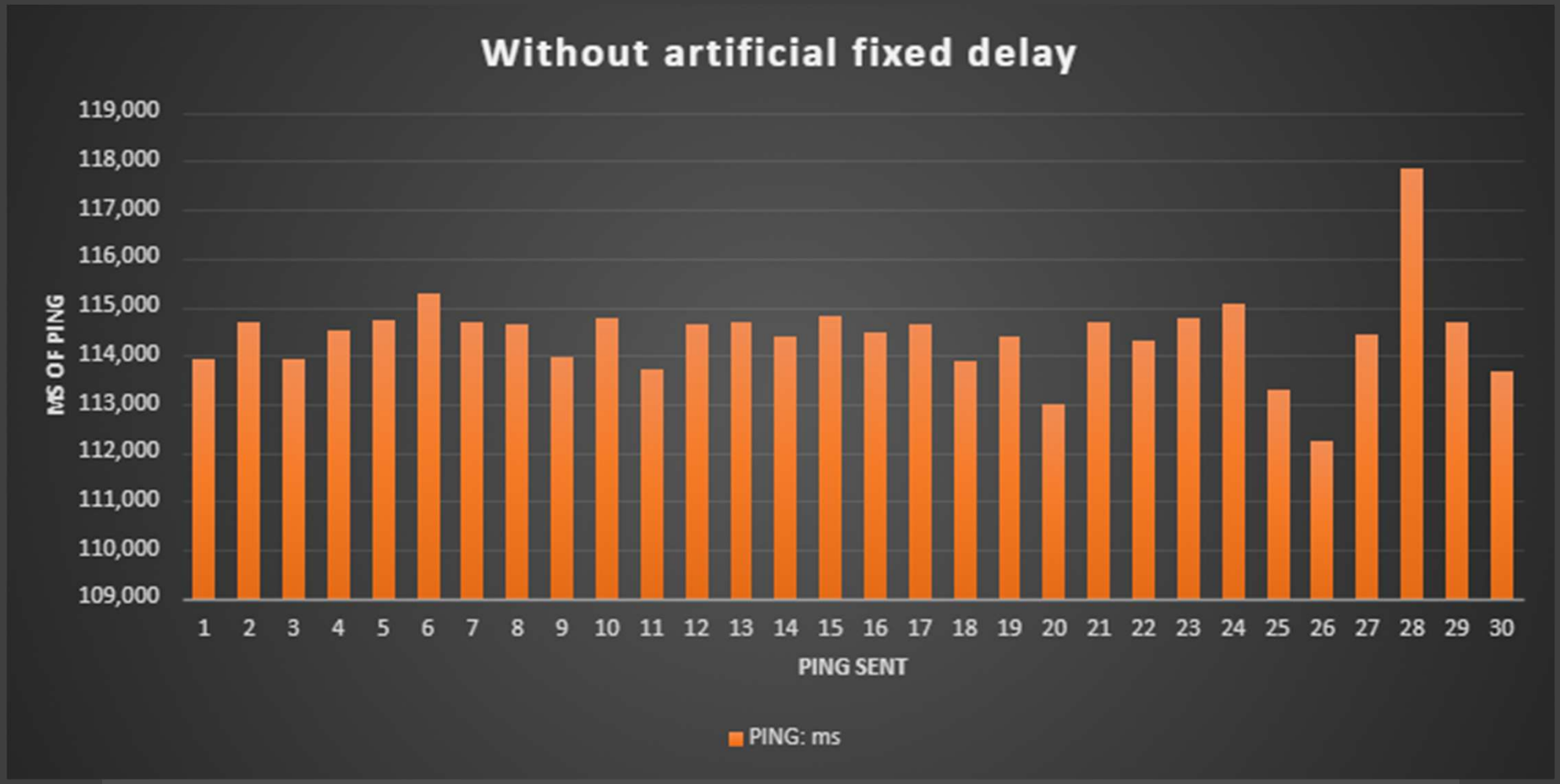


- Tre Host docker inviano pacchetti allo stesso sito web (www.example.com)
- Un docker Pumba aggiunge rumore alla connessione

MONITORAGGIO RETE SENZA RITARDI FISSI

- In questo caso abbiamo effettuato un monitoraggio del ping dall'HostA verso un server chiamato www.example.com.
- È possibile notare che il ritardo in millisecondi è stato uniforme.
- Abbiamo inoltre monitorato le risorse hardware utilizzate dall'HostA durante il test.
- Per il test abbiamo NON abbiamo applicato alcun ritardo.

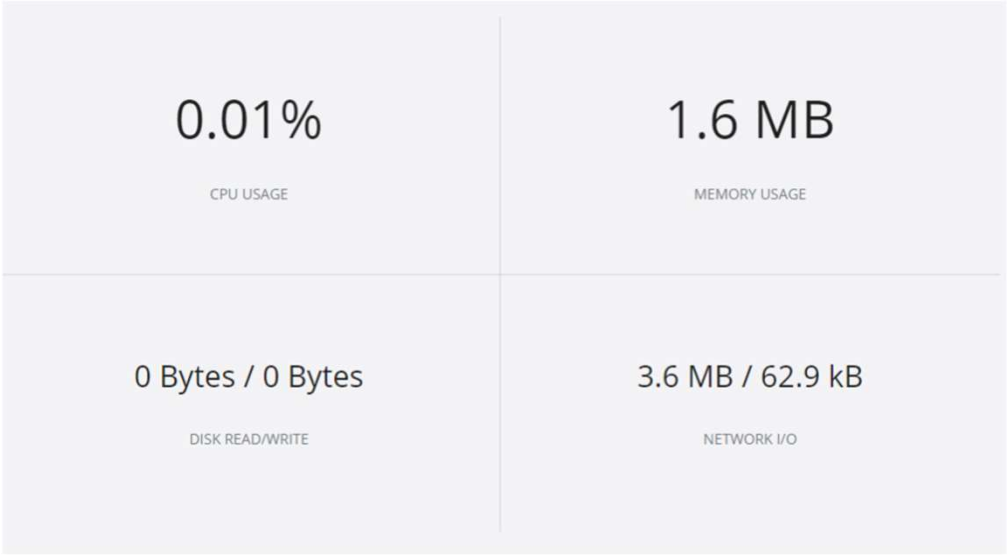


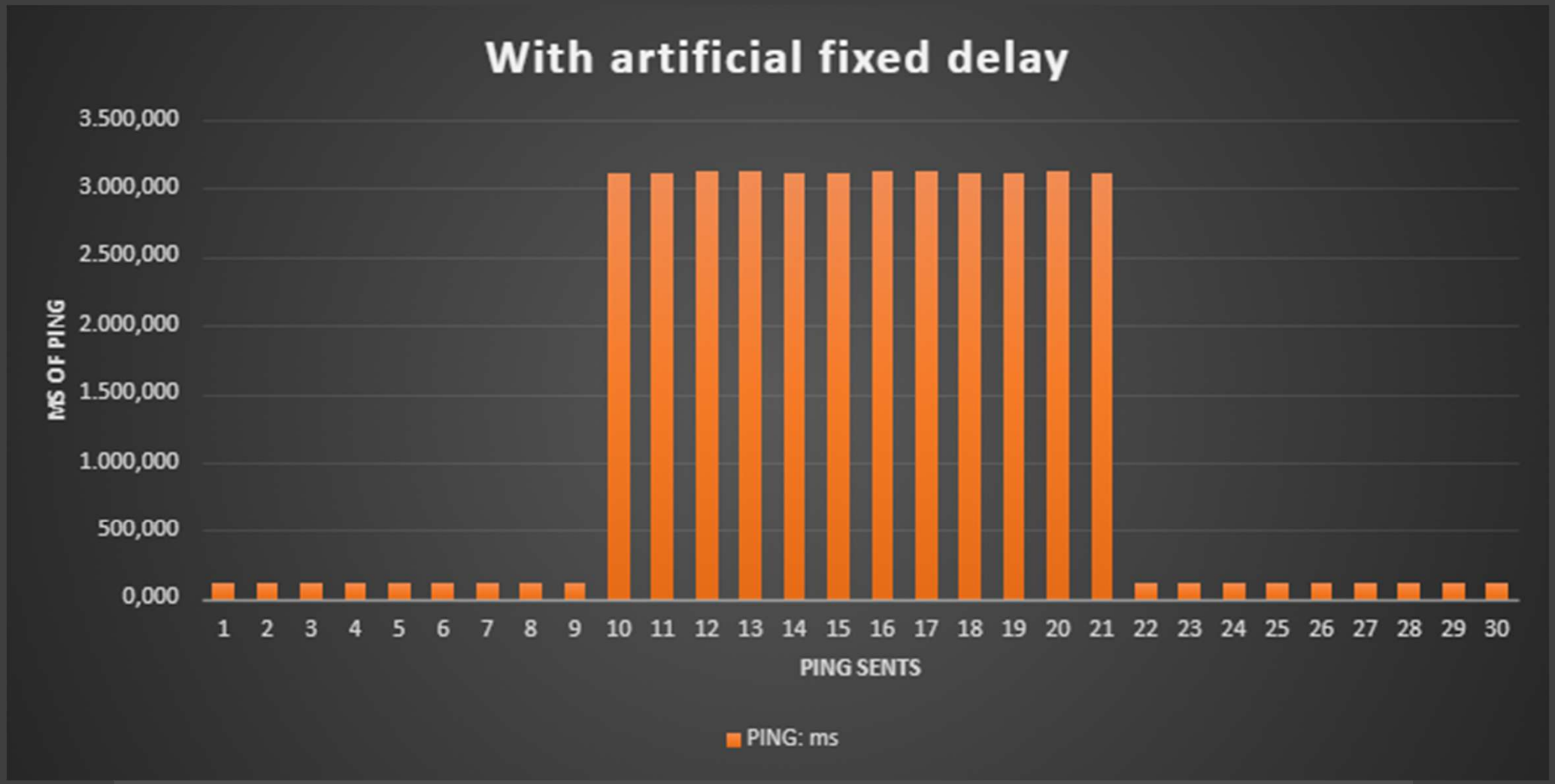


Dal grafico è visibile l’uniformità del ritardo rispetto al sito di prova.

