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

Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, Kay Römer
Graz University of Technology

February 2017 — NDSS 2017

Outline

- cache covert channels
- how do we get a covert channel working in the cloud?
- how do we get a covert channel working in a noisy environment?
- what are the applications of such covert channel?

CPU cache

- main memory is slow compared to the CPU

CPU cache

- main memory is slow compared to the CPU
- caches buffer frequently used data

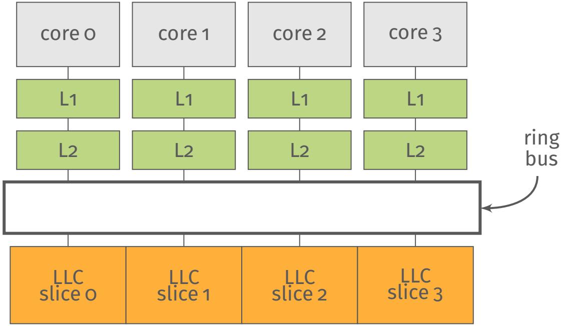
CPU cache

- main memory is slow compared to the CPU
- caches buffer frequently used data
- every data access goes through the cache

CPU cache

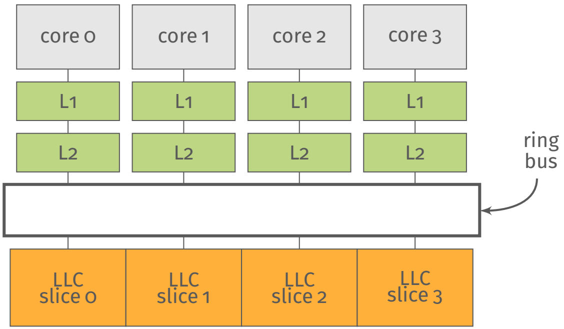
- main memory is slow compared to the CPU
- caches buffer frequently used data
- every data access goes through the cache
- caches are transparent to the OS and the software

Caches on Intel CPUs



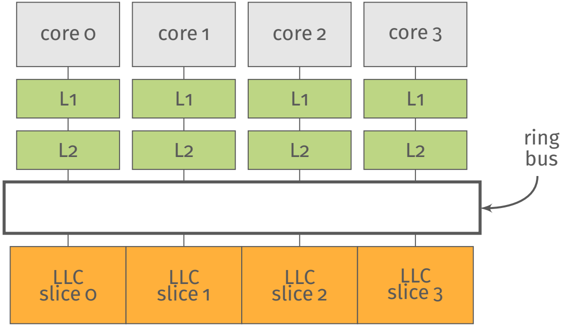
- L1 and L2 are private

Caches on Intel CPUs



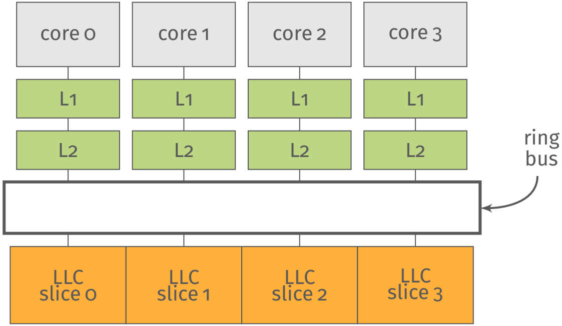
- L1 and L2 are private
- last-level cache

Caches on Intel CPUs



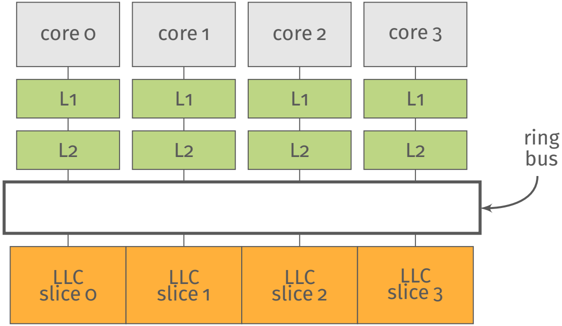
- L1 and L2 are private
- last-level cache
  - divided in slices

Caches on Intel CPUs



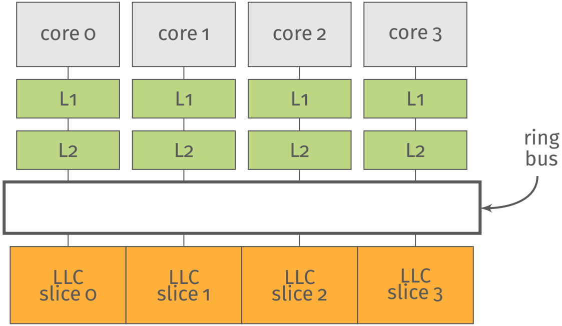
- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores

Caches on Intel CPUs



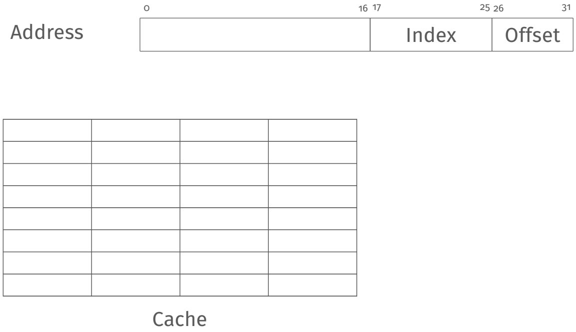
- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores
  - inclusive

Caches on Intel CPUs

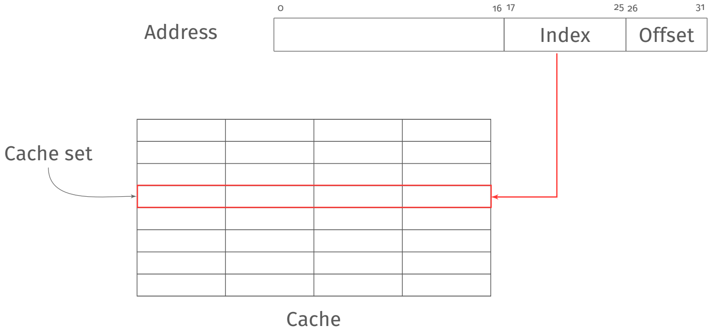


- L1 and L2 are private
- last-level cache
  - divided in slices
  - shared across cores
  - inclusive
  - hash function maps a physical address to a slice

