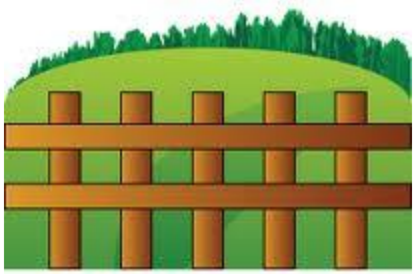
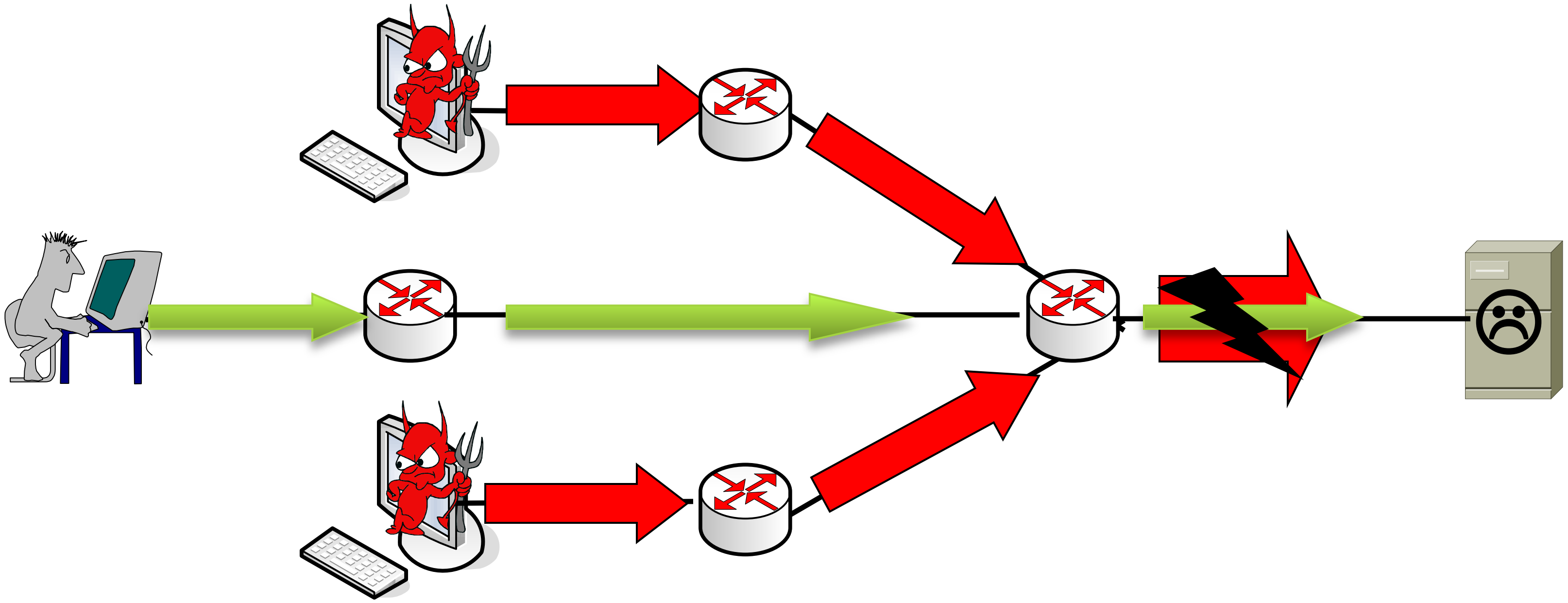


NetFence: Preventing Internet Denial of Service from Inside Out

Xiaowei Yang (Duke University)
with Xin Liu (Duke University)
Yong Xia (NEC Labs China)

Sigcomm 2010
Delhi, India

DoS is a Formidable Threat



- Distributed attacks: many bots send packet floods to exhaust shared resources
  - Bandwidth, memory, or CPU

Largest DDoS Attack - 49 Gigabits Per Second

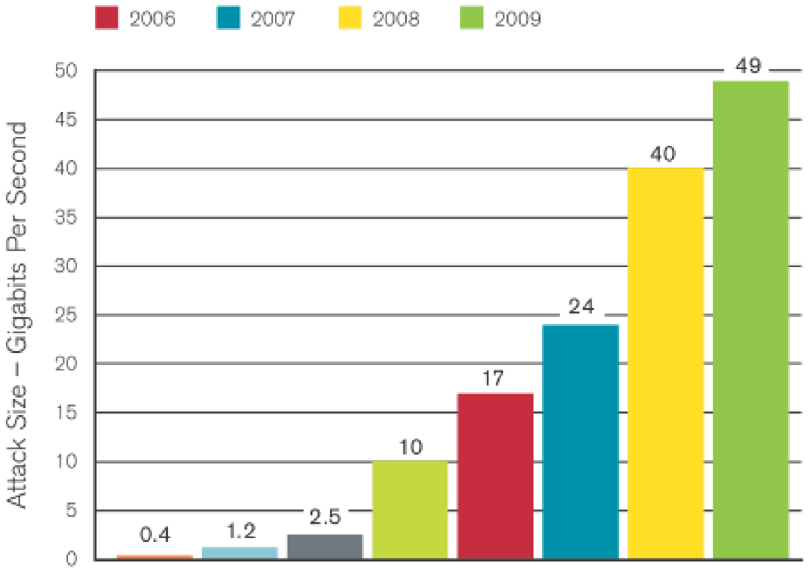


Figure 1: Largest DDoS Attack – 49 Gigabits Per Second

Source: Arbor Networks, Inc.

- *2009 Survey results by Arbor Networks, Inc. among 132 network operators*

Largest Anticipated Threat - Next 12 Months

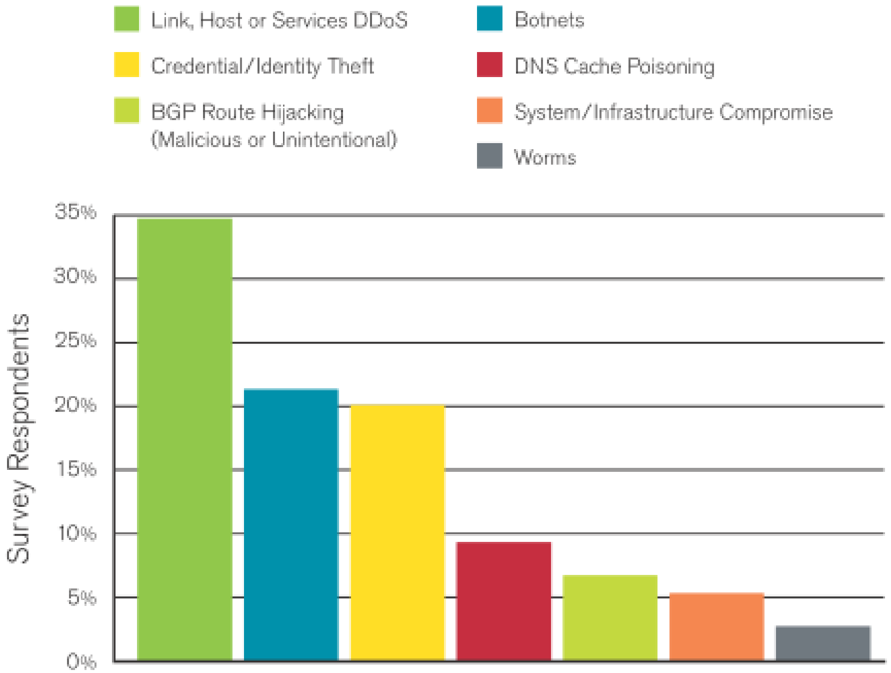


Figure 4: Largest Anticipated Threat – Next 12 Months

Source: Arbor Networks, Inc.

Largest DDoS Attack
- 49 Gigabits Per Second

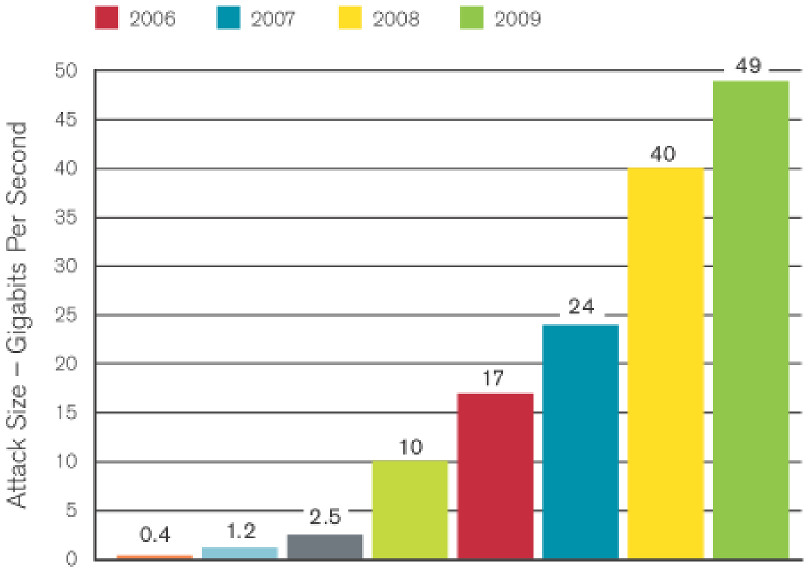


Figure 1: Largest DDoS Attack – 49 Gigabits Per Second

Source: Arbor Networks, Inc.

Largest Anticipated Threat
- Next 12 Months

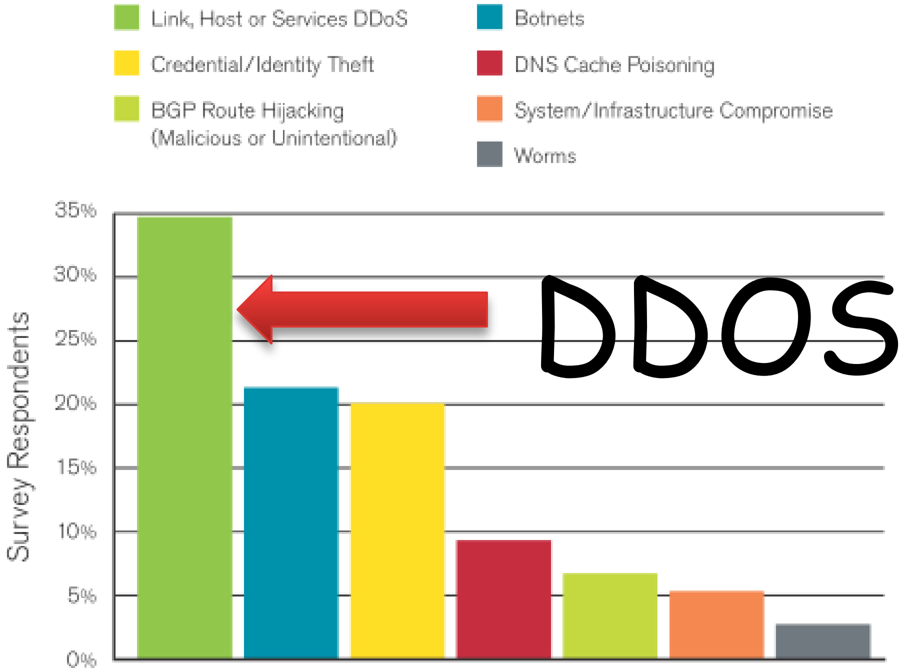


Figure 4: Largest Anticipated Threat – Next 12 Months

Source: Arbor Networks, Inc.

- *2009 Survey results by Arbor Networks, Inc. among 132 network operators*

Largest DDoS Attack
- 49 Gigabits Per Second

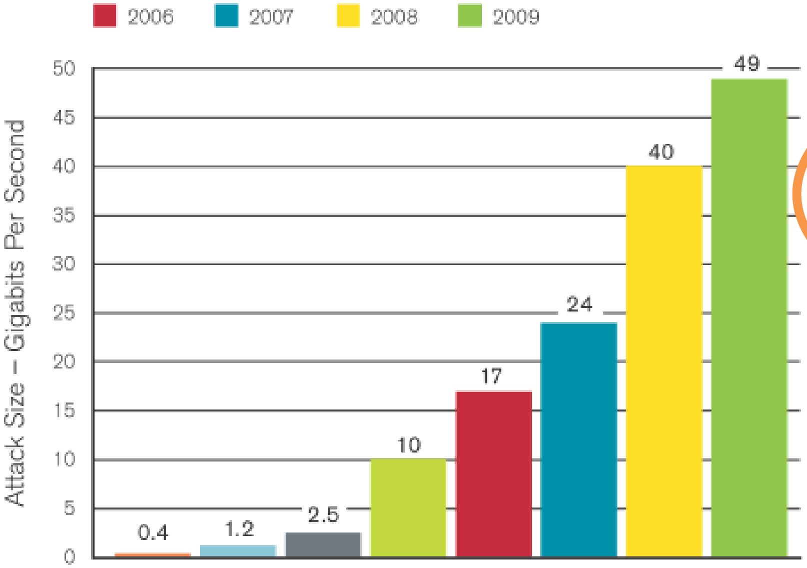


Figure 1: Largest DDoS Attack – 49 Gigabits Per Second

Source: Arbor Networks, Inc.

Largest Anticipated Threat
- Next 12 Months

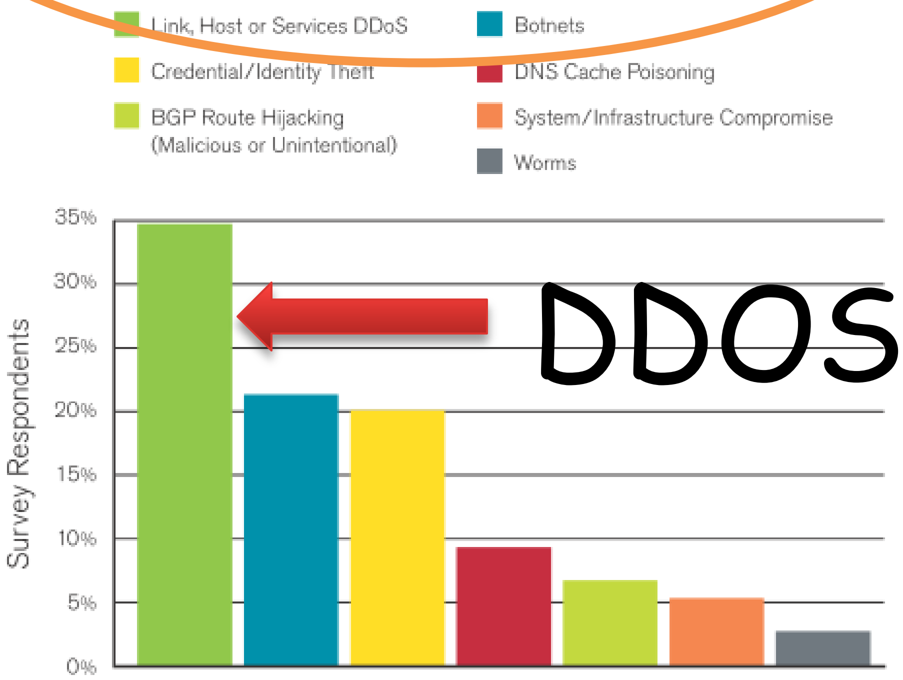


Figure 4: Largest Anticipated Threat – Next 12 Months

Source: Arbor Networks, Inc.

- *2009 Survey results by Arbor Networks, Inc. among 132 network operators*

Largest DDoS Attack
- 49 Gigabits Per Second

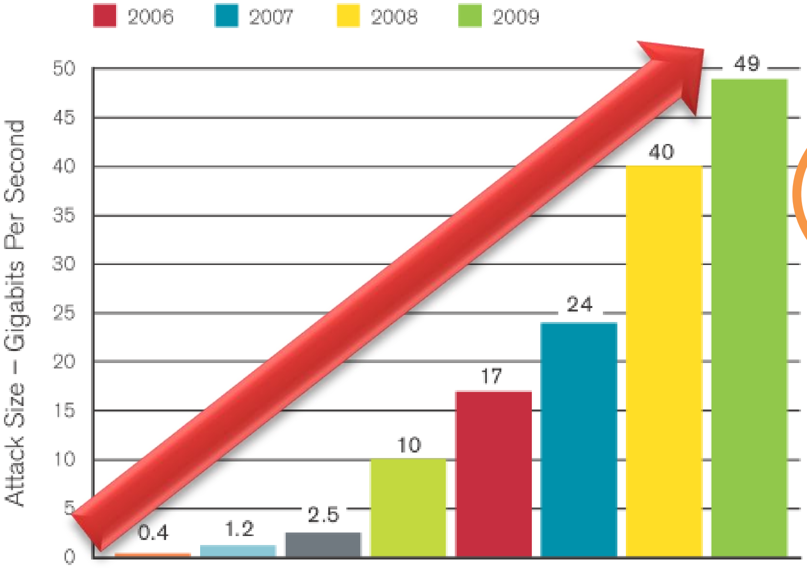
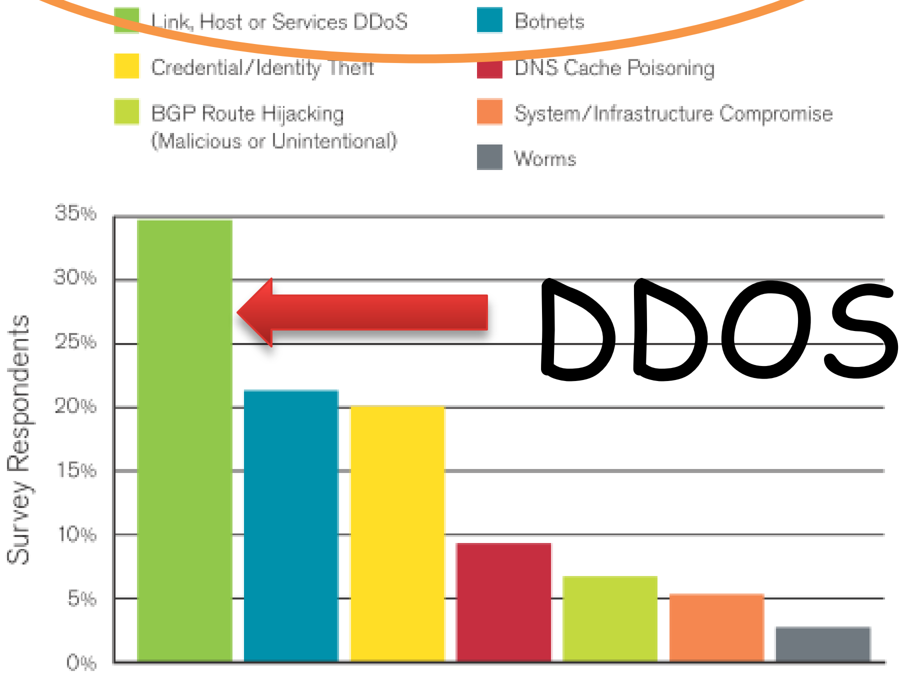


Figure 1: Largest DDoS Attack – 49 Gigabits Per Second

Source: Arbor Networks, Inc.