MONITORAGGIO RETE CON RITARDI FISSI

- In questo caso abbiamo effettuato un monitoraggio del ping dall'HostB verso un server chiamato www.example.com.
- È possibile notare che con l'aggiunta del ritardo fisso il ping è aumentato per una durata pari a circa 10 secondi in maniera uniforme
- Abbiamo monitorato le risorse hardware dell'HostB e non abbiamo notato cambiamenti nell'utilizzo delle risorse.
- Per il test abbiamo applicato un ritardo di circa 3000 ms.





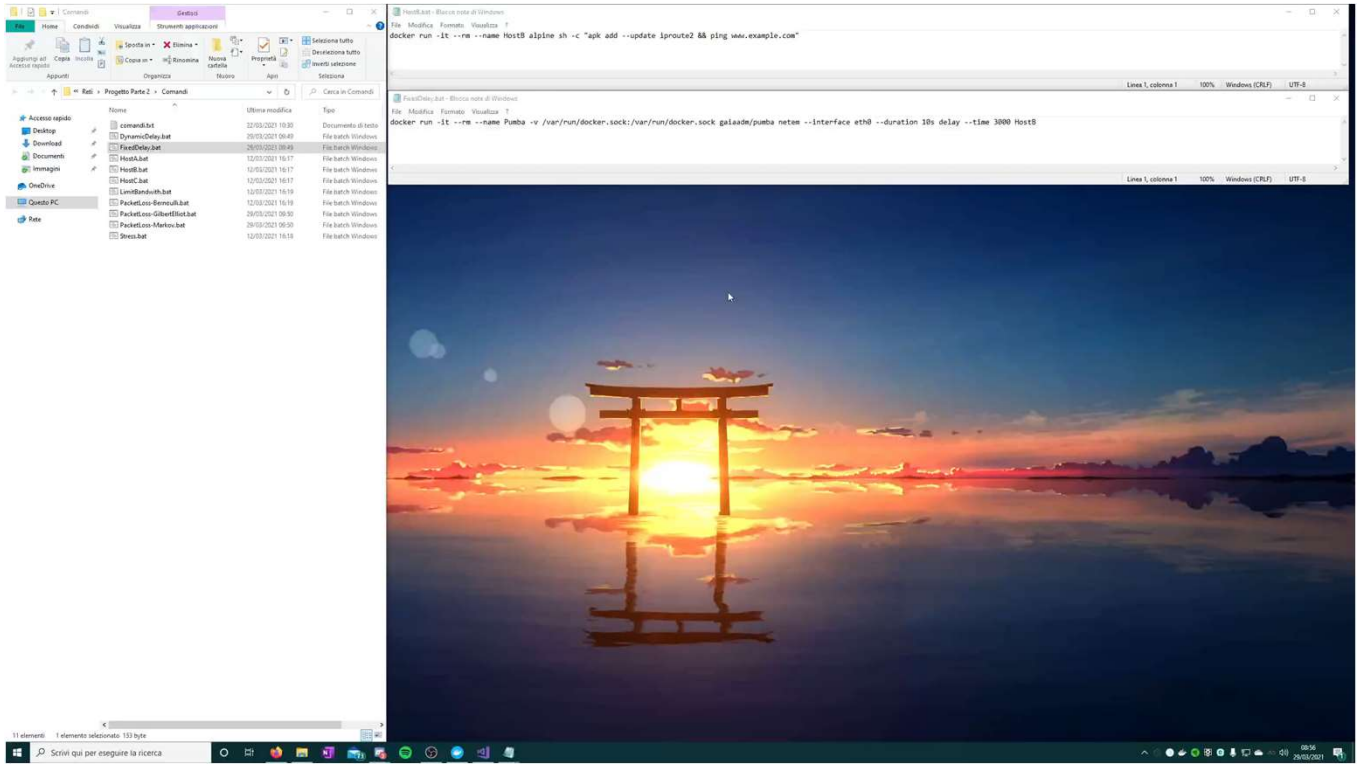
Dal grafico è visibile l'uniformità del ritardo nel tempo e il momento in cui il ritardo è stato immesso rispetto al sito di prova.

Comandi utilizzati

Inserimento ritardo fisso:

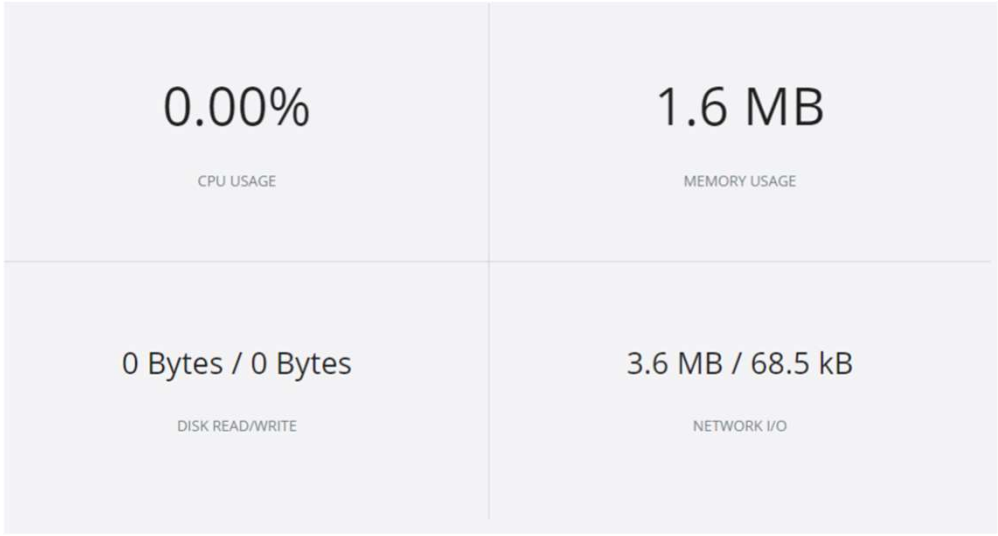
```
docker run -it --rm --name Pumba -v
/var/run/docker.sock:/var/run/docker.sock
gaiaadm/pumba netem --interface eth0 --
duration 60s delay --time 3000 HostB
```

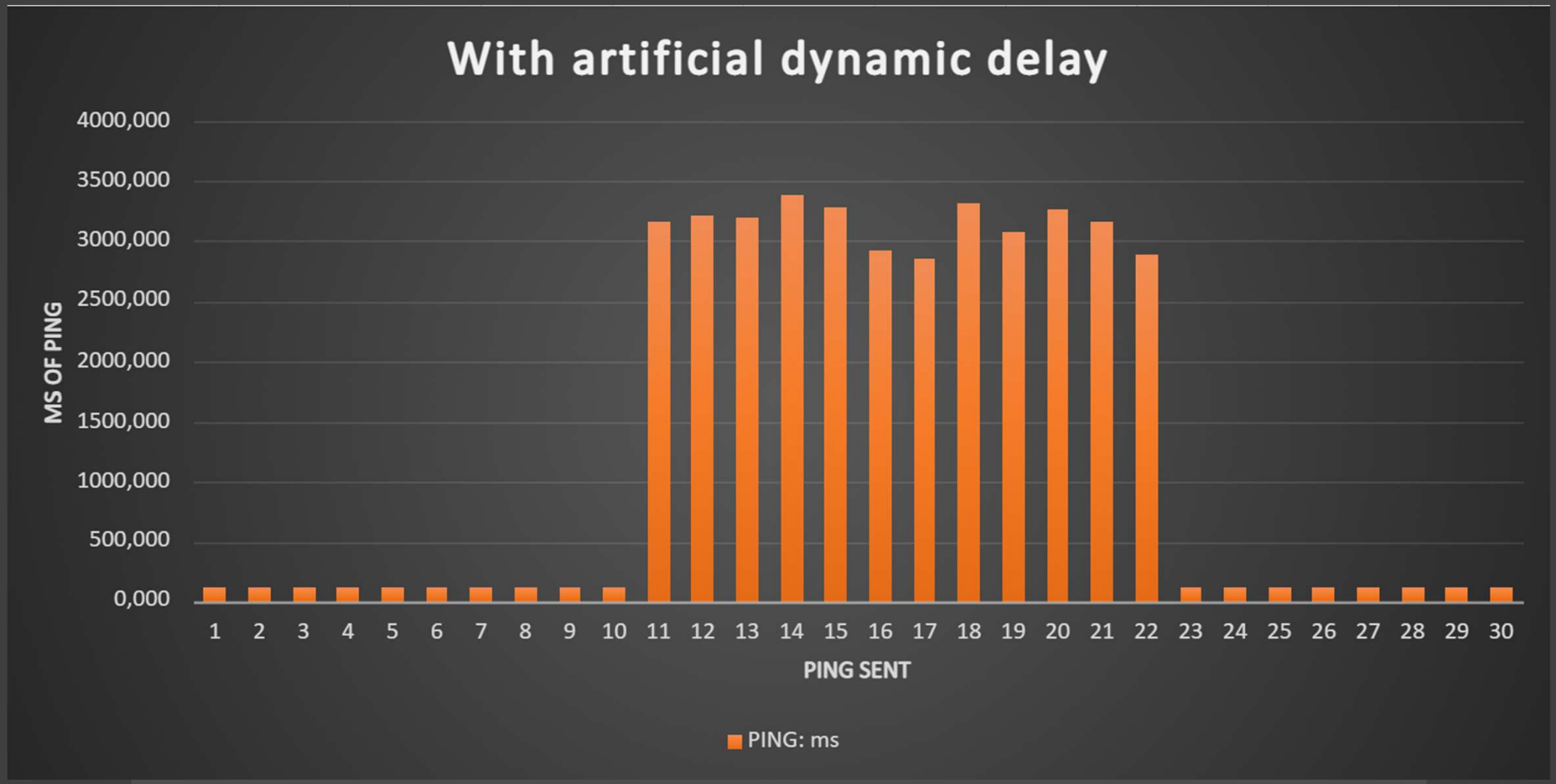
- **Interface eth0**: imposta l'ethernet di default del dispositivo in cui inserire il ritardo.
- **Duration 60s**: indica la durata dello script.
- **Delay –time 3000**: indica, in millisecondi, il ritardo fisso da aggiungere.
- **HostB**: indica l'host in cui è necessario immettere il ritardo



MONITORAGGIO RETE CON RITARDI DINAMICI (JITTER)

- In questo caso abbiamo effettuato un monitoraggio del ping dall'HostC verso un server chiamato www.example.com.
- È possibile notare che con l'aggiunta del ritardo variabile il ping è aumentato per una durata pari a circa 10 secondi in maniera **non** uniforme (varianzione +/- 300 ms su un fisso di 3000 ms)
- Abbiamo monitorato le risorse hardware dell'HostC e non abbiamo notato cambiamenti nell'utilizzo delle risorse rispetto ai precendenti esempi.