Set-associative caches

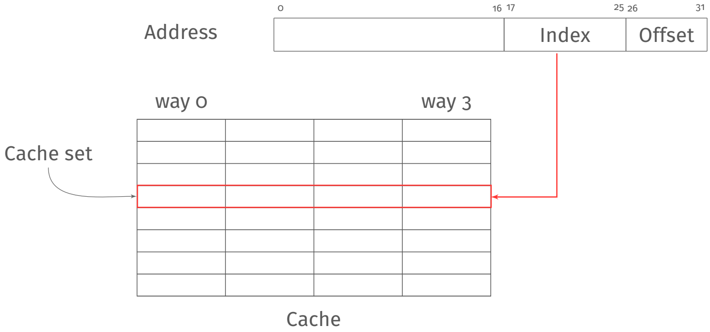


Set-associative caches



Data loaded in a specific set depending on its address

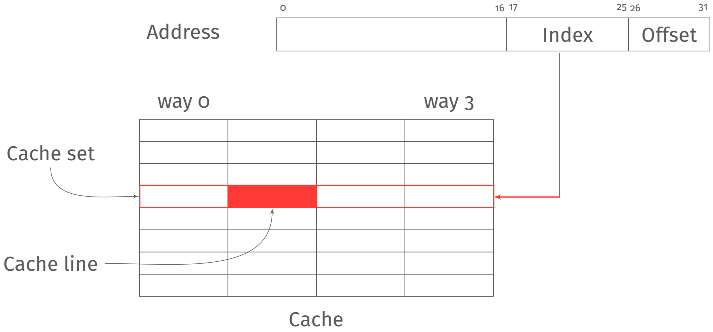
Set-associative caches



Data loaded in a specific set depending on its address

Several ways per set

Set-associative caches

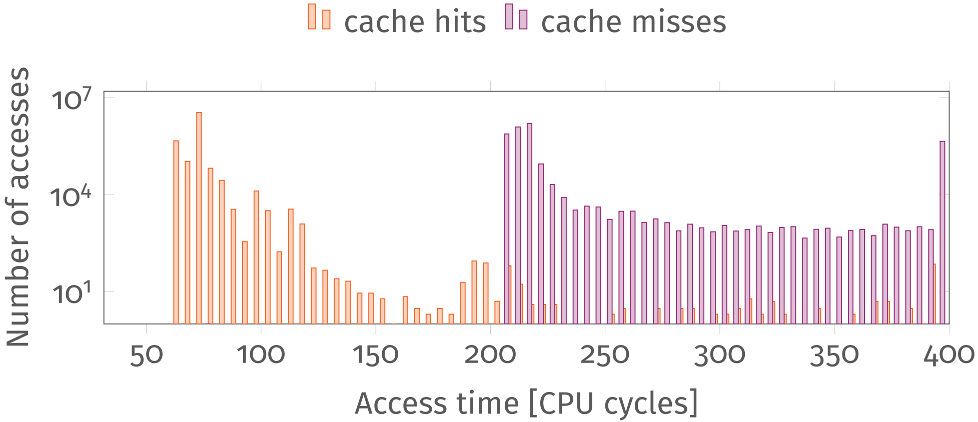


Data loaded in a specific set depending on its address

Several ways per set

Cache line loaded in a specific way depending on the replacement policy

Timing differences



Cache-based covert channels

- cache attacks $\rightarrow$ exploit timing differences of memory accesses

Cache-based covert channels

- cache attacks $\rightarrow$ exploit timing differences of memory accesses
- covert channel: two processes communicating with each other
    - not allowed to do so, e.g., across VMs

Cache-based covert channels

- cache attacks $\rightarrow$ exploit timing differences of memory accesses
- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
- literature: stops working with noise on the machine

Cache-based covert channels

- cache attacks $\rightarrow$ exploit timing differences of memory accesses
- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
- literature: stops working with noise on the machine
- solution? “Just use error-correcting codes”

Prime+Probe

- attacker knows which cache set the victim accessed, not the content

Prime+Probe

- attacker knows which cache set the victim accessed, not the content
- works across CPU cores as the last-level cache is shared

Prime+Probe

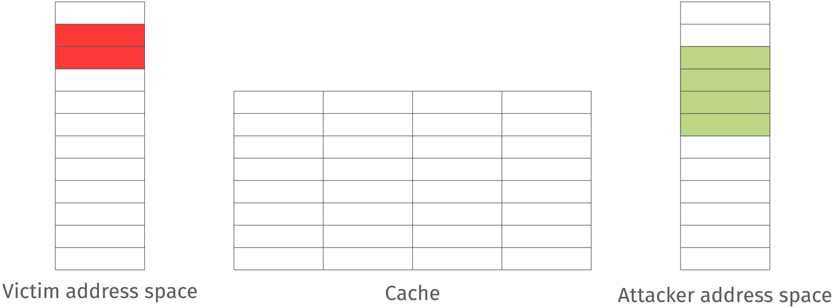
- attacker knows which cache set the victim accessed, not the content
- works across CPU cores as the last-level cache is shared
- does not need shared memory, e.g., memory de-deduplication

Prime+Probe

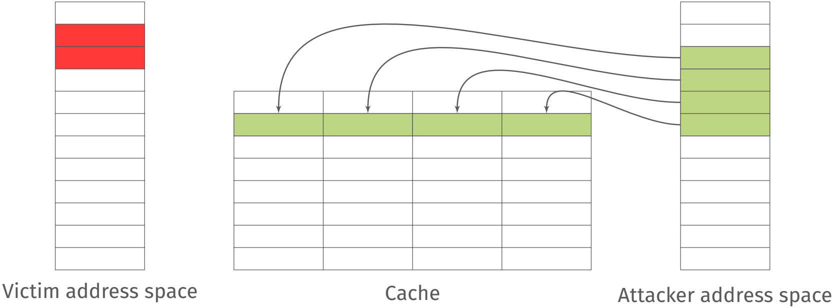
- attacker knows which cache set the victim accessed, not the content
- works across CPU cores as the last-level cache is shared
- does not need shared memory, e.g., memory de-deduplication

→ works across VM in the cloud, e.g., on Amazon EC2

Prime+Probe

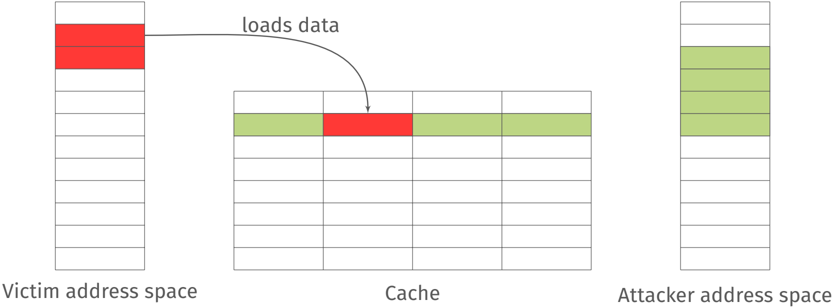


Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

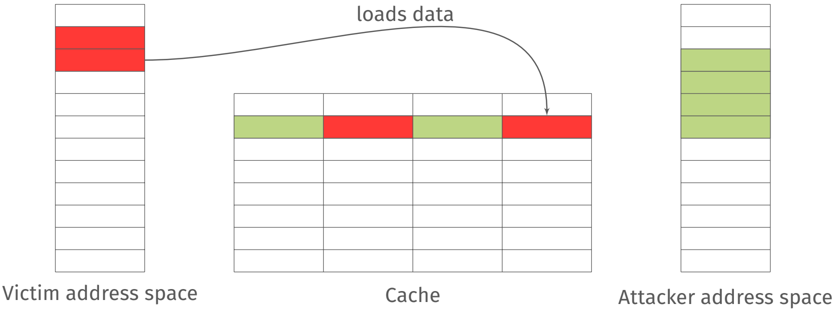
Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