- *2009 Survey results by Arbor Networks, Inc. among 132 network operators*

Largest Anticipated Threat
- Next 12 Months



DDOS

Figure 4: Largest Anticipated Threat – Next 12 Months

Source: Arbor Networks, Inc.

Largest DDoS Attack - 49 Gigabits Per Second

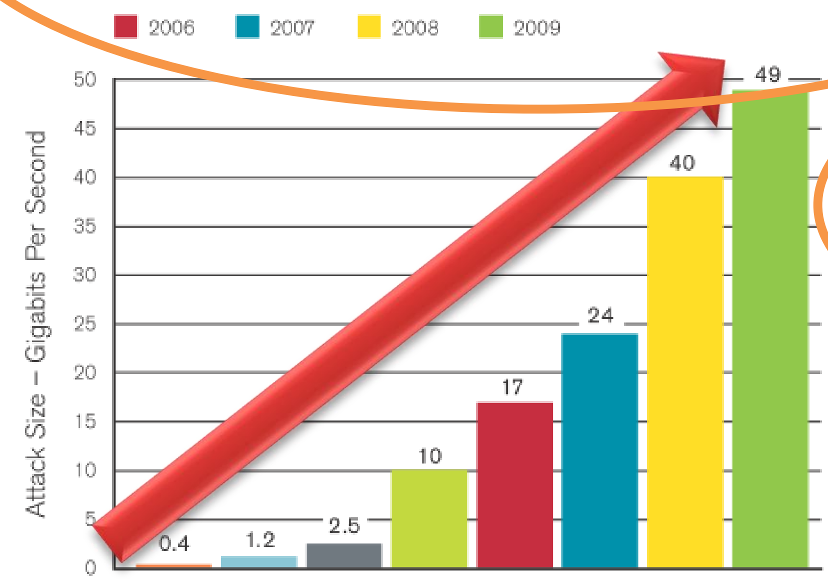
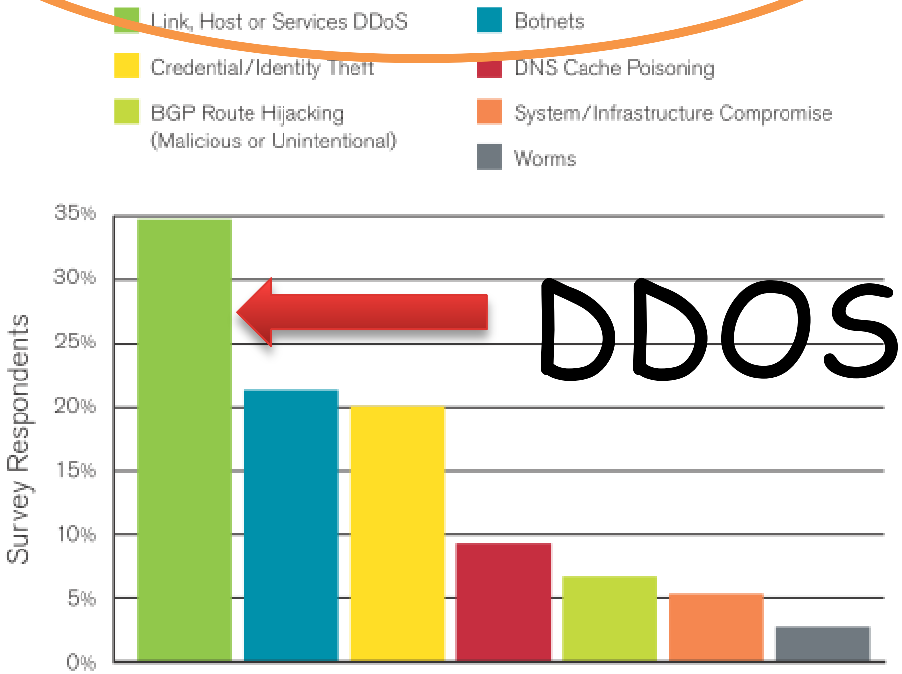


Figure 1: Largest DDoS Attack – 49 Gigabits Per Second

Source: Arbor Networks, Inc.

Largest Anticipated Threat - Next 12 Months



DDOS

Figure 4: Largest Anticipated Threat – Next 12 Months

Source: Arbor Networks, Inc.

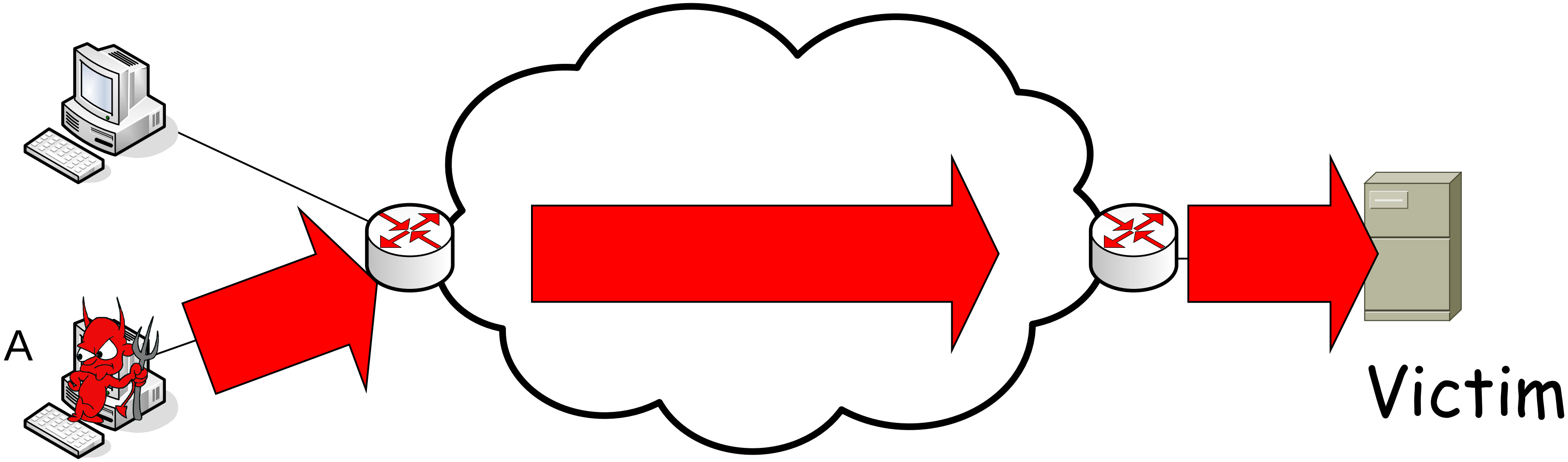
- *2009 Survey results by Arbor Networks, Inc. among 132 network operators*

Combating DoS is Difficult

- A fundamental architecture problem
  1. **Open**: Any to any communication, and new applications
  2. **Robust**: Non-disrupted communications despite compromised hosts and routers

- DoS defense must be built inside out
  - Rethinking the Internet architecture

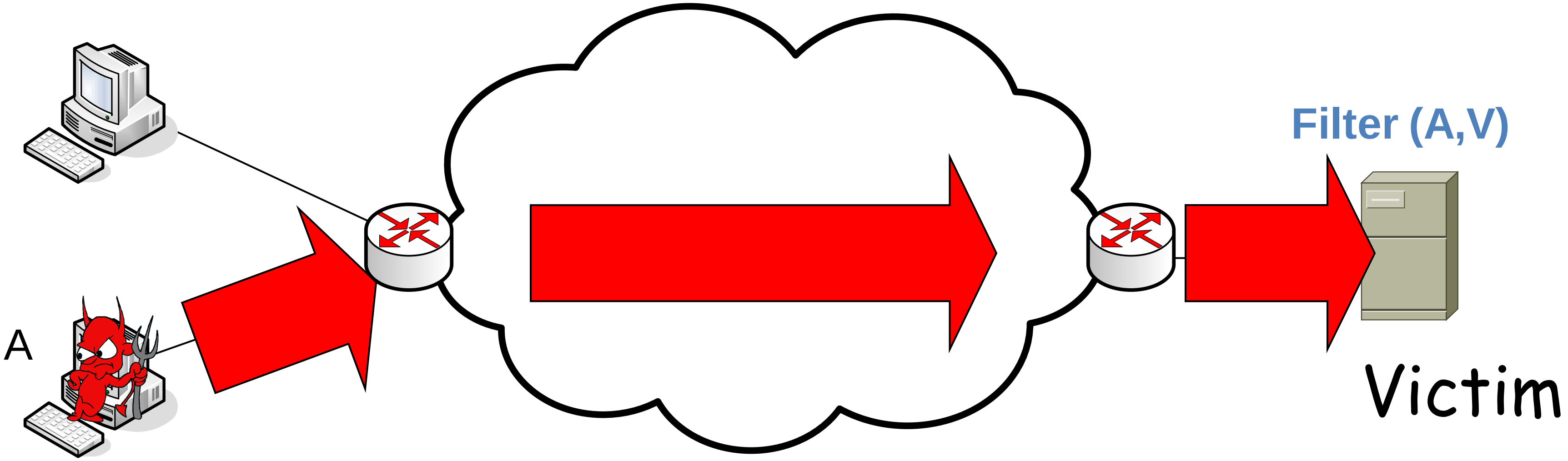
Previous Work: Receivers as Victims



- Much work: AIP, AITF, CenterTrack, dFence, Defense-by-Offense, FastPass, Flow-Cookies, Kill-a-Bot, LazySusan, Mayday, OverDoSe, PacketSymmetry, Phalanx, Pushback, Portcullis, SIFF, SOS, SpeakUp, StopIt, TVA…

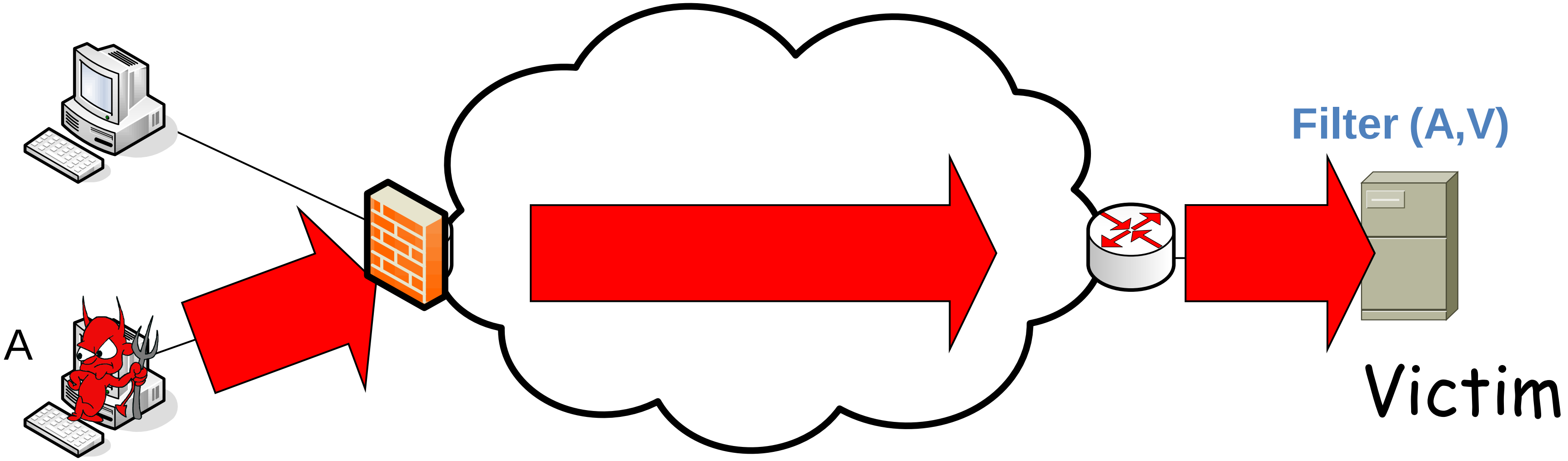
- Denial of Edge Service (DoES)
  - Enable receivers to suppress unwanted traffic
  - Network filters, network capabilities

Previous Work: Receivers as Victims



- Much work: AIP, AITF, CenterTrack, dFence, Defense-by-Offense, FastPass, Flow-Cookies, Kill-a-Bot, LazySusan, Mayday, OverDoSe, PacketSymmetry, Phalanx, Pushback, Portcullis, SIFF, SOS, SpeakUp, StopIt, TVA…
- Denial of Edge Service (DoES)
  - Enable receivers to suppress unwanted traffic
  - Network filters, network capabilities

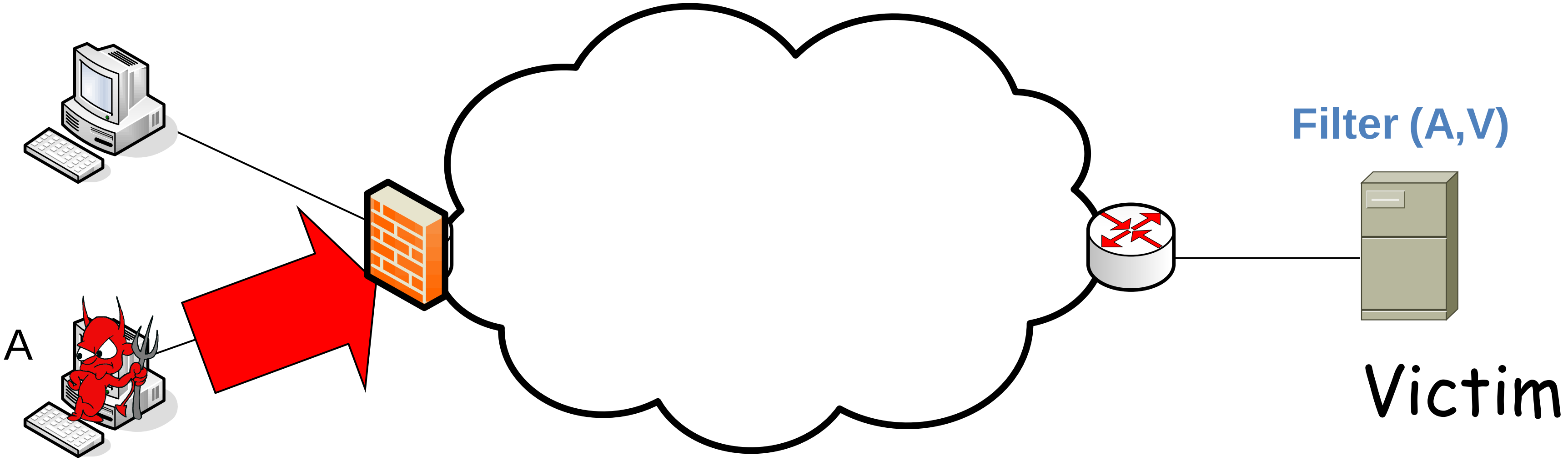
Previous Work: Receivers as Victims



- Much work: AIP, AITF, CenterTrack, dFence, Defense-by-Offense, FastPass, Flow-Cookies, Kill-a-Bot, LazySusan, Mayday, OverDoSe, PacketSymmetry, Phalanx, Pushback, Portcullis, SIFF, SOS, SpeakUp, StopIt, TVA…

- Denial of Edge Service (DoES)
  - Enable receivers to suppress unwanted traffic
  - Network filters, network capabilities

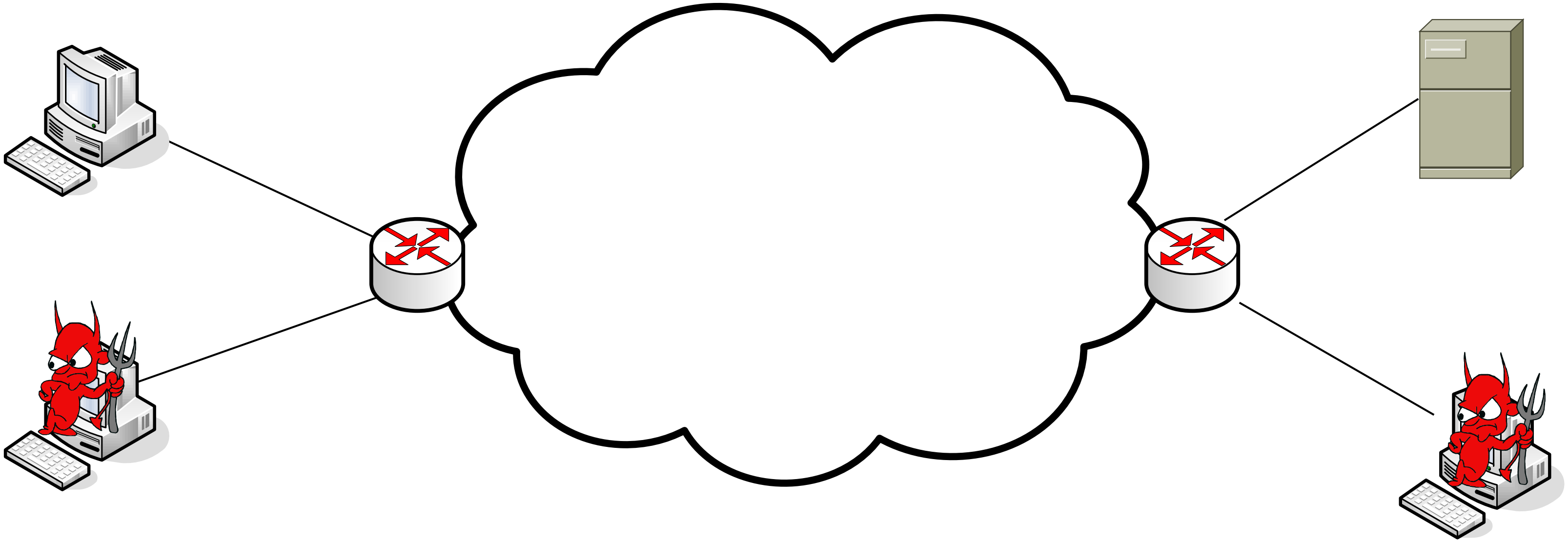
Previous Work: Receivers as Victims



- Much work: AIP, AITF, CenterTrack, dFence, Defense-by-Offense, FastPass, Flow-Cookies, Kill-a-Bot, LazySusan, Mayday, OverDoSe, PacketSymmetry, Phalanx, Pushback, Portcullis, SIFF, SOS, SpeakUp, StopIt, TVA…

