Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud

Michael Schwarz and Manuel Weber

March 30th, 2017

About this presentation

This talks shows how caches allow to circumvent the isolation of virtual machines

- It is not about software bugs
- The attack vector is due to hardware design
- We demonstrate a robust covert channel on the Amazon cloud
- And we have a really cool live demo at the end

Take aways

Take aways

- Cache-based covert channels are practical and a real threat
- Virtual machines are not a perfect isolation mechanism
- There is no known countermeasure for what we present

Introduction

Whoami

- **Manuel Weber**
- PhD Student, Graz University of Technology
- Interested in IoT, networks and security
- `@WeberOnNetworks`
- ✉ `manuel.weber@tugraz.at`

Whoami

- **Michael Schwarz**
- PhD Student, Graz University of Technology
- Likes to break stuff
- `@misc0110`
- ✉ `michael.schwarz@iaik.tugraz.at`

And the team

The research team

- Clémentine Maurice
- Lukas Giner
- Daniel Gruss
- Carlo Alberto Boano
- Kay Römer
- Stefan Mangard

from Graz University of Technology

Covert channel

What is a covert channel?

- Two programs would like to communicate

Covert channel

What is a covert channel?

- Two programs would like to communicate but are not allowed to do so

Covert channel

What is a covert channel?

- Two programs would like to communicate but are not allowed to do so
    - either because there is no communication channel...

Covert channel

What is a covert channel?

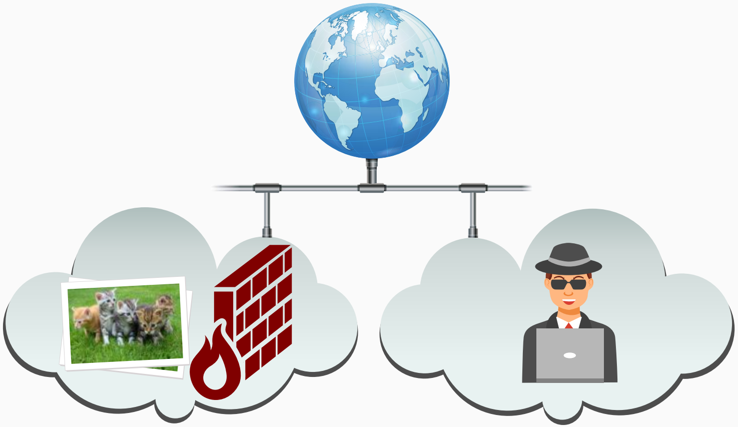
- Two programs would like to communicate but are not allowed to do so
    - either because there is no communication channel...
    - ...or the channels are monitored and programs are stopped on communication attempts

Covert channel

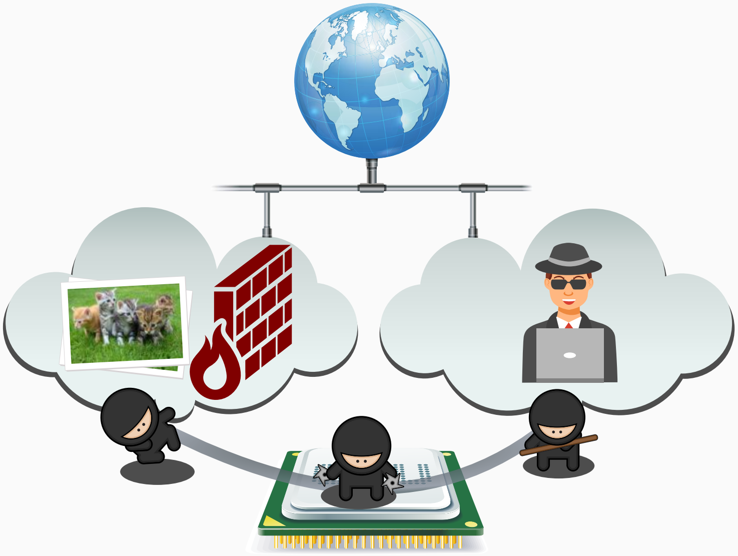
What is a covert channel?

- Two programs would like to communicate but are not allowed to do so
  - either because there is no communication channel...
  - ...or the channels are monitored and programs are stopped on communication attempts
- Use side channels and stay stealthy

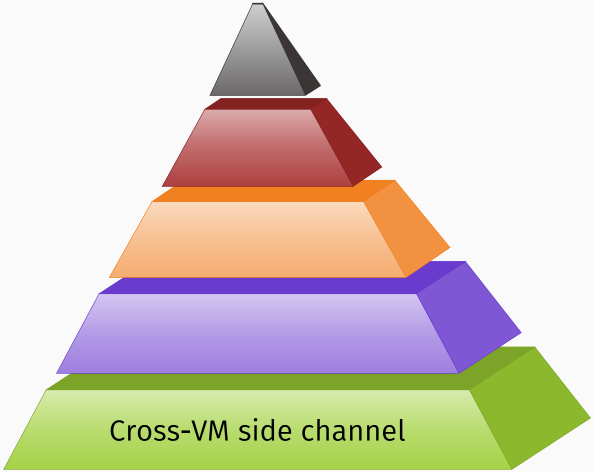
Covert channel



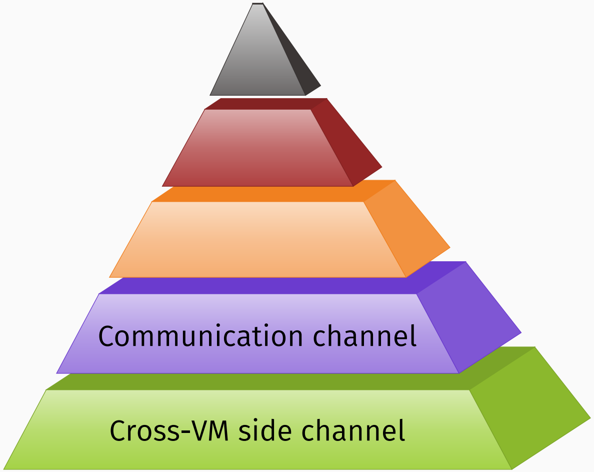
Covert channel



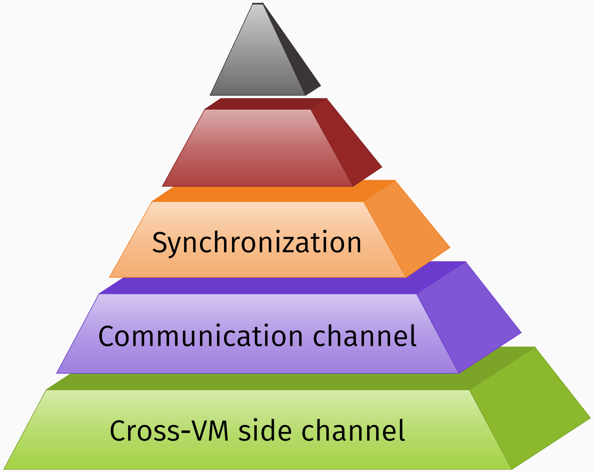
Challenges



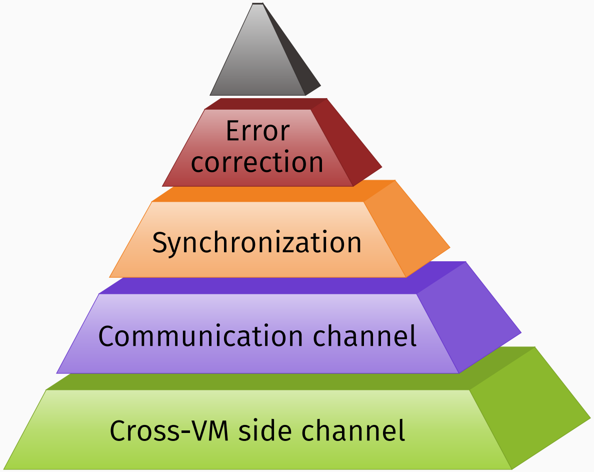
Challenges



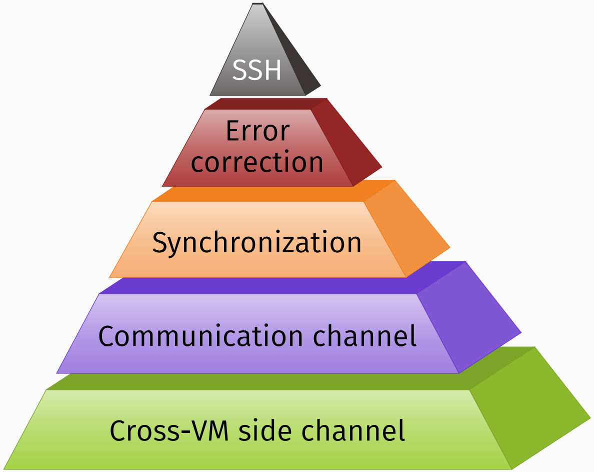
Challenges


Synchronization
Communication channel
Cross-VM side channel

Challenges


Error correction
Synchronization
Communication channel
Cross-VM side channel

Challenges


SSH
Error correction
Synchronization
Communication channel
Cross-VM side channel

CPU Caches

Motivation

- Main memory is slow compared to the CPU

Motivation

- Main memory is slow compared to the CPU
- Caches buffer frequently used data

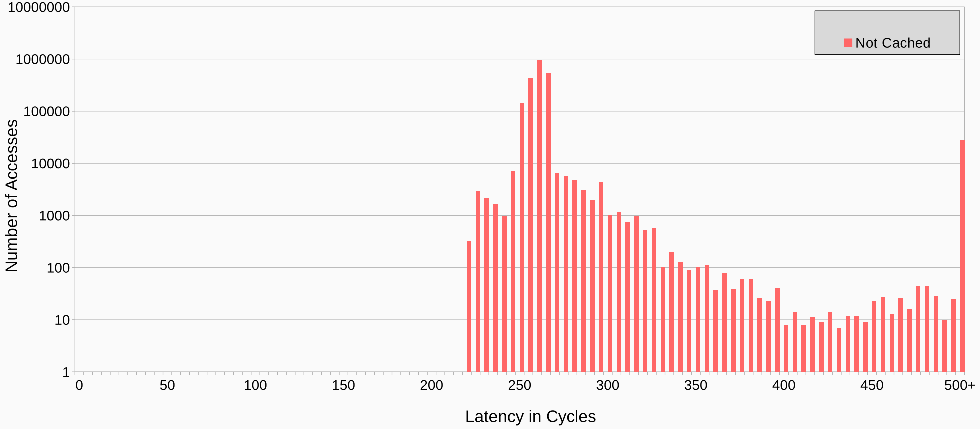
Motivation

- Main memory is slow compared to the CPU
- Caches buffer frequently used data
- Every data access goes through the cache

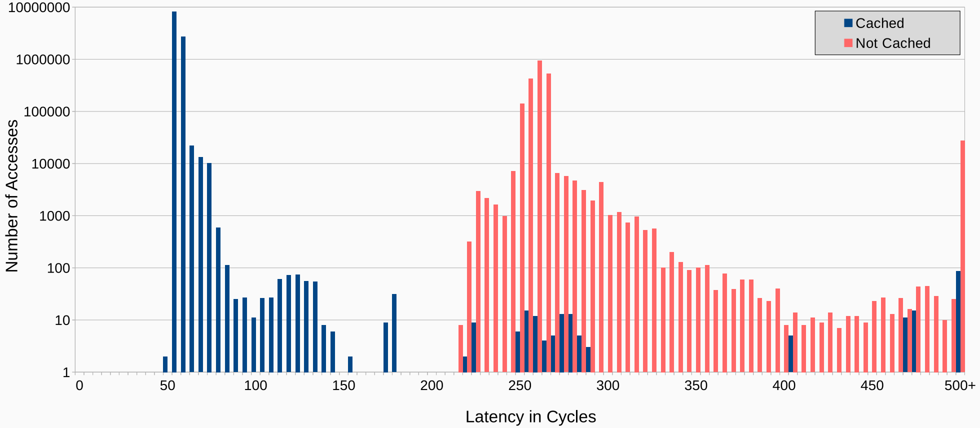
Motivation

- Main memory is slow compared to the CPU
- Caches buffer frequently used data
- Every data access goes through the cache
- Caches are transparent to the OS and the software

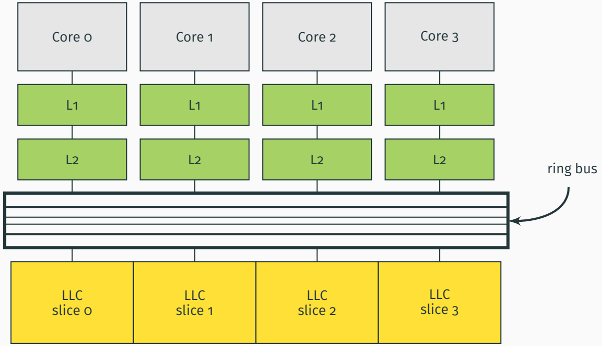
Memory access time



Memory access time

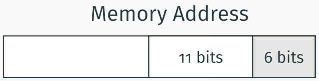


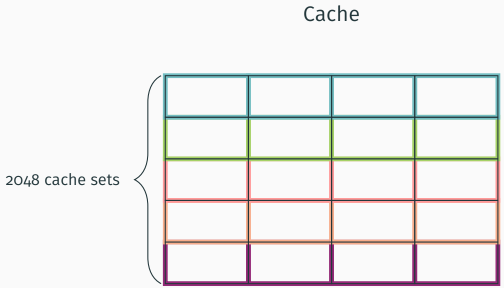
Cache hierarchy



- L1 and L2 are private
- Last-level cache is
    - divided into slices
    - shared across cores
    - inclusive

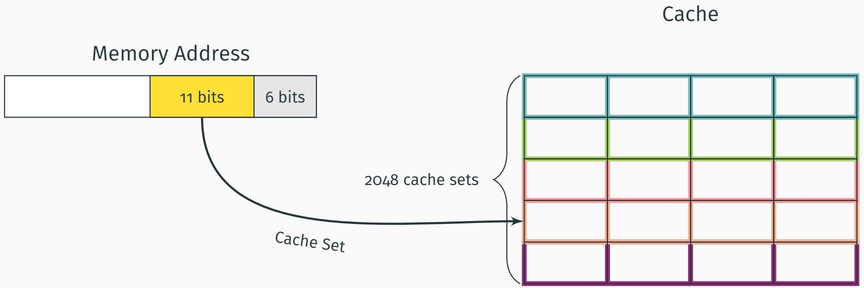
Set-associative Last-level Cache





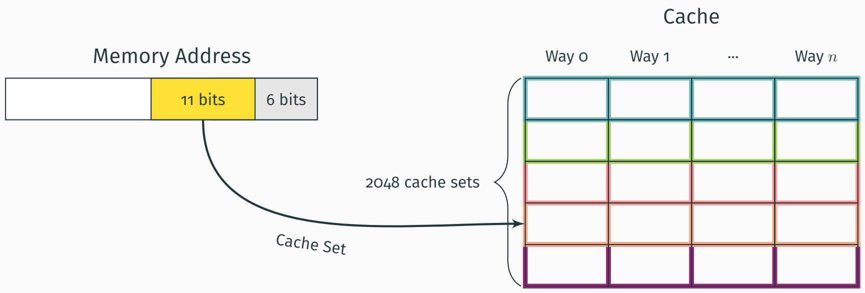
- Location in cache depends on the physical address of data

Set-associative Last-level Cache



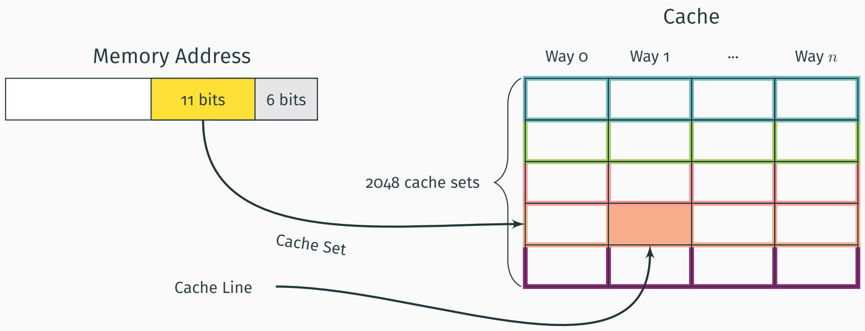
- Location in cache depends on the physical address of data
- Bits 6 to 16 determine the cache set

Set-associative Last-level Cache



- Location in cache depends on the physical address of data
- Bits 6 to 16 determine the cache set
- A cache set has multiple ways to store the data

Set-associative Last-level Cache



- Location in cache depends on the physical address of data
- Bits 6 to 16 determine the cache set
- A cache set has multiple ways to store the data
- A way inside a cache set is a cache line, determined by the cache replacement policy

Prime+Probe

Prime+Probe

Prime+Probe…

Prime+Probe

Prime+Probe...

- exploits the timing difference when accessing...

Prime+Probe

Prime+Probe...

- exploits the timing difference when accessing...
    - cached data (fast)

Prime+Probe

Prime+Probe...

- exploits the timing difference when accessing...
    - cached data (fast)
    - uncached data (slow)

Prime+Probe

Prime+Probe...

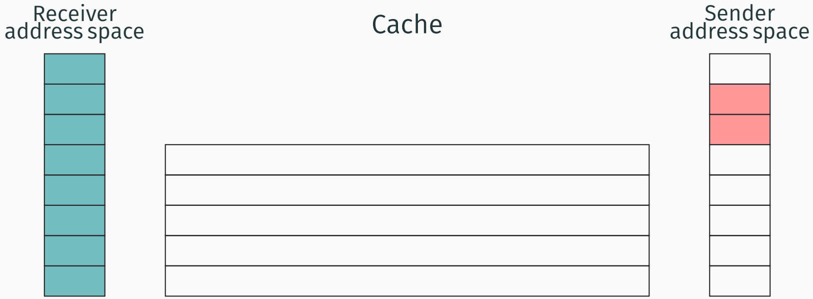
- exploits the timing difference when accessing...
    - cached data (fast)
    - uncached data (slow)
- is applied to one cache set

Prime+Probe

Prime+Probe...