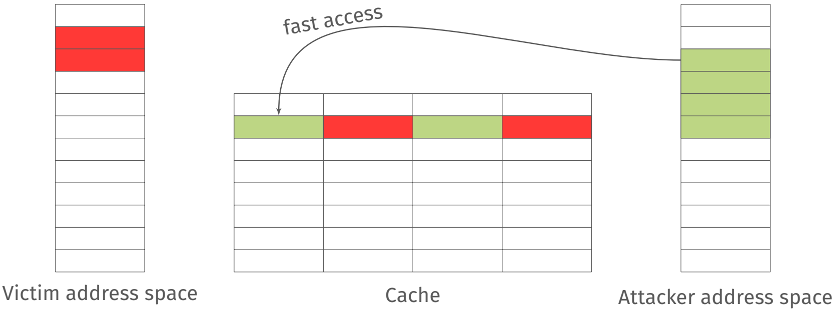
Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Prime+Probe

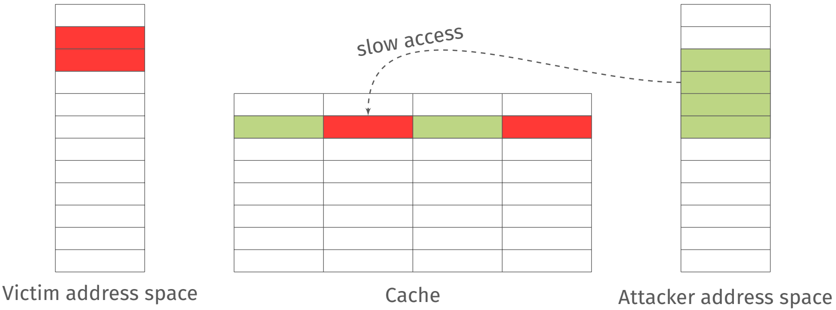


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Step 3: Attacker probes data to determine if set has been accessed

Prime+Probe

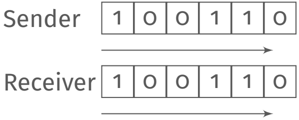


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

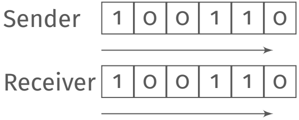
Step 3: Attacker probes data to determine if set has been accessed

Why can’t we just use error correcting codes?

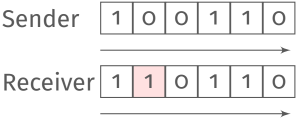


(a) Transmission without errors

Why can't we just use error correcting codes?

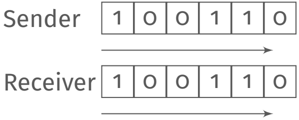


(a) Transmission without errors

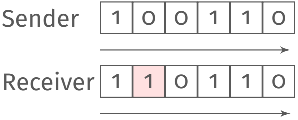


(b) Noise: substitution error

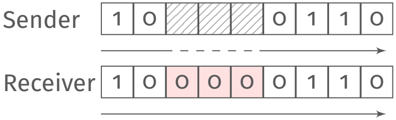
Why can’t we just use error correcting codes?



(a) Transmission without errors

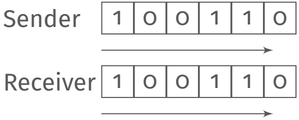


(b) Noise: substitution error

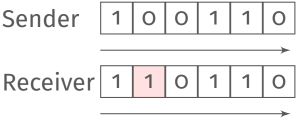


(c) Sender descheduled: insertions

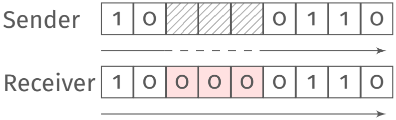
Why can’t we just use error correcting codes?



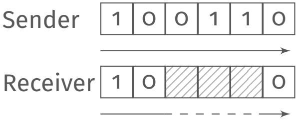
(a) Transmission without errors



(b) Noise: substitution error



(c) Sender descheduled: insertions



(d) Receiver descheduled: deletions

Our robust covert channel

- physical layer:
  - transmits words as a sequence of '0's and '1's
  - deals with synchronization errors
- data-link layer:
  - divides data to transmit into packets
  - corrects the remaining errors

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously
- sender transmits ’0’ doing nothing

  → lines of the receiver still in cache → fast access

Physical layer: Sending ‘0’s and ‘1’s

- sender and receiver agree on one set
- receiver probes the set continuously
- sender transmits ’0’ doing nothing
  - → lines of the receiver still in cache → fast access
- sender transmits ’1’ accessing addresses in the set
  - → evicts lines of the receiver → slow access

Eviction set generation

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

Eviction set generation

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

Eviction set generation

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

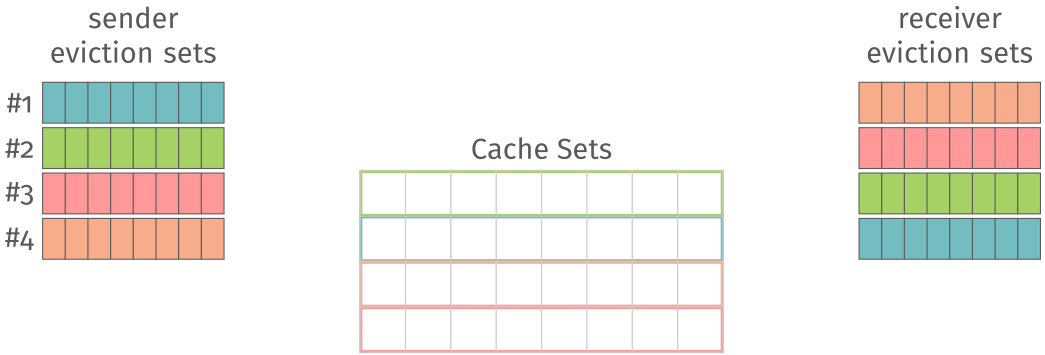
- we can build a set of addresses in the same cache set and same slice

