Achieving Linear CPU Scaling in WireGuard with an Efficient Multi-Tunnel Architecture

Mirco Barone
mirco.barone@studenti.polito.it

Federico Parola
federico.parola@polito.it

Fulvio Risso
fulvio.risso@polito.it

Davide Miola
davide.miola@polito.it

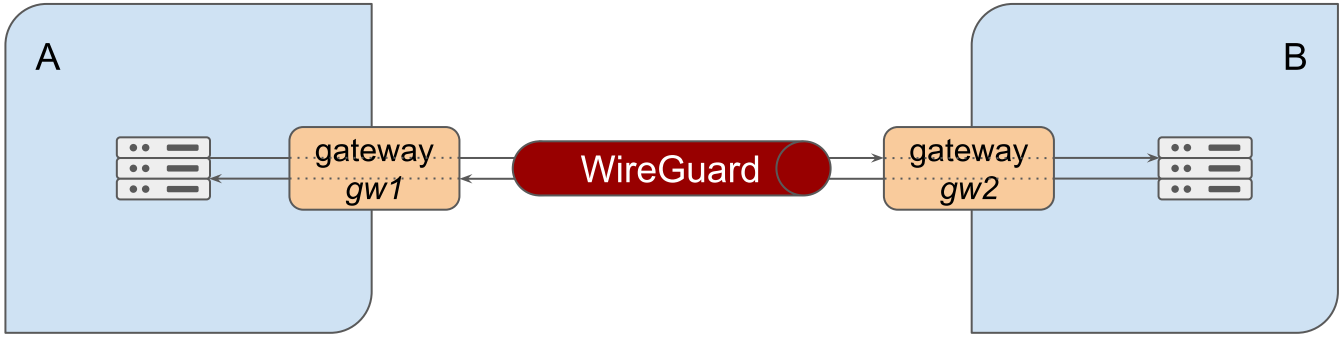
Politecnico di Torino

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

Agenda

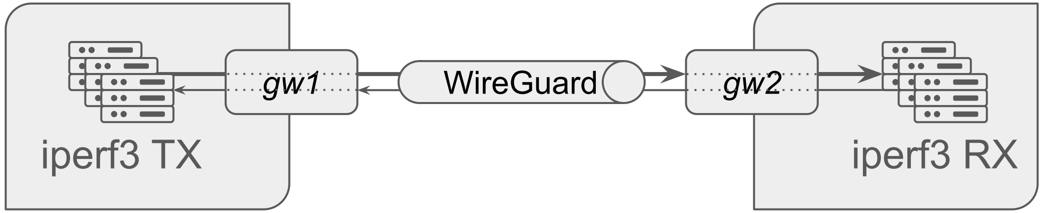
1. Introduction

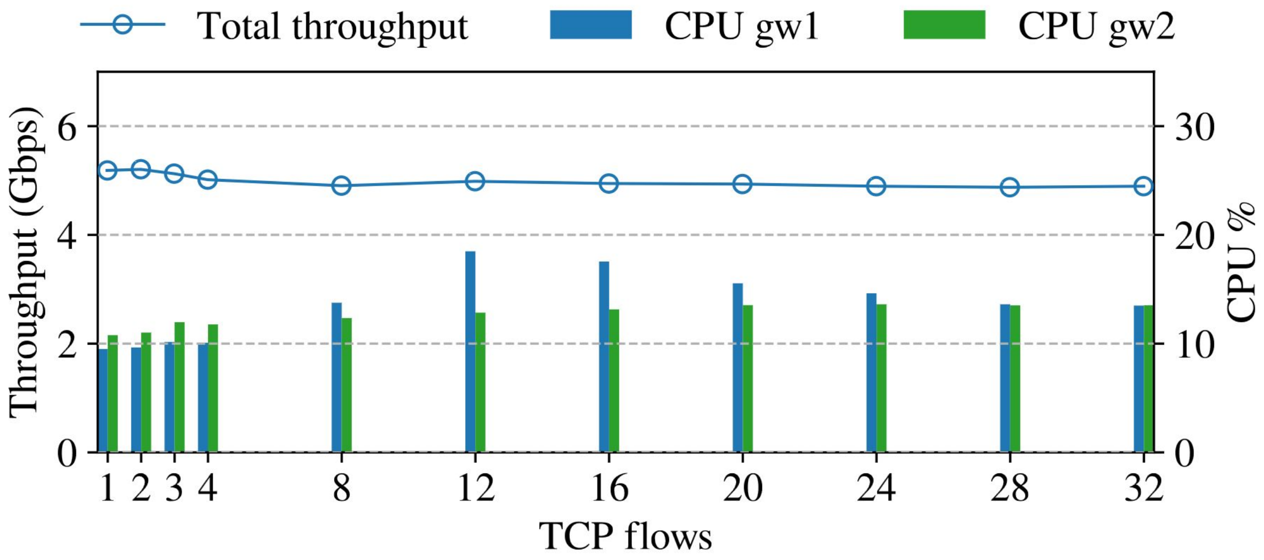
WireGuard is a ***modern***, ***simple*** and ***fast*** secure tunnel technology that is among the most commonly used in end-to-end and site-to-site deployment configurations.