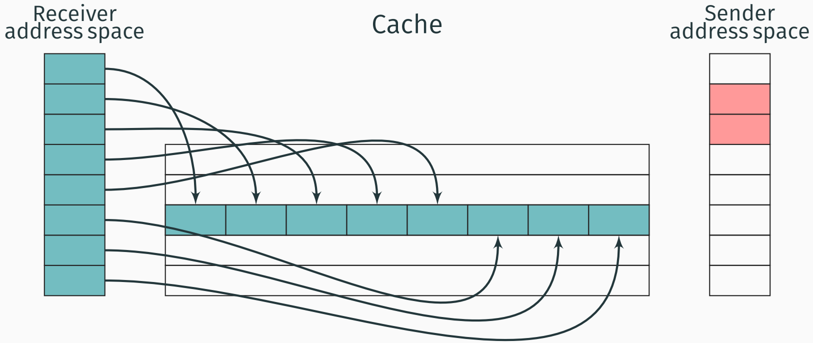
- exploits the timing difference when accessing...
    - cached data (fast)
    - uncached data (slow)
- is applied to one cache set
- works across CPU cores as the last-level cache is shared

Prime+Probe



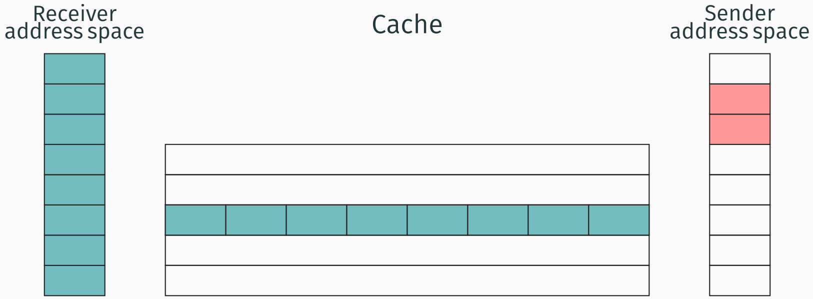
Step 0: Receiver fills the cache (prime)

Prime+Probe



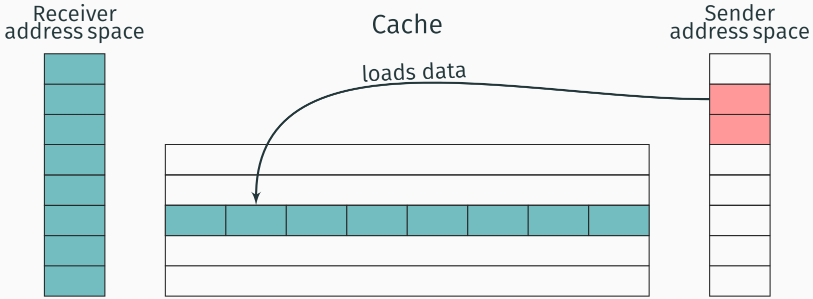
Step 0: Receiver fills the cache (prime)

Prime+Probe



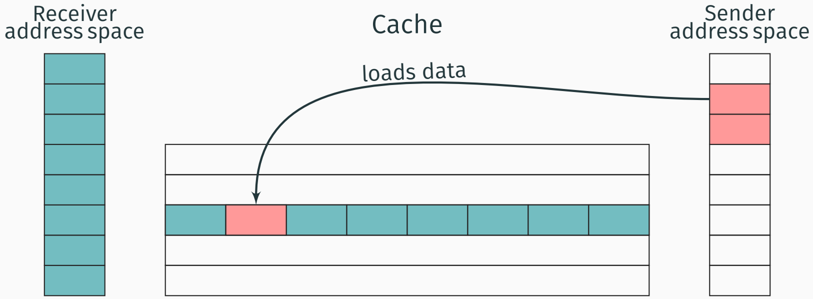
Step 0: Receiver fills the cache (prime)

Prime+Probe



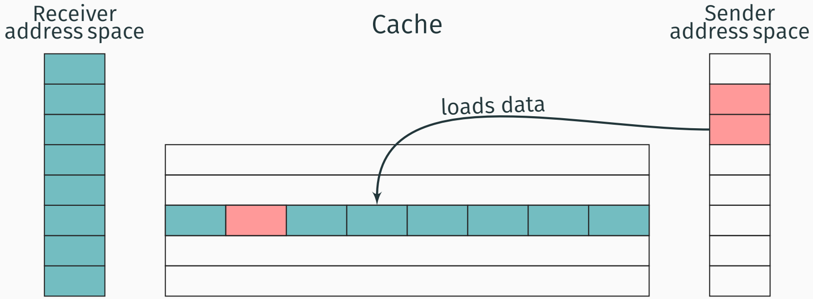
Step 0: Receiver fills the cache (prime)
Step 1: Sender evicts cache lines by accessing own data

Prime+Probe



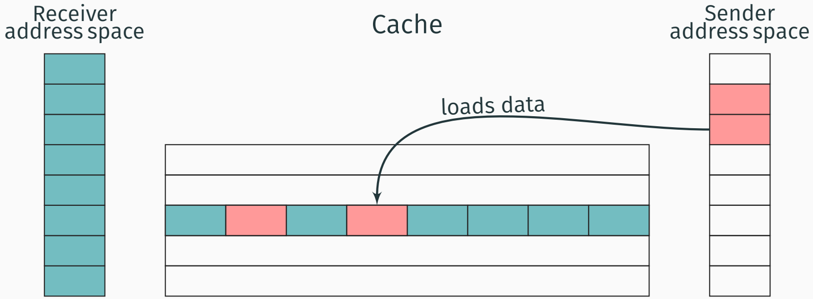
Step 0: Receiver fills the cache (prime)
Step 1: Sender evicts cache lines by accessing own data

Prime+Probe



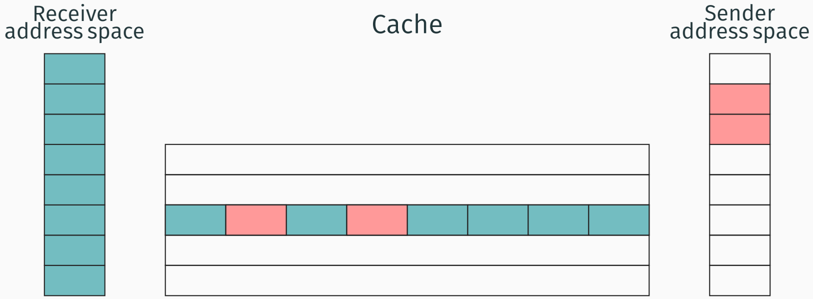
Step 0: Receiver fills the cache (prime)
Step 1: Sender evicts cache lines by accessing own data

Prime+Probe



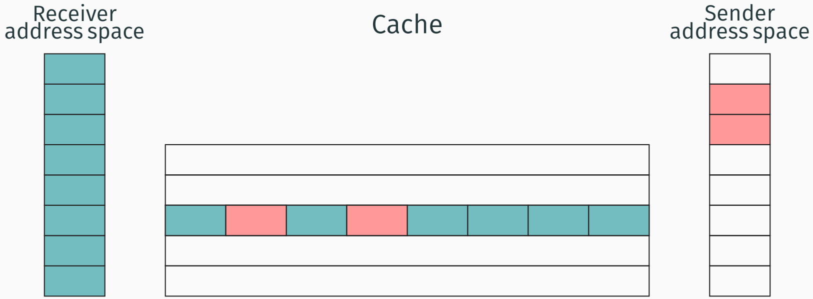
Step 0: Receiver fills the cache (prime)
Step 1: Sender evicts cache lines by accessing own data

Prime+Probe



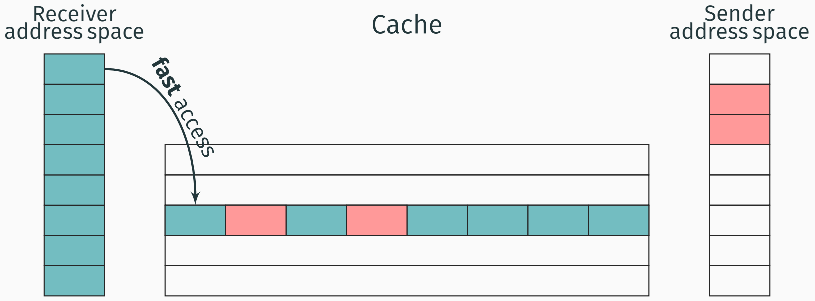
Step 0: Receiver fills the cache (prime)
Step 1: Sender evicts cache lines by accessing own data

Prime+Probe



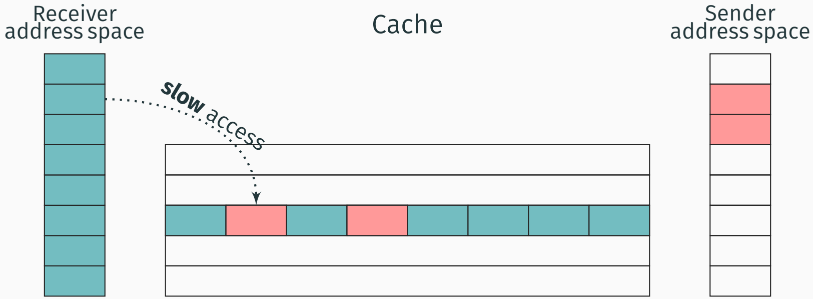
Step 0: Receiver fills the cache (prime)
Step 1: Sender evicts cache lines by accessing own data
Step 2: Receiver probes data to determine if the set was accessed

Prime+Probe



Step 0: Receiver fills the cache (prime)
Step 1: Sender evicts cache lines by accessing own data
Step 2: Receiver probes data to determine if the set was accessed

Prime+Probe



Step 0: Receiver fills the cache (prime)
Step 1: Sender evicts cache lines by accessing own data
Step 2: Receiver probes data to determine if the set was accessed

Building a robust covert channel

The goal

We want to build a covert channel which...

The goal

We want to build a covert channel which...

- works across virtual machines

The goal

We want to build a covert channel which...

- works across virtual machines
- runs on the Amazon cloud

The goal

We want to build a covert channel which...

- works across virtual machines
- runs on the Amazon cloud
- is fast (*i.e.*, multiple kB/s)

The goal

We want to build a covert channel which...

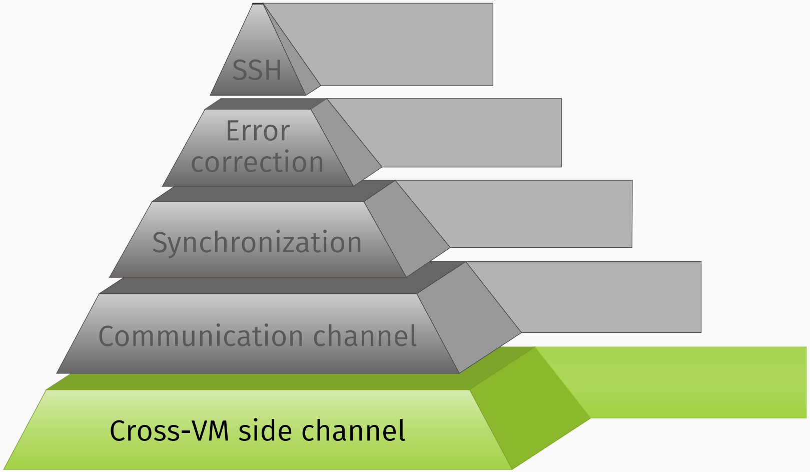
- works across virtual machines
- runs on the Amazon cloud
- is fast (*i.e.*, multiple kB/s)
- is free of transmission errors

The goal

We want to build a covert channel which...

- works across virtual machines
- runs on the Amazon cloud
- is fast (*i.e.*, multiple kB/s)
- is free of transmission errors
- is robust against system noise

Challenges


SSH
Error correction
Synchronization
Communication channel
Cross-VM side channel

Cross-VM side channel

We require a side channel which works across virtual machines

Cross-VM side channel

We require a side channel which works across virtual machines

- We do not want to rely on software bugs, they can be patched

Cross-VM side channel

We require a side channel which works across virtual machines

- We do not want to rely on software bugs, they can be patched
- We want to exploit the hardware

Cross-VM side channel

We require a side channel which works across virtual machines

- We do not want to rely on software bugs, they can be patched
- We want to exploit the hardware
- Memory is shared between all virtual machines

Cross-VM side channel

We require a side channel which works across virtual machines

- We do not want to rely on software bugs, they can be patched
- We want to exploit the hardware
- Memory is shared between all virtual machines
    - DRAM

Cross-VM side channel

We require a side channel which works across virtual machines

- We do not want to rely on software bugs, they can be patched
- We want to exploit the hardware
- Memory is shared between all virtual machines
    - DRAM $\rightarrow$ covert channel (Schwarz and Fogh 2016, BlackHat Europe)

Cross-VM side channel

We require a side channel which works across virtual machines

- We do not want to rely on software bugs, they can be patched
- We want to exploit the hardware
- Memory is shared between all virtual machines
    - DRAM $\rightarrow$ covert channel (Schwarz and Fogh 2016, BlackHat Europe)
    - Cache

Cross-VM side channel

We require a side channel which works across virtual machines

- We do not want to rely on software bugs, they can be patched
- We want to exploit the hardware
- Memory is shared between all virtual machines
    - DRAM $\rightarrow$ covert channel (Schwarz and Fogh 2016, BlackHat Europe)
    - Cache $\rightarrow$ this talk!

Cross-VM side channel

We can use Prime+Probe for the side channel

- Prime+Probe works with the last-level cache

Cross-VM side channel

We can use Prime+Probe for the side channel

- Prime+Probe works with the last-level cache
- The last-level cache is shared among all CPU cores

Cross-VM side channel

We can use Prime+Probe for the side channel

- Prime+Probe works with the last-level cache
- The last-level cache is shared among all CPU cores
- No requirement for any form of shared memory

Cross-VM side channel

We can use Prime+Probe for the side channel

- Prime+Probe works with the last-level cache
- The last-level cache is shared among all CPU cores
- No requirement for any form of shared memory
- We just need to build eviction sets and negotiate the used cache sets

Cross-VM side channel

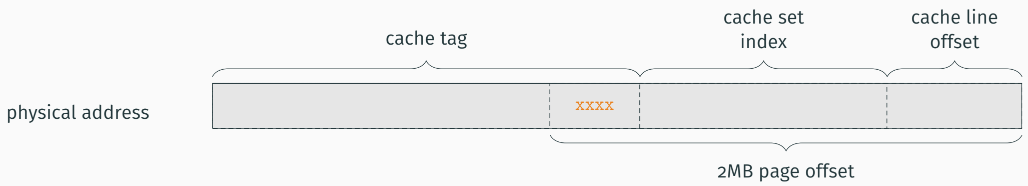
- We need a set of addresses in the same cache set and same slice

Cross-VM side channel

- We need a set of addresses in the same cache set and same slice
- Problem: slice number depends on all bits of the physical address

Cross-VM side channel

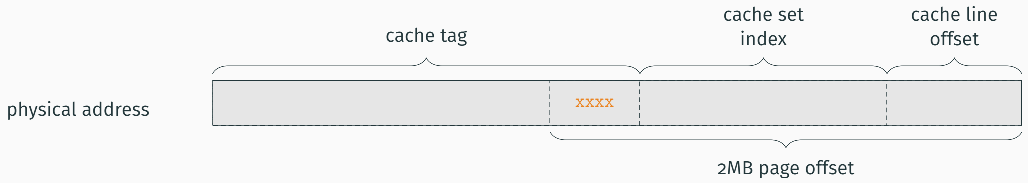
- We need a set of addresses in the same cache set and same slice
- Problem: slice number depends on all bits of the physical address



- We can build a set of addresses in the same cache set and same slice...