Eviction set generation

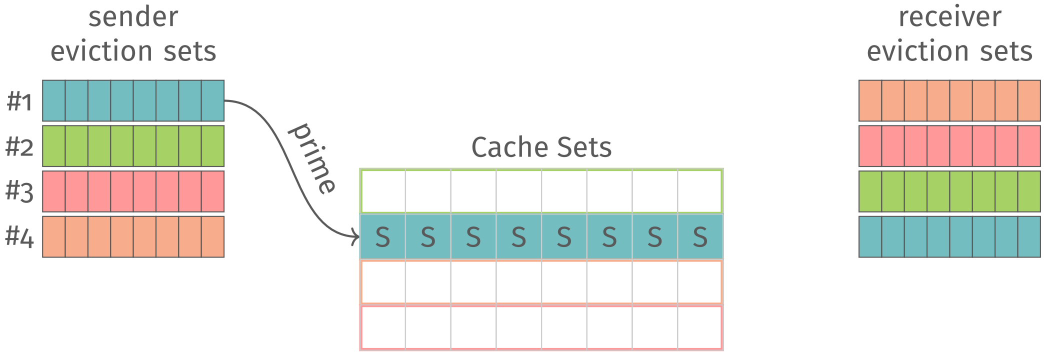
- need a set of addresses in the same cache set and same slice
- problem: slice number depends on all bits of the physical address