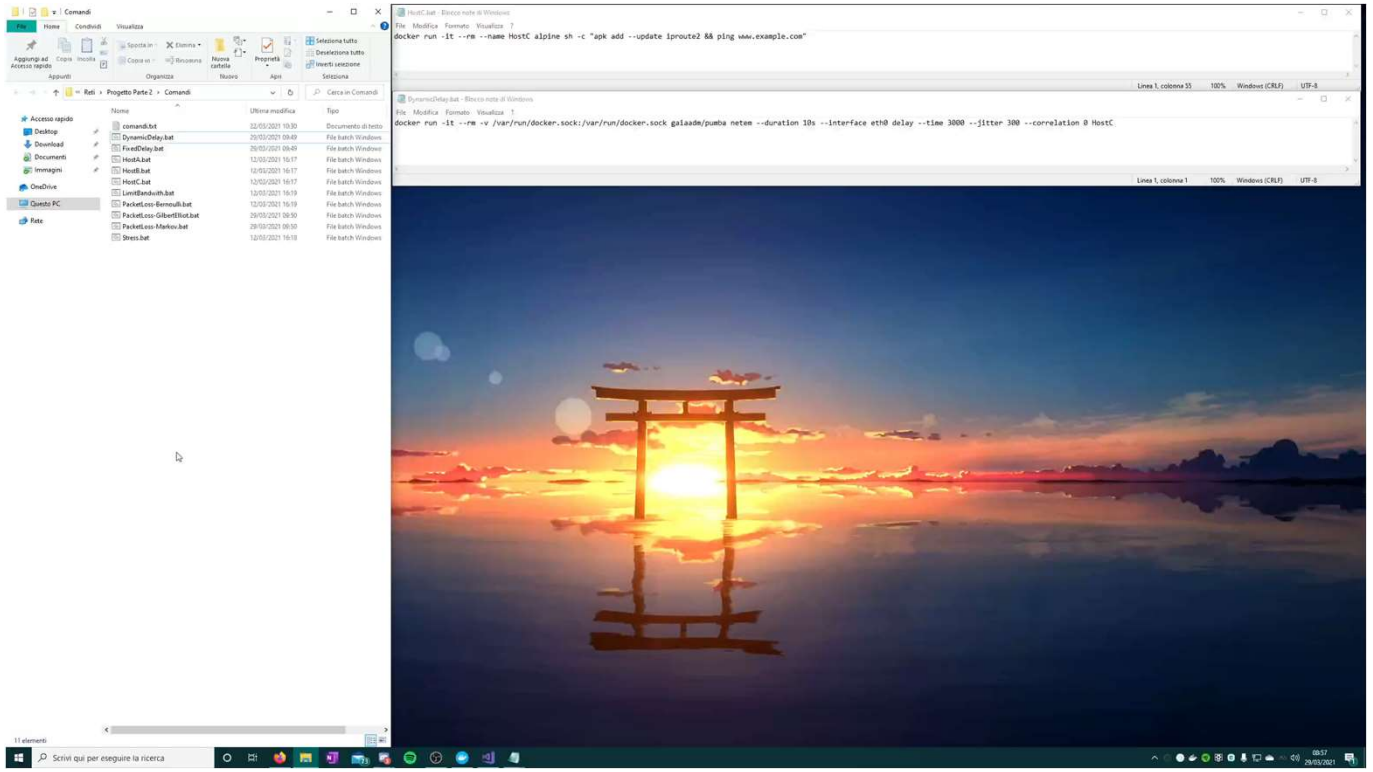
Dal grafico è visibile la disomogeneità del ritardo rispetto al sito di prova.

Comandi utilizzati

Inserimento ritardo dinamico:

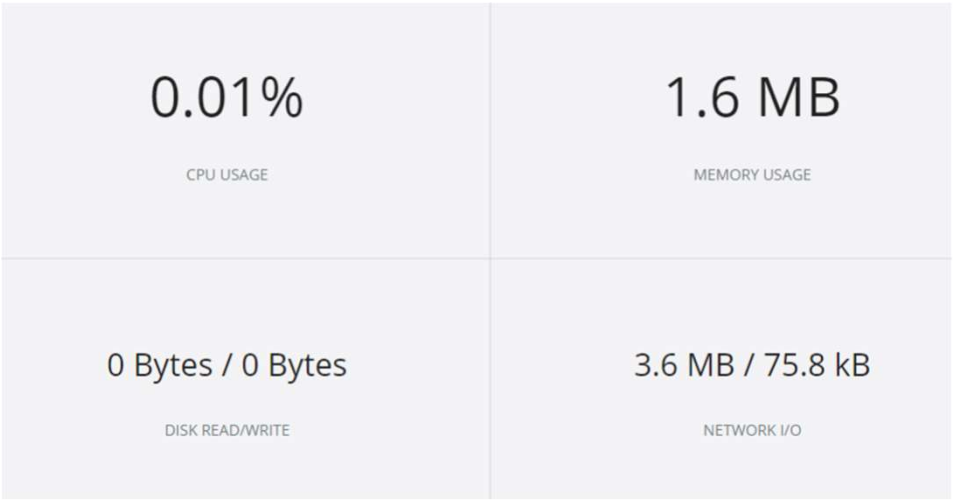
```
docker run -it --rm -v
/var/run/docker.sock:/var/run/docker.sock
gaiaadm/pumba netem --duration 60s --
interface eth0 delay --time 3000 --jitter
300 --correlation 0 HostC
```

- **Interface eth0**: imposta l'ethernet di default del dispositivo in cui inserire il ritardo.
- **Duration 60s**: indica la durata dello script.
- **Delay –time 3000**: indica, in millisecondi, il ritardo fisso da aggiungere.
- **HostC**: indica l'host in cui è necessario immettere il ritardo
- **Correlation 0**: corrisponde alla % di correlazione tra il ritardo di un pacchetto e l'altro
- **Jitter 300**: indica la variazione del ritardo



MONITORAGGIO RETE CON PERDITA DEI PACCHETTI – BERNOULLI

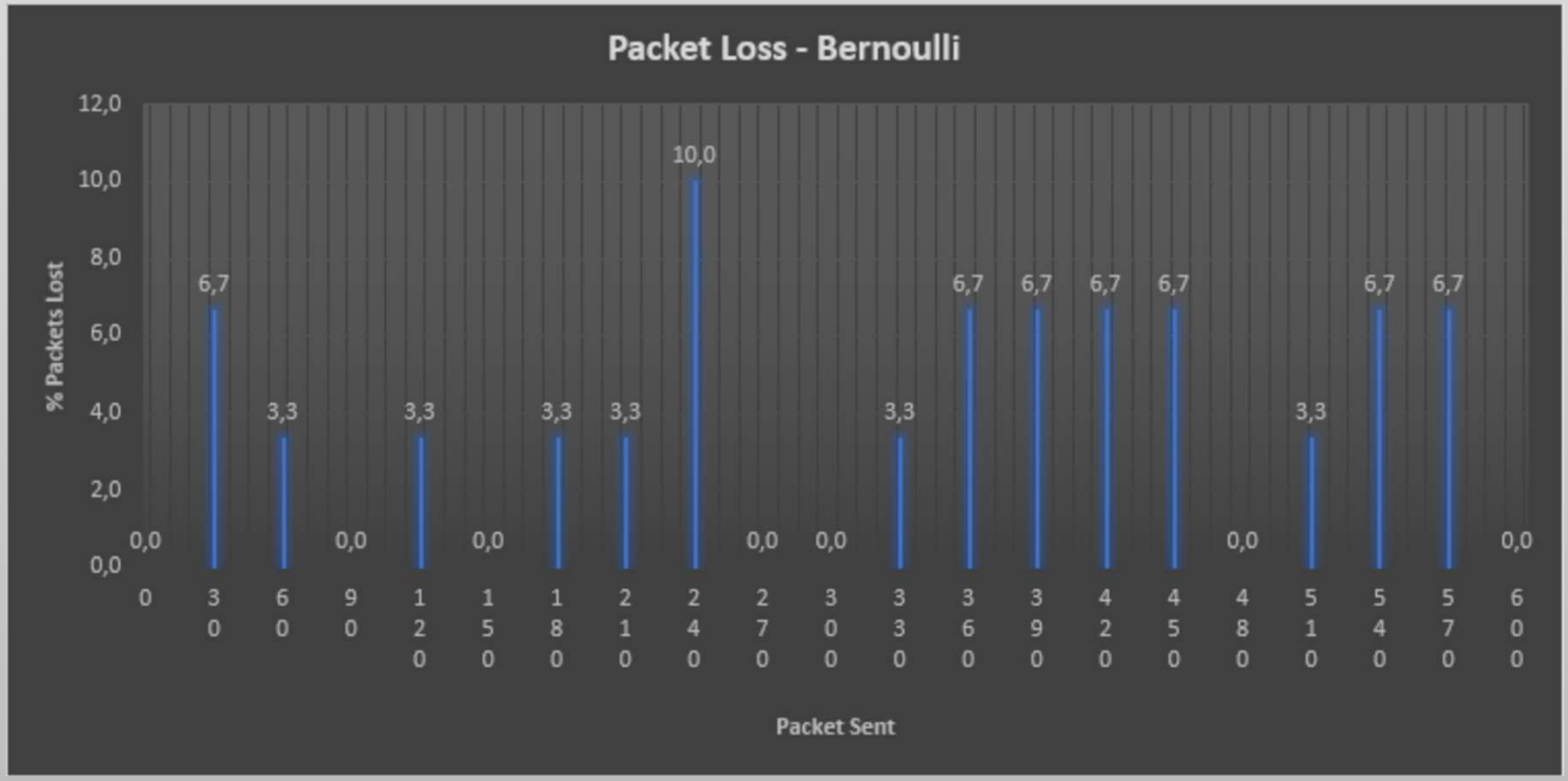
- In questo caso abbiamo effettuato un monitoraggio del ping dall'HostA verso un server chiamato www.example.com.
- Utilizzando Pumba abbiamo simulato delle perdite di pacchetti attraverso **il metodo di Bernoulli.**
- Abbiamo monitorato le risorse hardware dell'HostA e non abbiamo notato cambiamenti nell'utilizzo delle risorse.