Cross-VM side channel

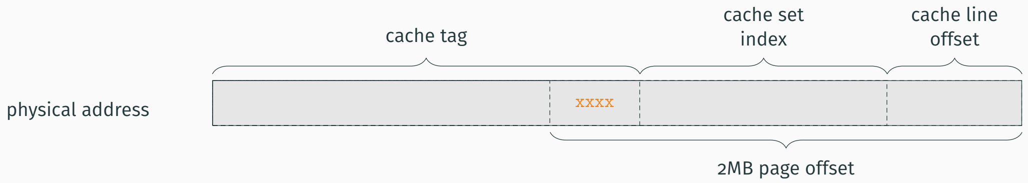
- We need a set of addresses in the same cache set and same slice
- Problem: slice number depends on all bits of the physical address



- We can build a set of addresses in the same cache set and same slice...
- ...without knowing which slice

Cross-VM side channel

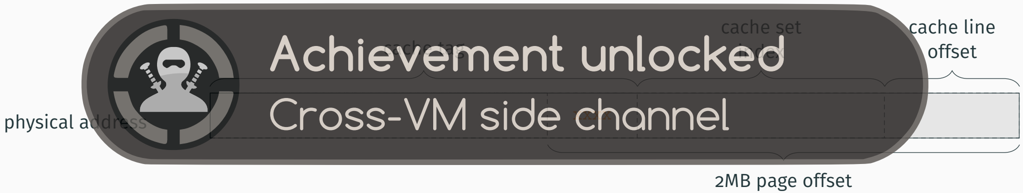
- We need a set of addresses in the same cache set and same slice
- Problem: slice number depends on all bits of the physical address



- We can build a set of addresses in the same cache set and same slice...
- ...without knowing which slice
- And then remove the addresses of the wrong slices afterwards

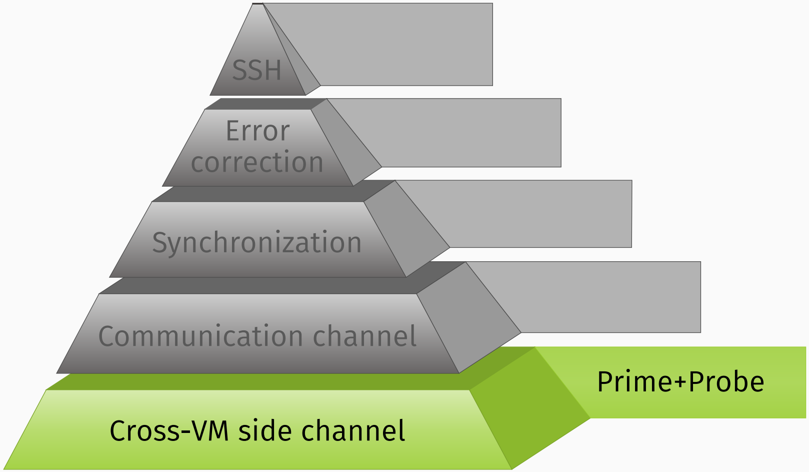
Cross-VM side channel

- We need a set of addresses in the same cache set and same slice
- Problem: slice number depends on all bits of the physical address

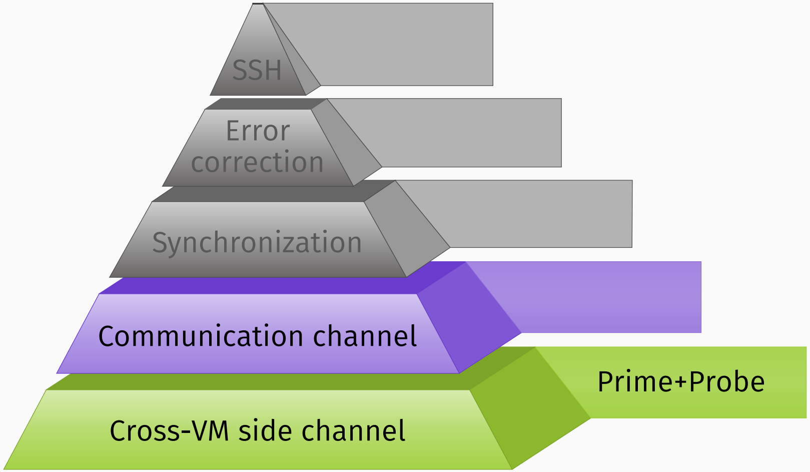


- We can build a set of addresses in the same cache set and same slice...
- ...without knowing which slice
- And then remove the addresses of the wrong slices afterwards

Challenges


SSH
Error correction
Synchronization
Communication channel
Cross-VM side channel
Prime+Probe

Challenges


SSH
Error correction
Synchronization
Communication channel
Cross-VM side channel
Prime+Probe

Communication Channel

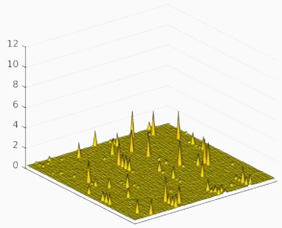
- For a communication, we have to agree on communication channels

Communication Channel

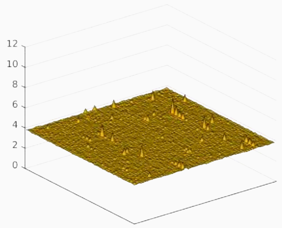
- For a communication, we have to agree on communication channels
- We have to negotiate them dynamically

Communication Channel

- For a communication, we have to agree on communication channels
- We have to negotiate them dynamically
- There is always noise on all cache sets



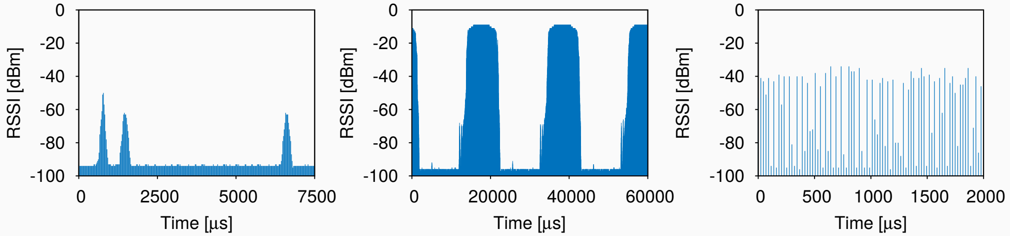
(a) Quiet system



(b) Watching an 1080p video

Communication Channel

Quite similar to a wireless communication channel



(a) Bluetooth **(b)** Microwave **(c)** WiFi

Figure 2: Noise in wireless channels (Boano et al. 2012)

Jamming Agreement

- Idea: »He who shouts loudest will be heard«

Jamming Agreement

- Idea: »He who shouts loudest will be heard«
- One party generates a lot of “noise” on the channel

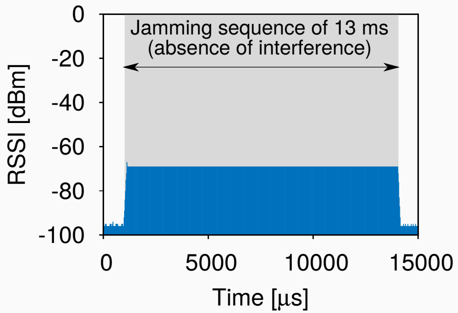
Jamming Agreement

- Idea: »He who shouts loudest will be heard«
- One party generates a lot of “noise” on the channel
- The other party monitors the channels

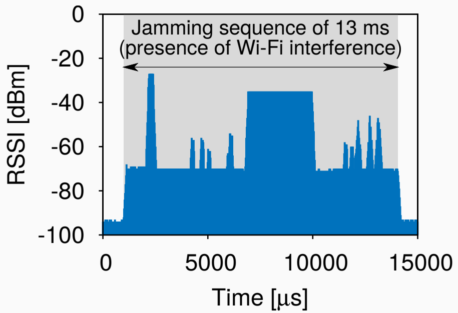
Jamming Agreement

- Idea: »He who shouts loudest will be heard«
- One party generates a lot of “noise” on the channel
- The other party monitors the channels
- Correct channel if the noise level never falls below a certain value

Jamming Agreement



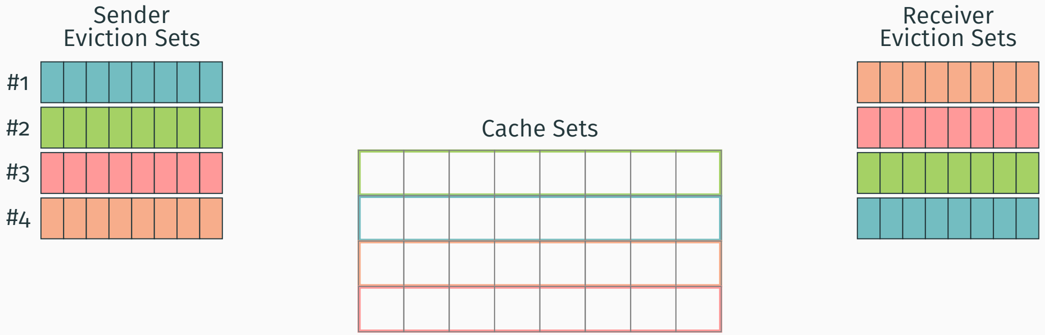
(a) No interference



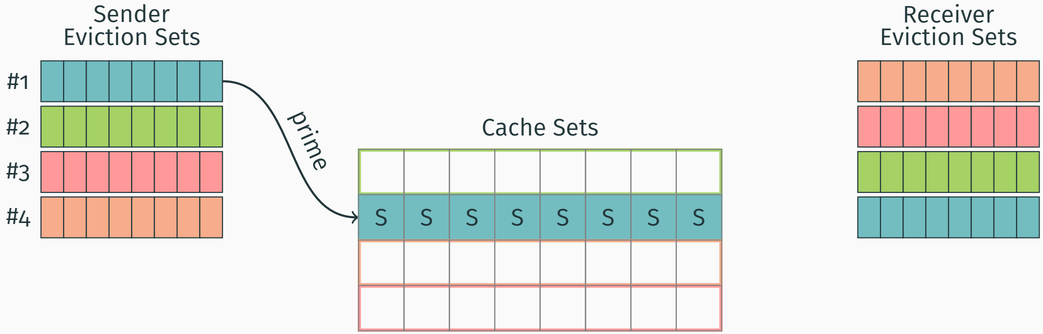
(b) WiFi interference

Figure 3: Jamming agreement in wireless channels (Boano et al. 2012)