1. Introduction

The problem

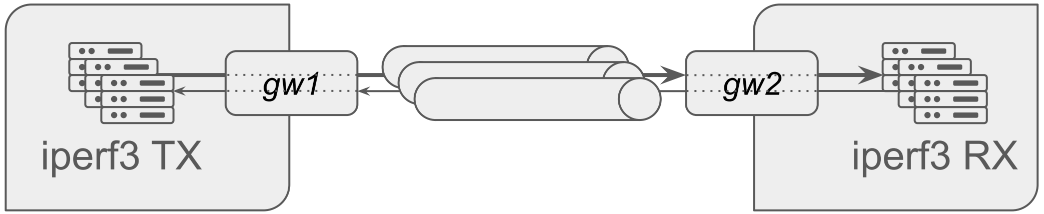


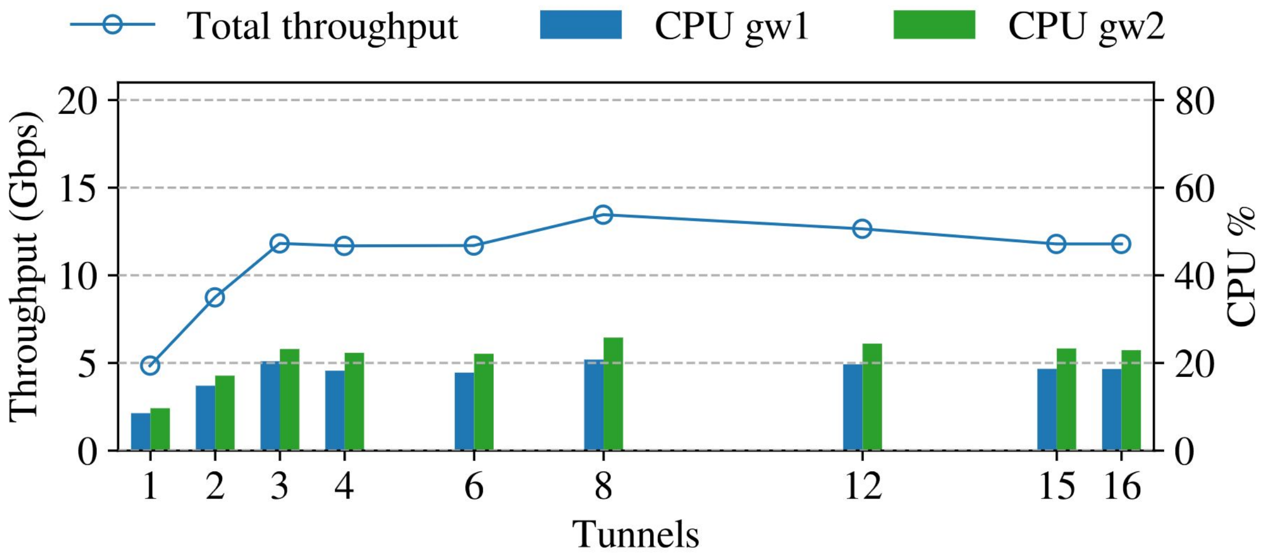


The default implementation is incapable of fully utilizing all available resources, penalizing throughput.