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

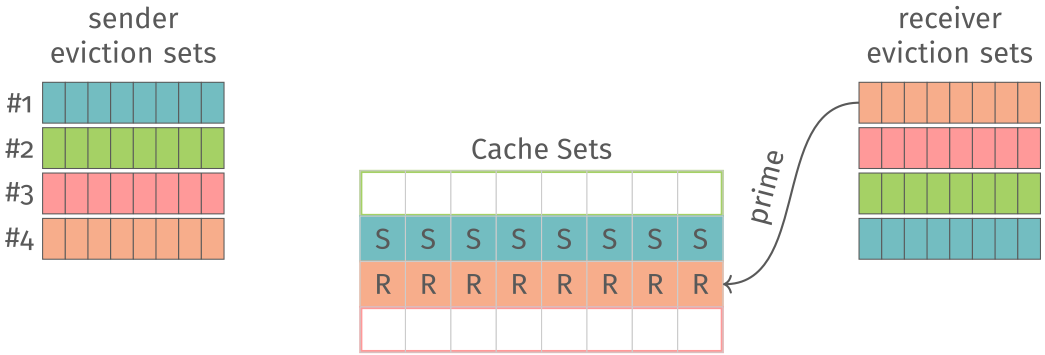
Jamming agreement



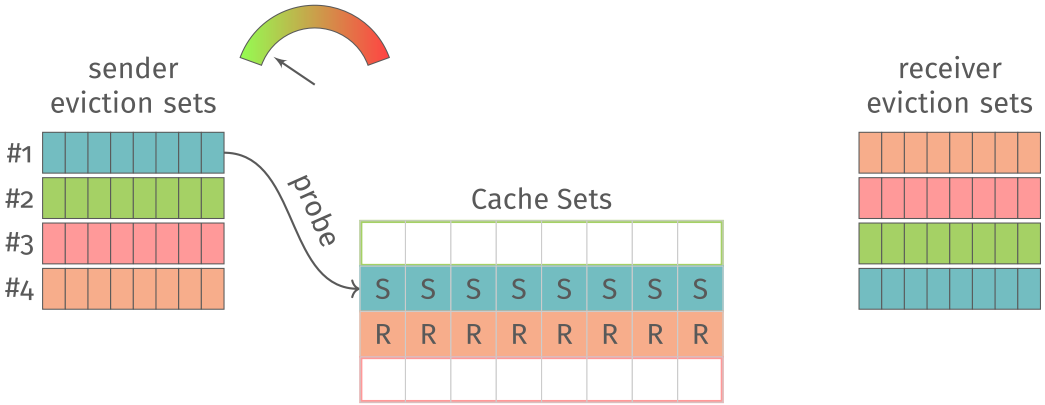
Jamming agreement



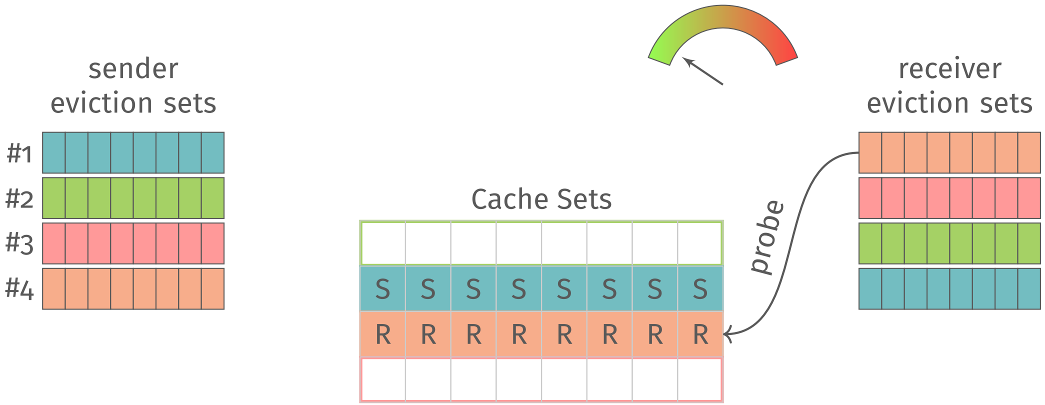
Jamming agreement



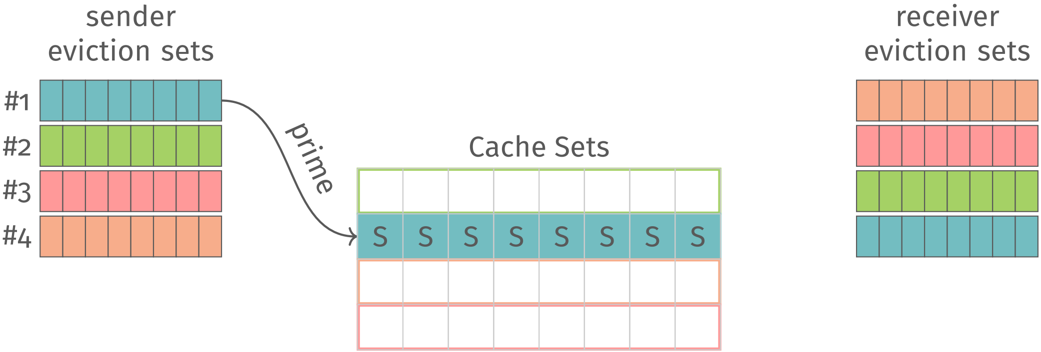
Jamming agreement



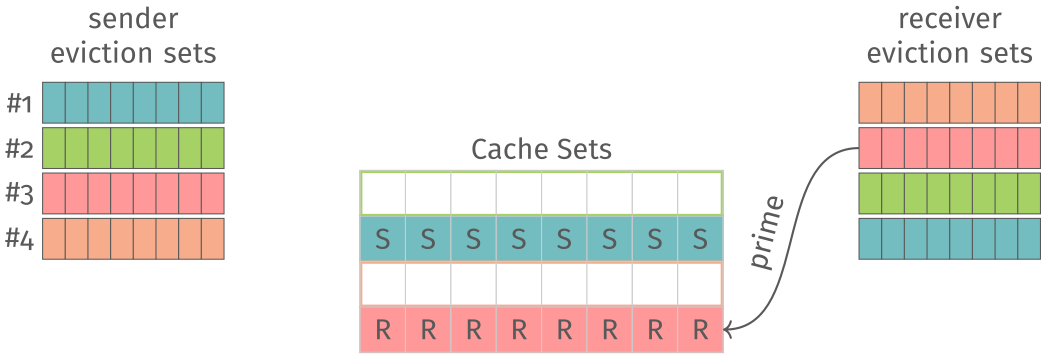
Jamming agreement



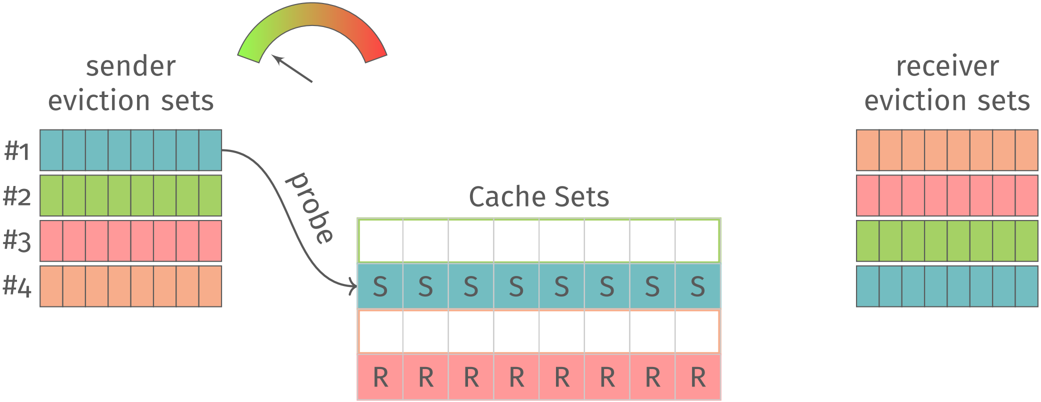
Jamming agreement



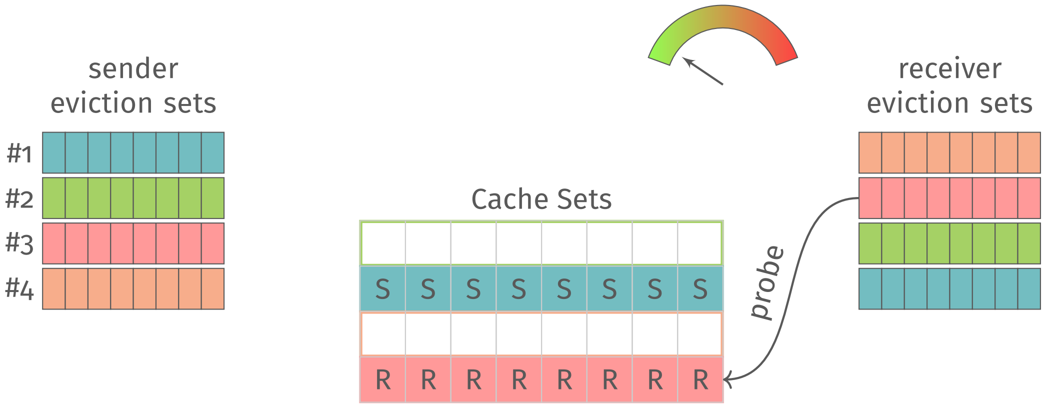
Jamming agreement



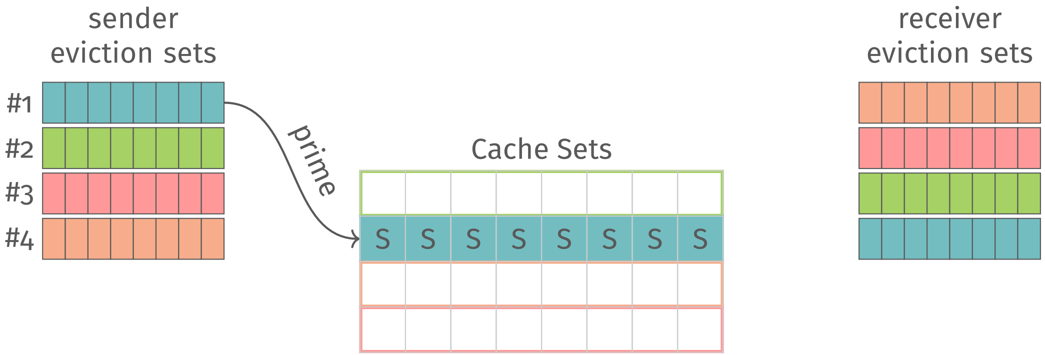
Jamming agreement



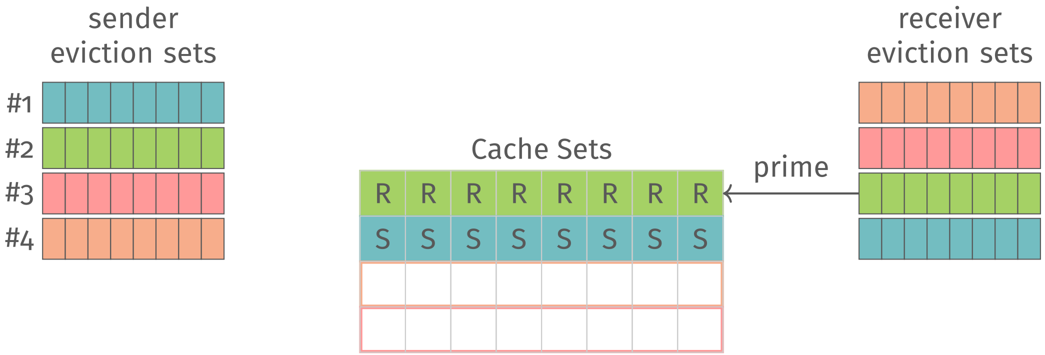
Jamming agreement



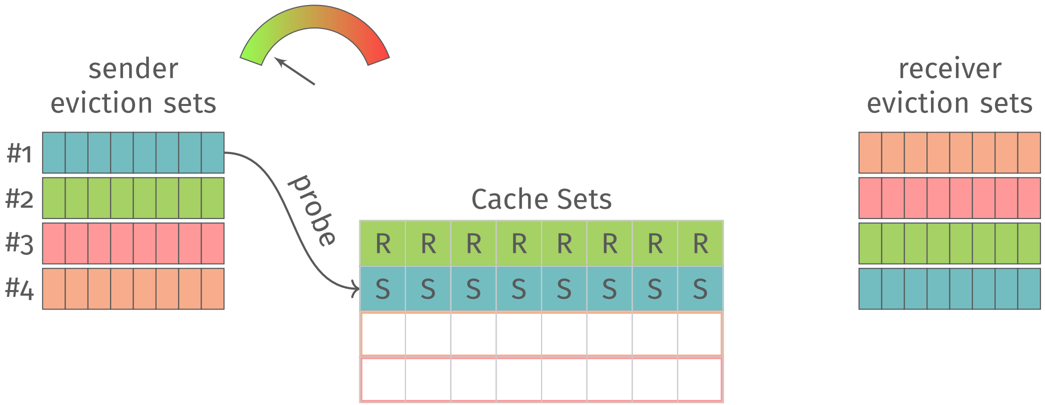
Jamming agreement



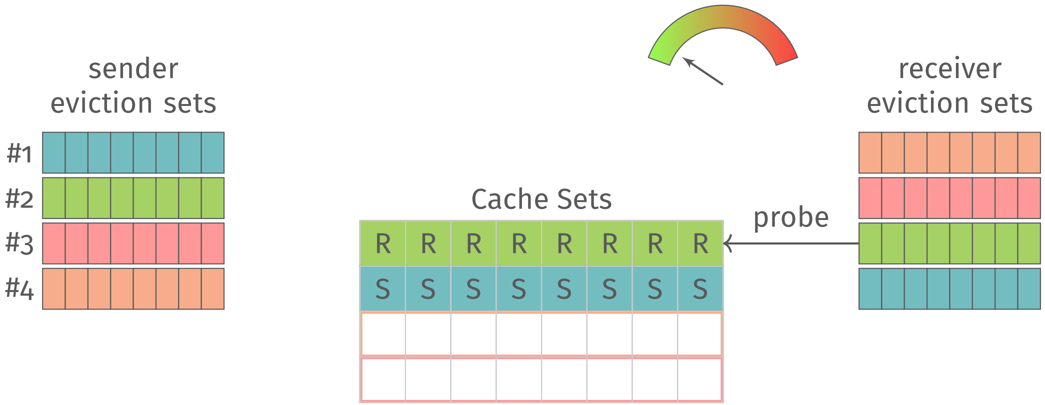
Jamming agreement



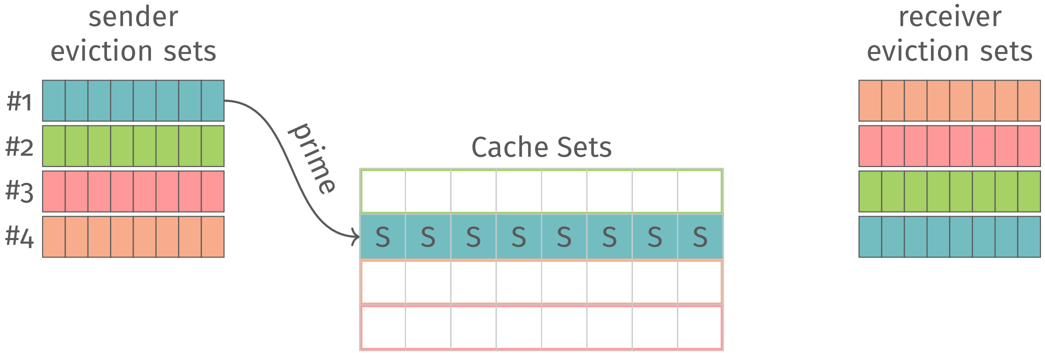
Jamming agreement



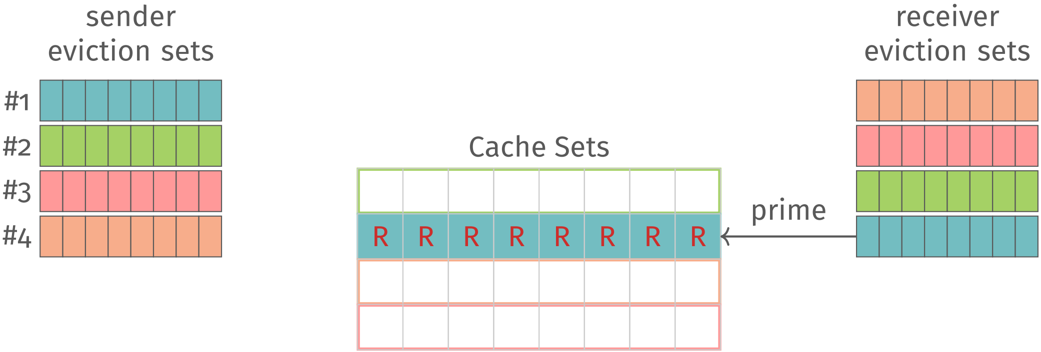
Jamming agreement



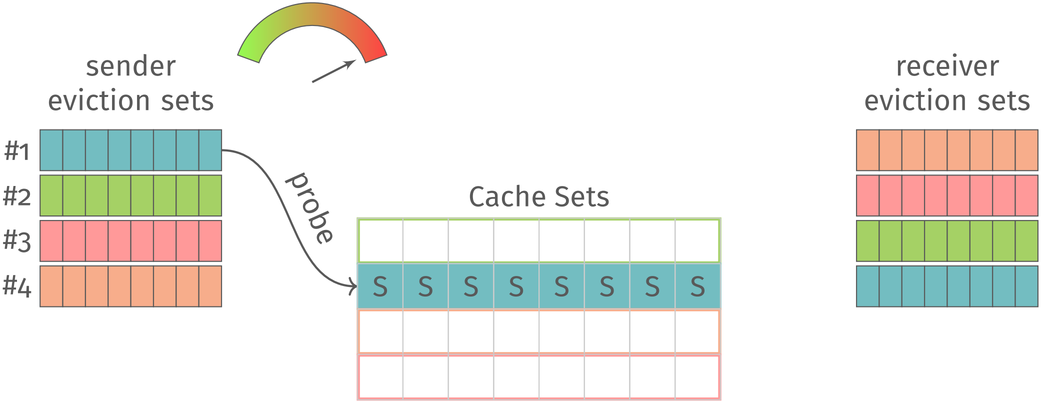
Jamming agreement



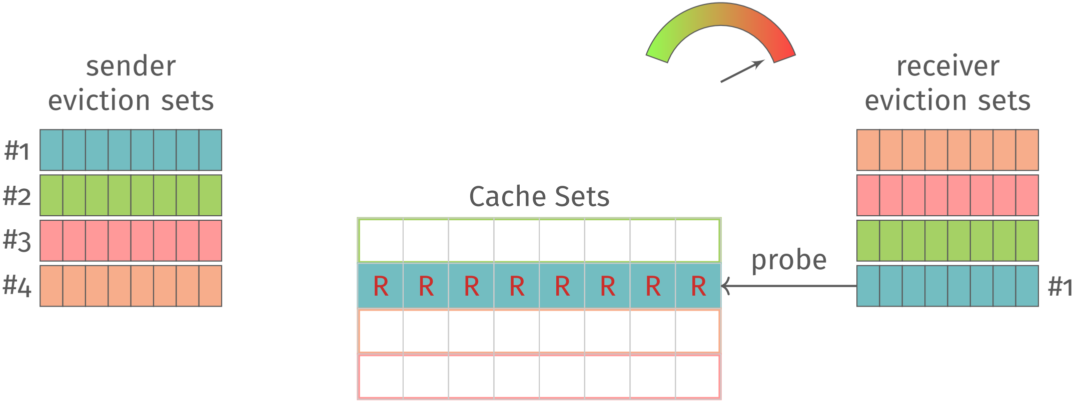
Jamming agreement



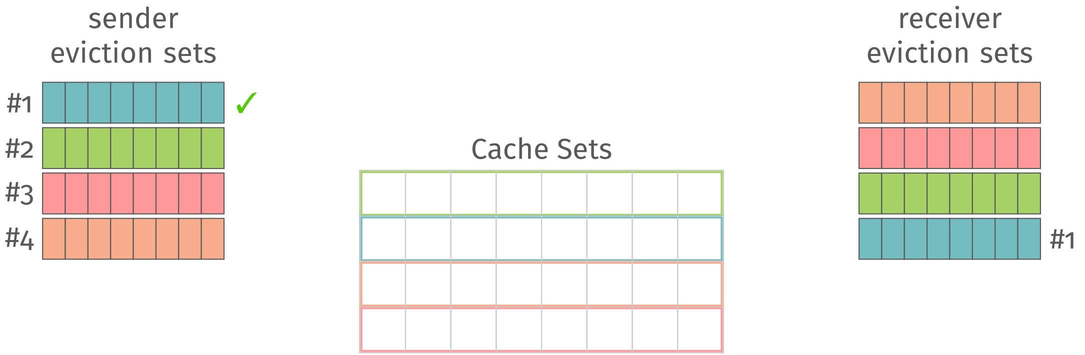
Jamming agreement



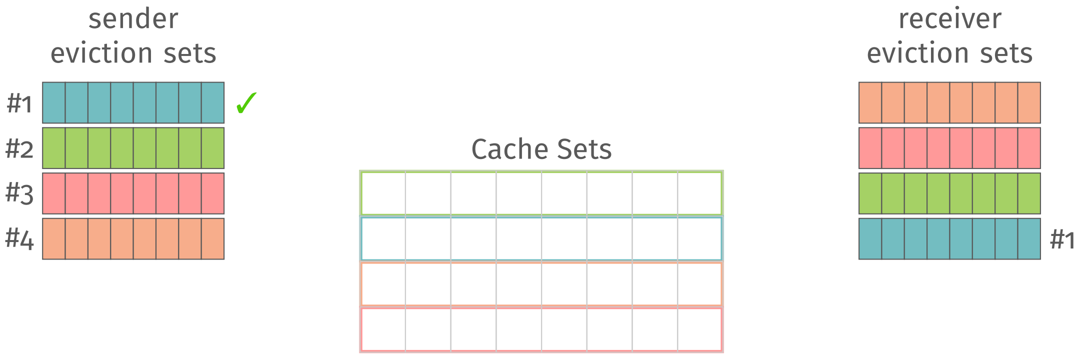
Jamming agreement



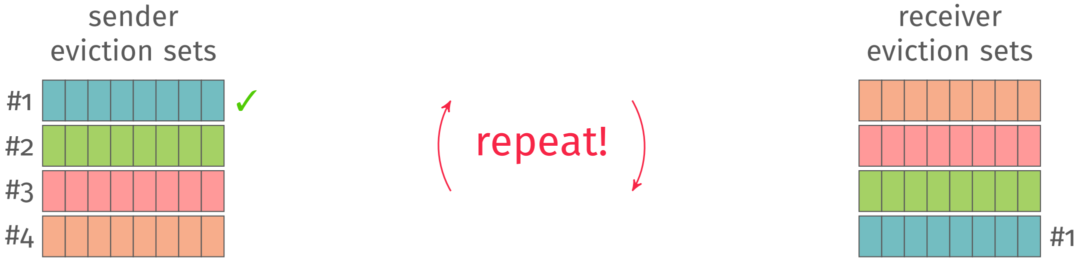
Jamming agreement



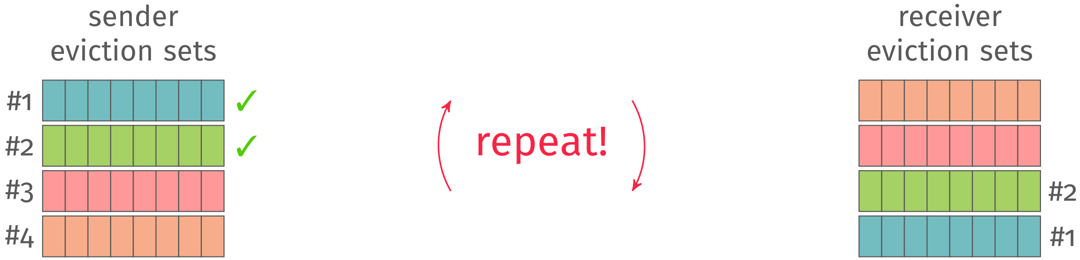