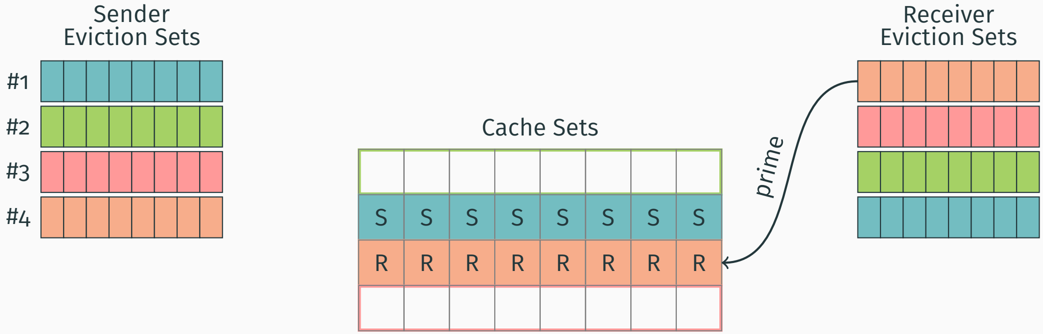
Jamming Agreement



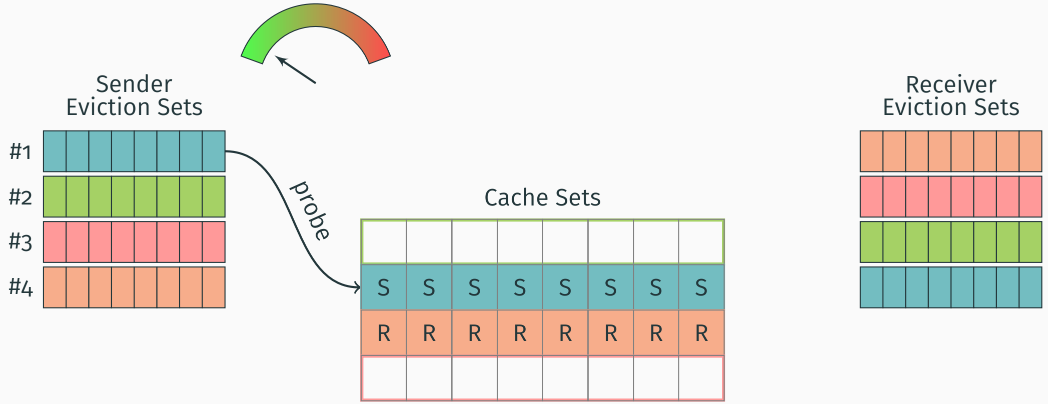
Jamming Agreement



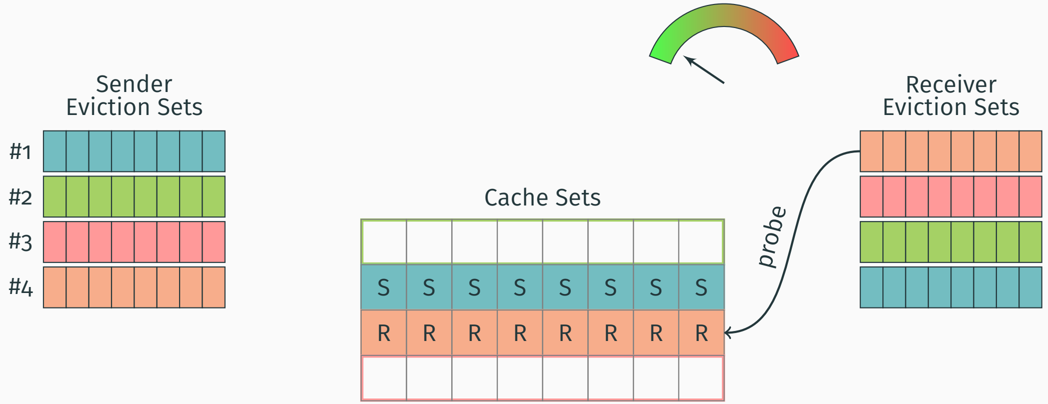
Jamming Agreement



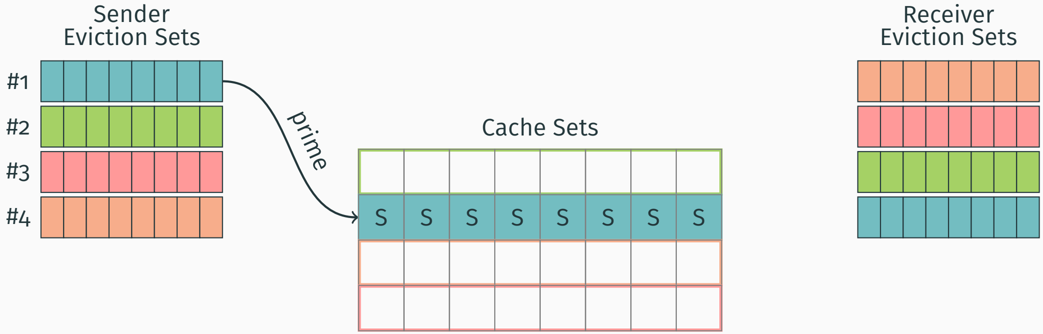
Jamming Agreement



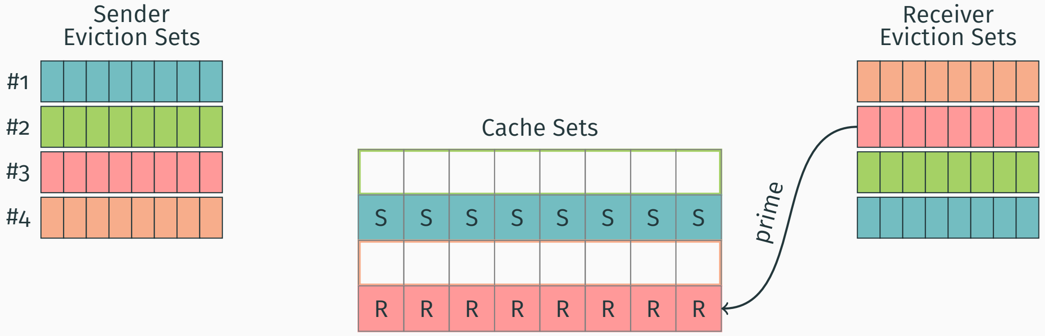
Jamming Agreement



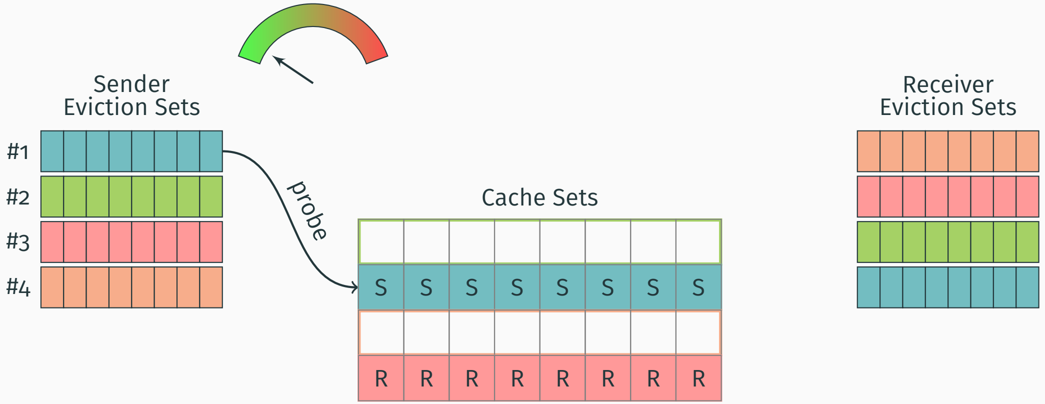
Jamming Agreement



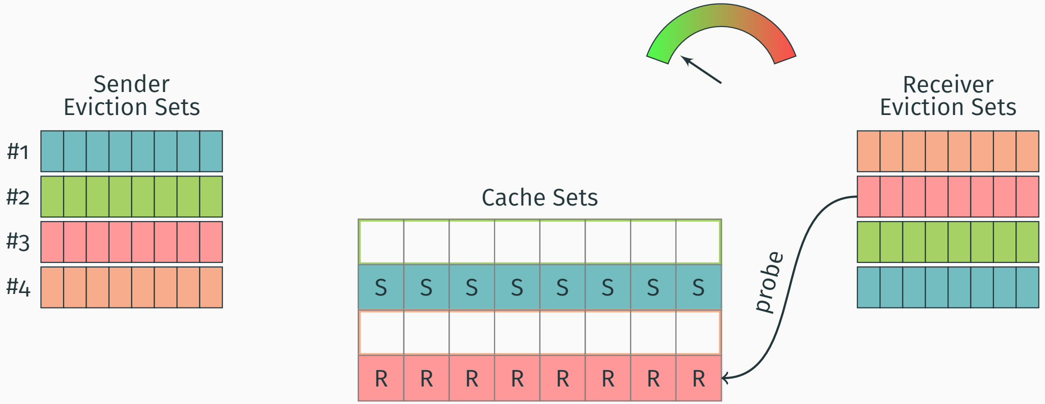
Jamming Agreement



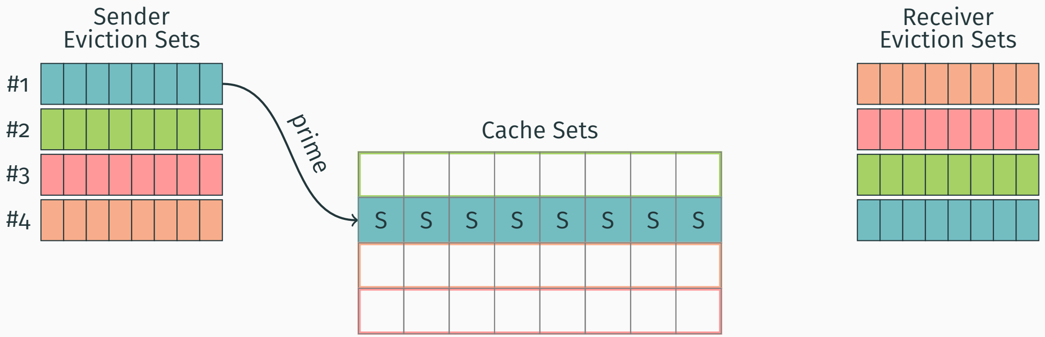
Jamming Agreement



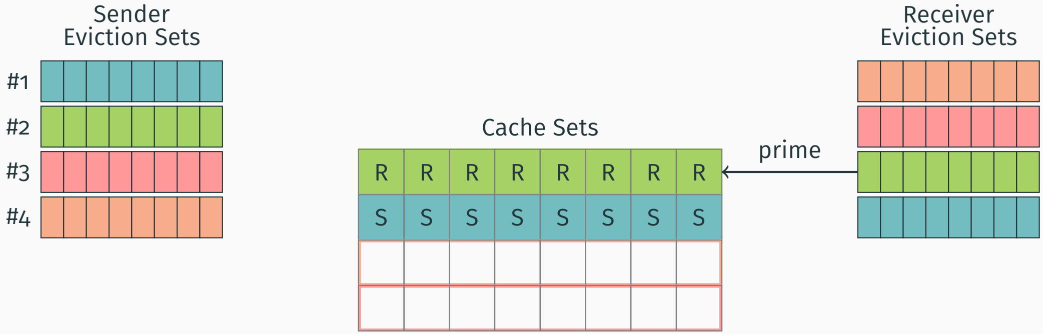
Jamming Agreement



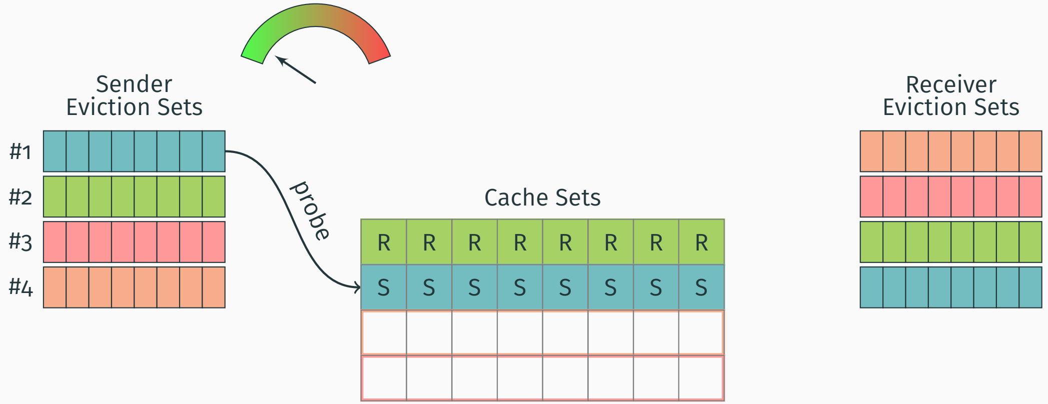
Jamming Agreement



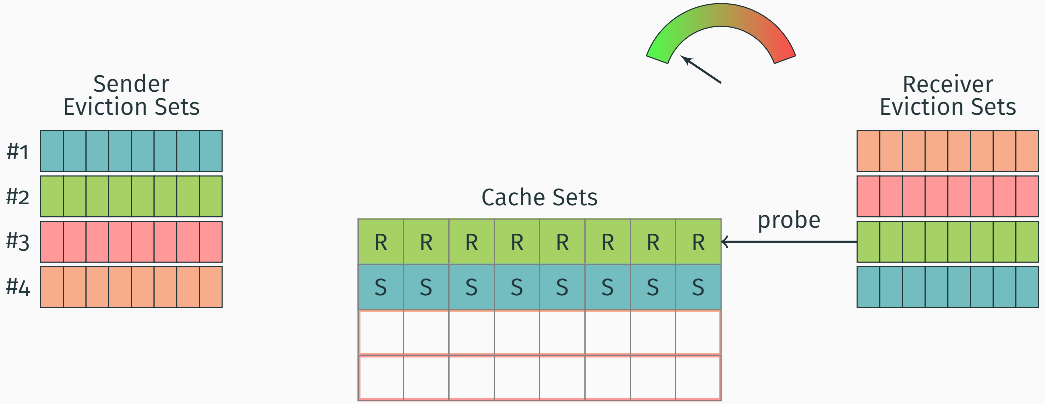
Jamming Agreement



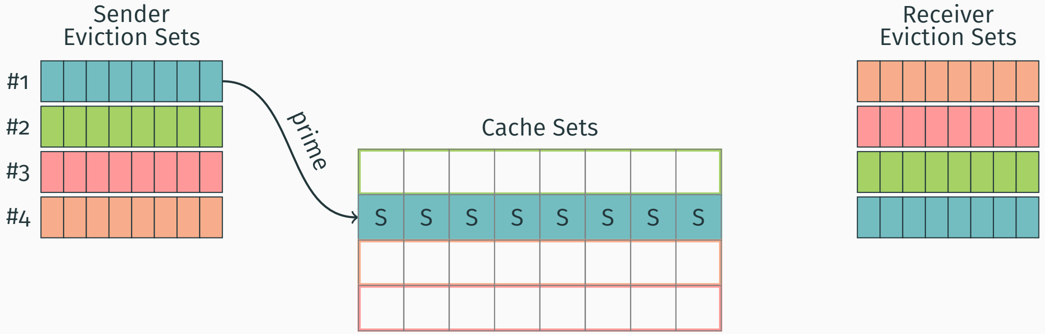
Jamming Agreement



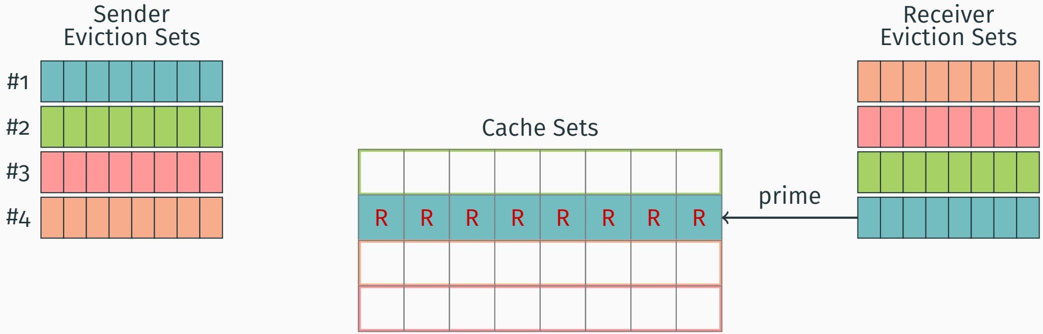
Jamming Agreement



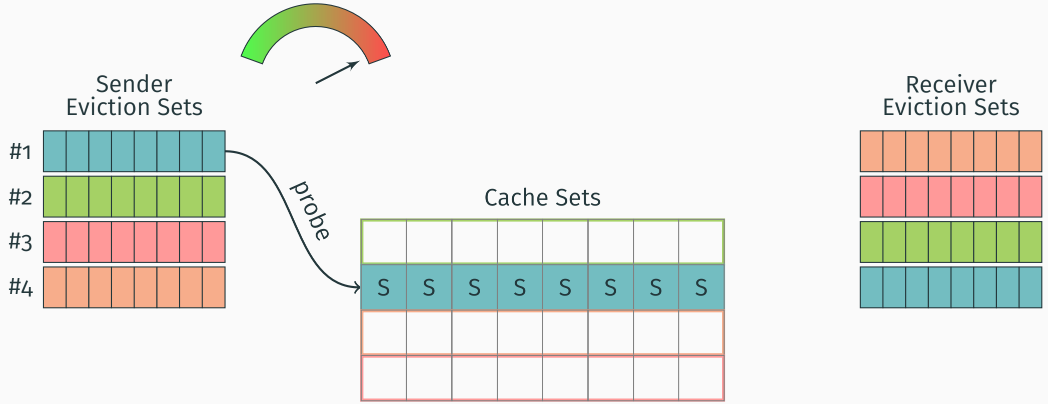
Jamming Agreement



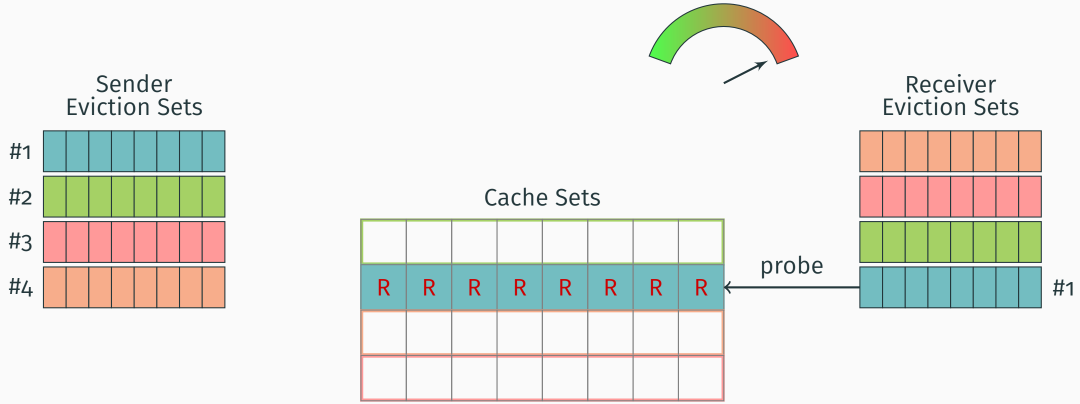
Jamming Agreement



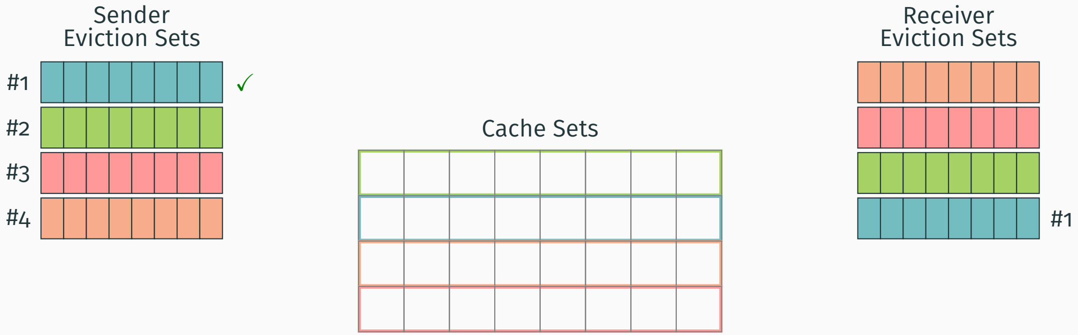
Jamming Agreement



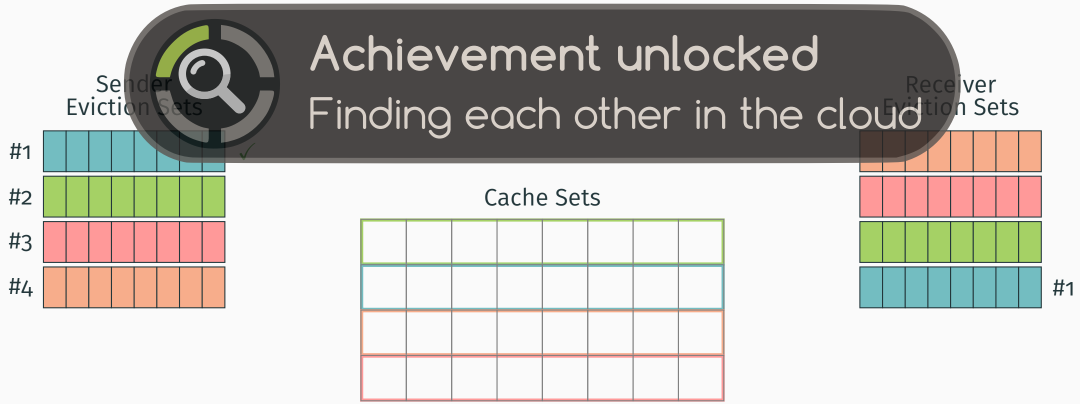
Jamming Agreement



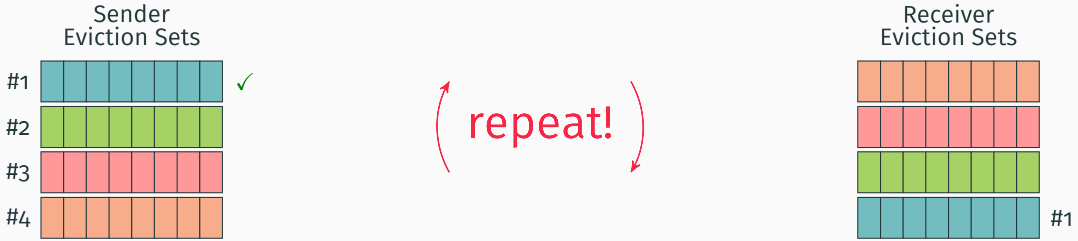
Jamming Agreement



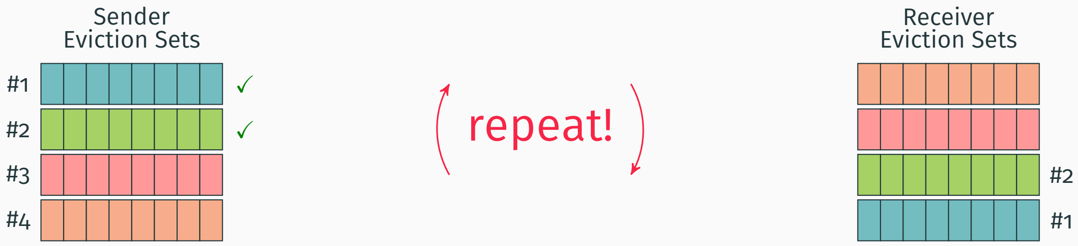
Jamming Agreement



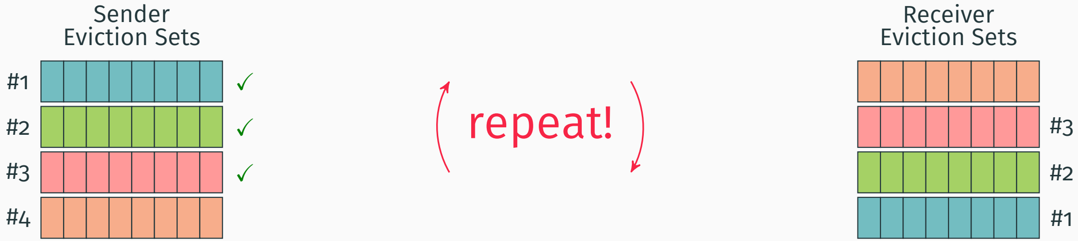
Jamming Agreement



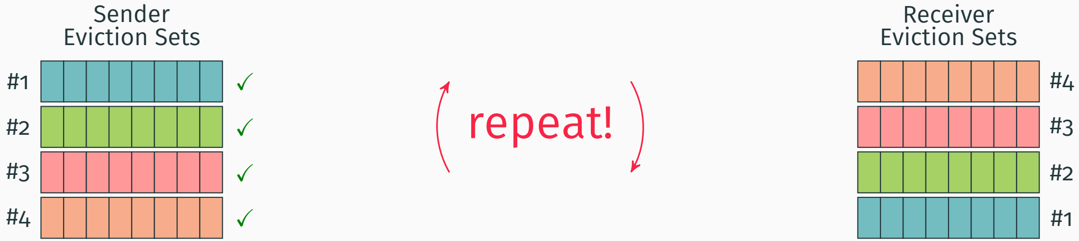
Jamming Agreement



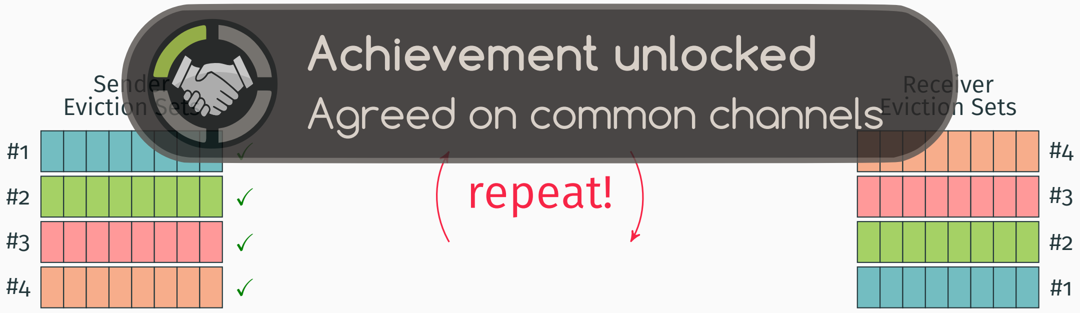
Jamming Agreement



Jamming Agreement



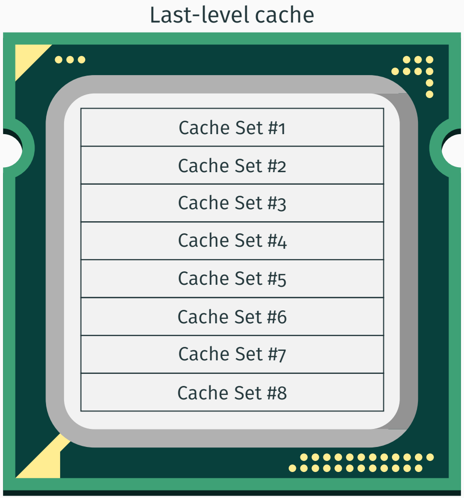
Jamming Agreement



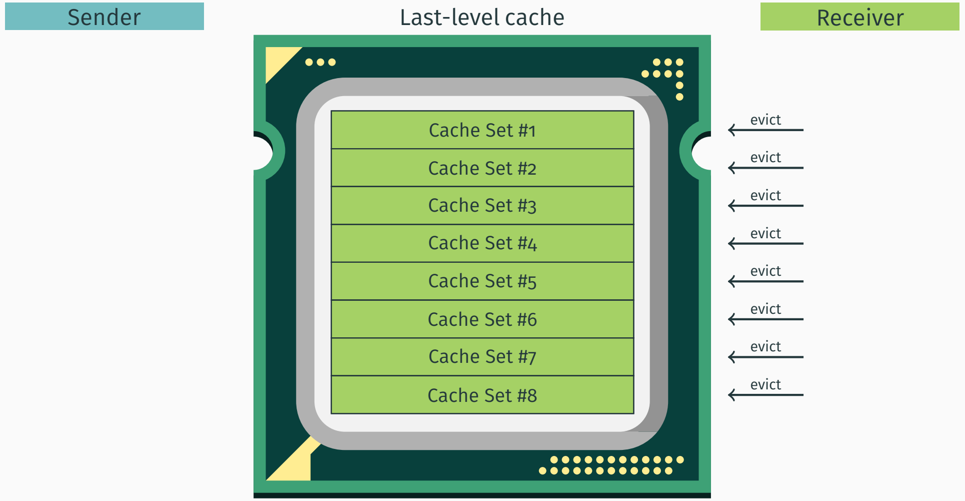
Sending Data

Sender

Receiver



Sending Data

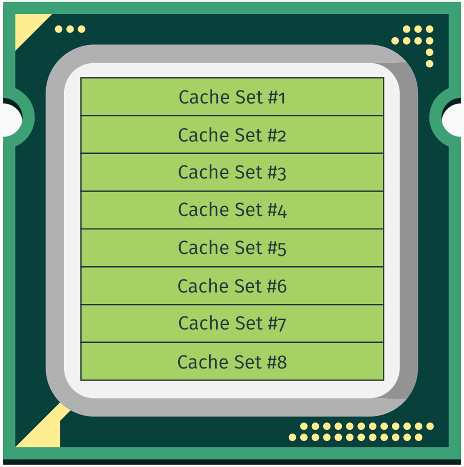


Sending Data

Sender

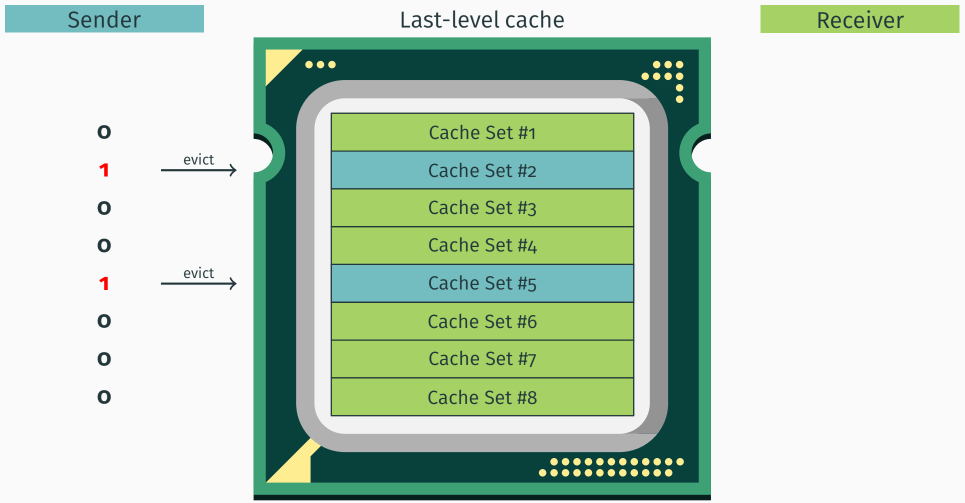
0
1
0
0
1
0
0
0

Last-level cache

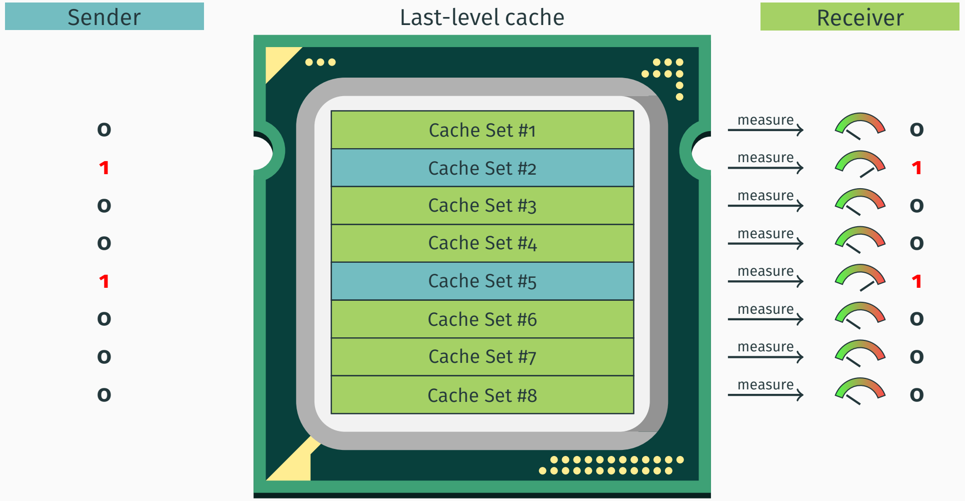


Receiver