1. Introduction

The problem



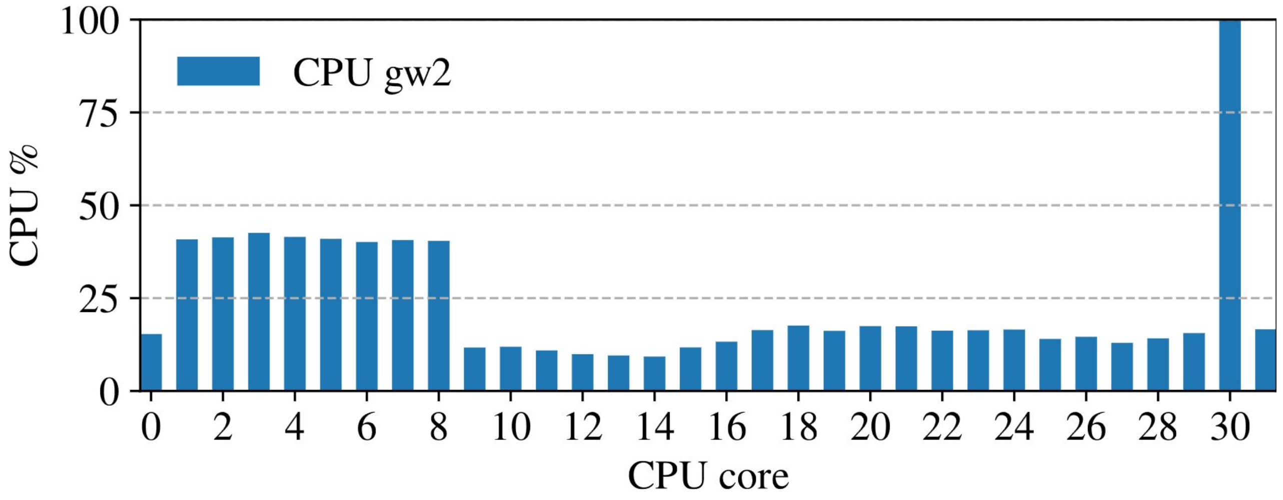


Distributing traffic over multiple tunnels shows suboptimal scaling behavior.