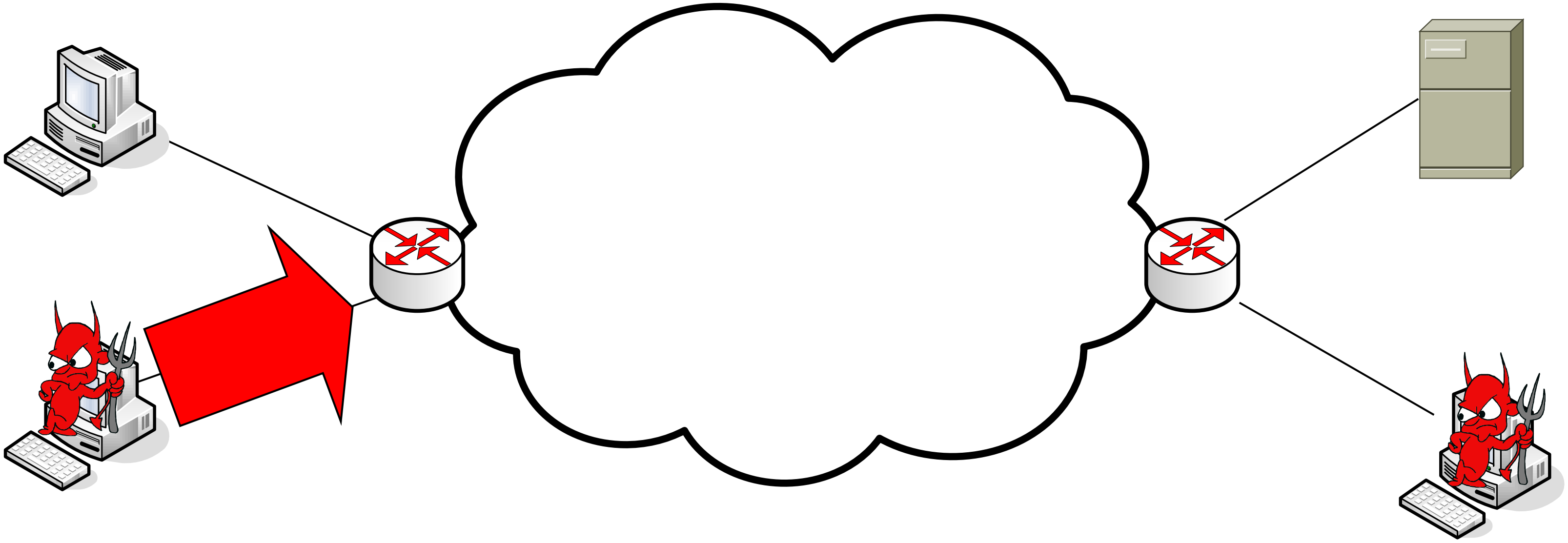
- Denial of Edge Service (DoES)
  - Enable receivers to suppress unwanted traffic
  - Network filters, network capabilities

New Threat: Denial of Network Service (DoNS)



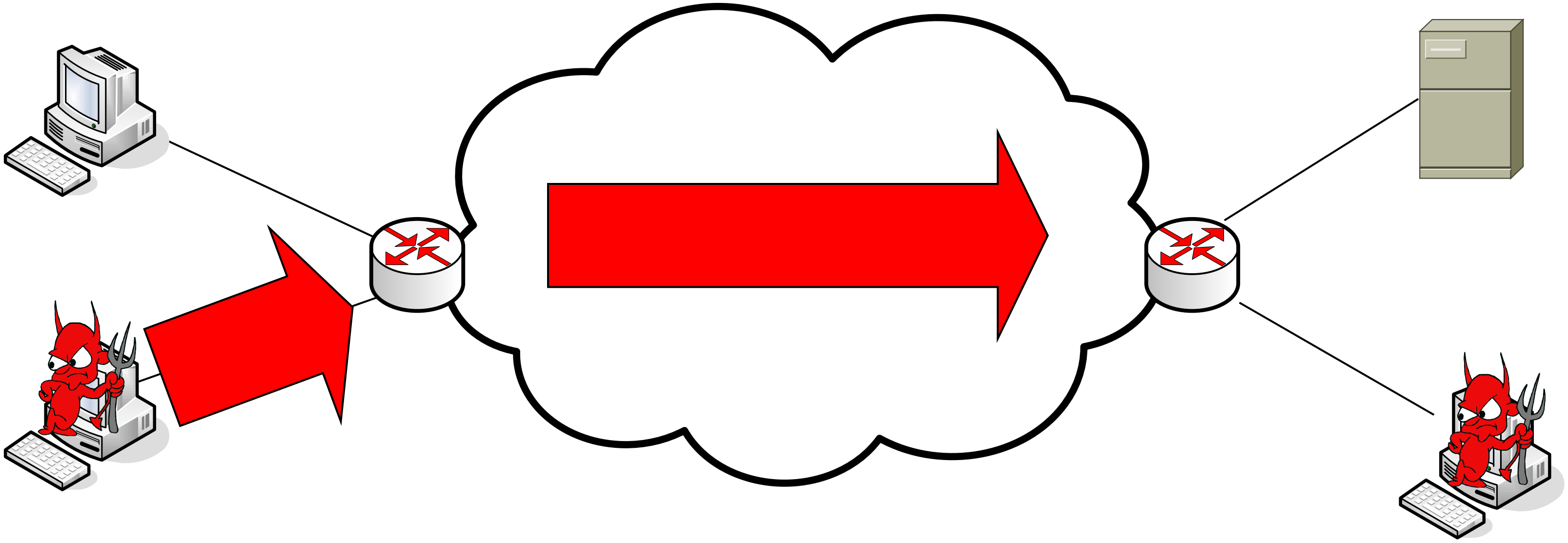
- Bots can collude to send packet floods
- Incapable of identifying attack traffic

New Threat: Denial of Network Service (DoNS)



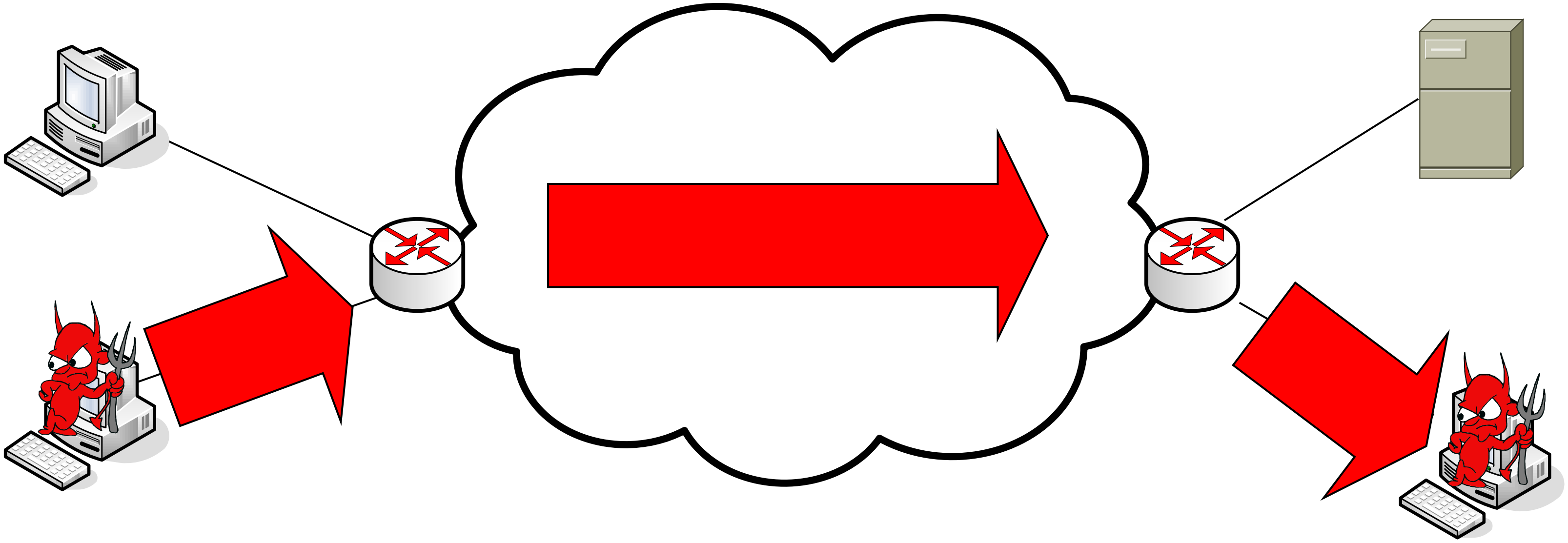
- Bots can collude to send packet floods
- Incapable of identifying attack traffic

New Threat: Denial of Network Service (DoNS)



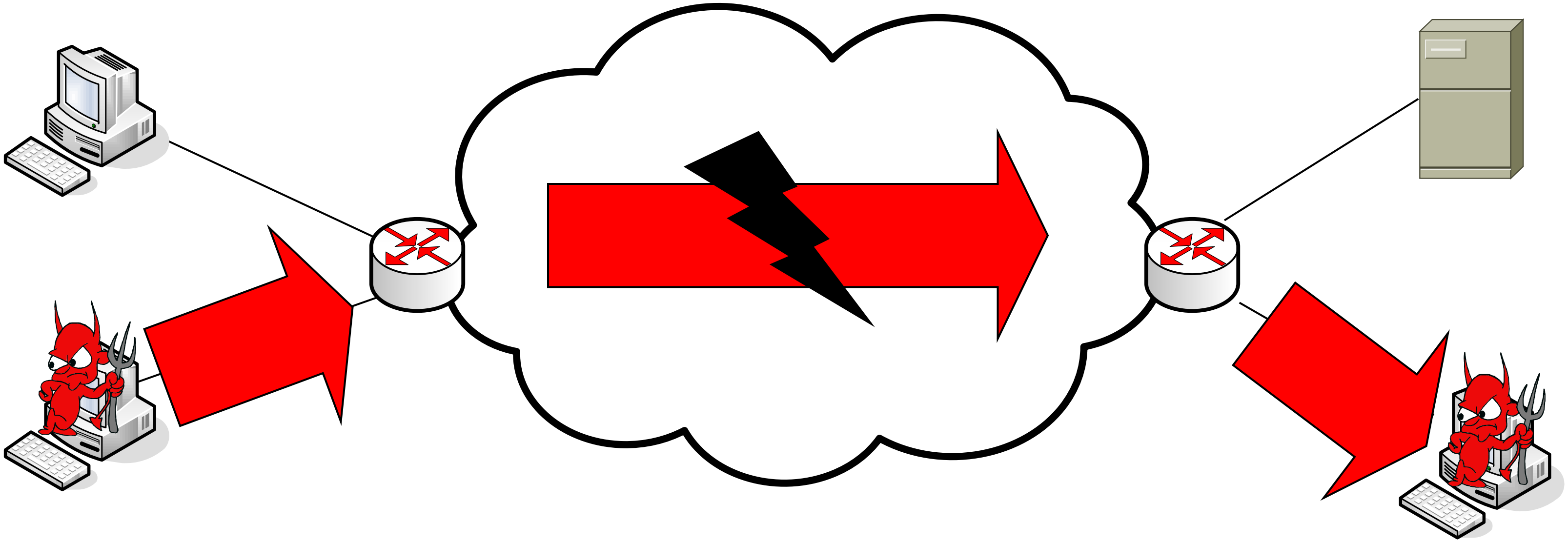
- Bots can collude to send packet floods
- Incapable of identifying attack traffic

New Threat: Denial of Network Service (DoNS)



- Bots can collude to send packet floods
- Incapable of identifying attack traffic

New Threat: Denial of Network Service (DoNS)



- Bots can collude to send packet floods
- Incapable of identifying attack traffic

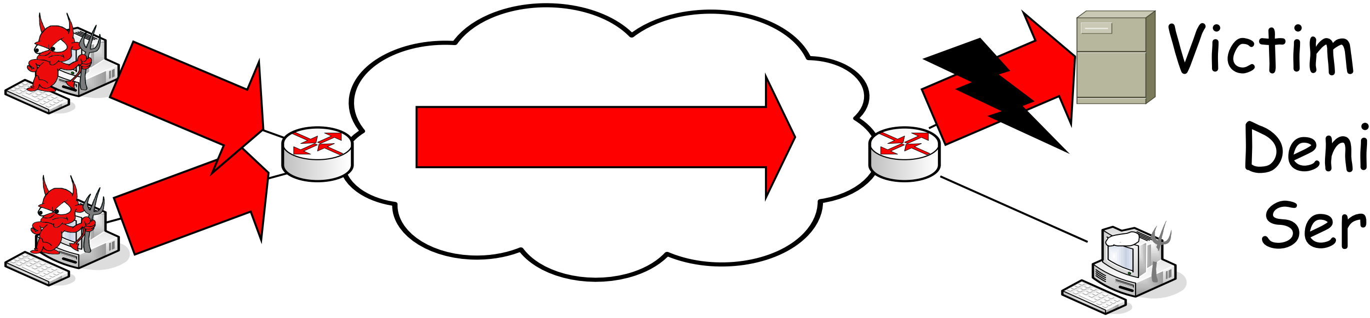
DoS

DoS

||

DoS

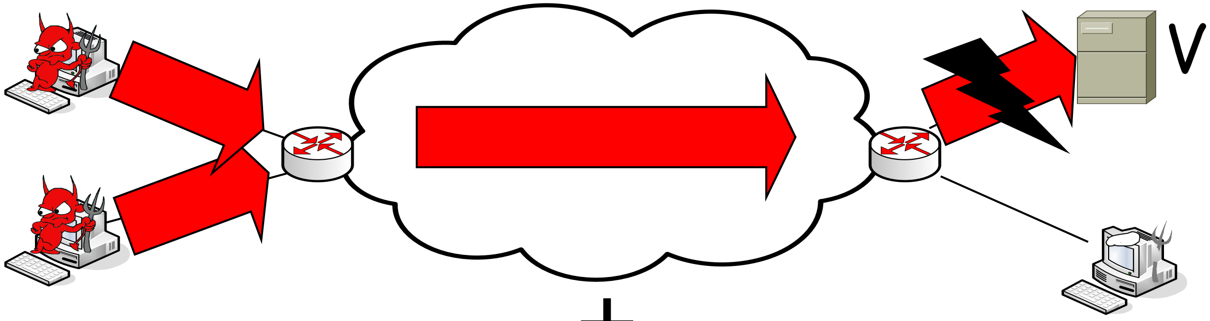
||



Denial of Edge Service (DoES)

DoS

=



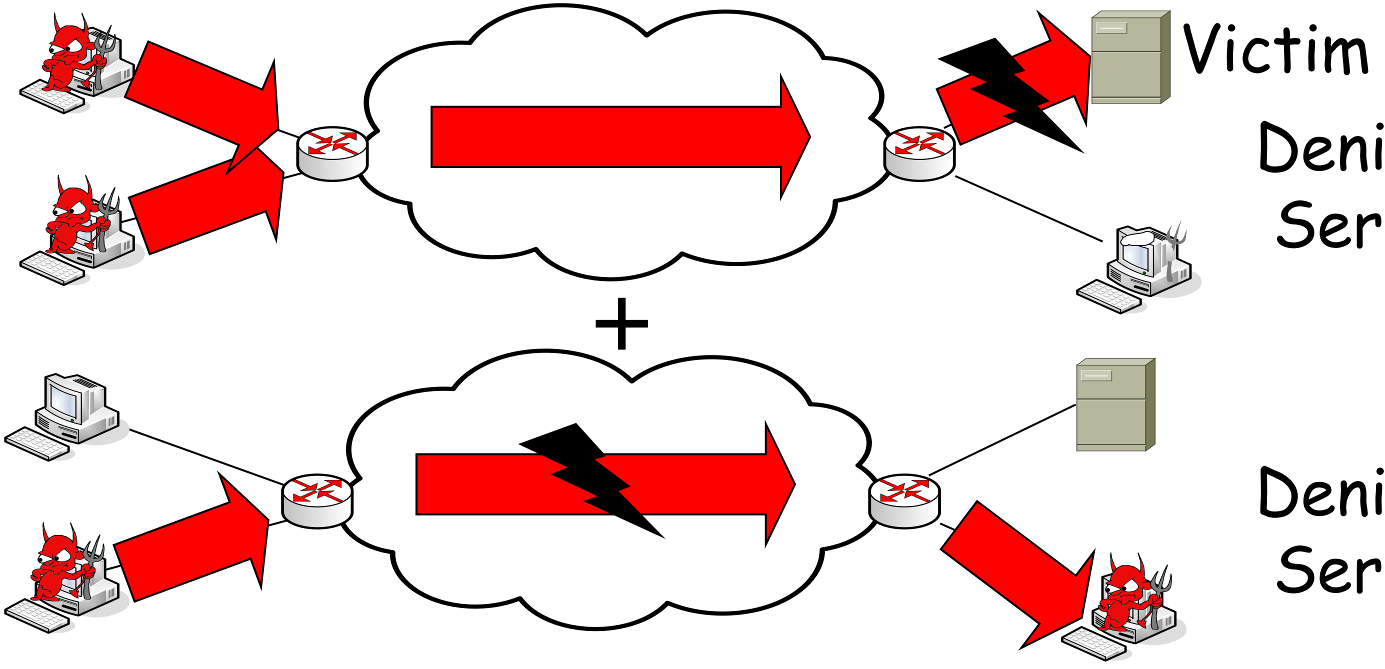
Victim

Denial of Edge Service (DoES)

+

DoS

=



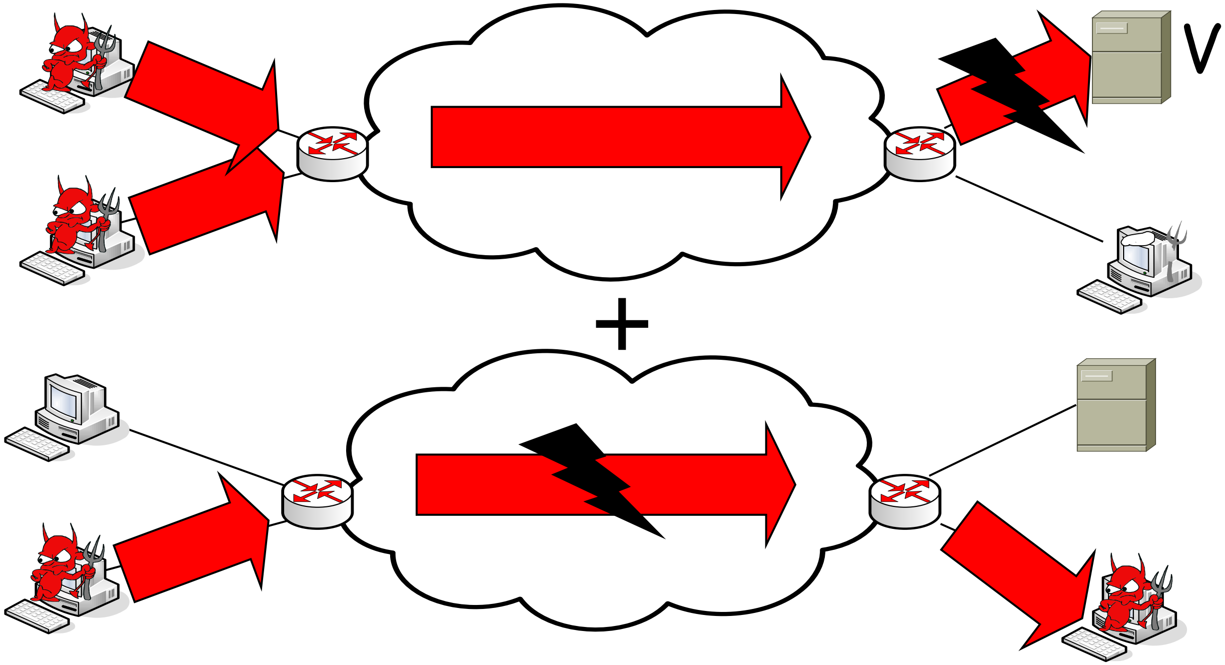
Denial of Edge Service (DoES)

+

Denial of Network Service (DoNS)

DoS

=



Victim

Denial of Edge Service (DoES)

+

Denial of Network Service (DoNS)

How can we design a network architecture that can combat both DoES and DoNS?