Sending Data



Sending Data

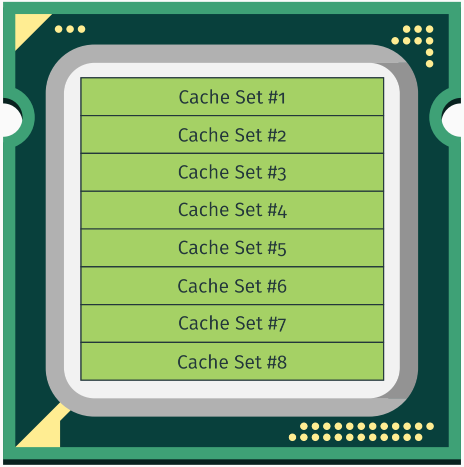


Sending Data

Sender

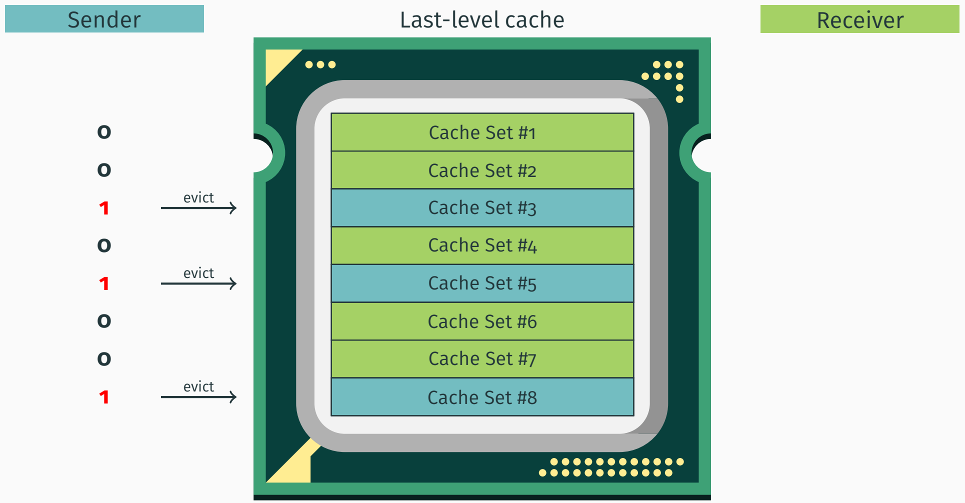
0
0
1
0
1
0
0
1

Last-level cache

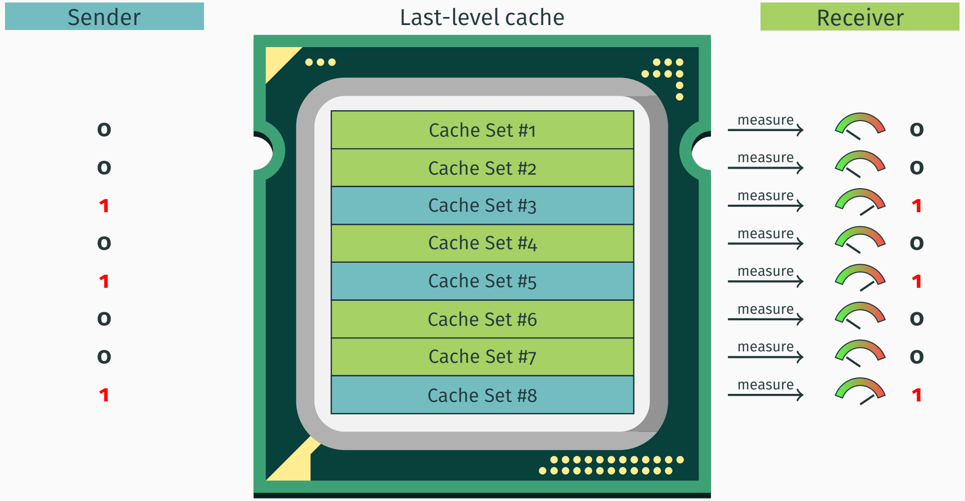


Receiver

Sending Data



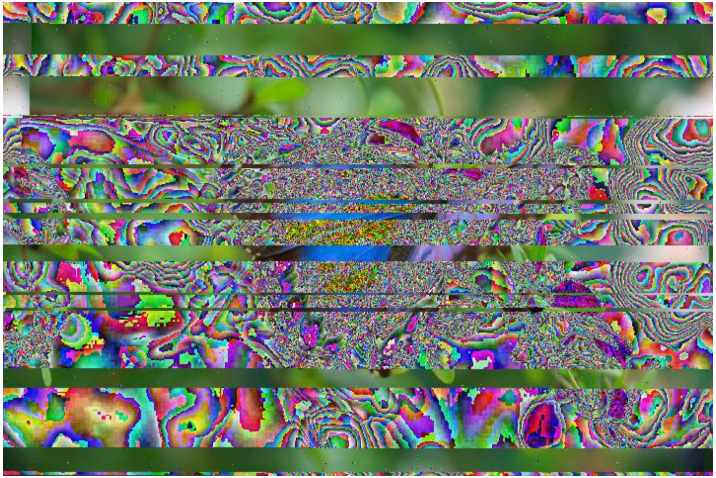
Sending Data



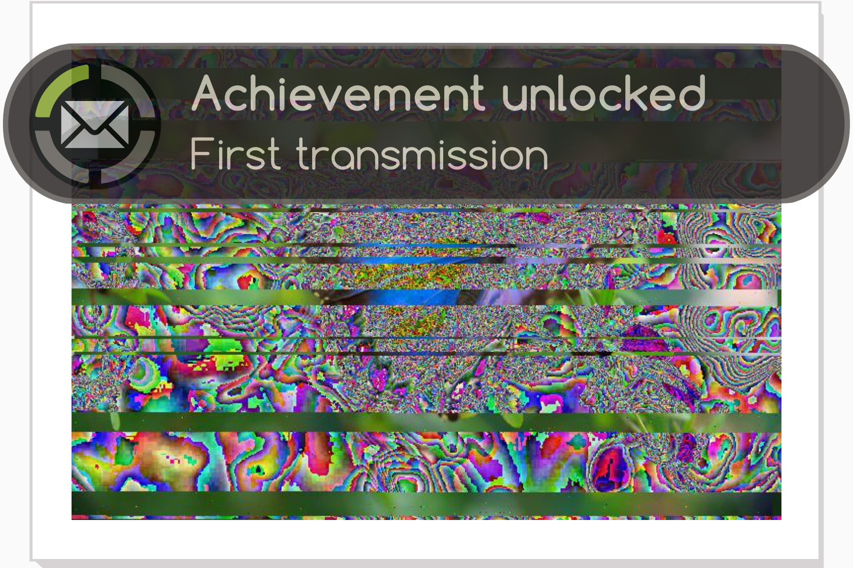
Why don’t we just take the file...

...and put it into the channel?

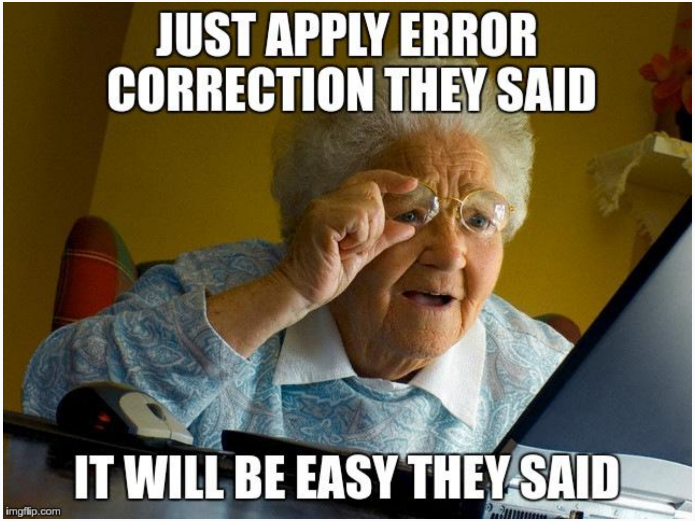
Sending the first image



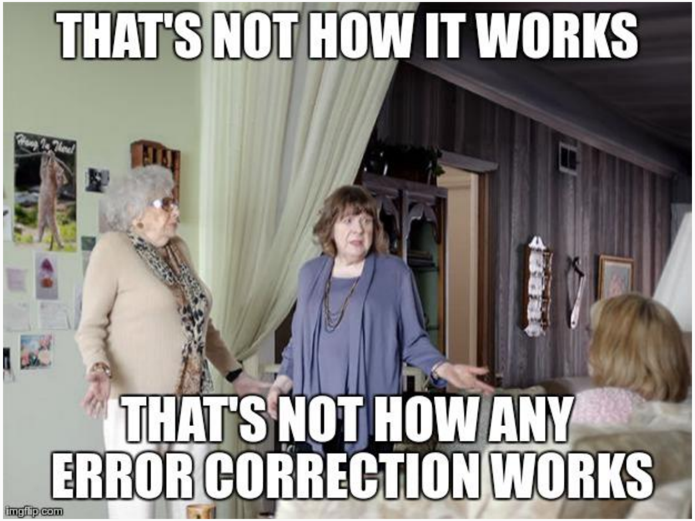
Sending the first image



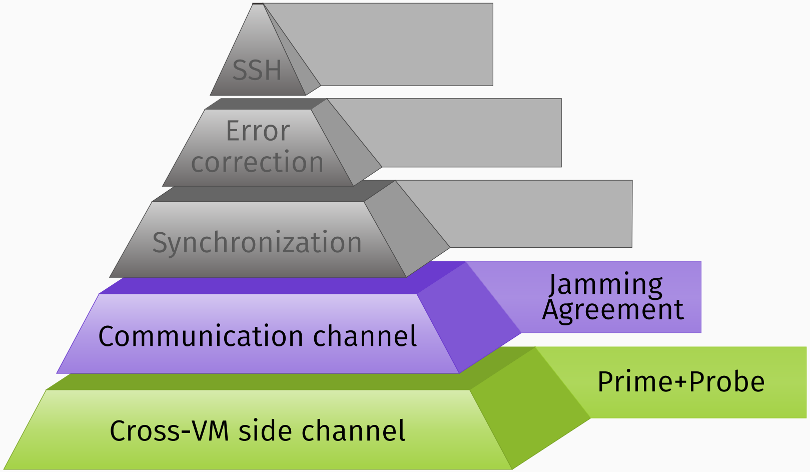
Sending the first image


JUST APPLY ERROR CORRECTION THEY SAID
IT WILL BE EASY THEY SAID
imgflip.com

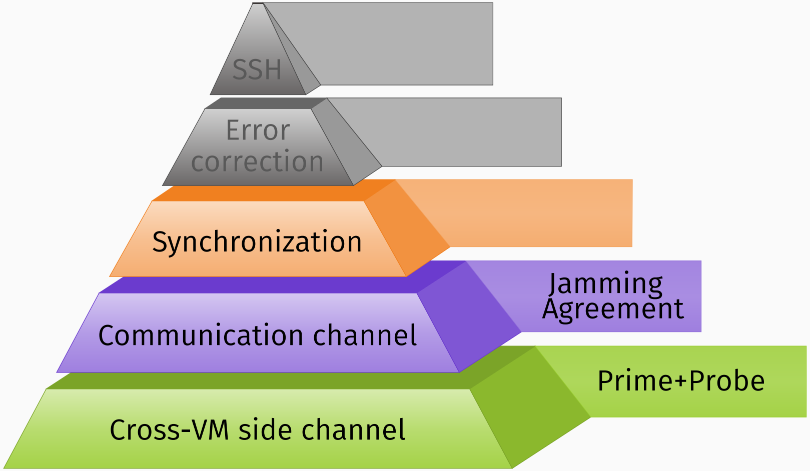
Sending the first image



Challenges

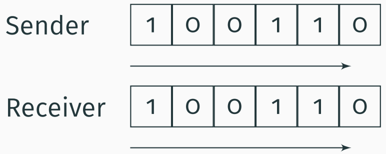

SSH
Error correction
Synchronization
Communication channel
Jamming Agreement
Cross-VM side channel
Prime+Probe

Challenges


SSH
Error correction
Synchronization
Communication channel
Jamming Agreement
Cross-VM side channel
Prime+Probe

Synchronization

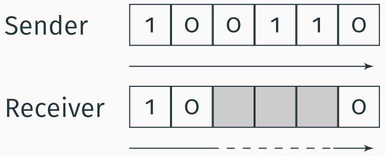
What we see are mostly synchronization errors



Normal transmission

Synchronization

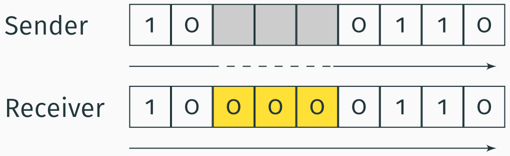
What we see are mostly synchronization errors



Deletion errors due to receiver not scheduled

Synchronization

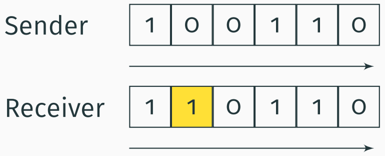
What we see are mostly synchronization errors



Insertion errors due to sender not scheduled

Synchronization

Only sometimes substitution errors which can be corrected



Substitution errors due to unrelated noise

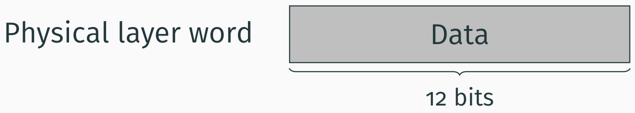
Synchronization

To cope with deletion errors, we use a request-to-send scheme.