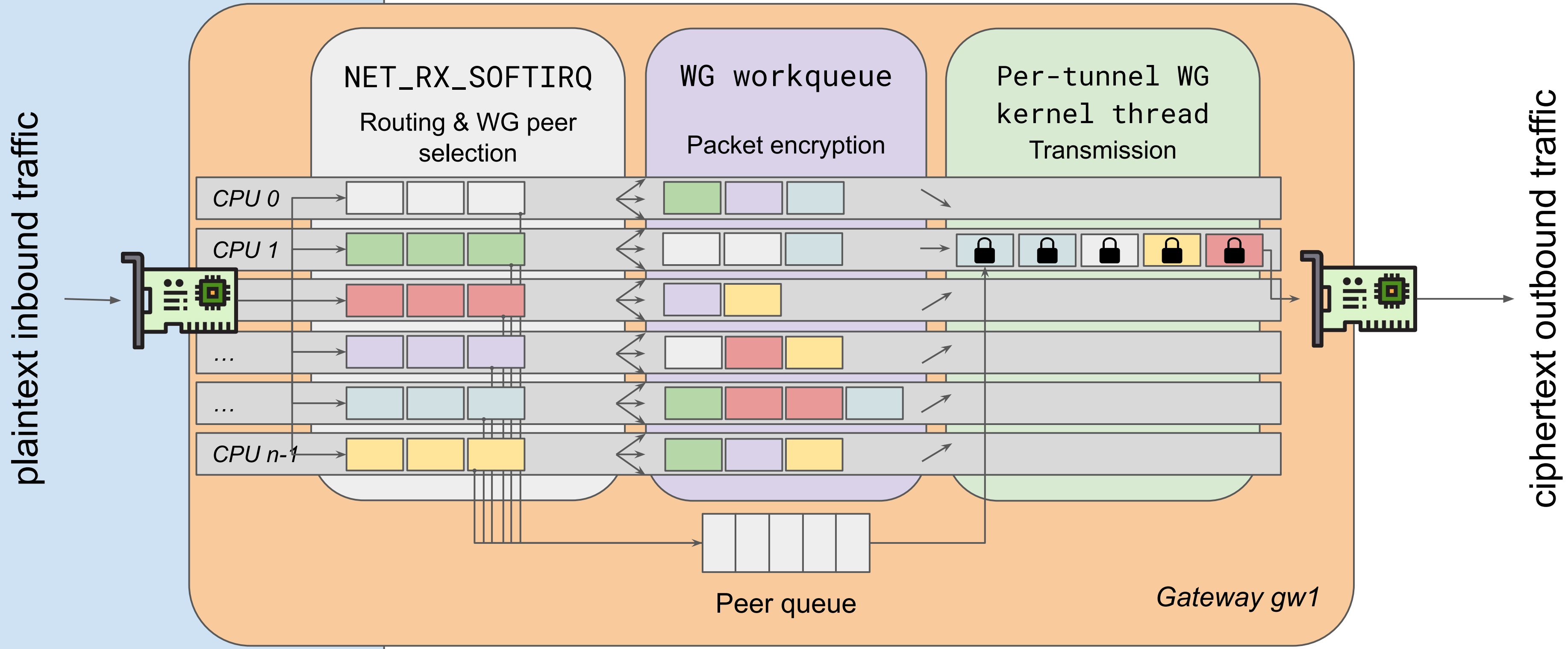
1. Introduction

The problem

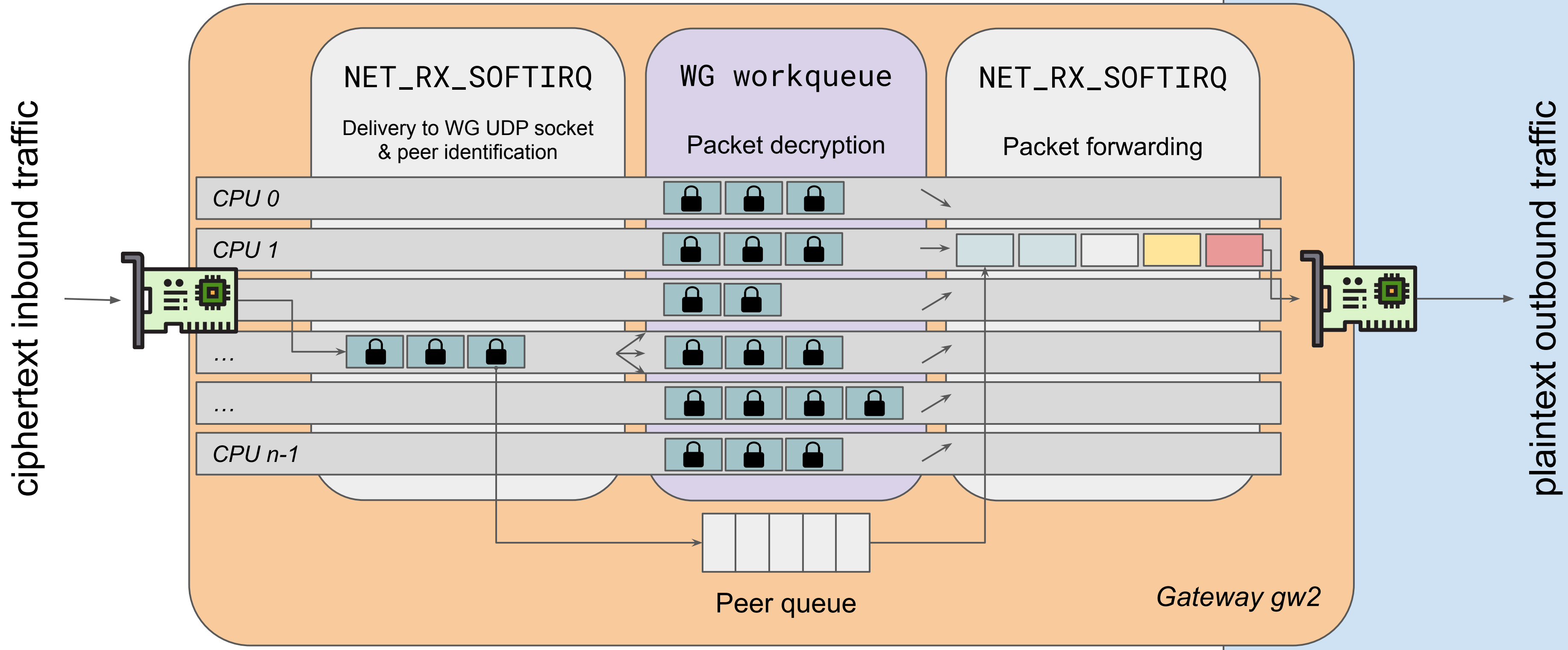
CPU utilization of *gw2* (receiver) with 8 WireGuard tunnels.