Jamming agreement



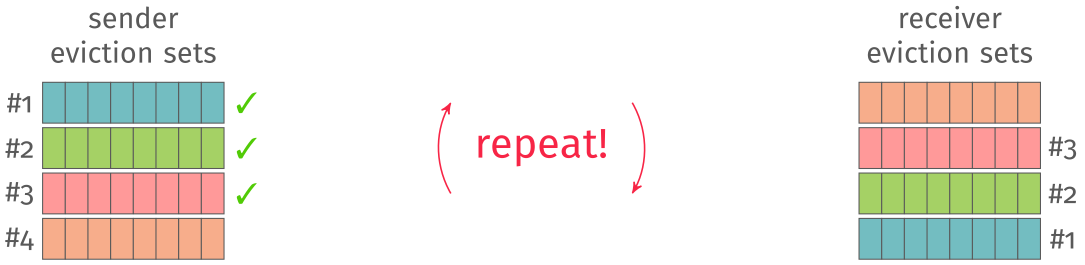
Jamming agreement



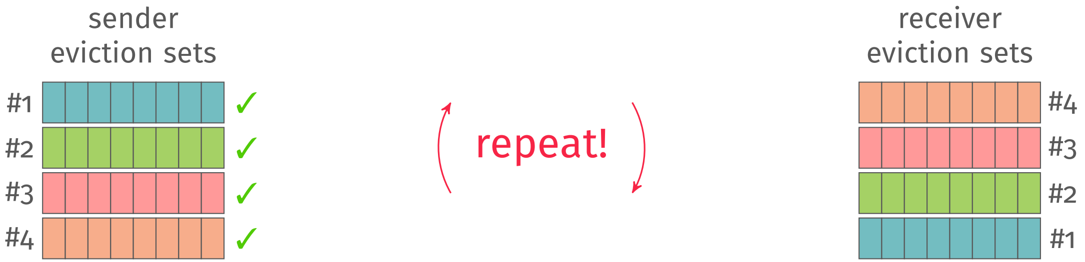
Jamming agreement



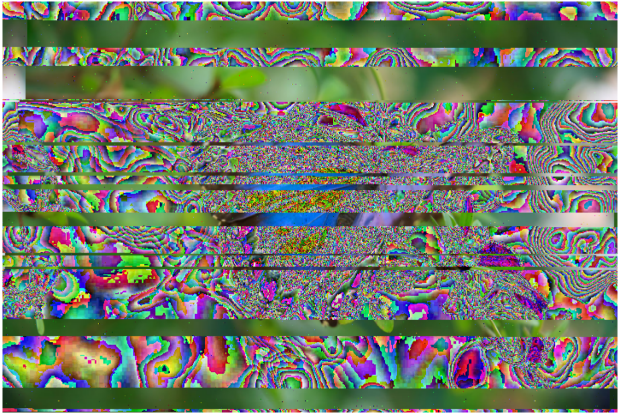
Jamming agreement



Jamming agreement



Sending the first image

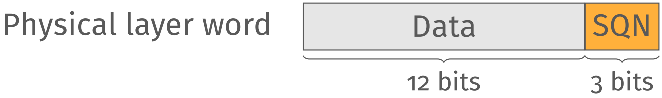


Handling synchronization errors

Physical layer word $\underbrace{\boxed{\text{Data}}}_{\text{12 bits}}$

Handling synchronization errors

- deletion errors: request-to-send scheme that also serves as ack
  - 3-bit sequence number
  - request: encoded sequence number (7 bits)

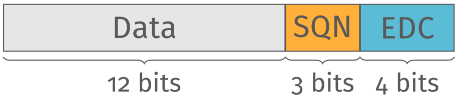


Handling synchronization errors

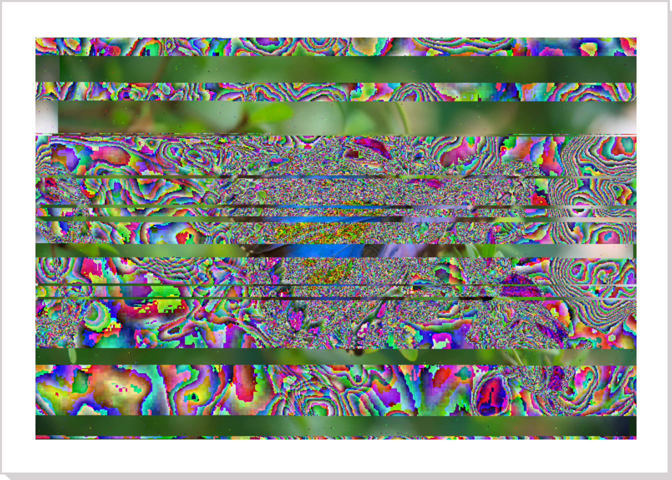
- deletion errors: request-to-send scheme that also serves as ack
  - 3-bit sequence number
  - request: encoded sequence number (7 bits)
- ’0’-insertion errors: error detection code → Berger codes
  - appending the number of ’0’s in the word to itself

  → property: a word cannot consist solely of ’0’s

Physical layer word



Synchronization (before)



Synchronization (after)

Synchronization (after)

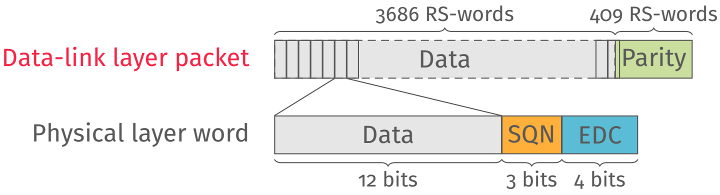
Synchronization (after)

Data-link layer: Error correction