Synchronization

To cope with deletion errors, we use a request-to-send scheme.

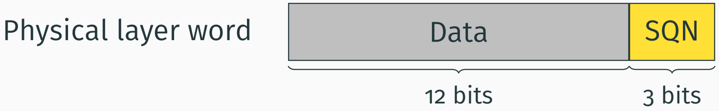
- Transmission uses packets



Synchronization

To cope with deletion errors, we use a request-to-send scheme.

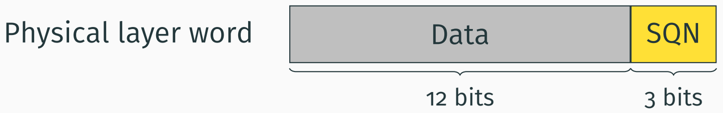
- Transmission uses packets with 3-bit sequence numbers



Synchronization

To cope with deletion errors, we use a request-to-send scheme.

- Transmission uses packets with 3-bit sequence numbers



- Receiver acknowledges by requesting the next sequence number

Synchronization

Important observation: insertion errors are almost always ‘o’s.

Synchronization

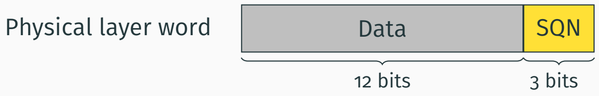
Important observation: insertion errors are almost always ‘O’s.

- Detecting additional ‘O’s detects (many) insertion errors

Synchronization

Important observation: insertion errors are almost always ‘0’s.

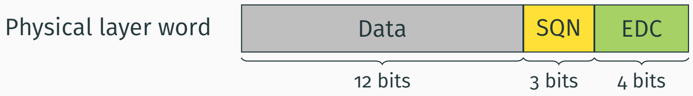
- Detecting additional ‘0’s detects (many) insertion errors
- We need an error detection code



Synchronization

Important observation: insertion errors are almost always ‘0’s.

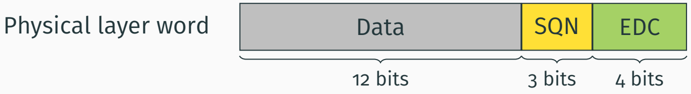
- Detecting additional ‘0’s detects (many) insertion errors
- We need an error detection code $\rightarrow$ Berger codes



Synchronization

Important observation: insertion errors are almost always ‘0’s.

- Detecting additional ‘0’s detects (many) insertion errors
- We need an error detection code $\rightarrow$ Berger codes

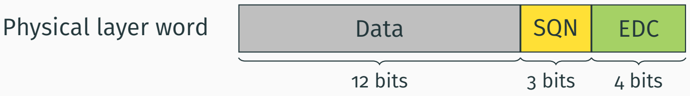


  - Count the number of ‘0’s in a word

Synchronization

Important observation: insertion errors are almost always ‘0’s.

- Detecting additional ‘0’s detects (many) insertion errors
- We need an error detection code $\rightarrow$ Berger codes



- Count the number of ‘0’s in a word
- Side effect: there is no ‘0’-word anymore

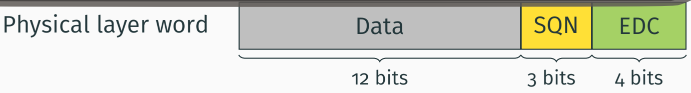
Synchronization

Achievement unlocked

Detect Interrupts

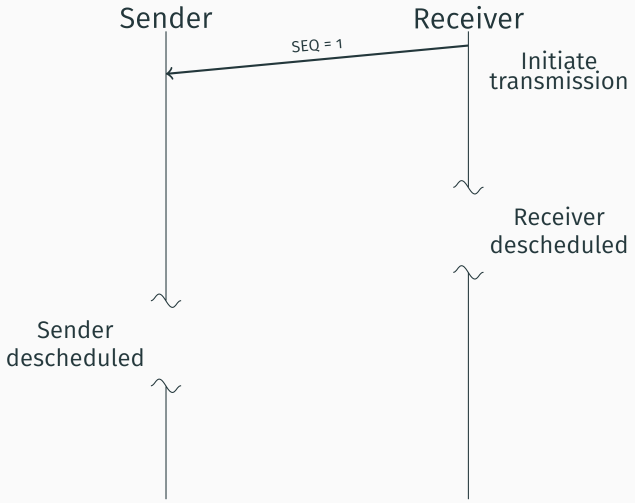
Important observation: insertion errors are almost always ‘0’s.

- Detecting additional ‘0’s detects (many) insertion errors
- We need an error detection code → Berger codes

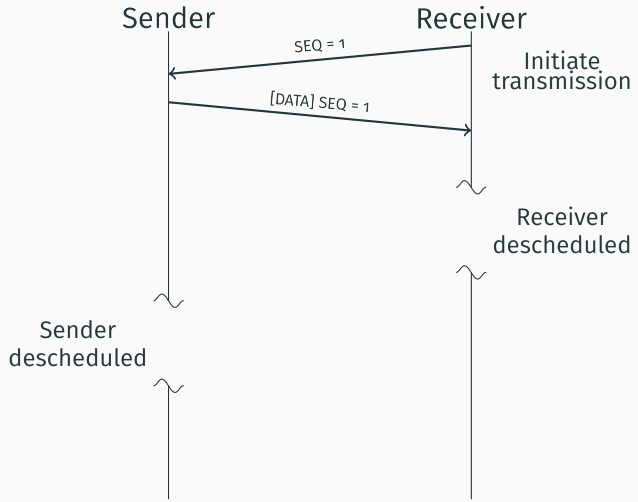


- Count the number of ‘0’s in a word
- Side effect: there is no ‘0’-word anymore

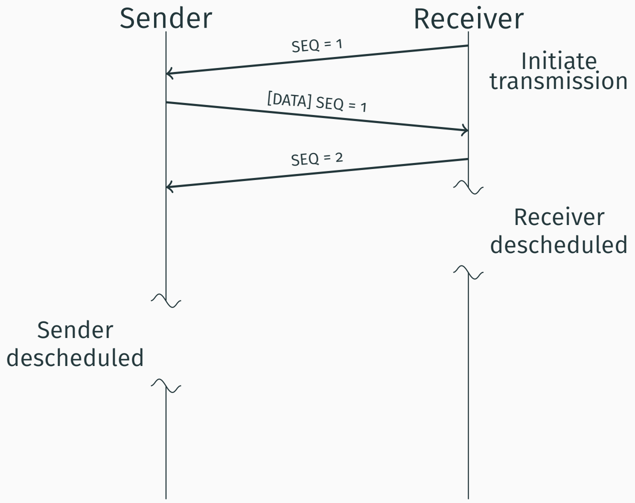
Synchronization



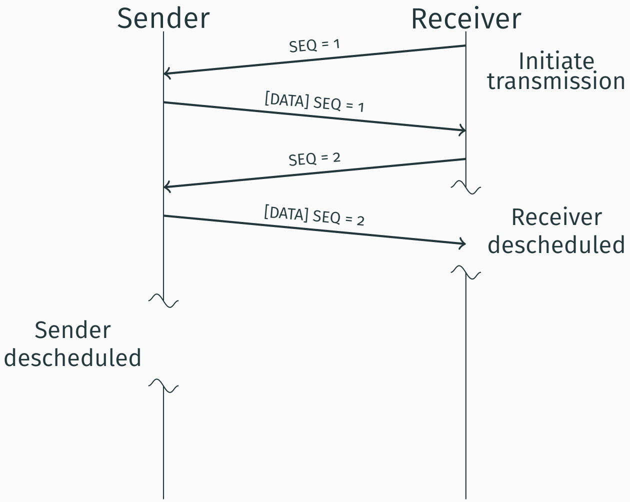
Synchronization



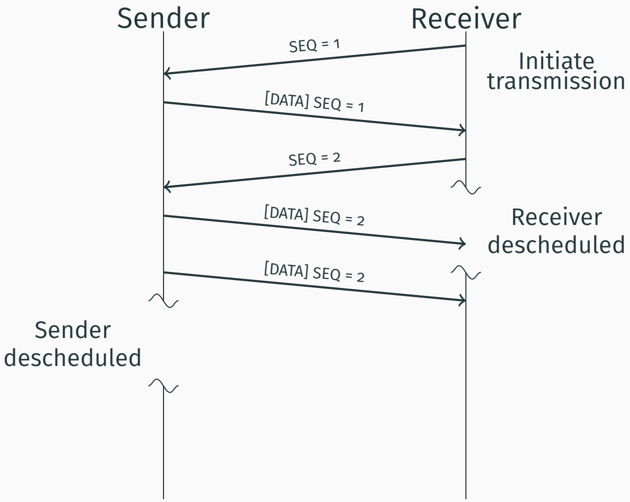
Synchronization



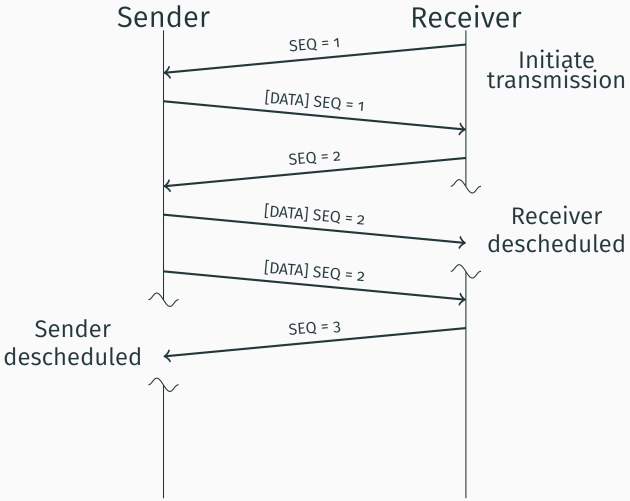
Synchronization



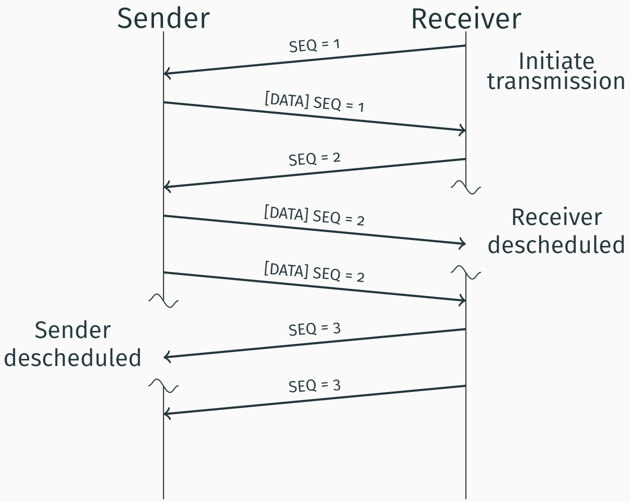
Synchronization



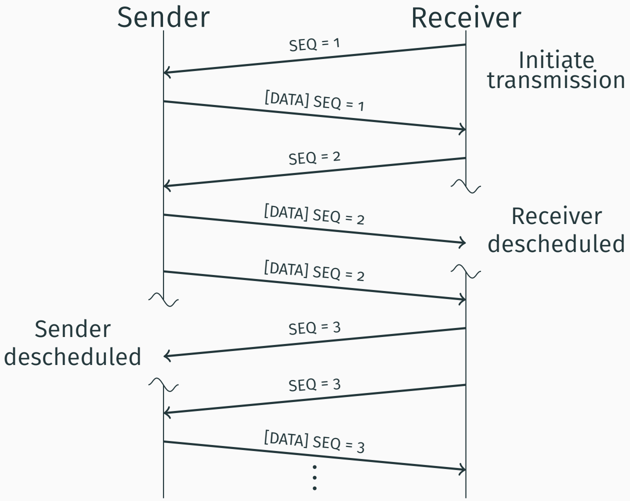
Synchronization



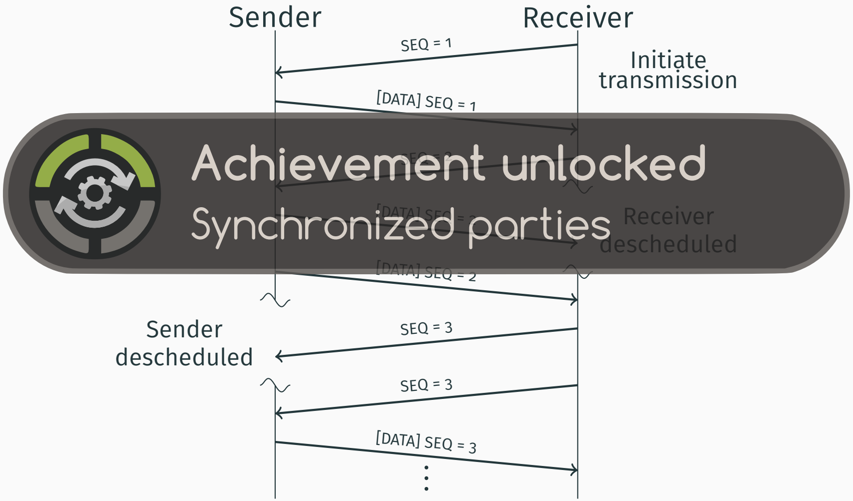
Synchronization



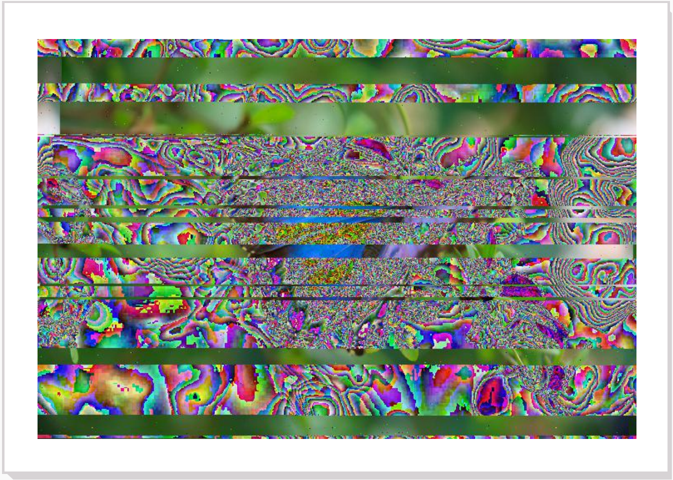
Synchronization



Synchronization

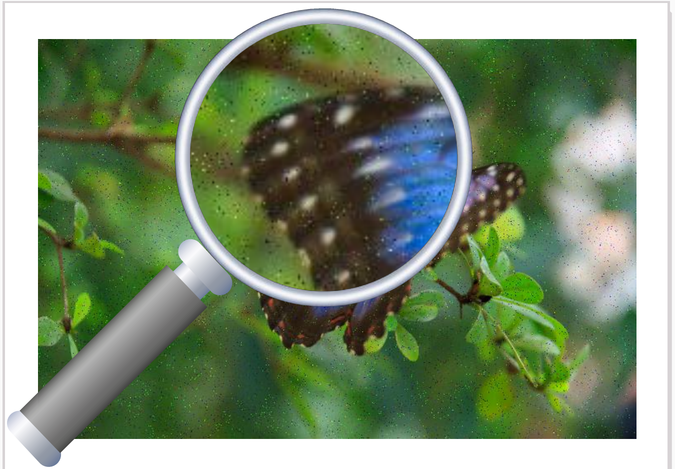


Without synchronization



Synchronization

Synchronization

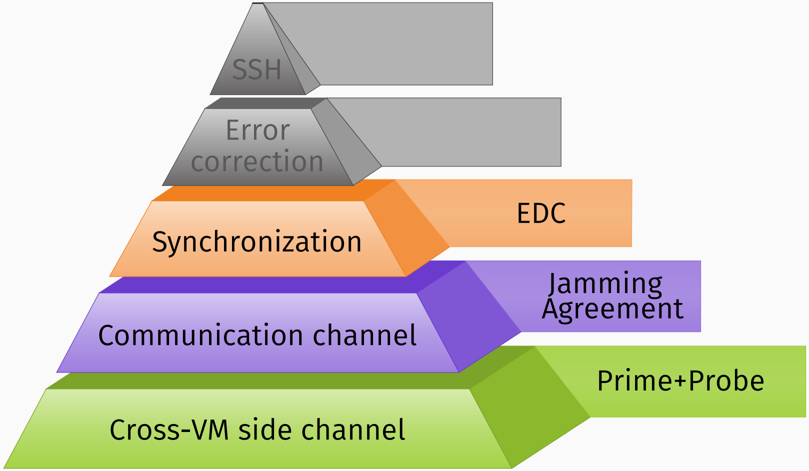


Synchronization

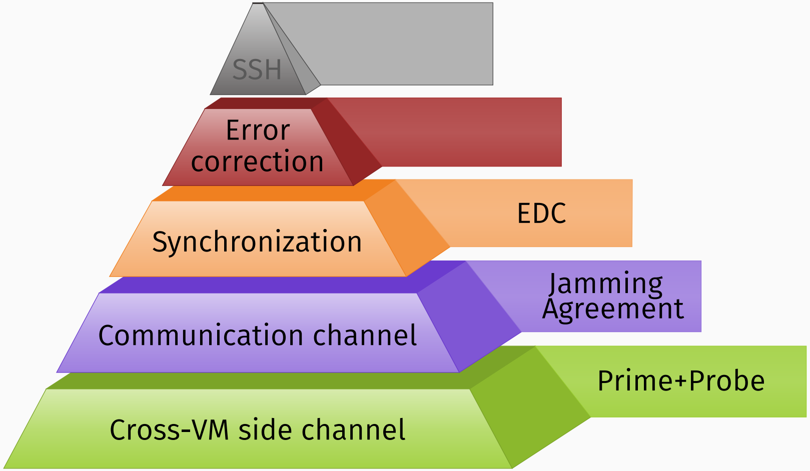
Synchronization

Synchronization

Challenges


SSH
Error correction
Synchronization
EDC
Communication channel
Jamming
Agreement
Cross-VM side channel
Prime+Probe