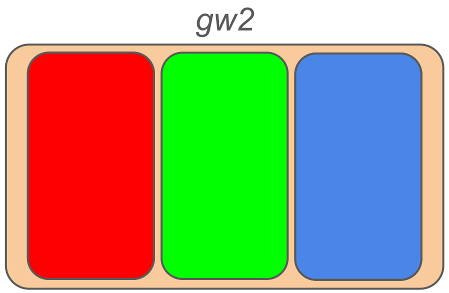
2. Understanding WireGuard's architecture


plaintext inbound traffic
NET_RX_SOFTIRQ
Routing & WG peer selection
WG workqueue
Packet encryption
Per-tunnel WG kernel thread
Transmission
CPU 0
CPU 1
…
…
CPU n-1
Peer queue
Gateway gw1
ciphertext outbound traffic

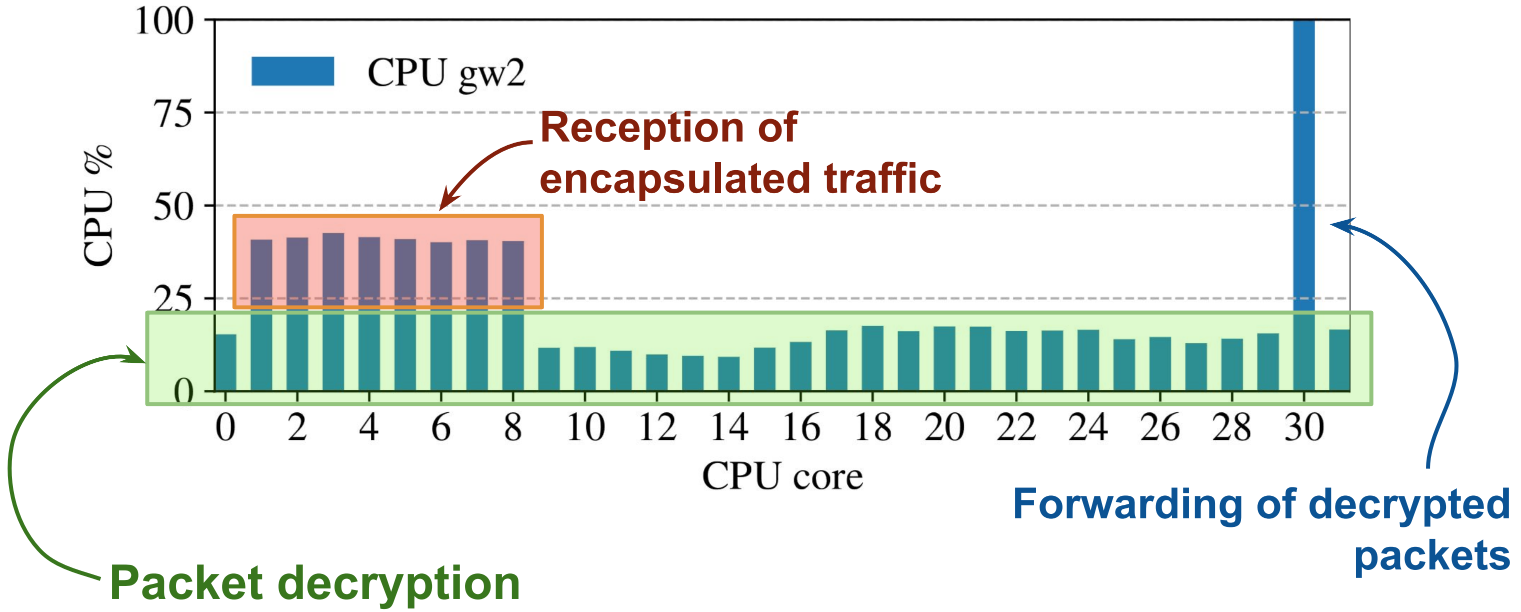
2. Understanding WireGuard's architecture