Solution: NetFence

- Design principle: inside-out, network-host joint lines of defense

  1. Network controls its resource allocation
     - Combating DoNS

  2. End systems controls what they receive
     - Combating DoES

Key Idea

1. Hierarchical,

+

2. Secure congestion policing in the network

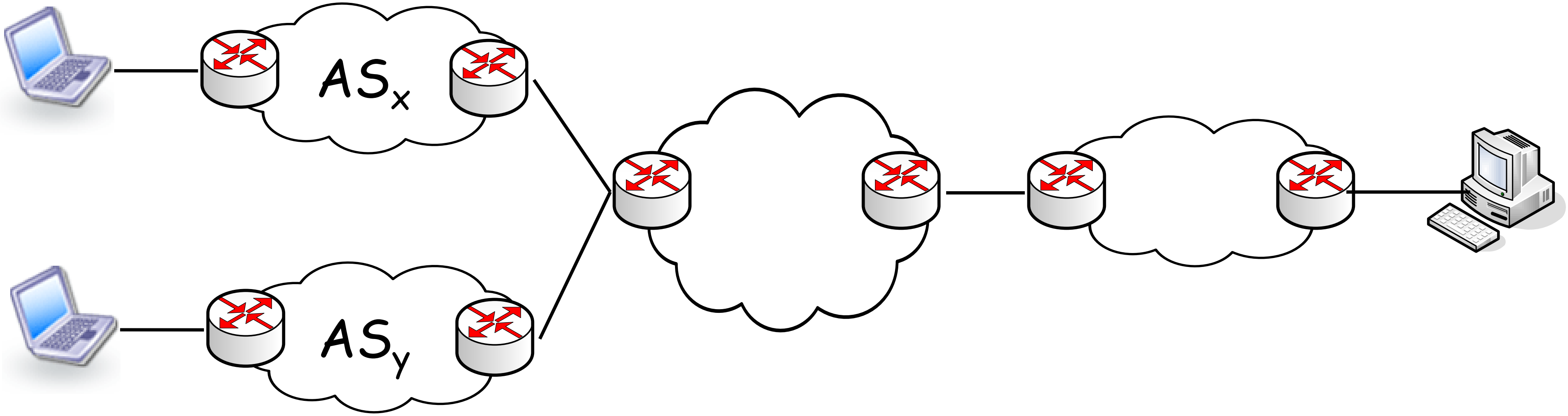
+

3. Coupled with network capabilities

⇓

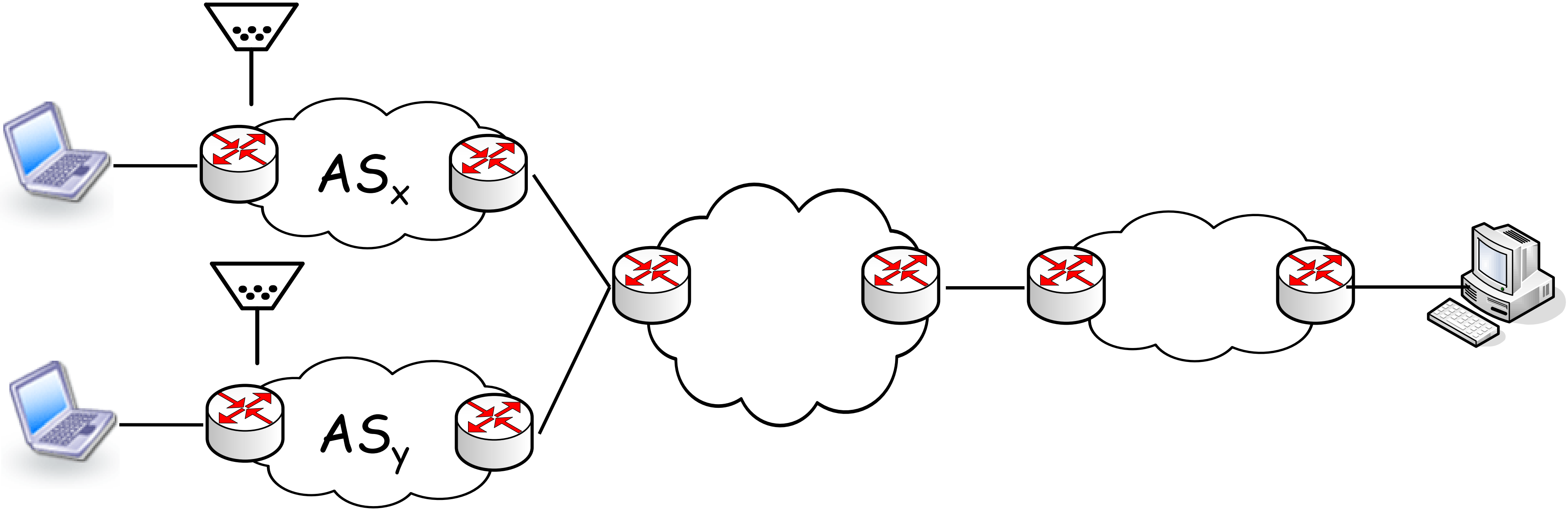
Goals: Scalable, Robust, Open

Hierarchical Congestion Policing



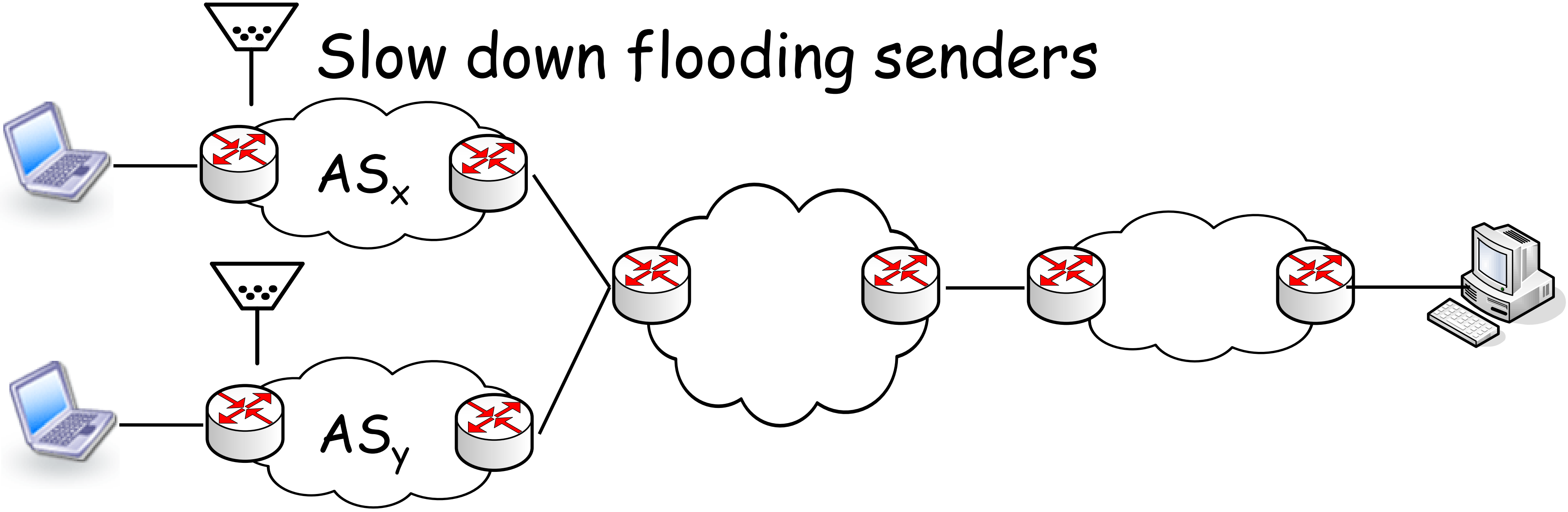
- Scalable: no per-flow state in the core
  1. Aggregate flow policing placed at edge routers [CSFQ]
  2. AS-level policing in the core
     - Fair queuing or rate limiting

Hierarchical Congestion Policing



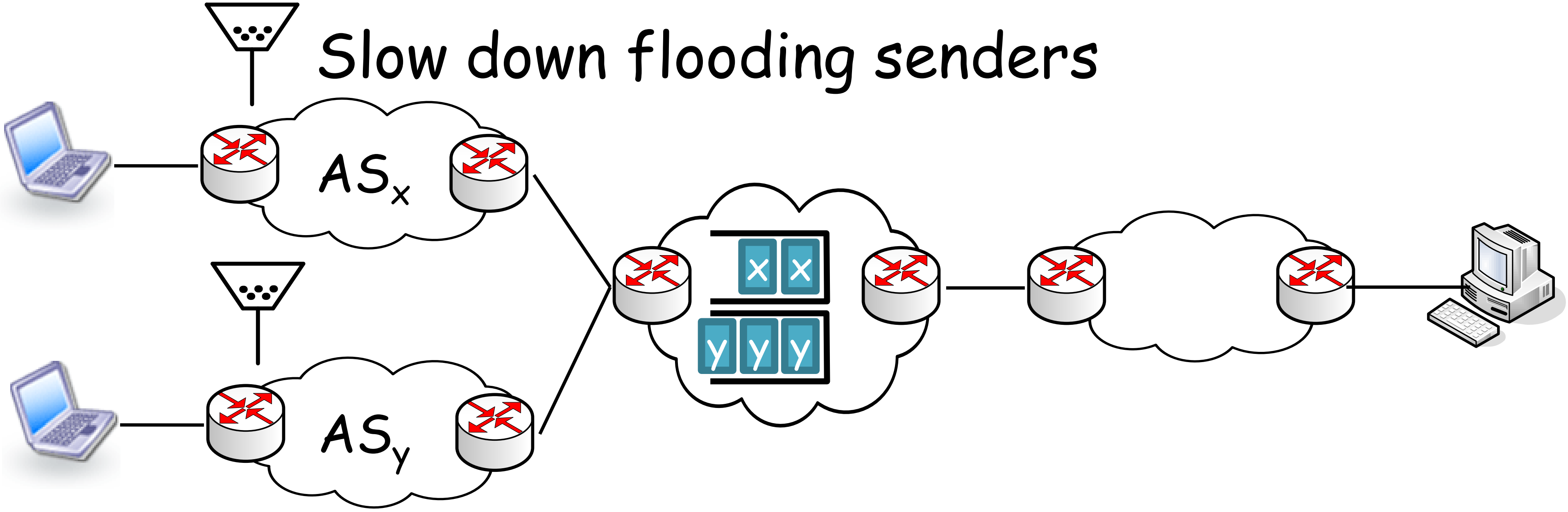
- Scalable: no per-flow state in the core
  1. Aggregate flow policing placed at edge routers [CSFQ]
  2. AS-level policing in the core
     - Fair queuing or rate limiting

Hierarchical Congestion Policing



- Scalable: no per-flow state in the core
  1. Aggregate flow policing placed at edge routers [CSFQ]
  2. AS-level policing in the core
     - Fair queuing or rate limiting

Hierarchical Congestion Policing

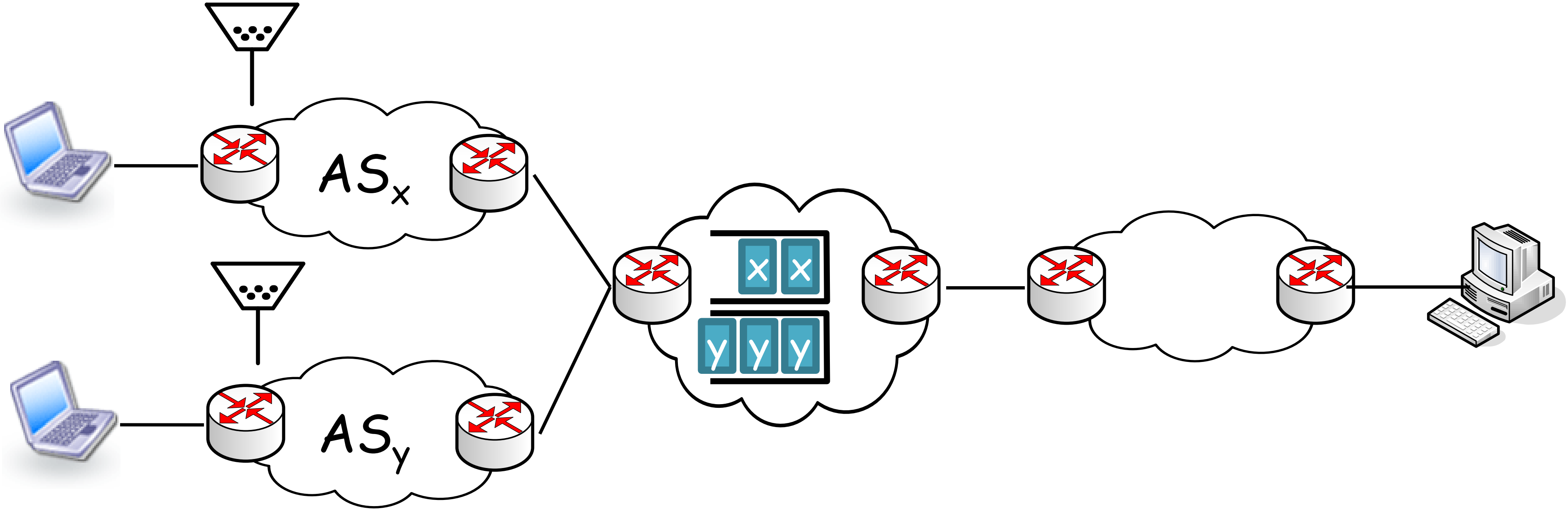


Slow down flooding senders

- Scalable: no per-flow state in the core

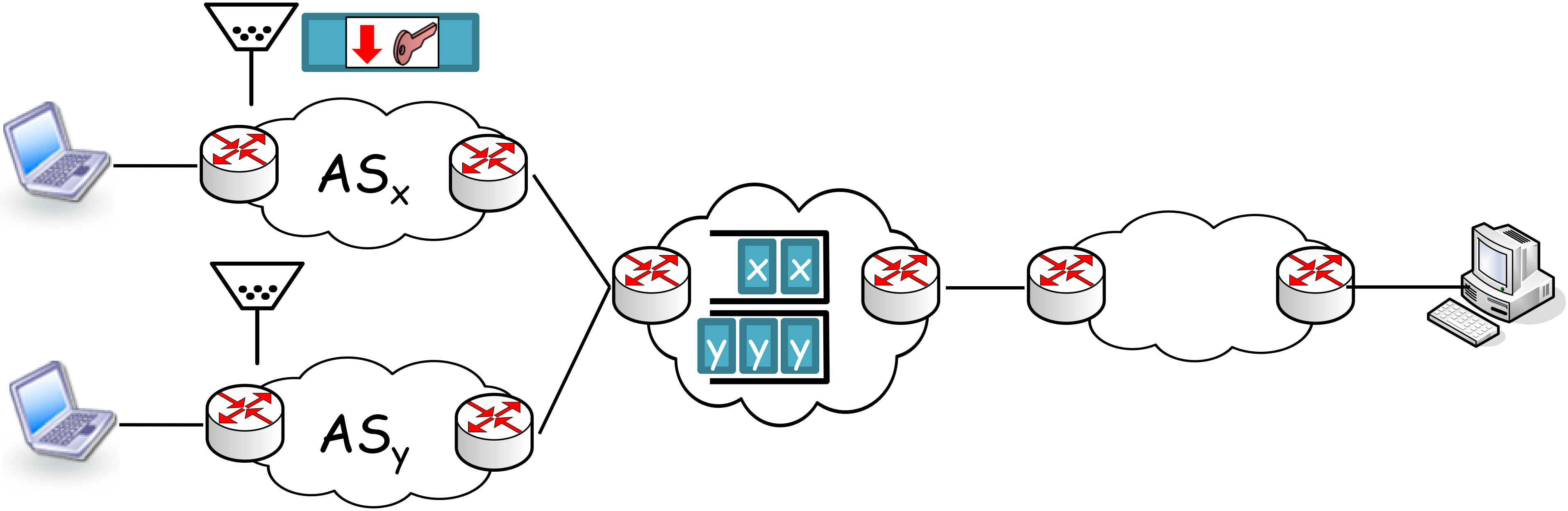
1. Aggregate flow policing placed at edge routers [CSFQ]
2. AS-level policing in the core
    - Fair queuing or rate limiting