Challenges


SSH
Error correction
EDC
Synchronization
Jamming Agreement
Communication channel
Prime+Probe
Cross-VM side channel

Error correction

- Substitution errors can be corrected using forward error correction

Error correction

- Substitution errors can be corrected using forward error correction
- We use wide-spread Reed-Solomon codes

Error correction

- Substitution errors can be corrected using forward error correction
- We use wide-spread Reed-Solomon codes
- Packets made of symbols

Error correction

- Substitution errors can be corrected using forward error correction
- We use wide-spread Reed-Solomon codes
- Packets made of symbols
    - Symbol size: 12 bits (“RS-word”)

Error correction

- Substitution errors can be corrected using forward error correction
- We use wide-spread Reed-Solomon codes
- Packets made of symbols
    - Symbol size: 12 bits (“RS-word”)
    - Packet size: 4095 symbols (= $2^{symbol} - 1$)

Error correction

- Substitution errors can be corrected using forward error correction
- We use wide-spread Reed-Solomon codes
- Packets made of symbols
    - Symbol size: 12 bits (“RS-word”)
    - Packet size: 4095 symbols (= $2^{symbol} - 1$)
- Packet consists of actual message and error correction symbols

Error correction

RS codes are a simple matrix multiplication

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

Error correction

RS codes are a simple matrix multiplication

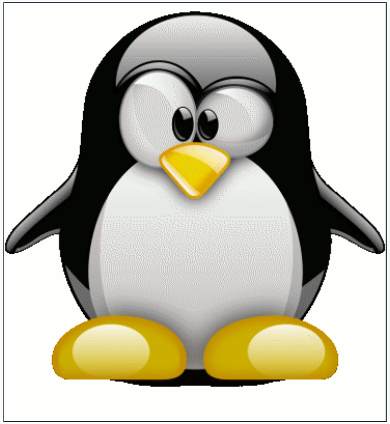
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \end{bmatrix} \times \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

Error correction

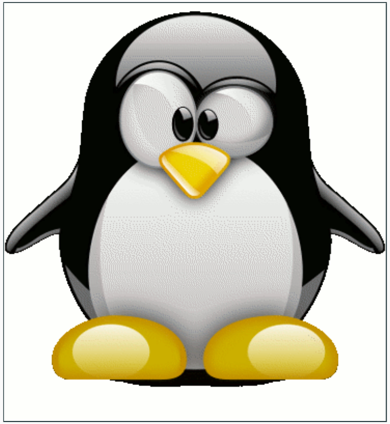
RS codes are a simple matrix multiplication

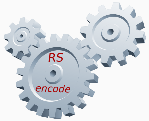
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \end{bmatrix} \times \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ c_0 \\ c_1 \end{bmatrix}$$

Error correction

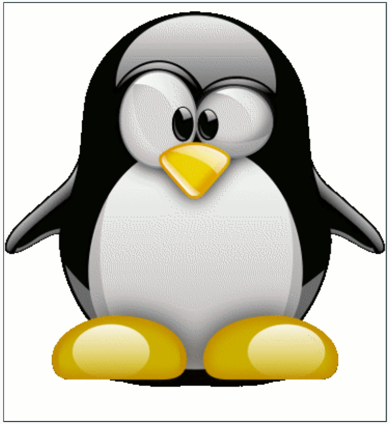


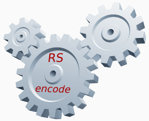
Error correction





Error correction



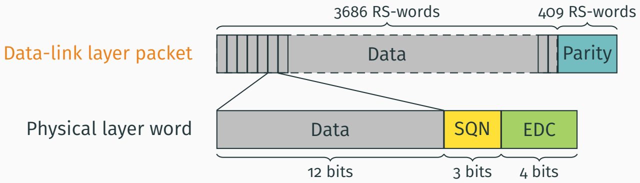


Error correction

- Better safe than sorry: 10% error-correcting code

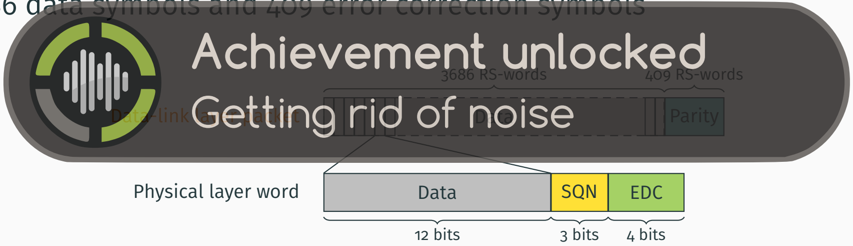
Error correction

- Better safe than sorry: 10% error-correcting code
- 3686 data symbols and 409 error correction symbols



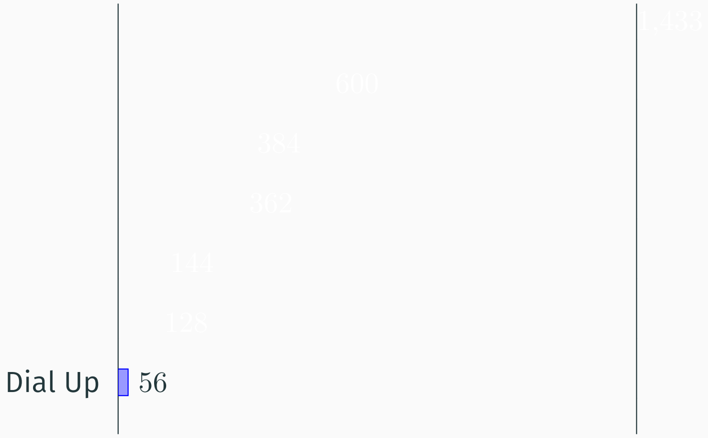
Error correction

- Better safe than sorry: 10% error-correcting code
- 3686 data symbols and 409 error correction symbols



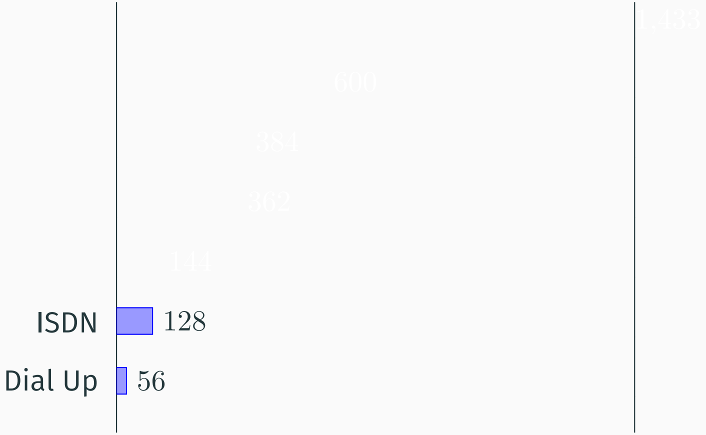
Error correction

Comparison of transmission speeds (in kbit/s)



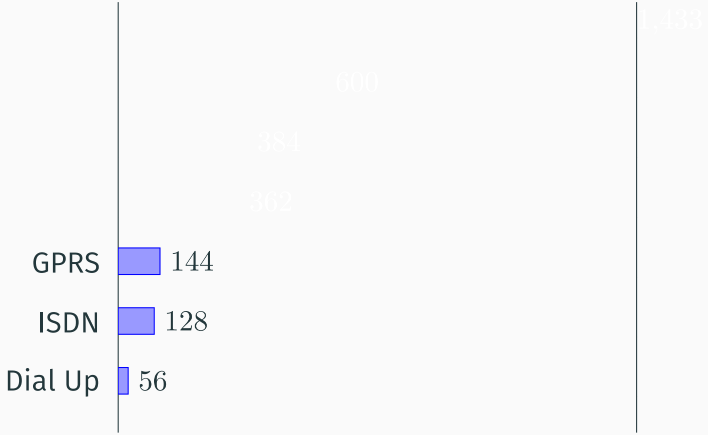
Error correction

Comparison of transmission speeds (in kbit/s)



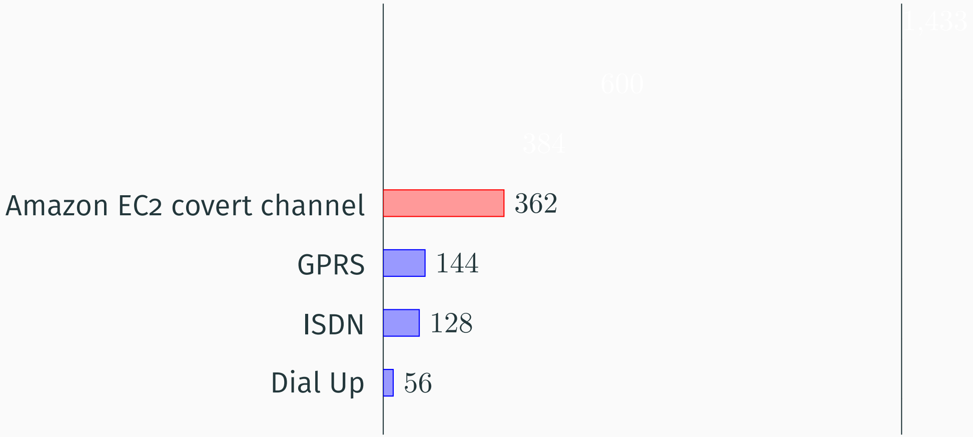
Error correction

Comparison of transmission speeds (in kbit/s)



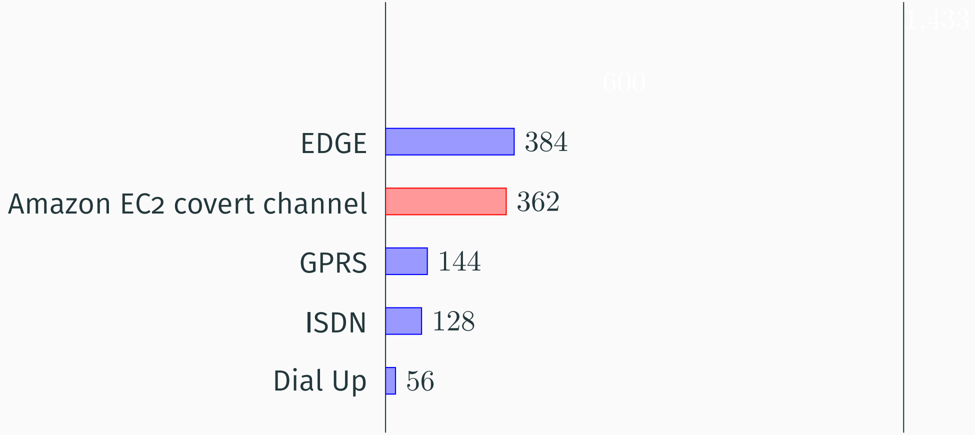
Error correction

Comparison of transmission speeds (in kbit/s)



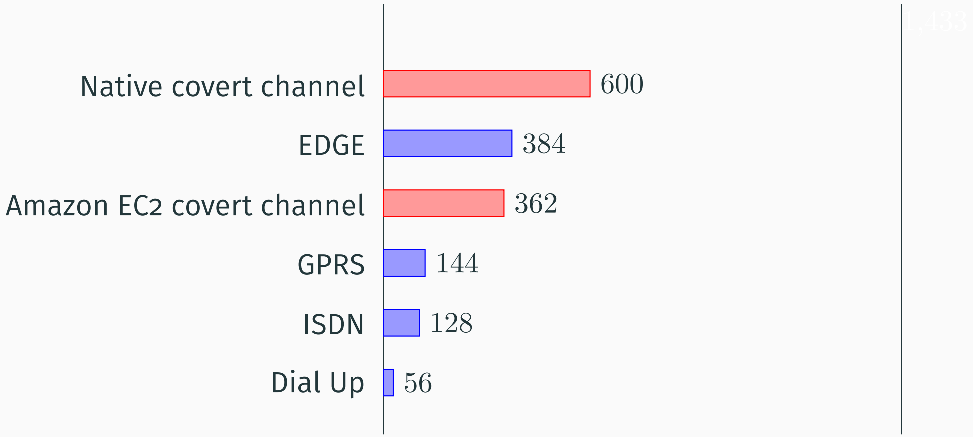
Error correction

Comparison of transmission speeds (in kbit/s)



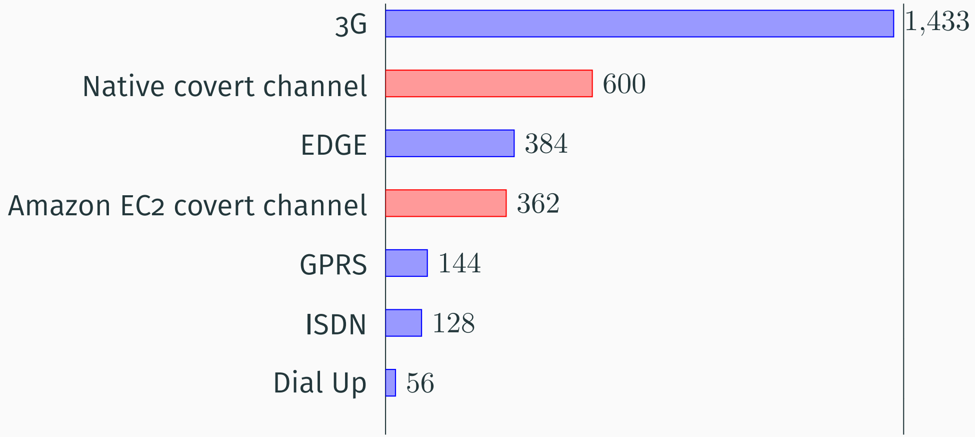
Error correction

Comparison of transmission speeds (in kbit/s)

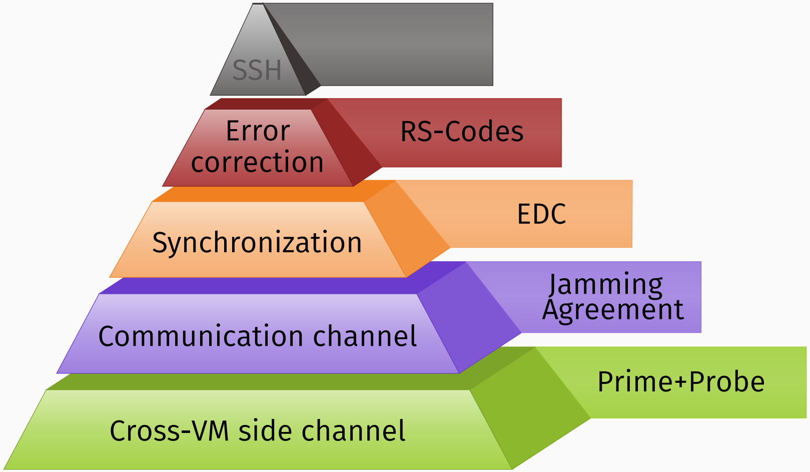


Error correction

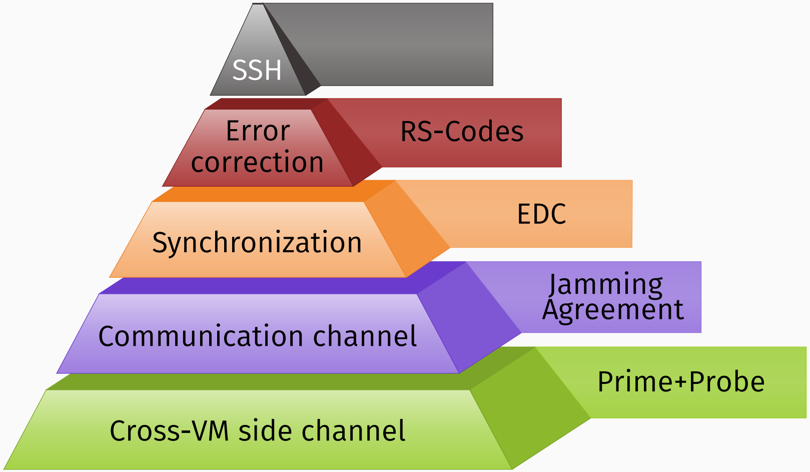
Comparison of transmission speeds (in kbit/s)



Challenges


SSH
Error correction
RS-Codes
Synchronization
EDC
Communication channel
Jamming Agreement
Cross-VM side channel
Prime+Probe

Challenges


SSH
Error correction
RS-Codes
Synchronization
EDC
Communication channel
Jamming Agreement
Cross-VM side channel
Prime+Probe

SSH

- The covert channel is fast and error free

SSH

- The covert channel is fast and error free
- We want it to be useful

SSH

- The covert channel is fast and error free
- We want it to be useful
- A remote shell without network access would be really nice...