3. The diagnosis



CPU utilization of *gw2* (receiver) with 8 WireGuard tunnels.



3. The diagnosis

Forwarding NAPI poll is equally as likely to be scheduled on any CPU core, for all the independent tunnels.

$\downarrow$ *random chance*

NAPI polls for 2+ tunnels end up being scheduled on the same CPU core

$\downarrow$

NAPI polls that share a CPU core are slowed down, hence taking more time to process the same amount of packets

$\downarrow$

Subsequently decrypted packets are more likely to find their polling function already running, so they simply get appended to the existing queue.
Forwarding NAPI polls become monolithic

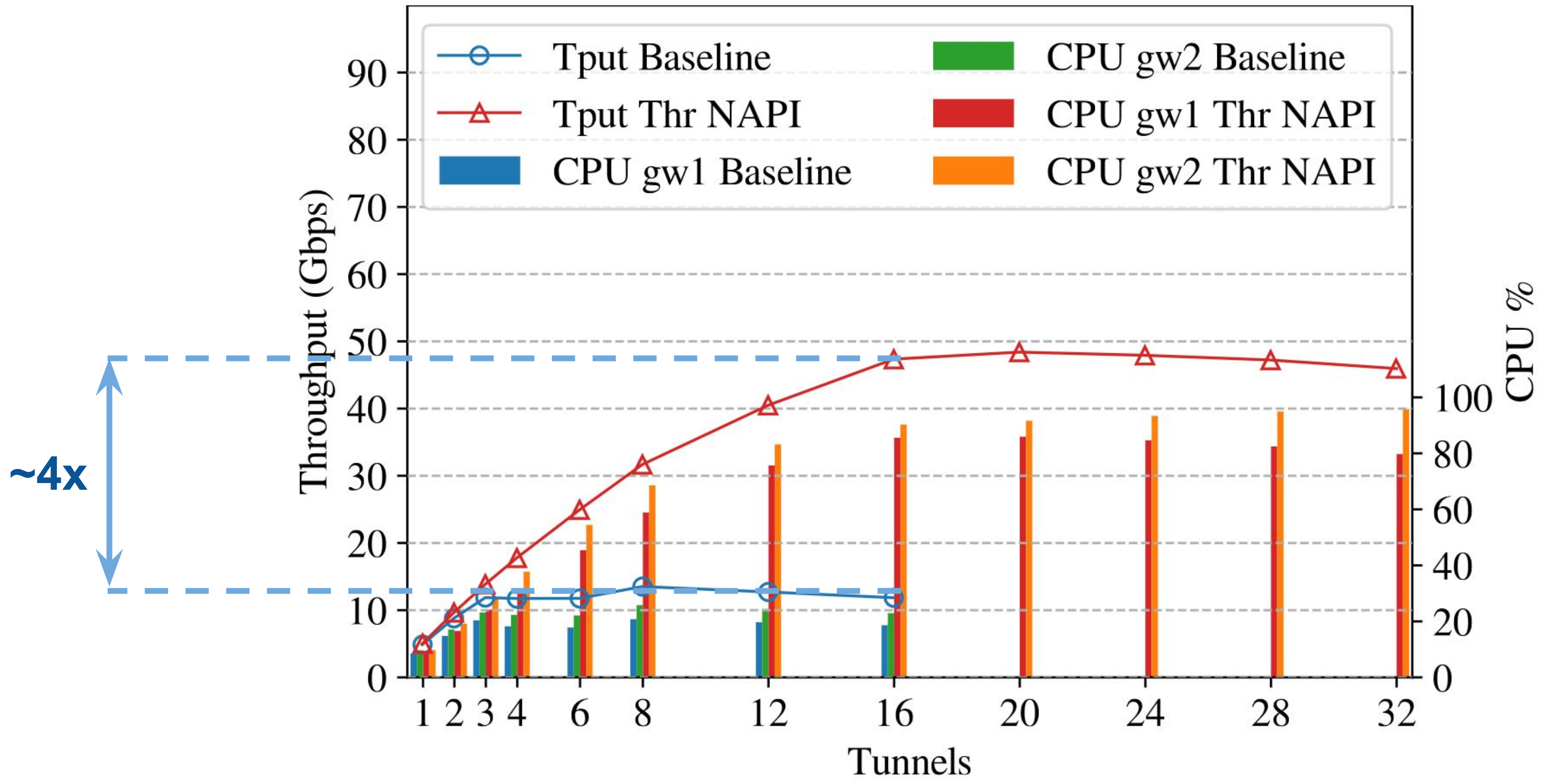
4. Circumventing the problem: *threaded NAPI*