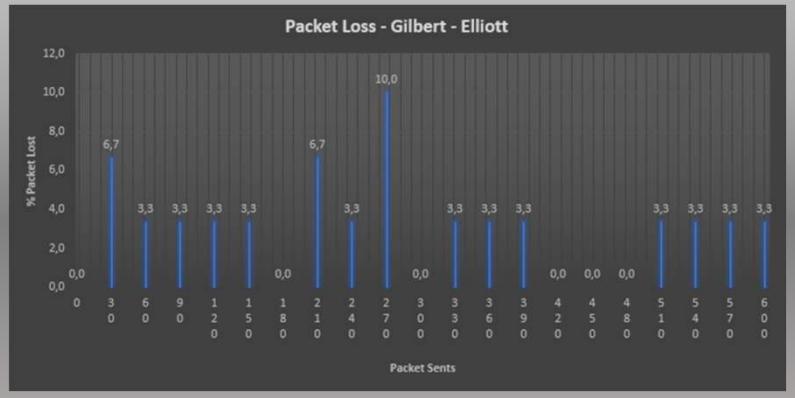


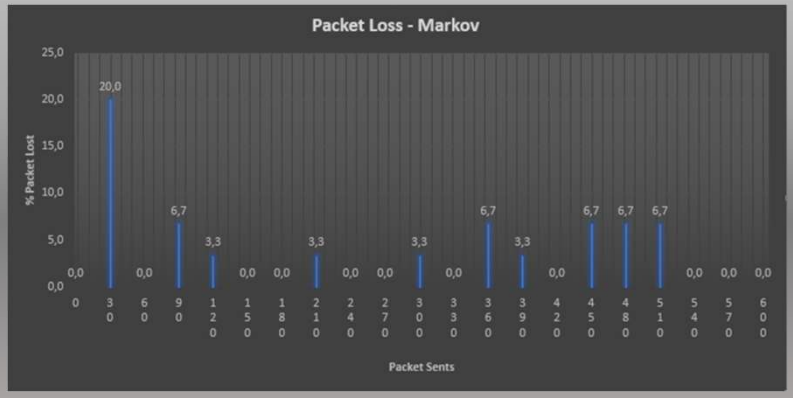
Parametri Utilizzati:

- Perdita: 4%
- Correlation: 0%

Dal grafico è facilmente visibile la suddivisione della perdita dei pacchetti.
L'algoritmo si è dimostrato affidabile.
Dalle nostre analisi sono stati persi il 3,83% dei pacchetti mostrando dei picchi fino al 10% in alcune zone e lo 0% in altre.

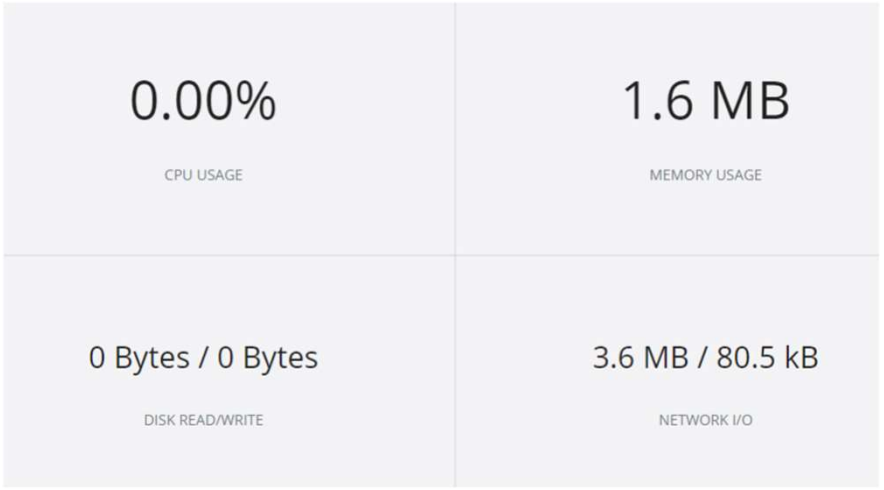






MONITORAGGIO RETE CON PERDITA DEI PACCHETTI – MARKOV

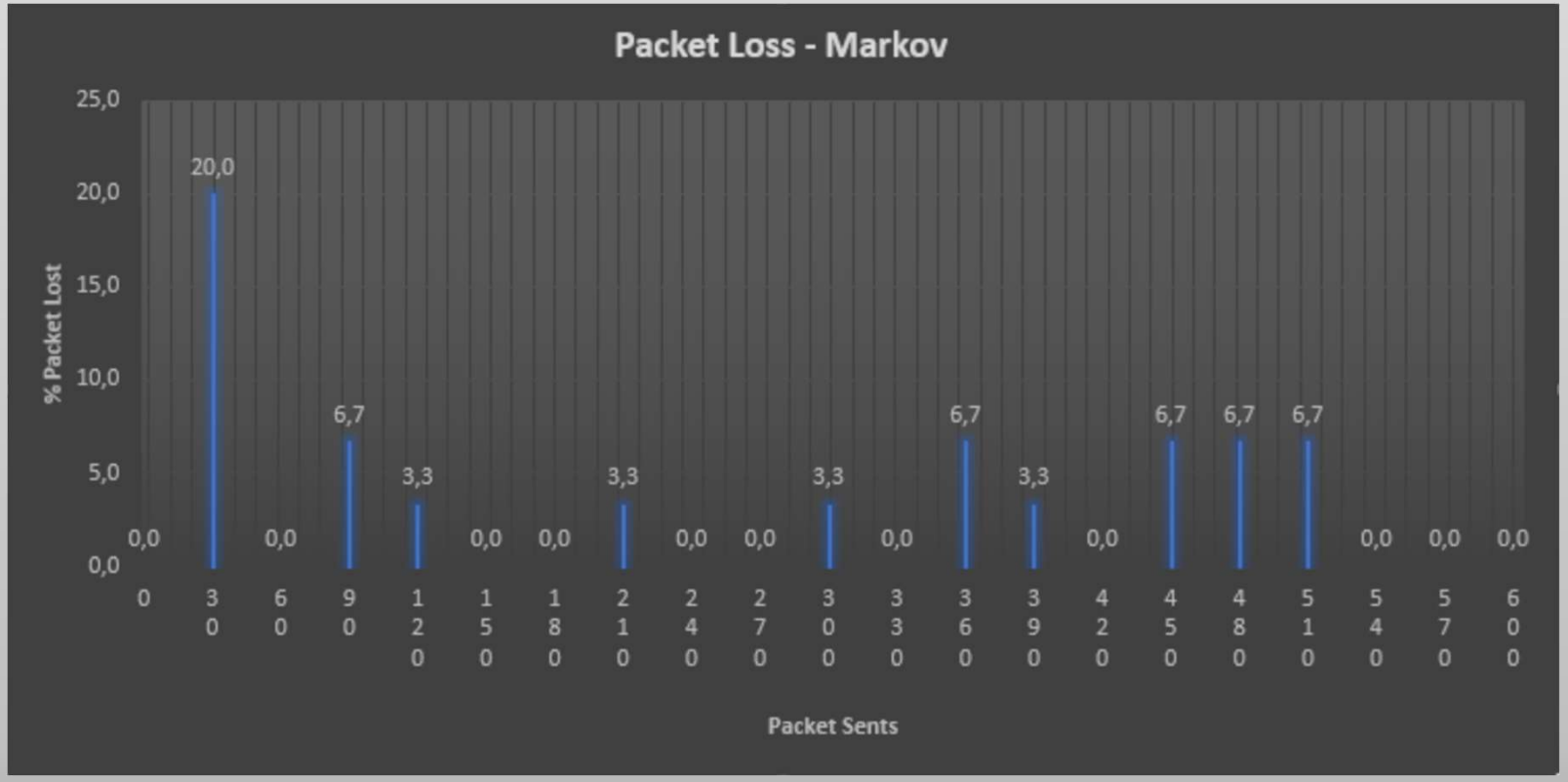
- In questo caso abbiamo effettuato un monitoraggio del ping dall'HostB verso un server chiamato www.example.com.
- Utilizzando Pumba abbiamo simulato delle perdite di pacchetti attraverso il **metodo di Markov.**
- Abbiamo monitorato le risorse hardware dell'HostB e non abbiamo notato cambiamenti nell'utilizzo delle risorse.