Secure Congestion Policing



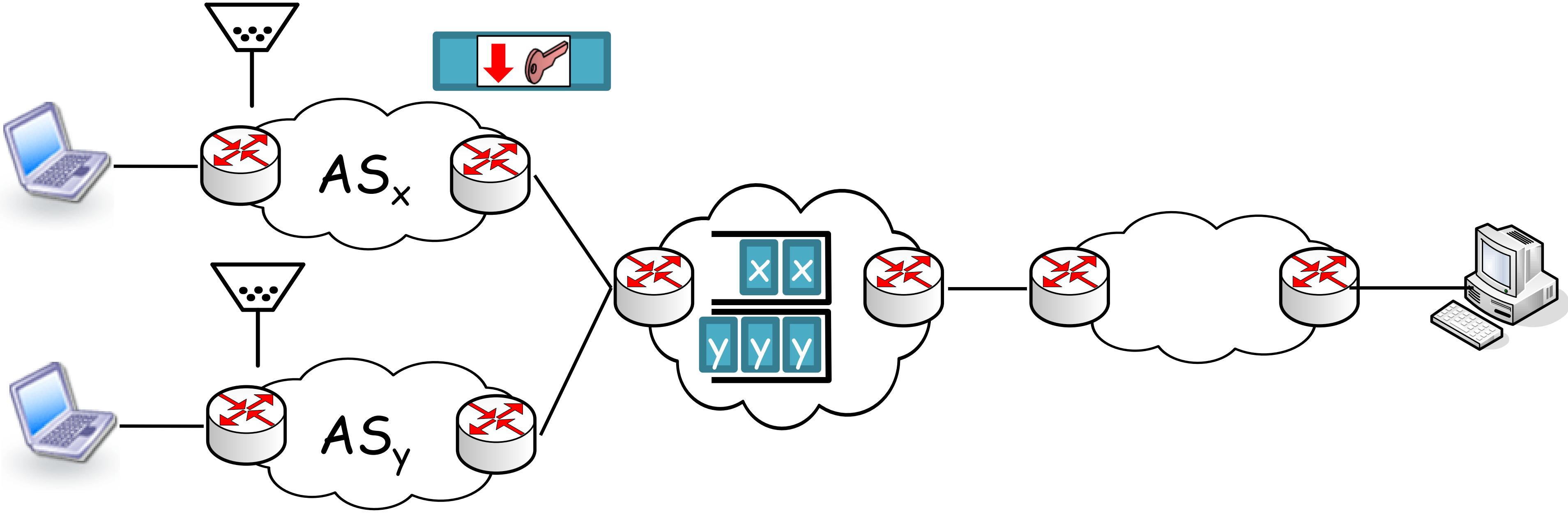
- Robust to compromised routers and hosts
  - Efficient symmetric key cryptography
  - Packets carry secure tokens
    - Source AS authenticators [Passport,NSDI08] → AS Accountability
    - Secure congestion policing feedback

Secure Congestion Policing



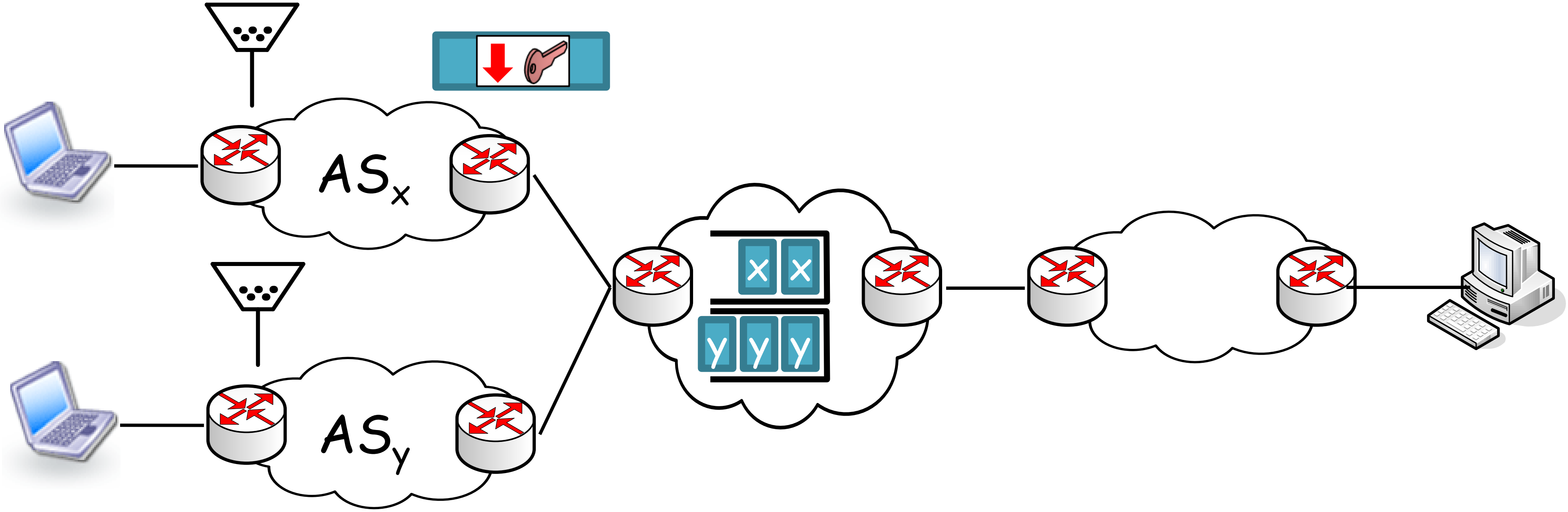
- Robust to compromised routers and hosts
  - Efficient symmetric key cryptography
  - Packets carry secure tokens
    - Source AS authenticators [Passport,NSDI08] → AS Accountability
    - Secure congestion policing feedback

Secure Congestion Policing Feedback as Network Capabilities



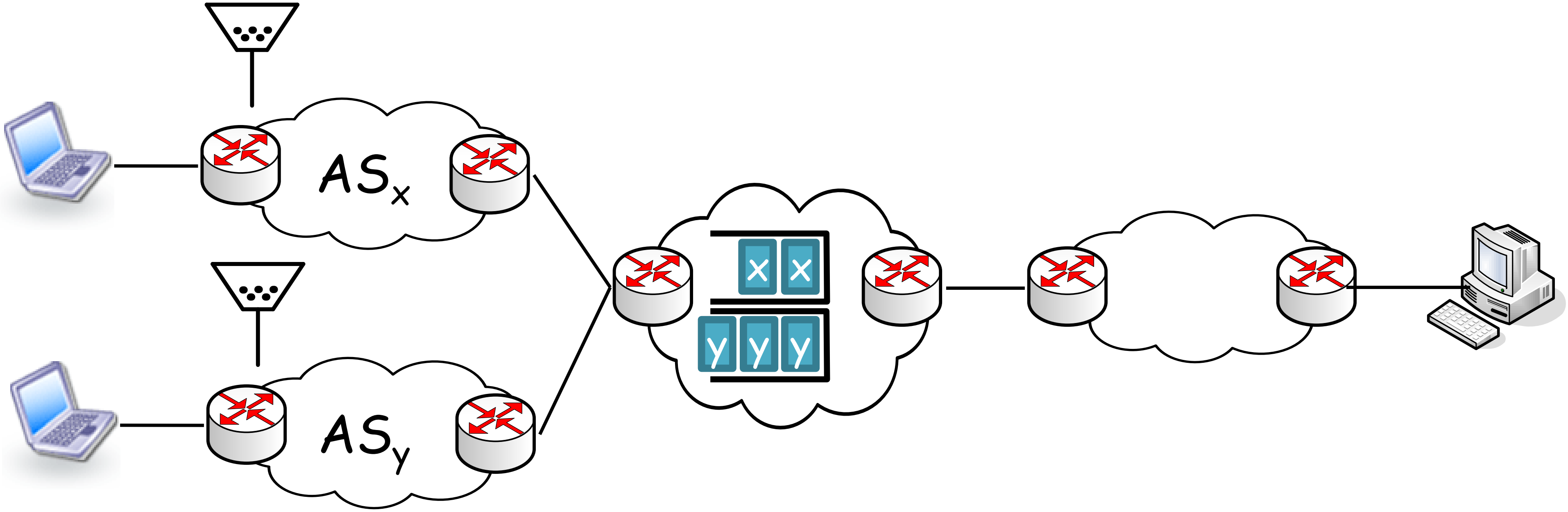
- **Open**
  - Receiver explicitly authorizes desired traffic
    - Return if wants to receive
    - Not, otherwise

Secure Congestion Policing Feedback as Network Capabilities



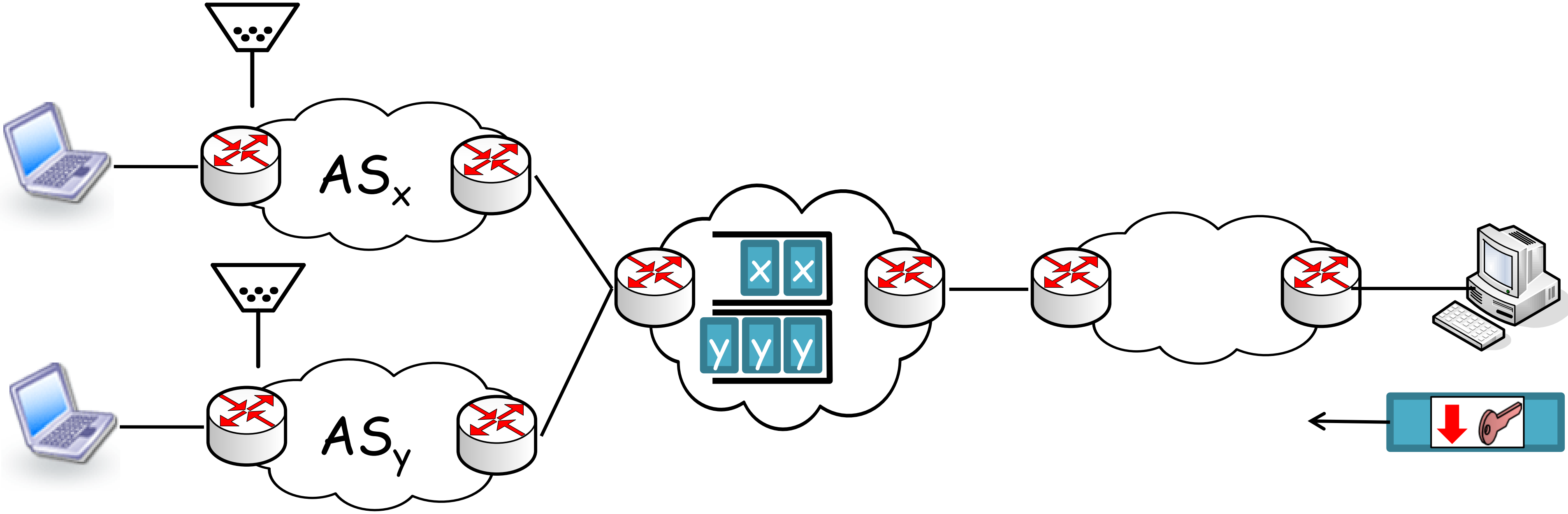
- Open
  - Receiver explicitly authorizes desired traffic
    - Return if wants to receive
    - Not, otherwise

Secure Congestion Policing Feedback as Network Capabilities



- Open
  - Receiver explicitly authorizes desired traffic
    - Return if wants to receive
    - Not, otherwise

Secure Congestion Policing Feedback as Network Capabilities



- Open
  - Receiver explicitly authorizes desired traffic
    - Return if wants to receive
    - Not, otherwise

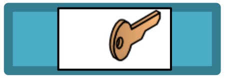
Now the Details…

How does NetFence Work?

- A sender sends two types of packets

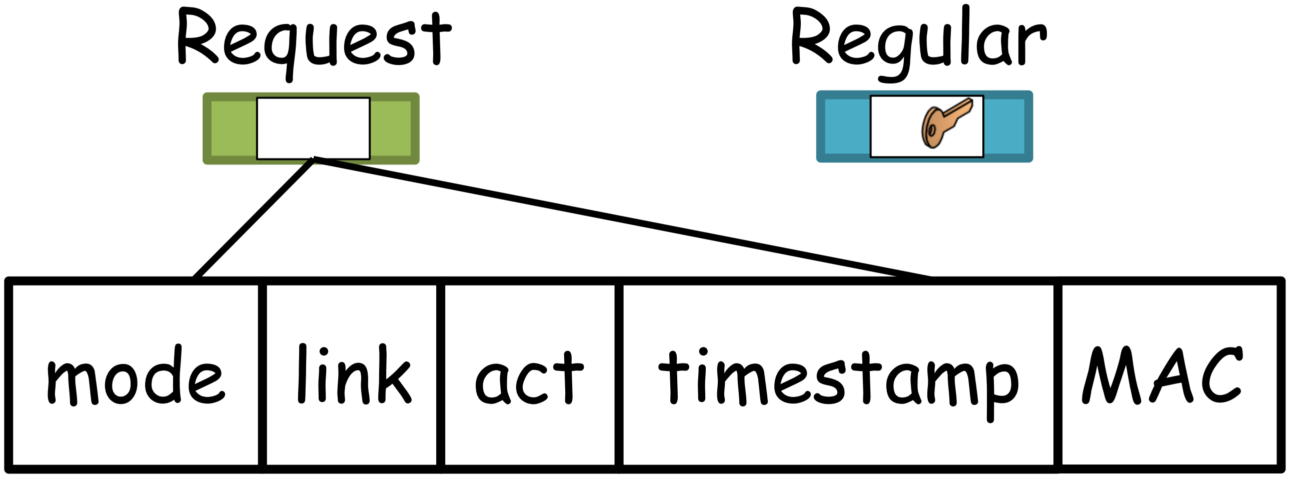
Request

Regular



How does NetFence Work?

- A sender sends two types of packets



Request

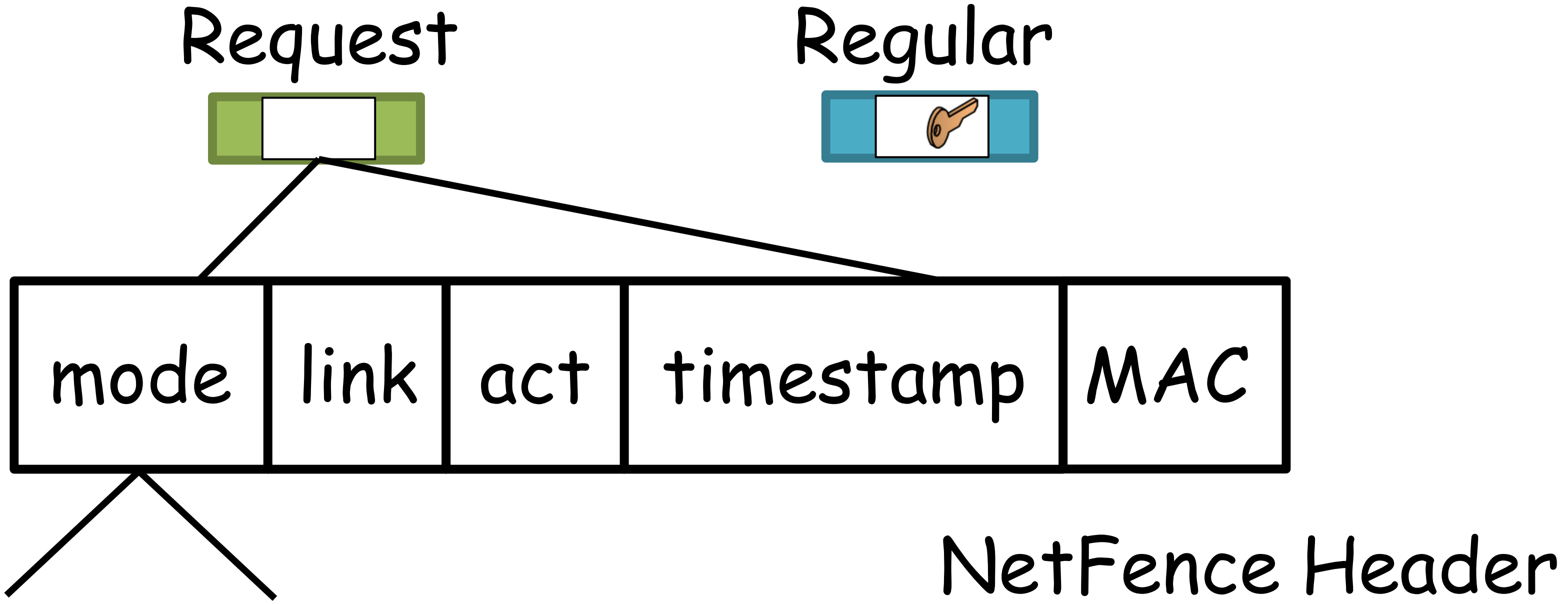
Regular

mode	link	act	timestamp	MAC

NetFence Header

How does NetFence Work?

- A sender sends two types of packets



Request

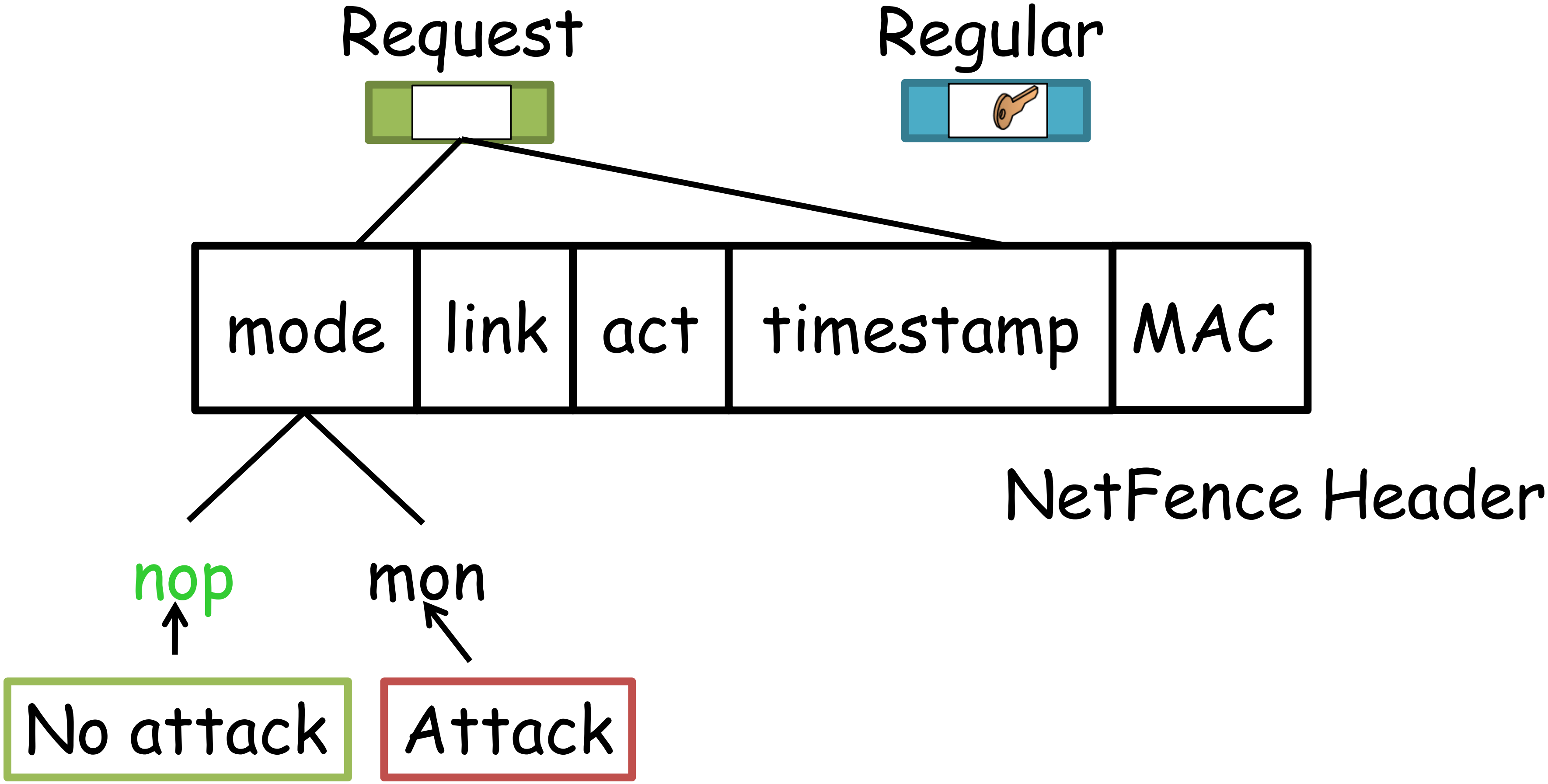
Regular

mode	link	act	timestamp	MAC

NetFence Header

How does NetFence Work?