By setting the WireGuard virtual interfaces' `threaded` flag to 1, the packet processing context is moved to a **preemptible** kernel thread.

```
> echo 1 | /sys/class/net/wg{0..n}/threaded
```

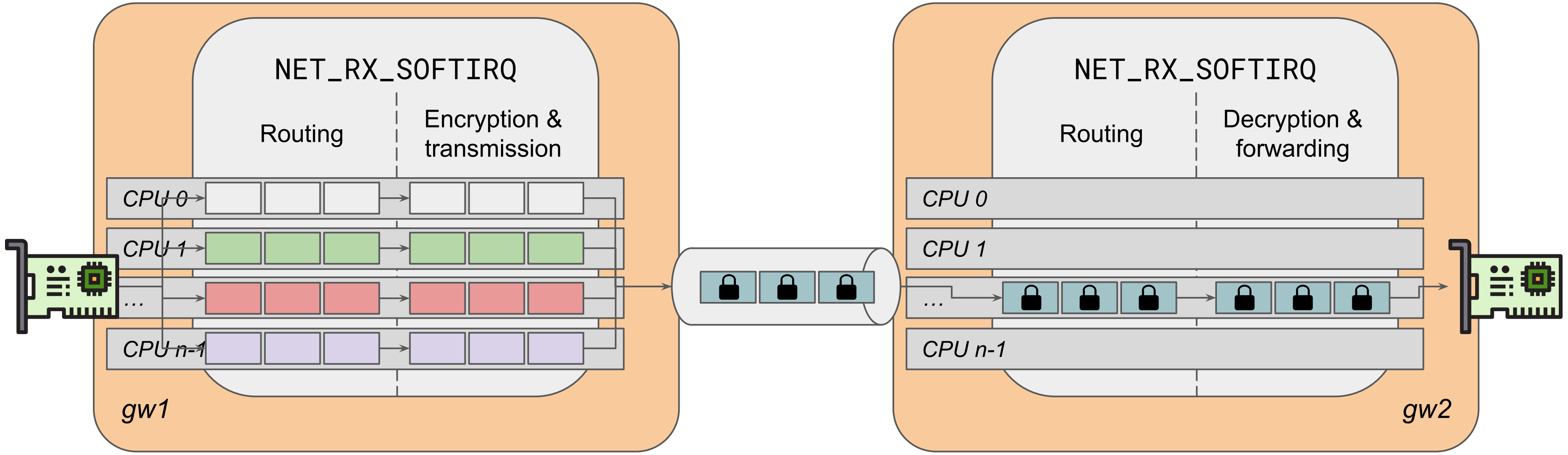
The Linux scheduler dynamically migrates these workers to ensure that no black hole happens.