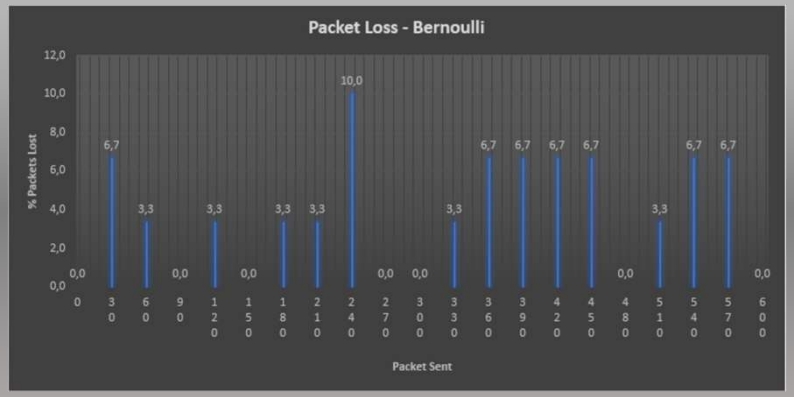


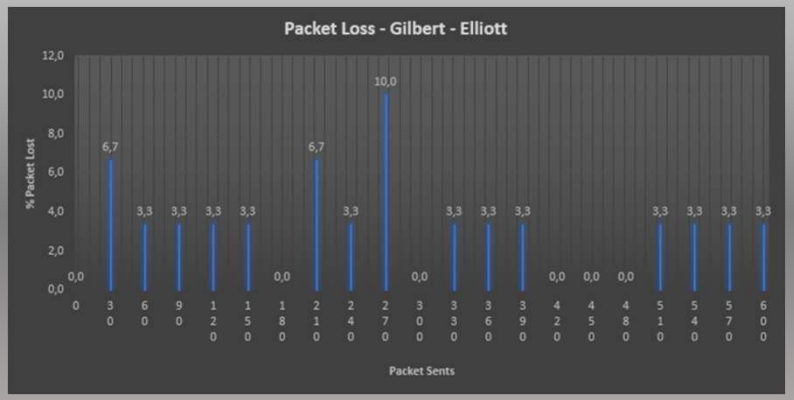
I parametri utilizzati sono stati:

- P13 al 2 %
- p31 al 80 %
- p32 al 12 %
- p23 al 5 %
- p14 al 11 %

Dal grafico è facilmente visibile la suddivisione della perdita dei pacchetti.
L'algoritmo si è dimostrato molto affidabile dopo il 30° ping.
Dalle nostre analisi sono stati persi il 3,33% dei pacchetti mostrando dei picchi fino al 20% in alcune zone e lo 0% in altre.

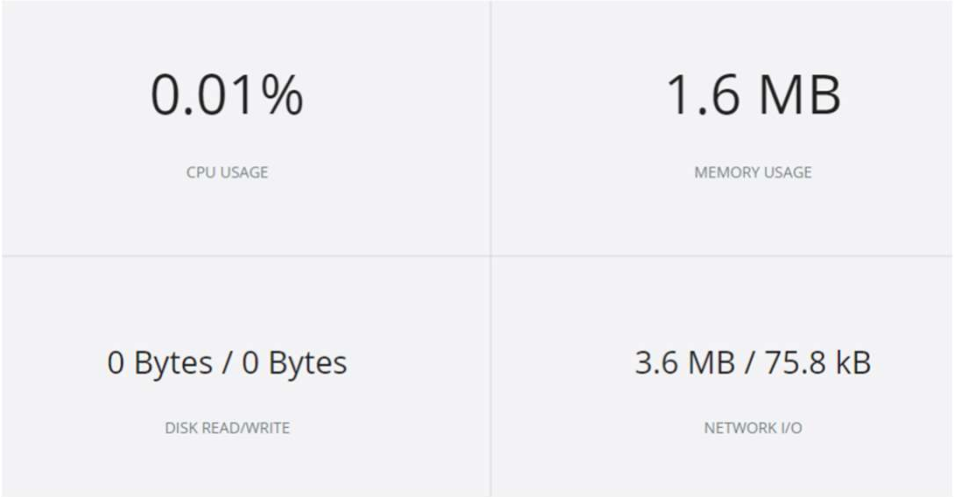






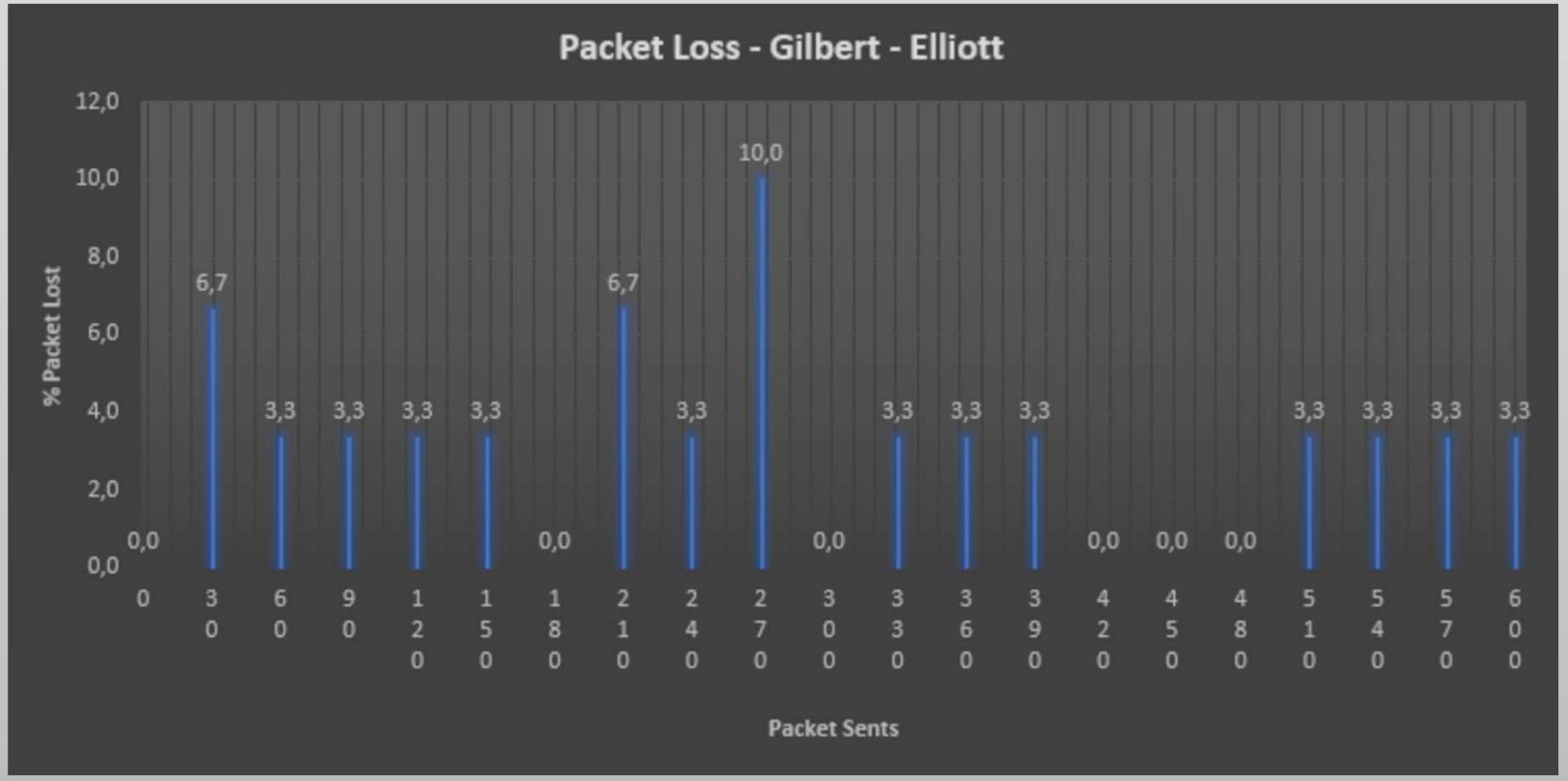
MONITORAGGIO RETE CON PERDITA DEI PACCHETTI – GILBERT ELLIOT

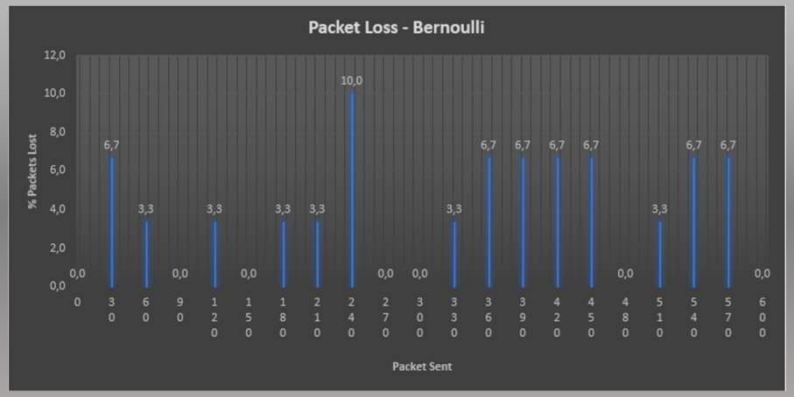
- In questo caso abbiamo effettuato un monitoraggio del ping dall'HostC verso un server chiamato www.example.com.
- Utilizzando Pumba abbiamo simulato delle perdite di pacchetti attraverso il **metodo di *Gilbert Elliott*.**
- Abbiamo monitorato le risorse hardware dell'HostC e non abbiamo notato cambiamenti nell'utilizzo delle risorse.

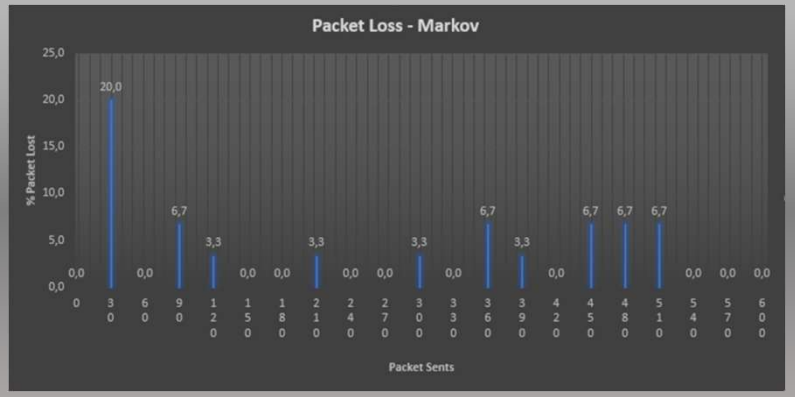


Transition probability to the bad state p: 3%
Transition probability in the good state r: 97%
Loss probability in the bad state h: 98%
Loss probability in the good state k: 2%

Dal grafico è facilmente visibile la suddivisione della perdita dei pacchetti.
L'algoritmo si è dimostrato molto affidabile.
Dalle nostre analisi sono stati persi il 3,16% dei pacchetti mostrando dei picchi fino al 10% in alcune zone e lo 0% in altre.