- A sender sends two types of packets



Request

Regular

mode	link	act	timestamp	MAC

NetFence Header

nop

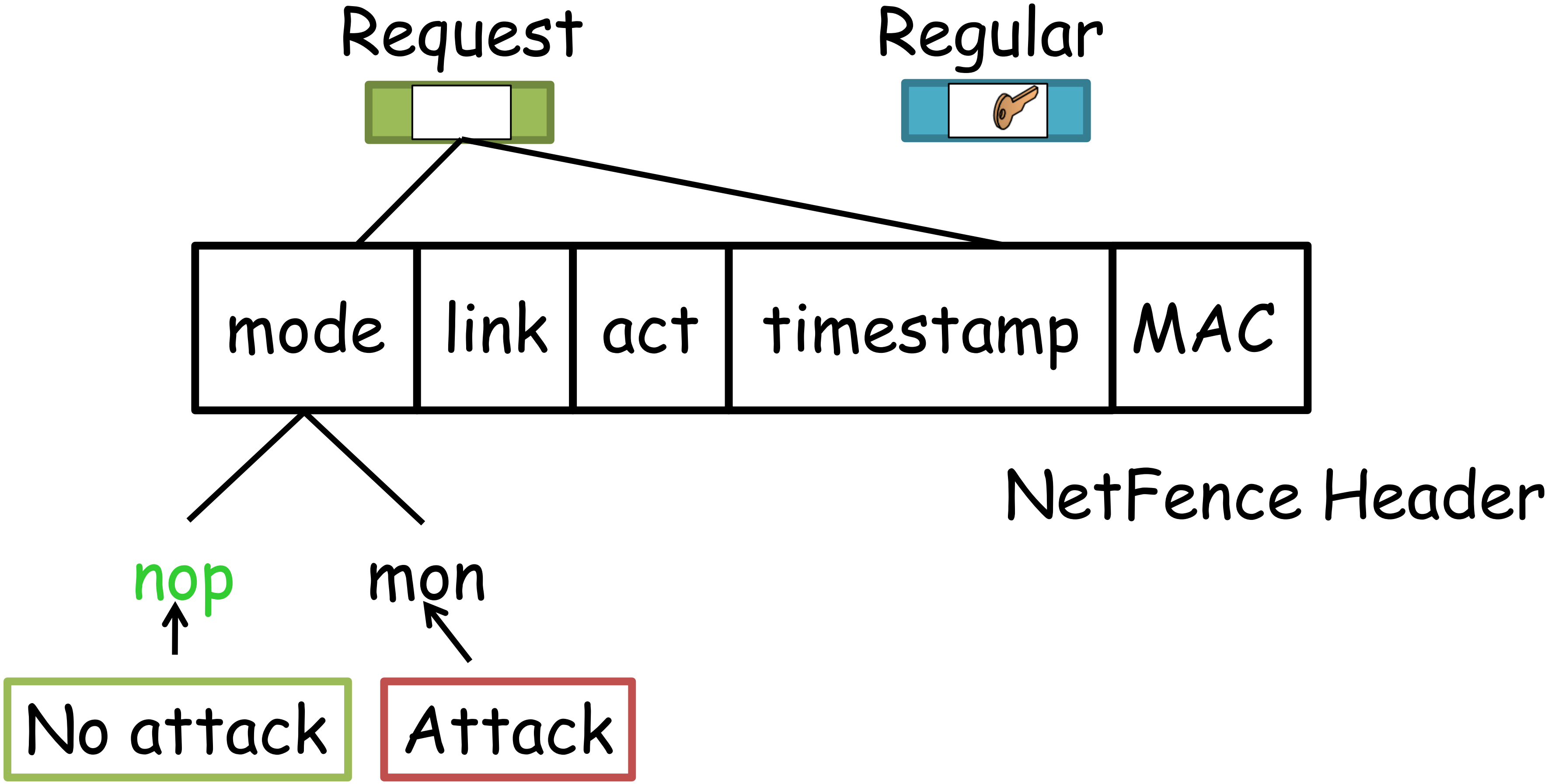
mon

No attack

Attack

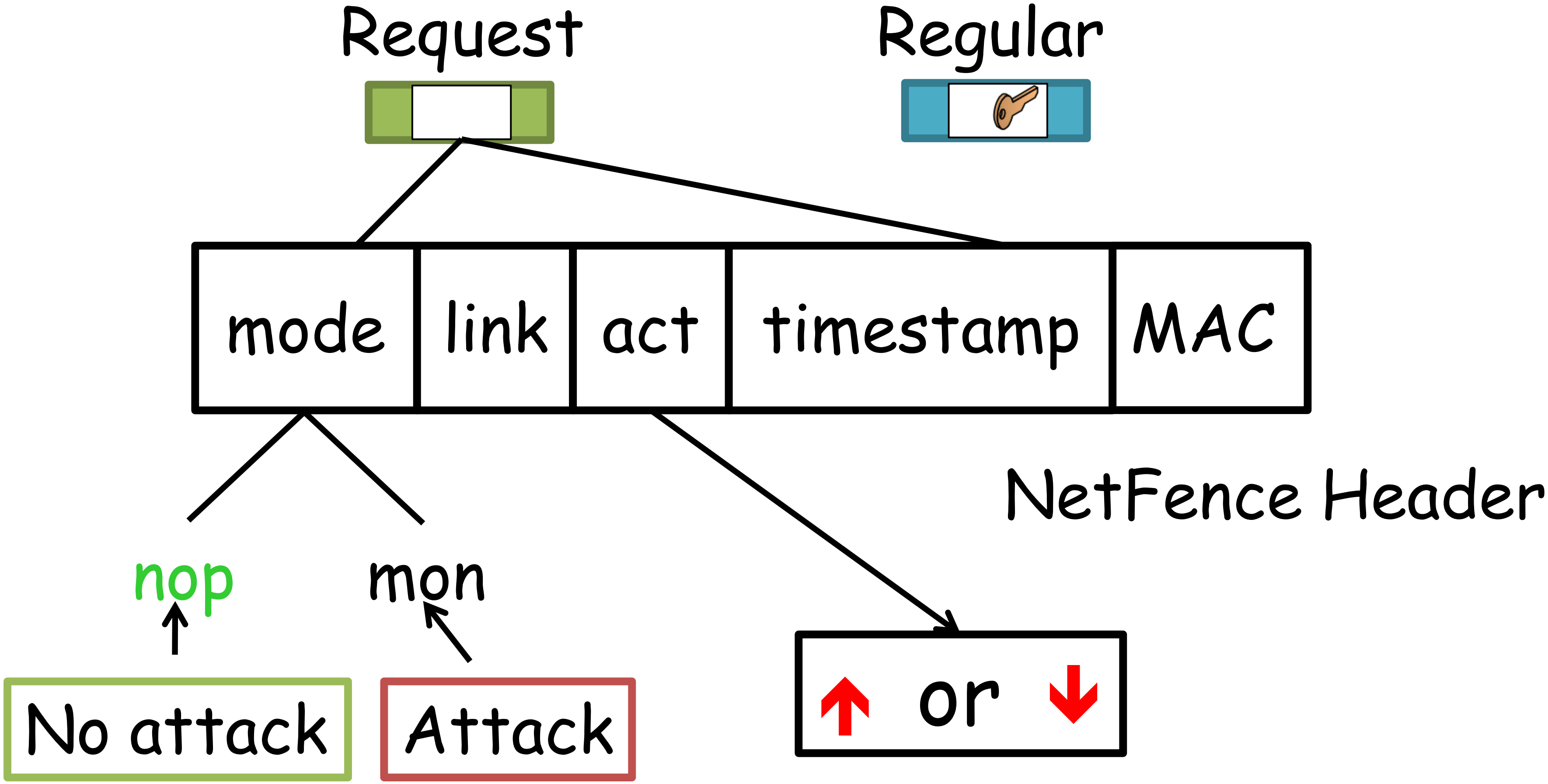
How does NetFence Work?

- A sender sends two types of packets



How does NetFence Work?

- A sender sends two types of packets



Request

Regular

mode | link | act | timestamp | MAC

NetFence Header

nop

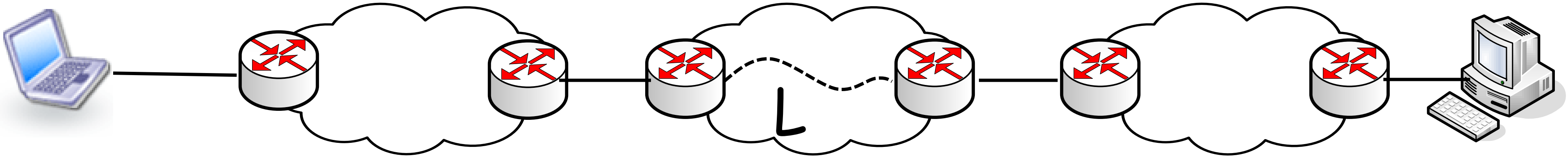
mon

No attack

Attack

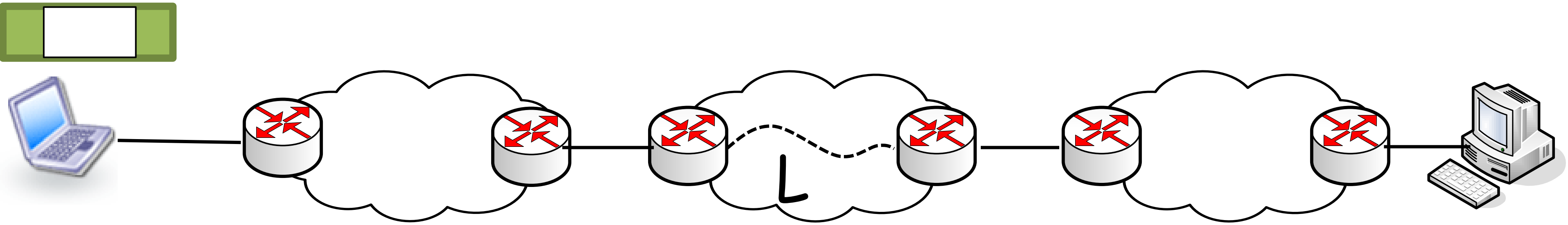
↑ or ↓

How does NetFence Work?



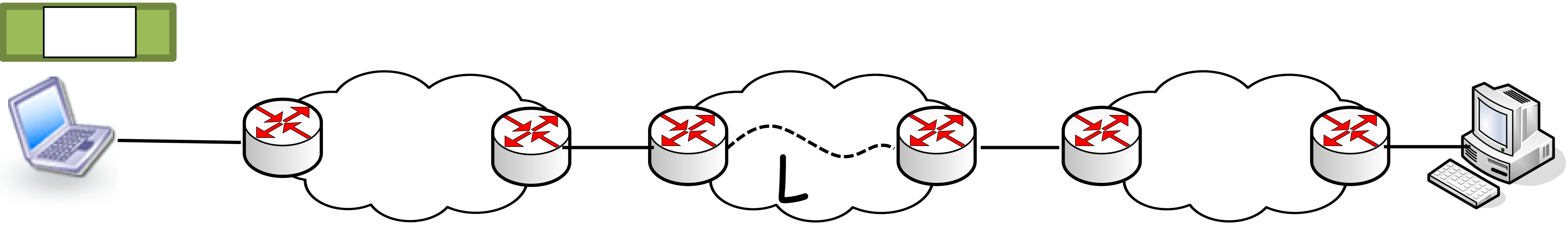
- A sender first sends a request packet
- Its access router stamps nop
    - now → ts (timestamp), null → link, nop → mode
    - ▬🔑 = MAC🔑(src, dst, ts, null, nop)

How does NetFence Work?



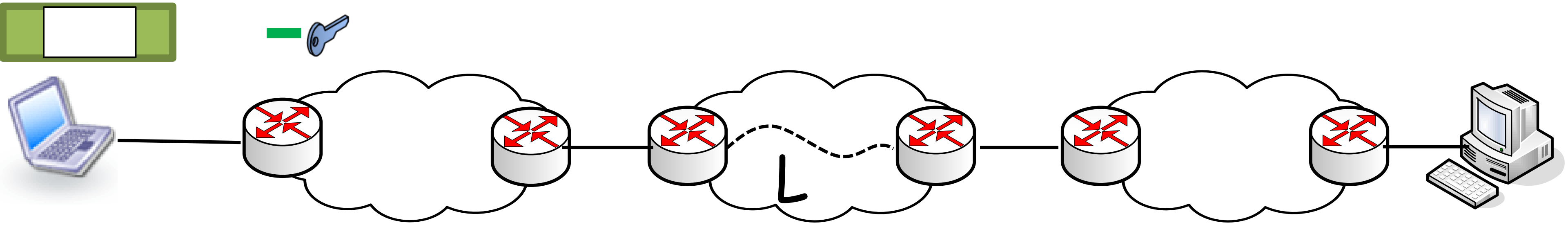
- A sender first sends a request packet
- Its access router stamps nop
  - now → ts (timestamp), null → link, nop → mode
  - ▬🔑 = MAC🔑(src, dst, ts, null, nop)

How does NetFence Work?



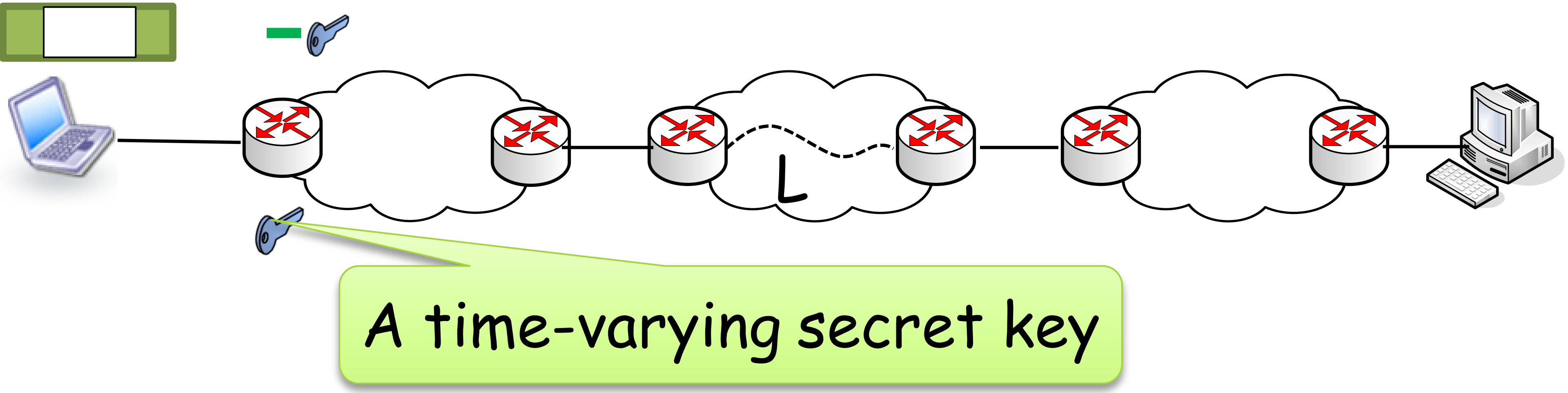
- A sender first sends a request packet
- Its access router stamps nop
  - now → ts (timestamp), null → link, nop → mode
  - ▬🔑 = MAC🔑(src, dst, ts, null, nop)

How does NetFence Work?



- A sender first sends a request packet
- Its access router stamps nop
  - now → ts (timestamp), null → link, nop → mode
  - ▬🔑 = MAC🔑(src, dst, ts, null, nop)

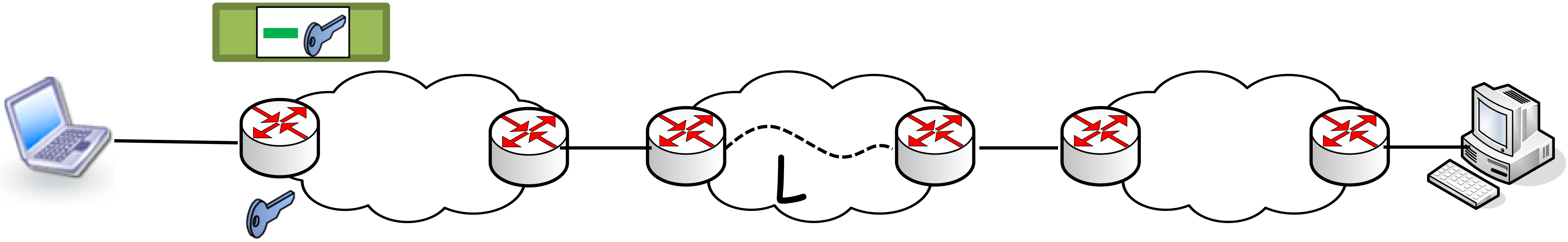
How does NetFence Work?



A time-varying secret key

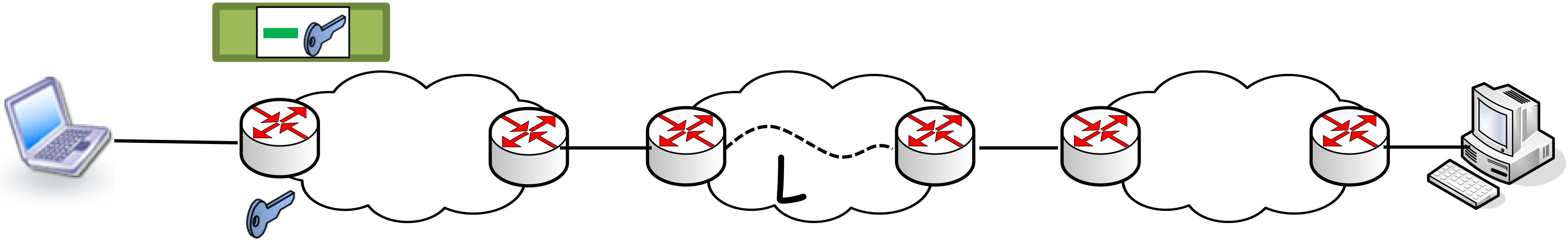
- A sender first sends a request packet
- Its access router stamps nop
  - now → ts (timestamp), null → link, nop → mode
  - ▬🔑 = MAC🔑(src, dst, ts, null, nop)

How does NetFence Work?