4. Circumventing the problem: *threaded NAPI*

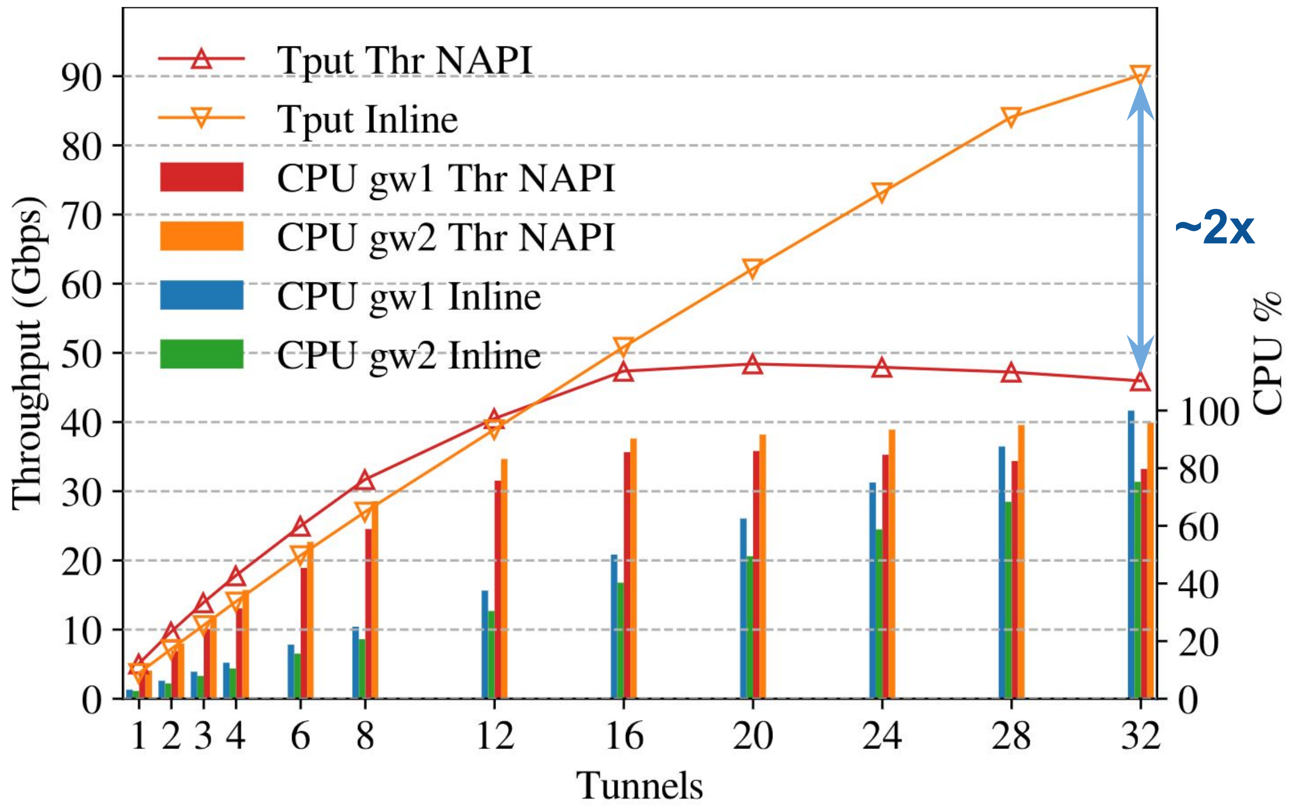


5. The better solution: *WireGuard Inline*

Move all of the pipeline's functions to the softirq that first receives the packets, minimizing cache locality and synchronization issues.



5. The better solution: *WireGuard Inline*



6. Conclusions

The current WireGuard architecture is incapable of sustaining today’s datacenter bandwidths, even when distributing traffic over multiple independent tunnels.

Our *Inline* design solves the problem, at the cost of **lower single-tunnel throughput**.

Thank you!