Comandi utilizzati

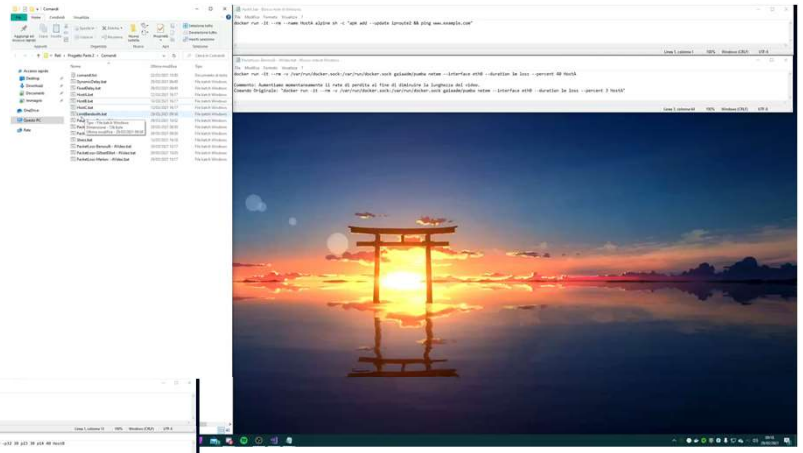
Inserimento perdita pacchetti:

Bernulli:
docker run -it --rm -v /var/run/docker.sock:/var/run/docker.sock gaiaadm/pumba netem --interface eth0 --duration 1m loss --percent 3 HostA

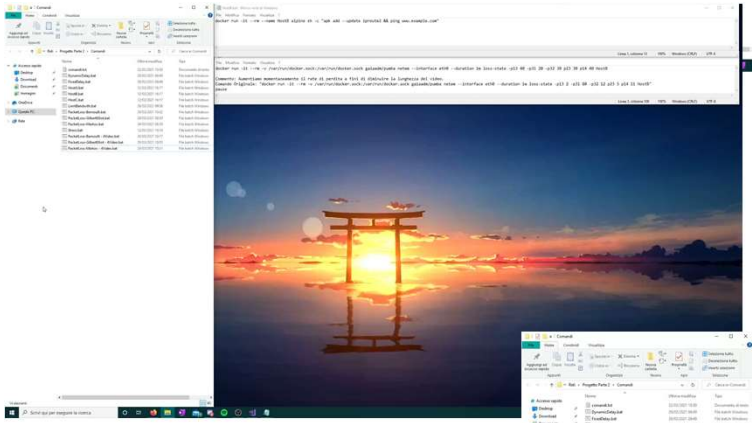
Markov:
docker run -it --rm -v /var/run/docker.sock:/var/run/docker.sock gaiaadm/pumba netem --interface eth0 --duration 1m loss-state -p13 2 -p31 80 -p32 12 p23 5 p14 11 HostB

Gilbert-Elliot:
docker run -it --rm -v /var/run/docker.sock:/var/run/docker.sock gaiaadm/pumba netem --interface eth0 --duration 1m loss-gemodel -p 3 -r 97 --one-h 98 --one-k 2 HostC

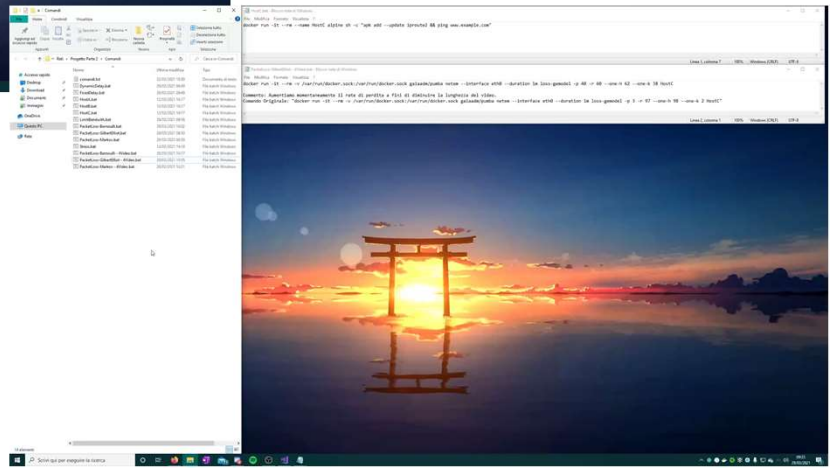
Bernulli: →



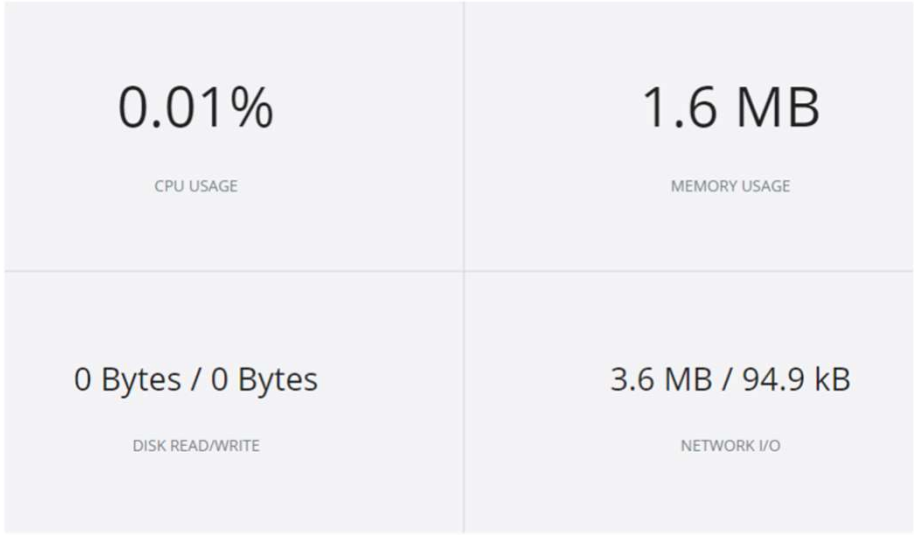
← Markov:



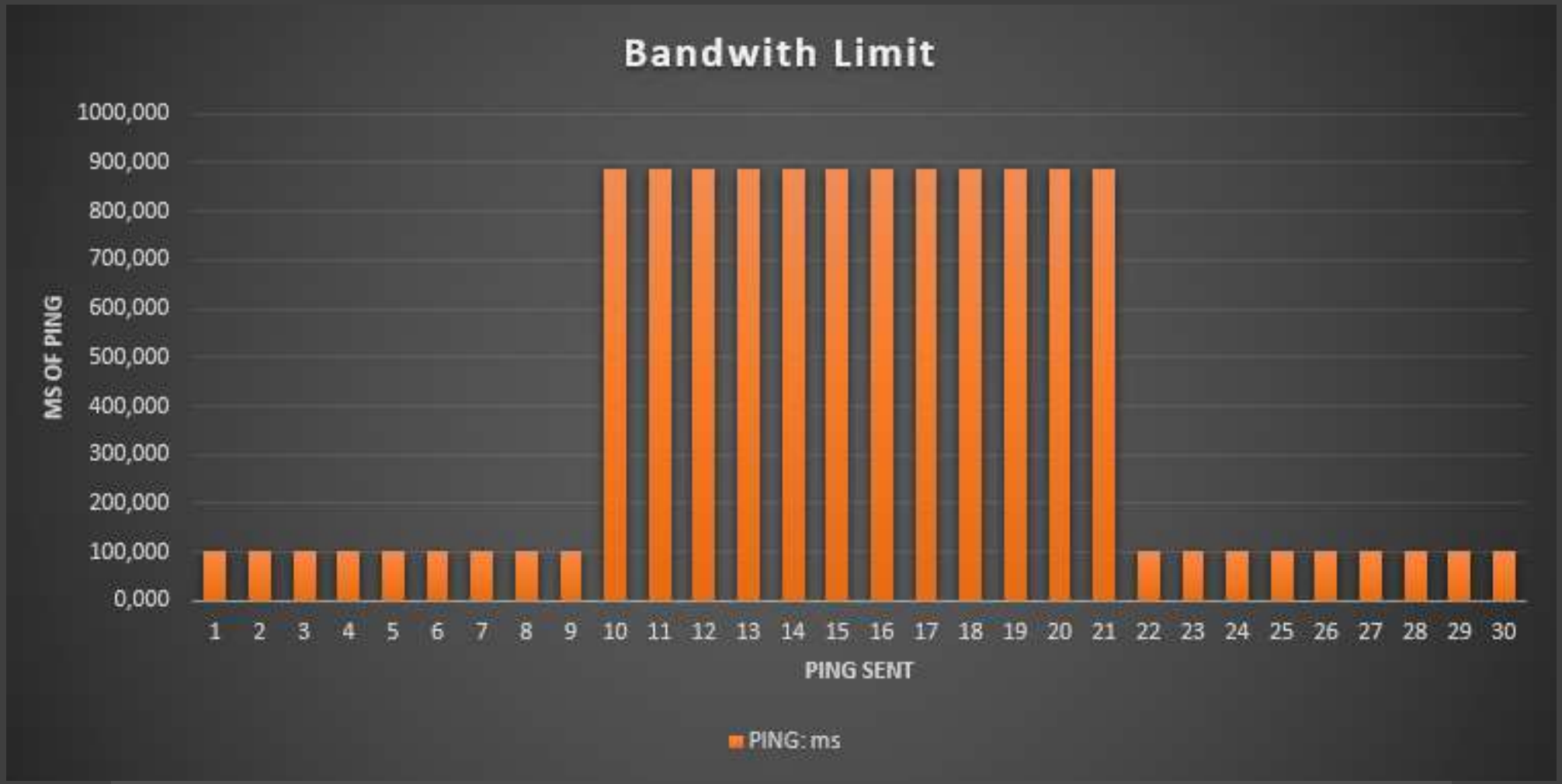
Gilbert-Elliot: →



MONITORAGGIO RETE CON BANDA LIMITATA



- In questo caso abbiamo effettuato un monitoraggio del ping dall'HostA verso un server chiamato www.example.com.
- Utilizzando Pumba abbiamo simulato una riduzione di banda, portandola ad 1 Kbit/s
- Abbiamo monitorato le risorse hardware dell'HostA e non abbiamo notato cambiamenti nell'utilizzo delle risorse.