SSH

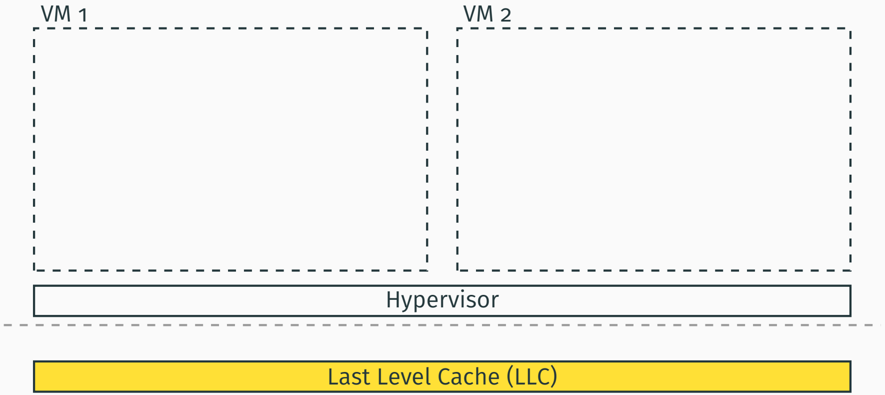
- The covert channel is fast and error free
- We want it to be useful
- A remote shell without network access would be really nice...

SSH

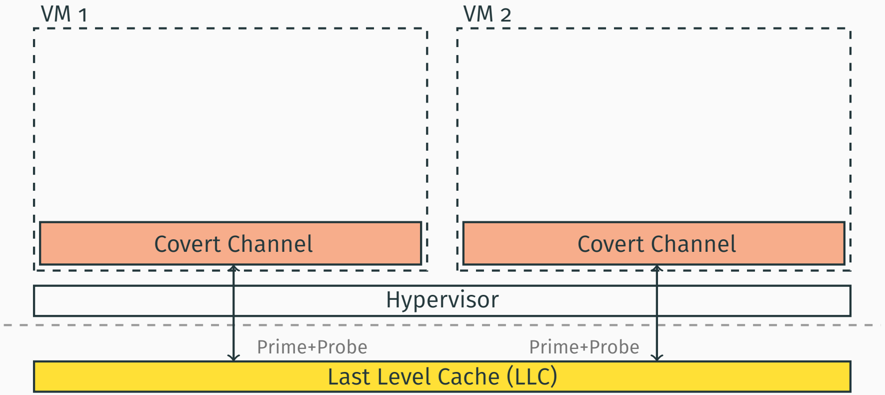
- The covert channel is fast and error free
- We want it to be useful
- A remote shell without network access would be really nice...
- Prerequisites: just TCP

SSH

TCP-over-Cache

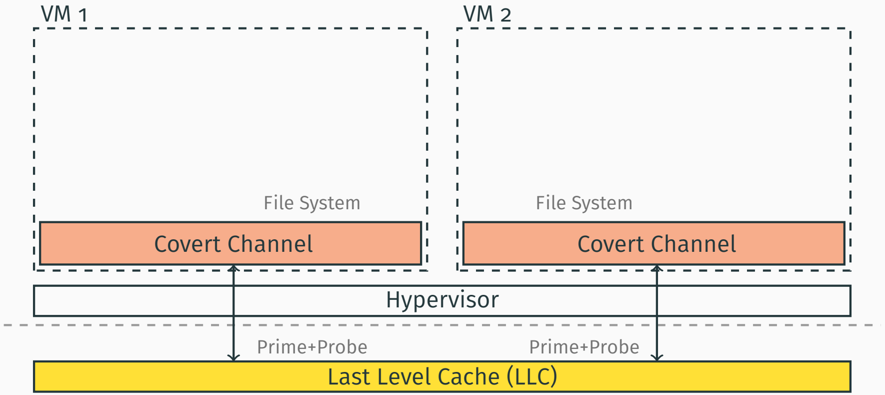


TCP-over-Cache



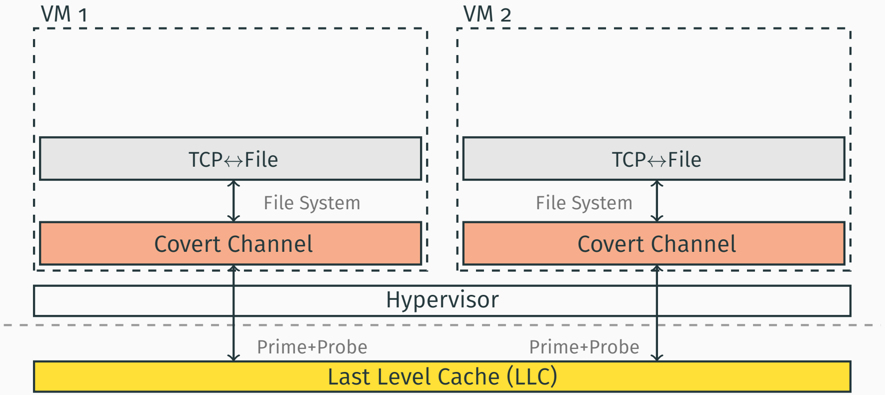
SSH

TCP-over-Cache



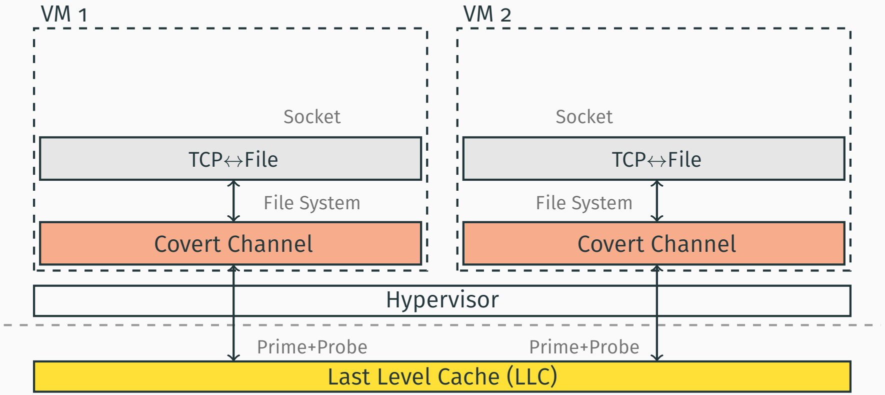
SSH

TCP-over-Cache



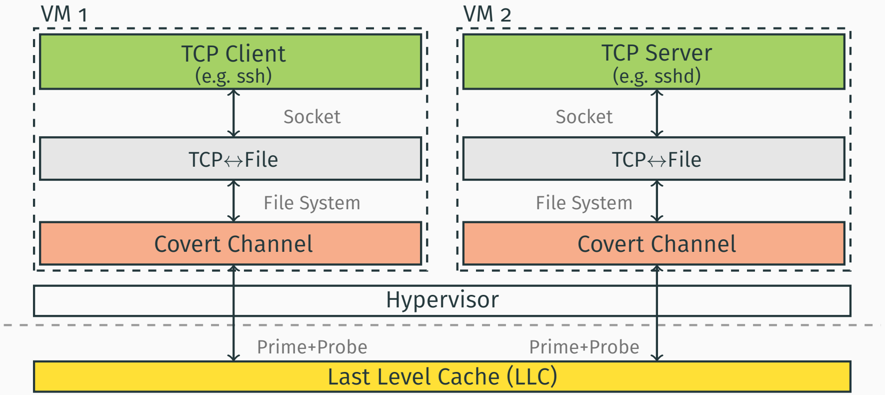
SSH

TCP-over-Cache



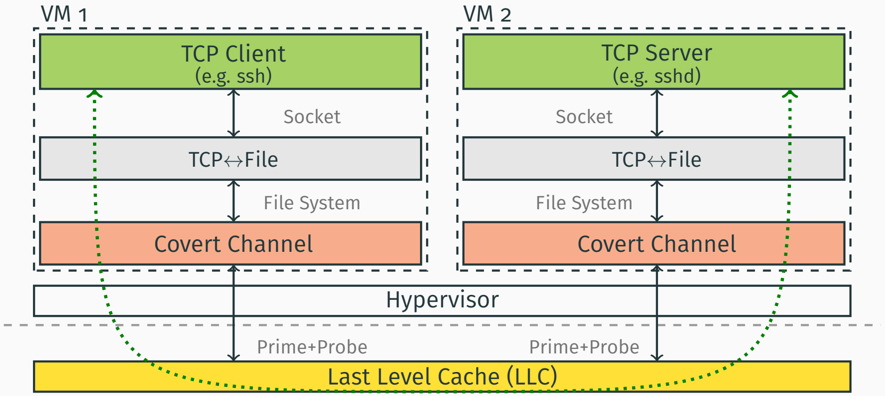
SSH

TCP-over-Cache



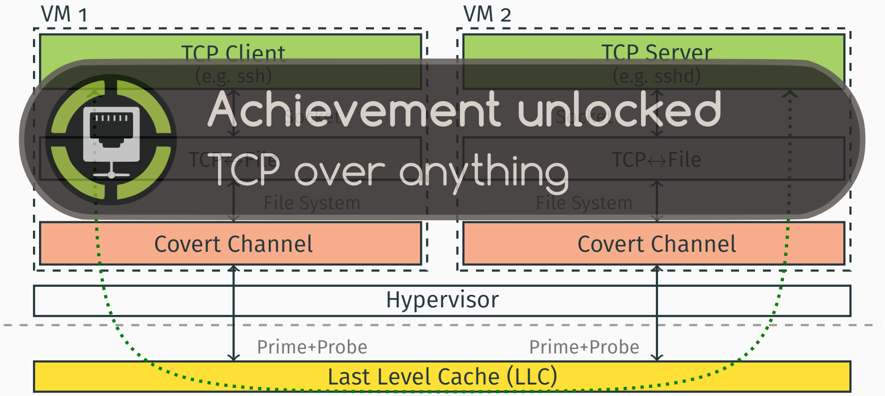
SSH

TCP-over-Cache



SSH

TCP-over-Cache



SSH

SSH between two instances on Amazon EC2

Noise	Connection
No noise	✓

SSH

SSH between two instances on Amazon EC2

Noise	Connection
No noise	✓
`stress -m 8` on third VM	✓

SSH

SSH between two instances on Amazon EC2

Noise	Connection
No noise	✓
`stress -m 8` on third VM	✓
Web server on third VM	✓

SSH

SSH between two instances on Amazon EC2

Noise	Connection
No noise	✓
`stress -m 8` on third VM	✓
Web server on third VM	✓
Web server on all VMs	✓

SSH

SSH between two instances on Amazon EC2

Noise	Connection
No noise	✓
`stress -m 8` on third VM	✓
Web server on third VM	✓
Web server on all VMs	✓
`stress -m 1` on server side	unstable

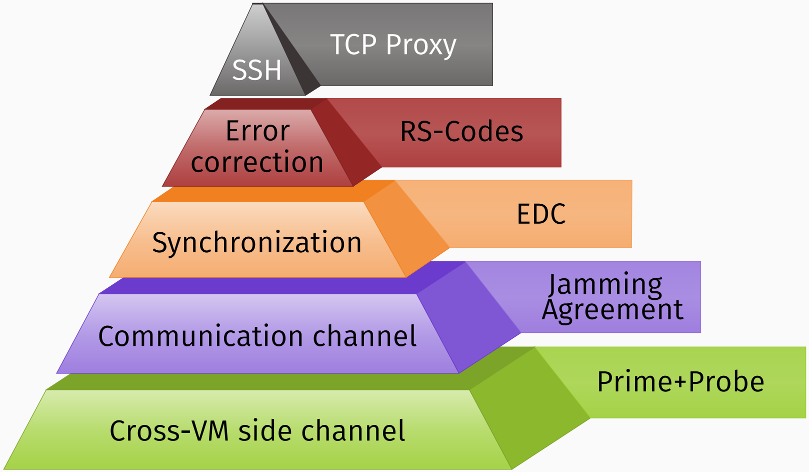
SSH

SSH between two instances on Amazon EC2

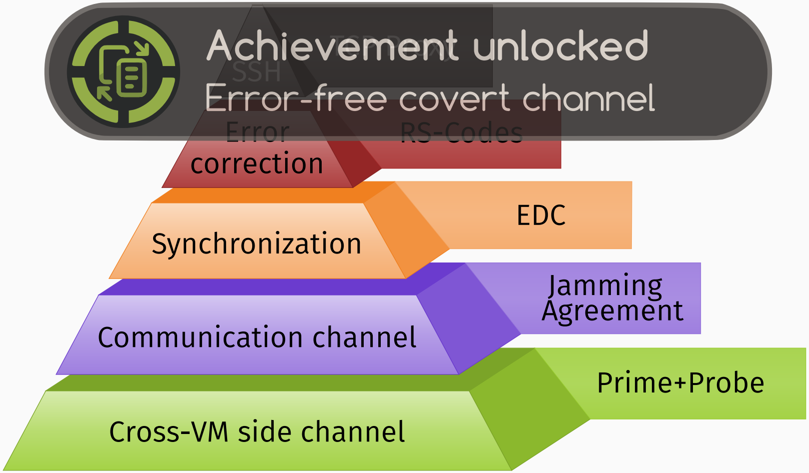
Noise	Connection
No noise	✓
`stress -m 8` on third VM	✓
Web server on third VM	✓
Web server on all VMs	✓
`stress -m 1` on server side	unstable

Telnet also works with occasional corrupted bytes with `stress -m 1`

Challenges


SSH
TCP Proxy
Error correction
RS-Codes
Synchronization
EDC
Communication channel
Jamming Agreement
Cross-VM side channel
Prime+Probe

Challenges



Achievement unlocked
Error-free covert channel
SSH
Error correction
RS-Codes
Synchronization
EDC
Communication channel
Jamming Agreement
Cross-VM side channel
Prime+Probe

Conclusion

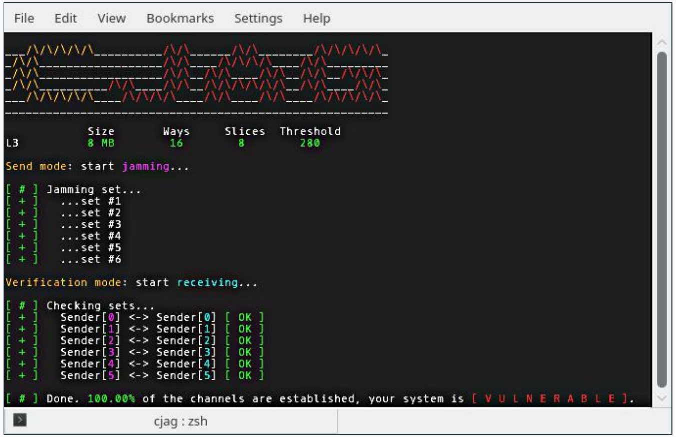
Black Hat Sound Bytes

Black Hat Sound Bytes.

- Cache covert channels are practical
- We can get a noise-free and fast channel, even in the cloud
- Noise does not protect against covert channels

Try it!

Is my cloud (provider) vulnerable?



```
___/\/\/\/\/\___________/\/\______/\/\________/\/\/\/\/\_
_/\/\_________________/\/\____/\/\/\/\____/\/\_________
_/\/\_________________/\/\__/\/\____/\/\__/\/\__/\/\/\_
_/\/\__________/\/\____/\/\__/\/\/\/\/\/\__/\/\____/\/\_
___/\/\/\/\/\____/\/\/\/\____/\/\____/\/\____/\/\/\/\/\_
--------------------------------------------------------

            Size        Ways       Slices  Threshold
L3          8 MB         16          8        280

Send mode: start jamming...

[ # ] Jamming set...
[ + ]   ...set #1
[ + ]   ...set #2
[ + ]   ...set #3
[ + ]   ...set #4
[ + ]   ...set #5
[ + ]   ...set #6

Verification mode: start receiving...

[ # ] Checking sets...
[ + ]   Sender[0] <-> Sender[0] [ OK ]
[ + ]   Sender[1] <-> Sender[1] [ OK ]
[ + ]   Sender[2] <-> Sender[2] [ OK ]
[ + ]   Sender[3] <-> Sender[3] [ OK ]
[ + ]   Sender[4] <-> Sender[4] [ OK ]
[ + ]   Sender[5] <-> Sender[5] [ OK ]

[ # ] Done. 100.00% of the channels are established, your system is [ V U L N E R A B L E ].
```

https://github.com/IAIK/CJAG

Demo

What you just saw

We extended Amazon’s product portfolio

What you just saw

We extended Amazon's product portfolio

amazon.com Prime

What you just saw

We extended Amazon’s product portfolio

amazon.com
Prime+Probe

Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud

Michael Schwarz and Manuel Weber

March 30th, 2017

https://github.com/IAIK/CJAG

Bibliography I

References

Boano, Carlo Alberto et al. (2012). “Jag: Reliable and predictable wireless agreement under external radio interference”. In: IEEE 33rd Real-Time Systems Symposium (RTSS).

Schwarz, Michael and Anders Fogh (2016). “DRAMA: How your DRAM becomes a security problem”. In: Black Hat Europe 2016.