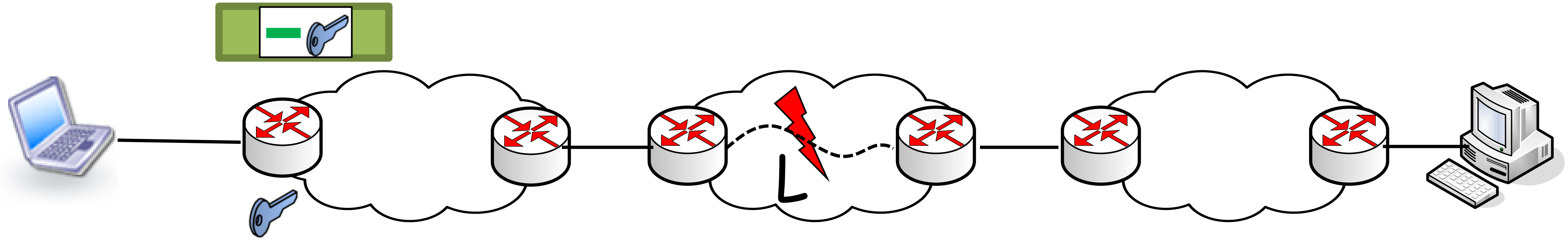
- A router under attack replaces nop with L↓
  - All traffic
  - Signal congestion to access router
  - L→ link, ↓ → act, mon → mode
  - ↓🔑 = MAC🔑(src, dst, ts, L, mon, ↓, –🔑)
  - No downstream overwrite

How does NetFence Work?



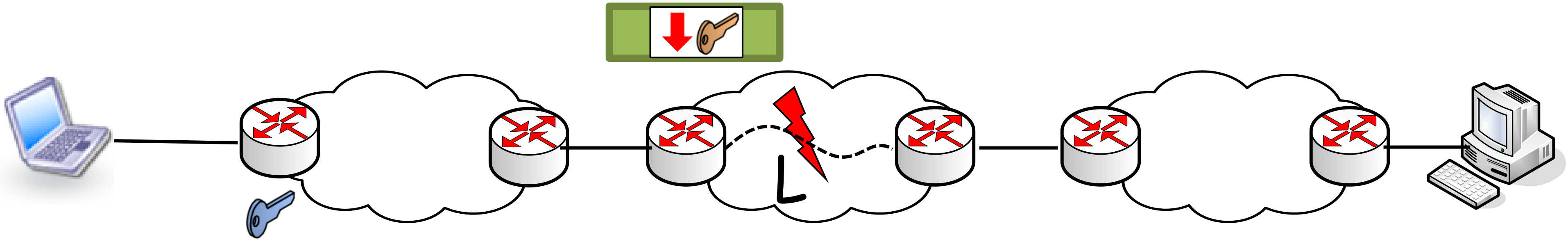
- A router under attack replaces nop with L↓
  - All traffic
  - Signal congestion to access router
  - L→ link, ↓ → act, mon → mode
  - ↓🔑 = MAC🔑(src, dst, ts, L, mon, ↓, —🔑)
  - No downstream overwrite

How does NetFence Work?



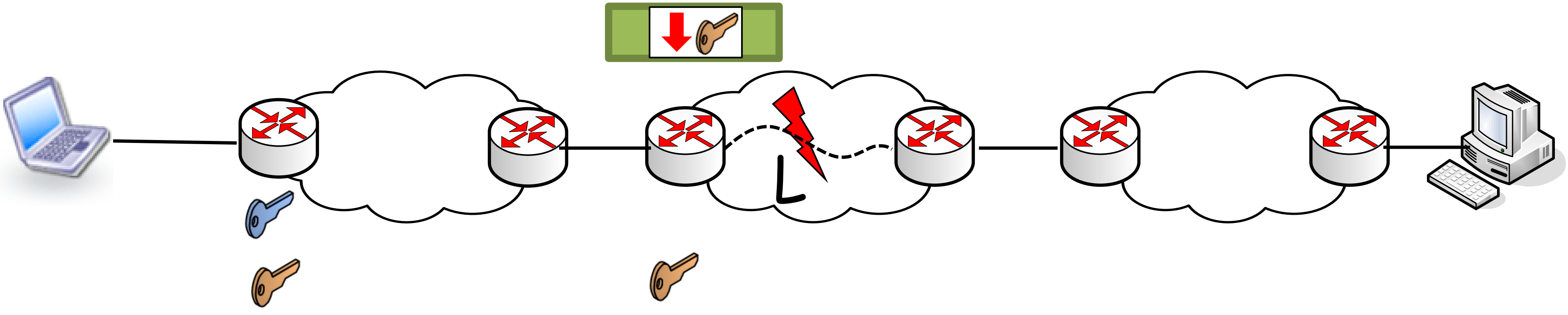
- A router under attack replaces nop with L↓
  - All traffic
  - Signal congestion to access router
  - L→ link, ↓ → act, mon → mode
  - ↓🔑 = MAC🔑(src, dst, ts, L, mon, ↓, –🔑)
  - No downstream overwrite

How does NetFence Work?



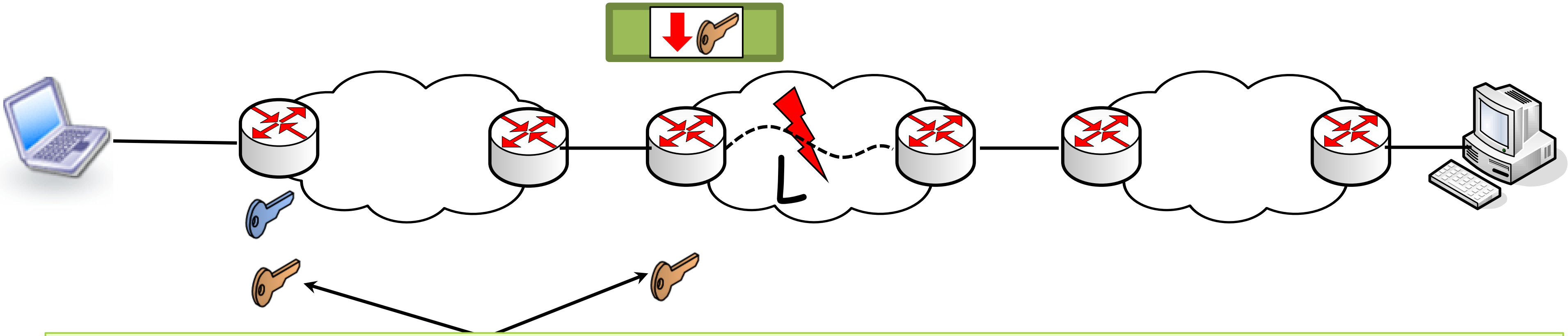
- A router under attack replaces nop with L↓
  - All traffic
  - Signal congestion to access router
  - L→ link, ↓ → act, mon → mode
  - ↓🔑 = MAC🔑(src, dst, ts, L, mon, ↓, –🔑)
  - No downstream overwrite

How does NetFence Work?



- A router under attack replaces nop with L↓
  - All traffic
  - Signal congestion to access router
  - L→ link, ↓ → act, mon → mode
  - ↓🔑 = MAC🔑(src, dst, ts, L, mon, ↓, −🔑)
  - No downstream overwrite

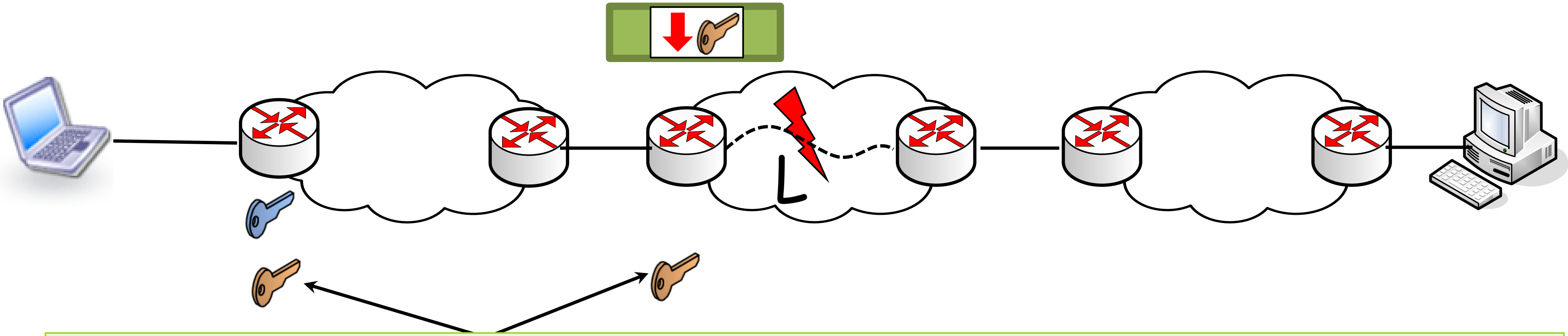
How does NetFence Work?



A shared time-varying secret key
via distributed Diffie-Hellman via BGP [Passport]

- A router under attack replaces nop with L↓
  - All traffic
  - Signal congestion to access router
  - L→ link, ↓ → act, mon → mode
  - ↓🔑 = MAC🔑(src, dst, ts, L, mon, ↓, –🔑)
  - No downstream overwrite

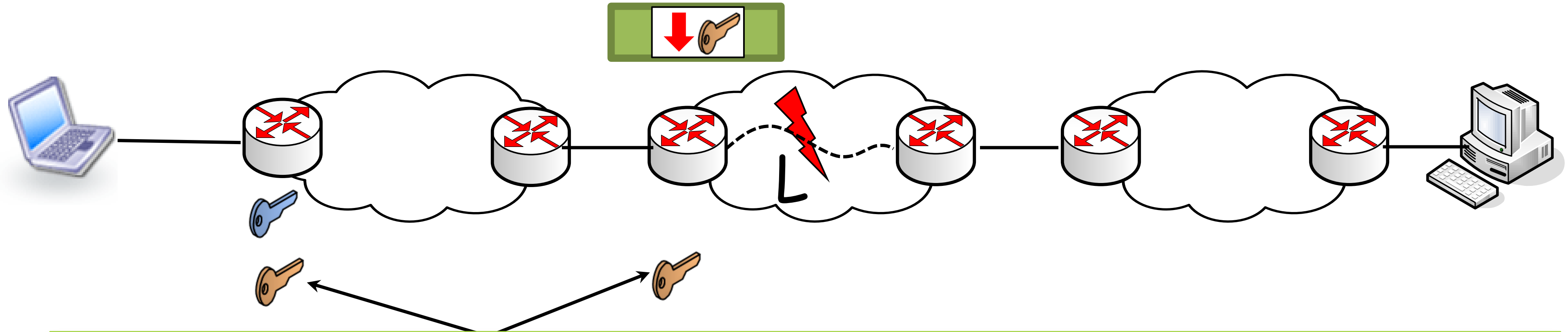
How does NetFence Work?



A shared time-varying secret key
via distributed Diffie-Hellman via BGP [Passport]

- A router under attack replaces nop with L↓
  - All traffic
  - Signal congestion to access router
  - L→ link, ↓ → act, mon → mode
  - ↓🔑 = MAC🔑(src, dst, ts, L, mon, ↓, −🔑)
  - No downstream overwrite

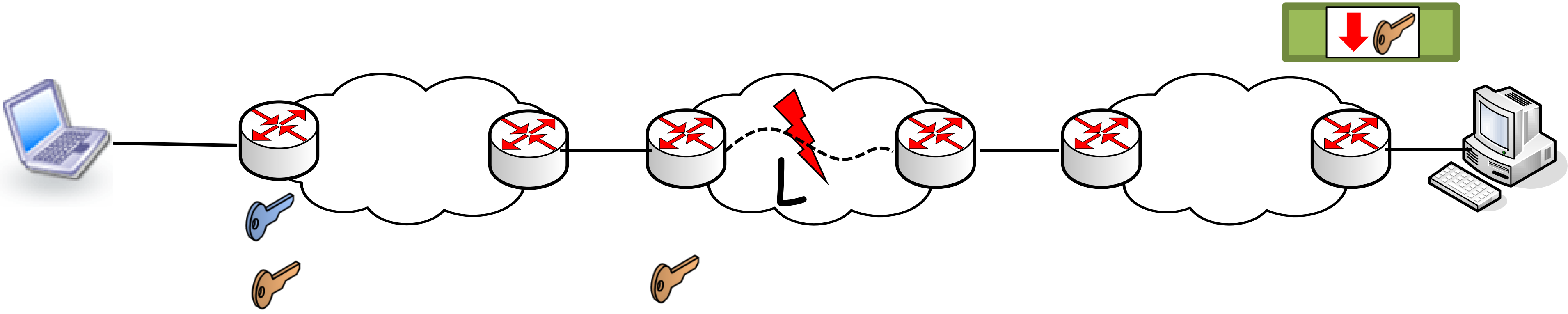
How does NetFence Work?



A shared time-varying secret key
via distributed Diffie-Hellman via BGP [Passport]

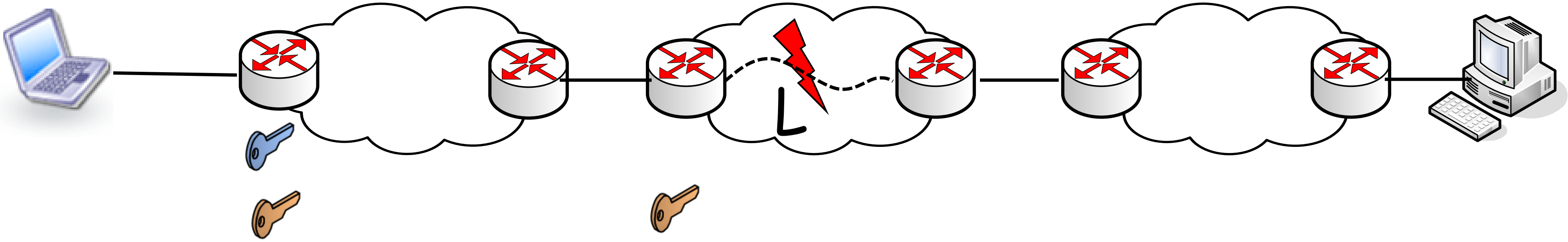
- A router under attack replaces nop with L⬇
  - All traffic
  - Signal congestion to access router
  - L→ link, ⬇ → act, mon → mode
  - ⬇🔑 = MAC$_{🔑}$(src, dst, ts, L, mon, ⬇, –🔑)
  - No downstream overwrite

How does NetFence Work?



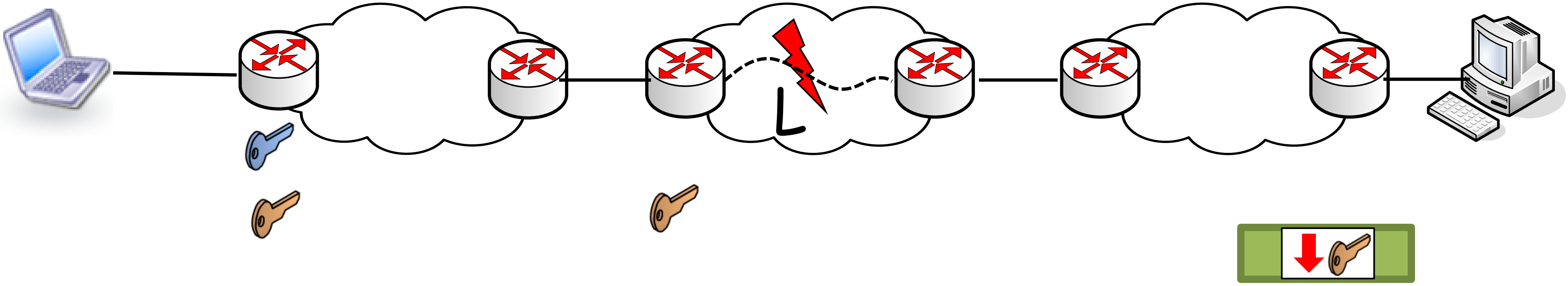
- A receiver use the feedback as capabilities
- Sender sends regular packets that carry the congestion policing feedback
  - Could be nop when there is no attack
  - Can't send if receiving no feedback from receiver

How does NetFence Work?



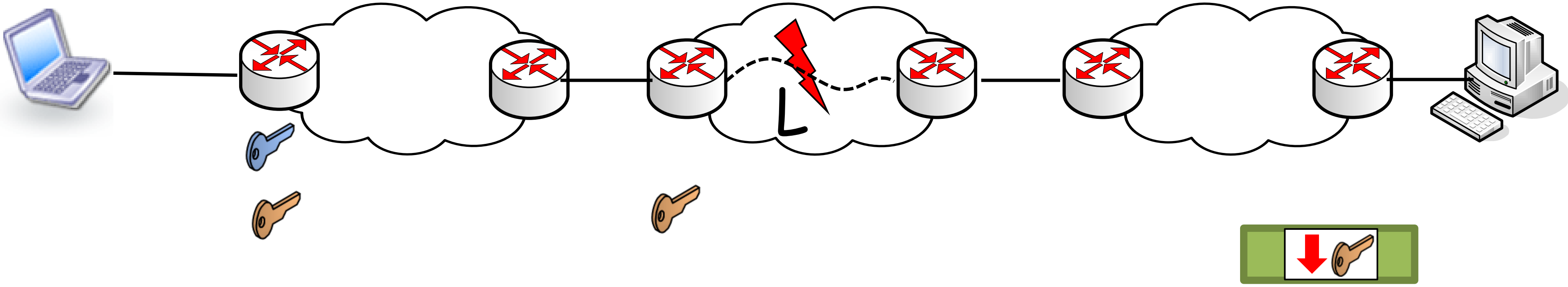
- A receiver use the feedback as capabilities
- Sender sends regular packets that carry the congestion policing feedback
  - Could be nop when there is no attack
  - Can't send if receiving no feedback from receiver

How does NetFence Work?