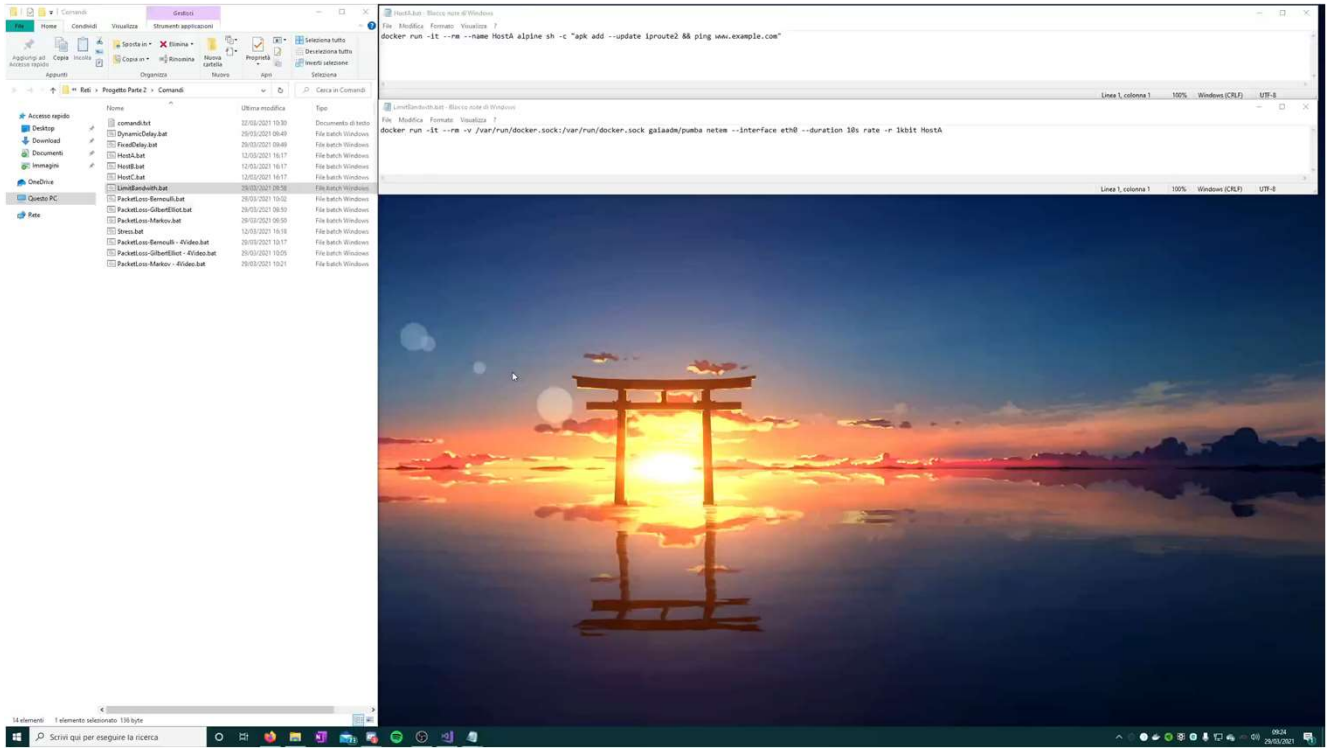
Dal grafico è visibile il comportamento del ritardo a causa della limitazione di banda a 1kbit per secondo.

Comandi utilizzati

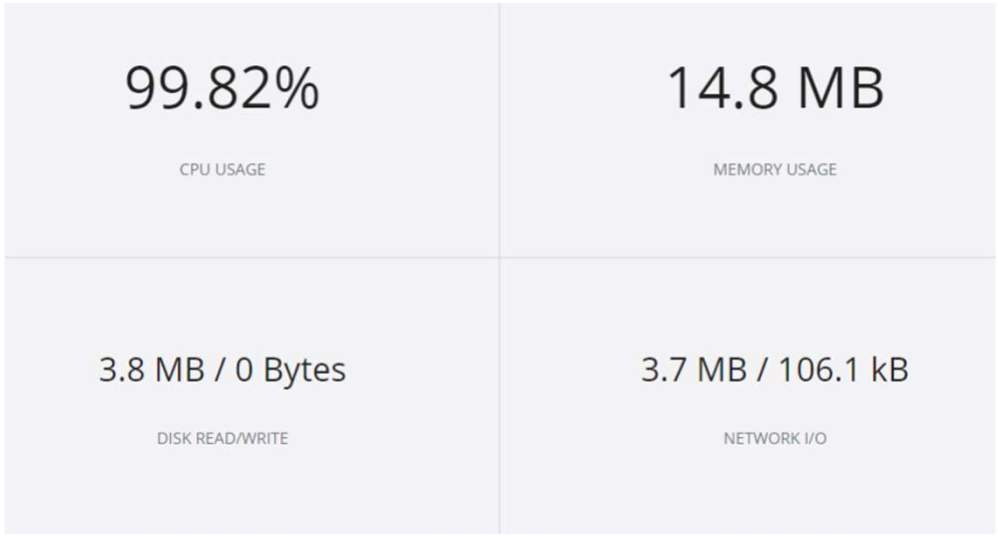
Inserimento limitazione di banda:

```
docker run -it --rm -v
/var/run/docker.sock:/var/run/docker.sock
gaiaadm/pumba netem --interface eth0 --
duration 60s rate -r 1kbit HostA
```

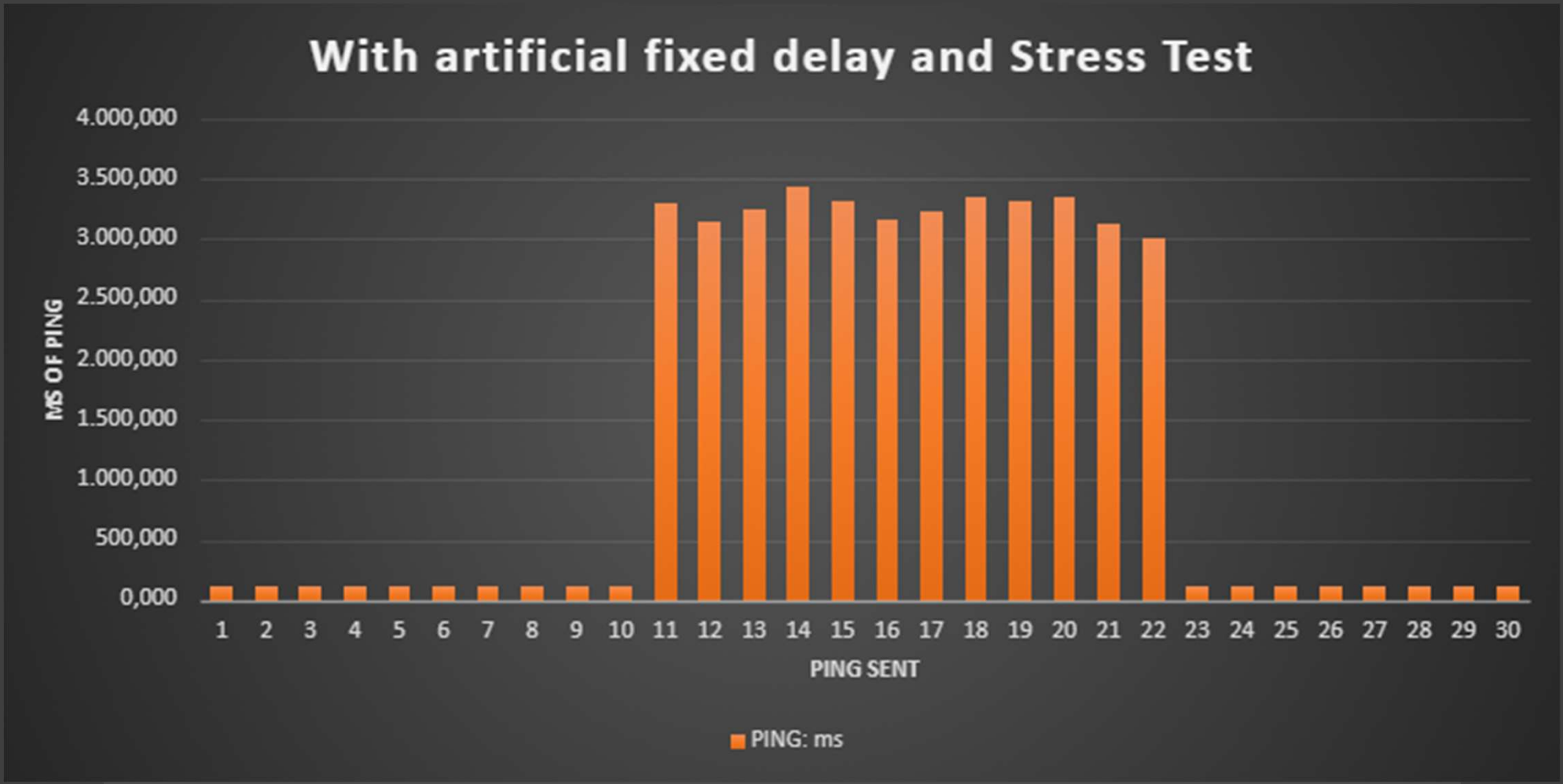
- **Interface eth0**: imposta l'ethernet di default del dispositivo in cui inserire il ritardo.
- **Duration 60s**: indica la durata dello script.
- **Rate –r 1kbit:** imposta il limite massimo di banda utilizzabile dall'host.
- **HostA**: indica l'host in cui è necessario limitare la banda.