- Reed-Solomon codes to correct the remaining errors

Data-link layer: Error correction

- Reed-Solomon codes to correct the remaining errors
- RS word size = physical layer word size = 12 bits
- packet size = $2^{12} - 1$ = 4095 RS words
- 10% error-correcting code: 409 parity and 3686 data RS words



Error correction (after)

Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–

Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–
Native	36.03 KBps	0.00%	`stress -m 1`

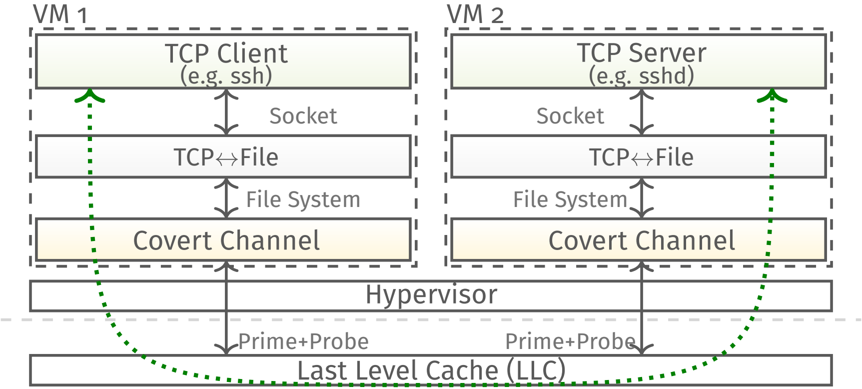
Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–
Native	36.03 KBps	0.00%	`stress -m 1`
Amazon EC2	45.25 KBps	0.00%	–

Evaluation

Environment	Bit rate	Error rate	Noise
Native	75.10 KBps	0.00%	–
Native	36.03 KBps	0.00%	`stress -m 1`
Amazon EC2	45.25 KBps	0.00%	–
Amazon EC2	45.09 KBps	0.00%	web server serving files on sender VM
Amazon EC2	42.96 KBps	0.00%	`stress -m 2` on sender VM
Amazon EC2	42.26 KBps	0.00%	`stress -m 1` on receiver VM
Amazon EC2	37.42 KBps	0.00%	web server on all 3 VMs, `stress -m 4` on 3rd VM, `stress -m 1` on sender and receiver VMs
Amazon EC2	34.27 KBps	0.00%	`stress -m 8` on third VM

Building an SSH connection



SSH evaluation

Between two instances on Amazon EC2

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

SSH evaluation

Between two instances on Amazon EC2

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

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

Conclusion

- cache covert channels are practical

Conclusion

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise

Conclusion

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise
- our robust covert channel supports an SSH connection

Conclusion

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise
- our robust covert channel supports an SSH connection
- we extended Amazon’s product portfolio :)

Conclusion

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise
- our robust covert channel supports an SSH connection
- we extended Amazon's product portfolio :)

Conclusion

- cache covert channels are practical
- even in the cloud, even in presence of extraordinary noise
- our robust covert channel supports an SSH connection
- we extended Amazon’s product portfolio :)

amazon.com

Prime+Probe

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

Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, Kay Römer
Graz University of Technology

February 2017 — NDSS 2017