MONITORAGGIO RETE CON STRESS TEST E RITARDO FISSO



- In questo caso abbiamo effettuato un monitoraggio del ping dall'HostA verso un server chiamato www.example.com.
- Utilizzando Pumba abbiamo stressato il container aumentando l'utilizzo della CPU e della memoria e abbiamo aggiunto parallelamente un ritardo fisso alla rete.
- Abbiamo monitorato le risorse hardware dell'HostA.



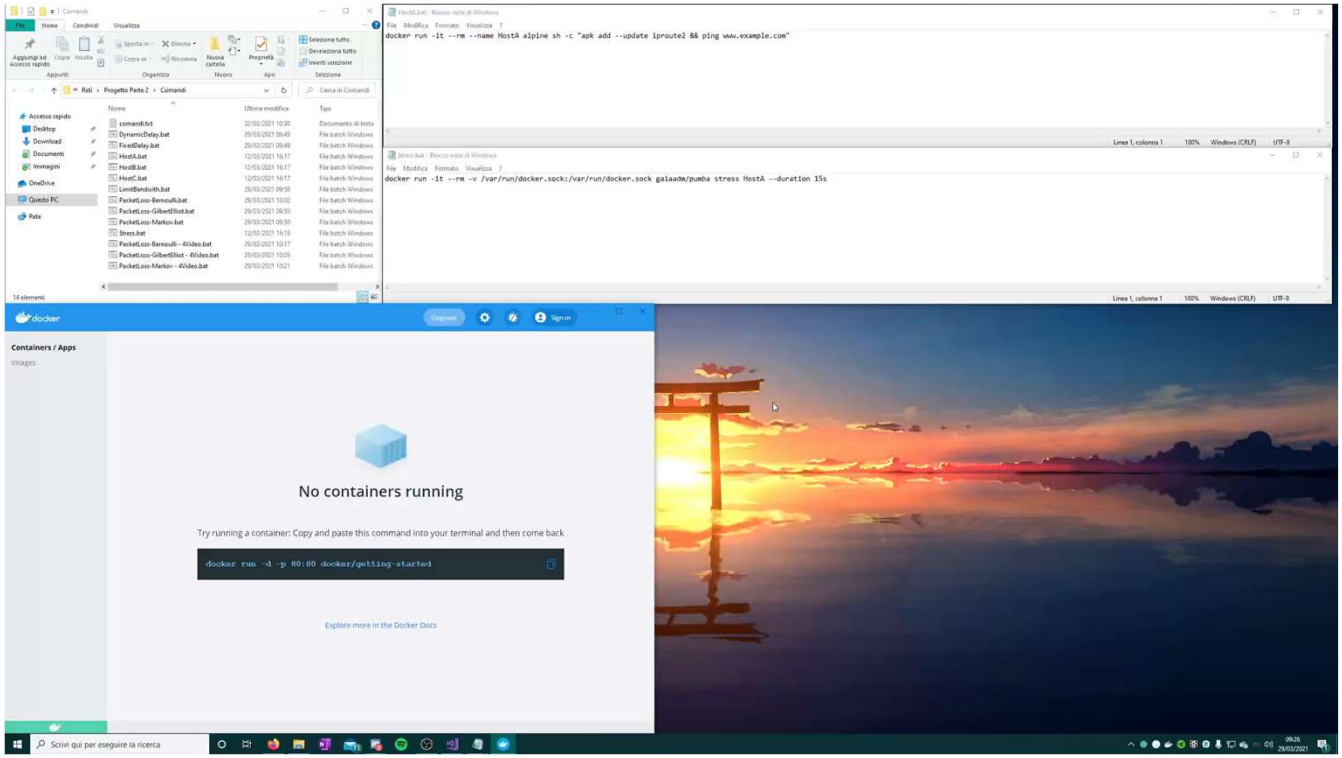
Dal grafico è visibile il comportamento del container durante uno stress test.
In questo caso abbiamo attivato sia lo stress test che il ritardo fisso di 3000 ms. Da questa immagine è quindi intuibile che quando la cpu è sotto stress il ritardo da fisso può diventare variabile 3000ms + massimo circa 300ms

Comandi utilizzati

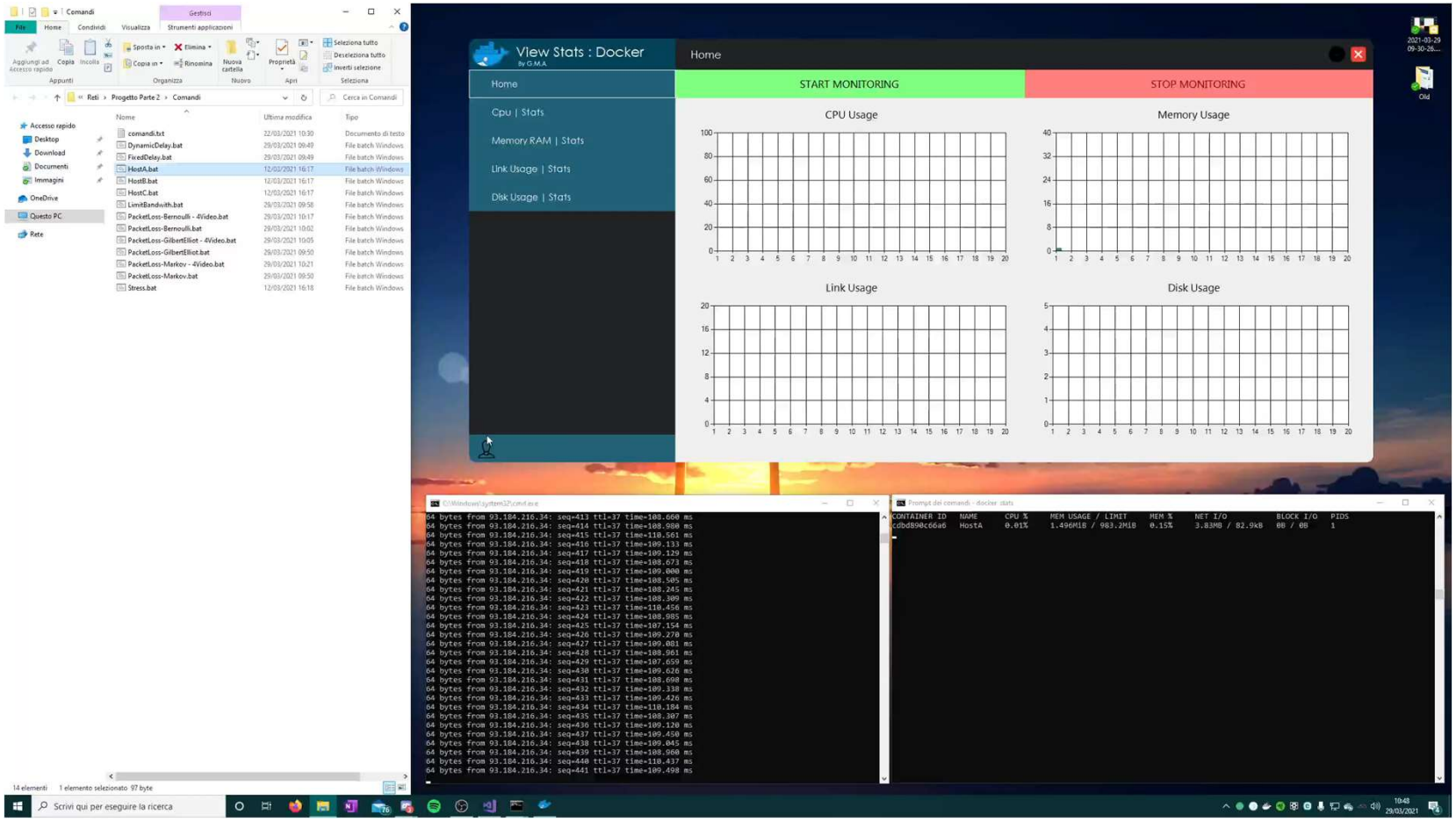
Inserimento stress test:

```
docker run -it --rm -v
/var/run/docker.sock:/var/run/docker.sock
gaiaadm/pumba stress HostA --duration
20s
```

- **Duration 20s**: indica la durata dello script.
- **Stress:** indica l'attivazione dello stress test (CPU e RAM) sull'host indicato.
- **HostA**: indica l'host in cui è necessario effettuare lo stress test.



Applicazione per monitoraggio delle Risorse



Conclusione

Abbiamo notato che in tutti i nostri test, l'utilizzo delle risorse rimaneva invariato, attivando sia i comandi per aggiungere ritardi che per inserire la perdita di pacchetti.

L'unico test che ha dato risultati inusuali è stato lo stress test, dove in alcuni casi, attivando sia lo stress test che un ritardo fisso sullo stesso Host portava a il ritardo fisso a diventare variabile. Inoltre, l'utilizzo delle risorse aumentava notevolmente.

Abbiamo sviluppato un'applicazione per monitorare, in real time, l'utilizzo delle risorse tramite dei grafici che mostrano l'andamento della CPU, della RAM, l'uso del disco e il collegamento (link).
Abbiamo sviluppato l'intero applicativo in C# suddividendoci i compiti nella maniera più uniforme possibile.

Fonti

- https://github.com/alexei-led/pumba
- https://www.docker.com/
- https://www.microsoft.com/it-it/software-download/windows10
- https://alexei-led.github.io/post/pumba_docker_netem/