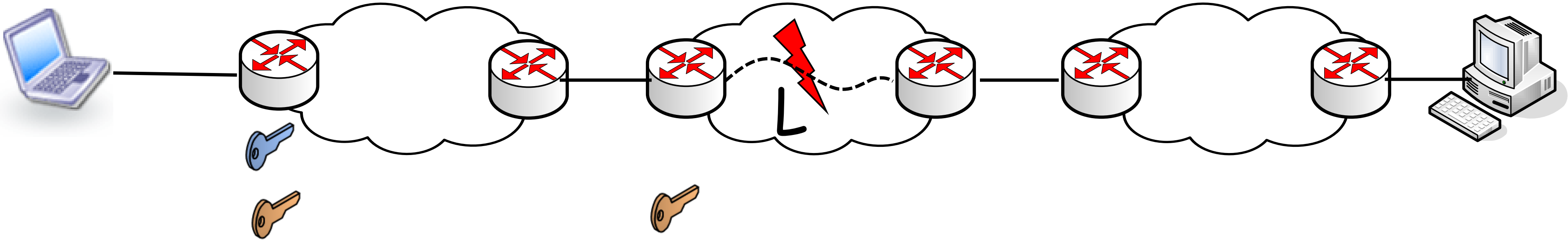
- A receiver use the feedback as capabilities
- Sender sends regular packets that carry the congestion policing feedback
  - Could be nop when there is no attack
  - Can't send if receiving no feedback from receiver

How does NetFence Work?



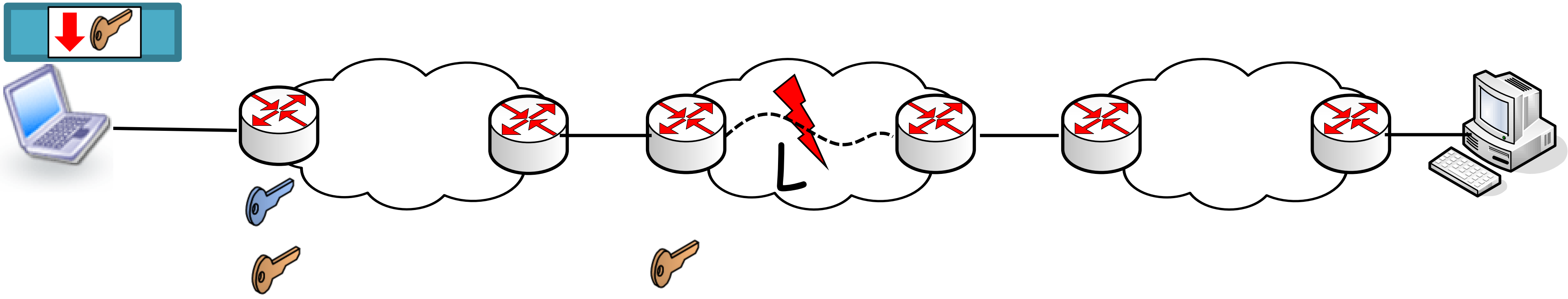
- A receiver use the feedback as capabilities
- Sender sends regular packets that carry the congestion policing feedback
  - Could be nop when there is no attack
  - Can't send if receiving no feedback from receiver

How does NetFence Work?



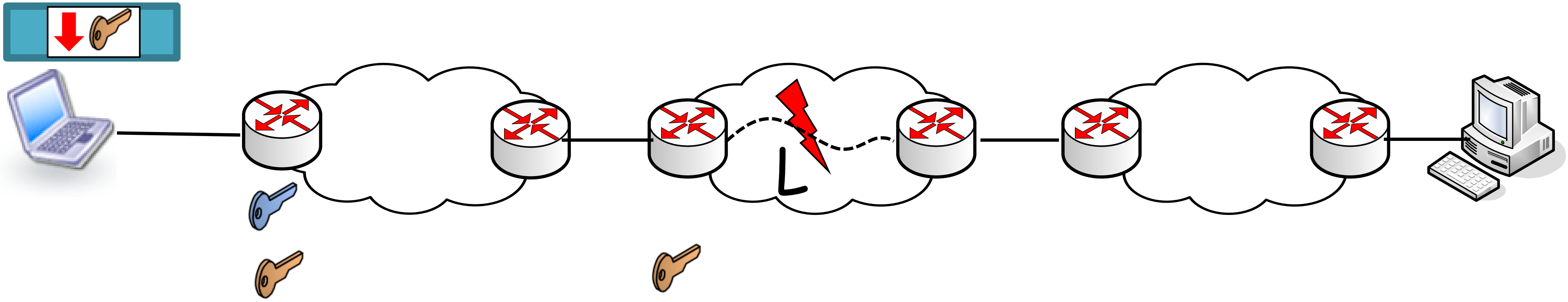
- A receiver use the feedback as capabilities
- Sender sends regular packets that carry the congestion policing feedback
  - Could be nop when there is no attack
  - Can't send if receiving no feedback from receiver

How does NetFence Work?



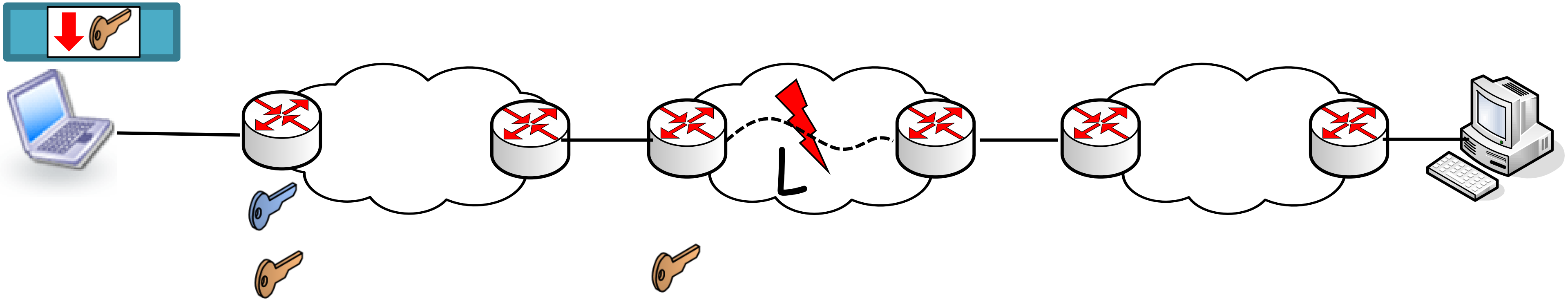
- A receiver use the feedback as capabilities
- Sender sends regular packets that carry the congestion policing feedback
  - Could be nop when there is no attack
  - Can't send if receiving no feedback from receiver

How does NetFence Work?



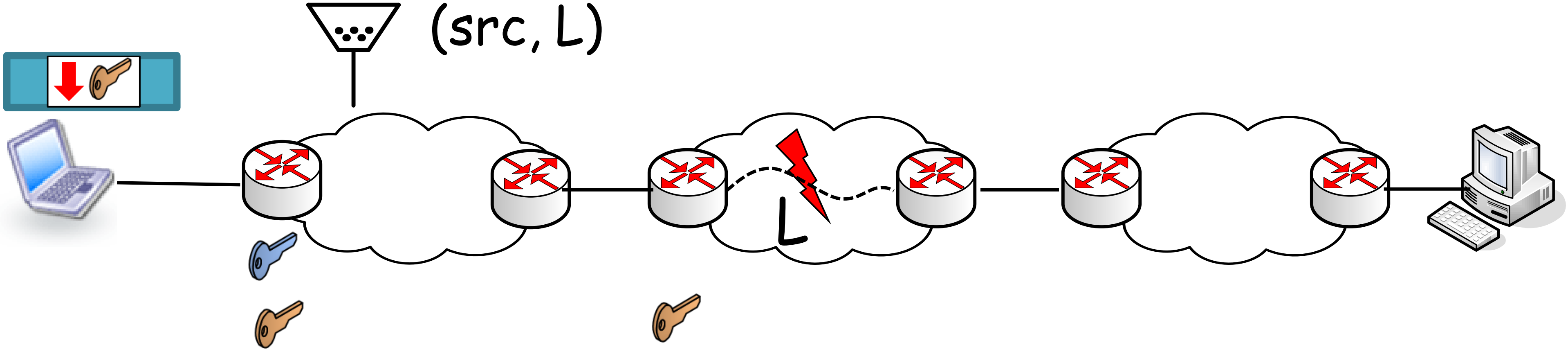
- Access router validates feedback
- Starts congestion policing
  - One leaky bucket per (src, L) limits sending rate
  - Not distinguish legitimate/malicious senders
- Resets L↑
  - now → ts, ↑ → act
  - ↑🔑 = MAC🔑 (src, dst, ts, L, mon, ↑)

How does NetFence Work?



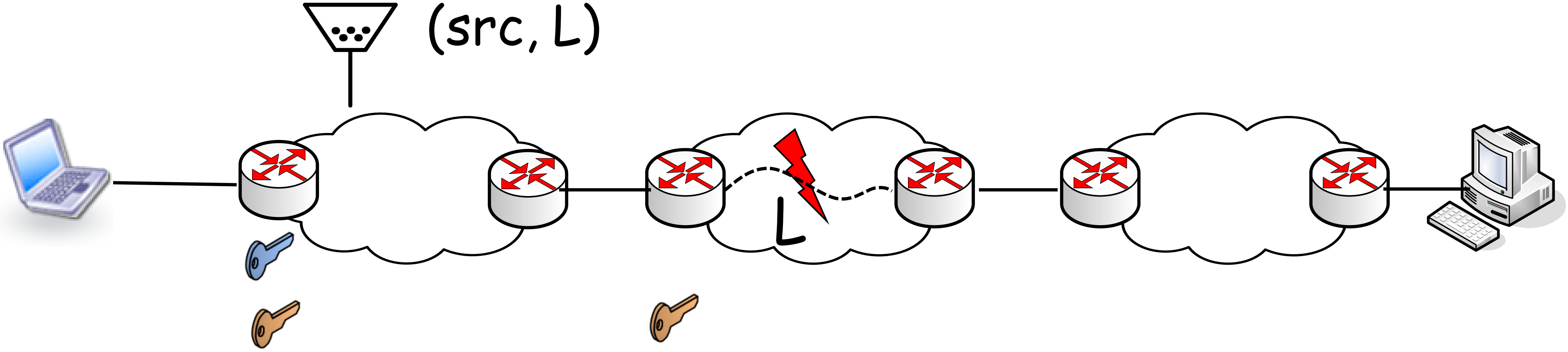
- Access router validates feedback
- Starts congestion policing
  - One leaky bucket per (src, L) limits sending rate
  - Not distinguish legitimate/malicious senders
- Resets L↑
  - now → ts, ↑ → act
  - ↑🔑 = MAC🔑 (src, dst, ts, L, mon, ↑)

How does NetFence Work?



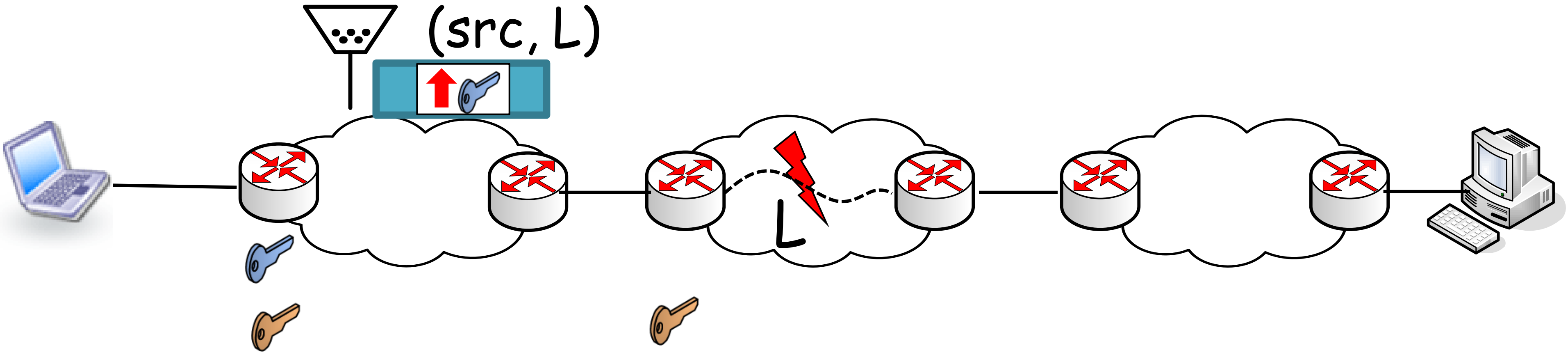
- Access router validates feedback
- Starts congestion policing
  - One leaky bucket per (src, L) limits sending rate
  - Not distinguish legitimate/malicious senders
- Resets L↑
  - now → ts, ↑ → act
  - ↑🔑 = MAC🔑 (src, dst, ts, L, mon, ↑)

How does NetFence Work?



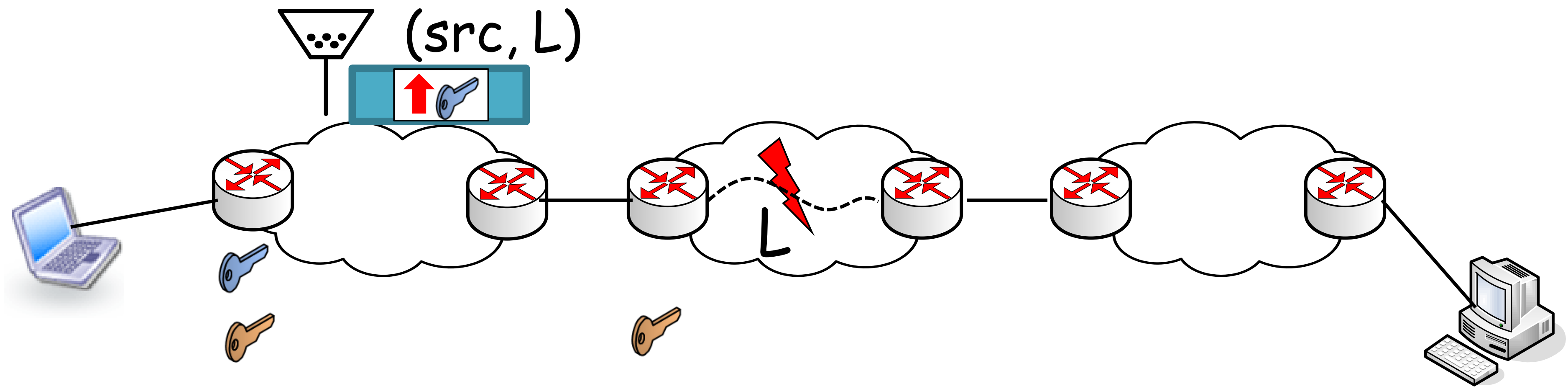
- Access router validates feedback
- Starts congestion policing
  - One leaky bucket per (src, L) limits sending rate
  - Not distinguish legitimate/malicious senders
- Resets L↑
  - now → ts, ↑ → act
  - ↑🔑 = MAC🔑 (src, dst, ts, L, mon, ↑)

How does NetFence Work?

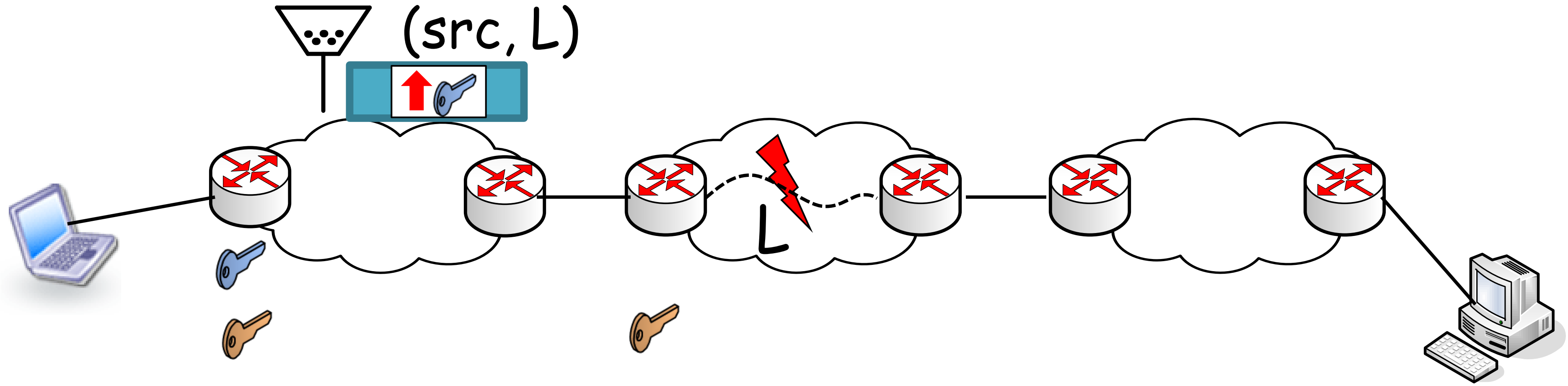


- Access router validates feedback
- Starts congestion policing
  - One leaky bucket per (src, L) limits sending rate
  - Not distinguish legitimate/malicious senders
- Resets L↑
  - now → ts, ↑ → act
  - ↑🔑 = MAC🔑 (src, dst, ts, L, mon, ↑)

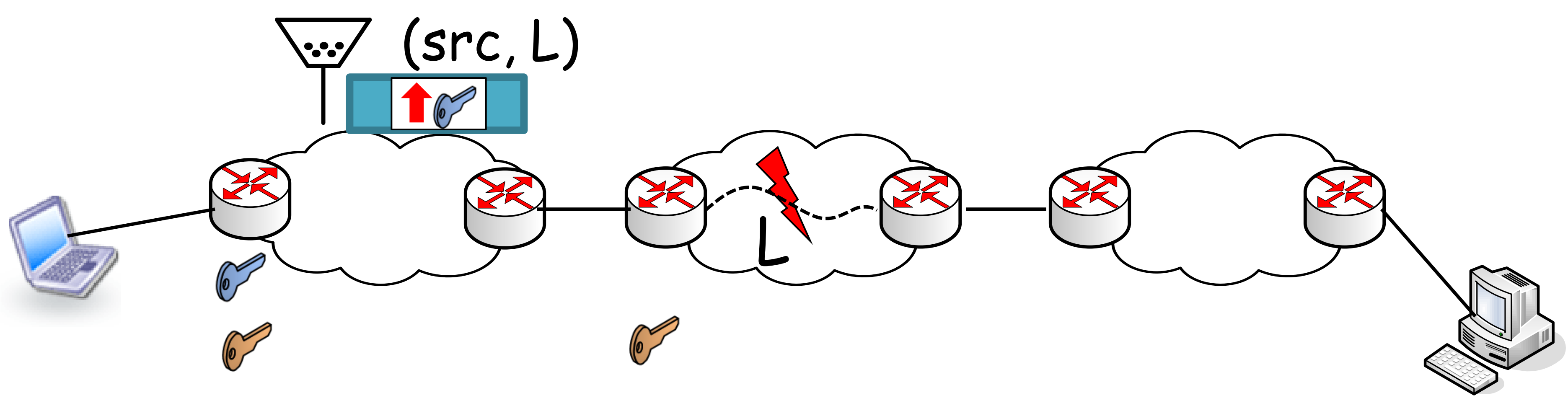
How does NetFence Work?



How does NetFence Work?

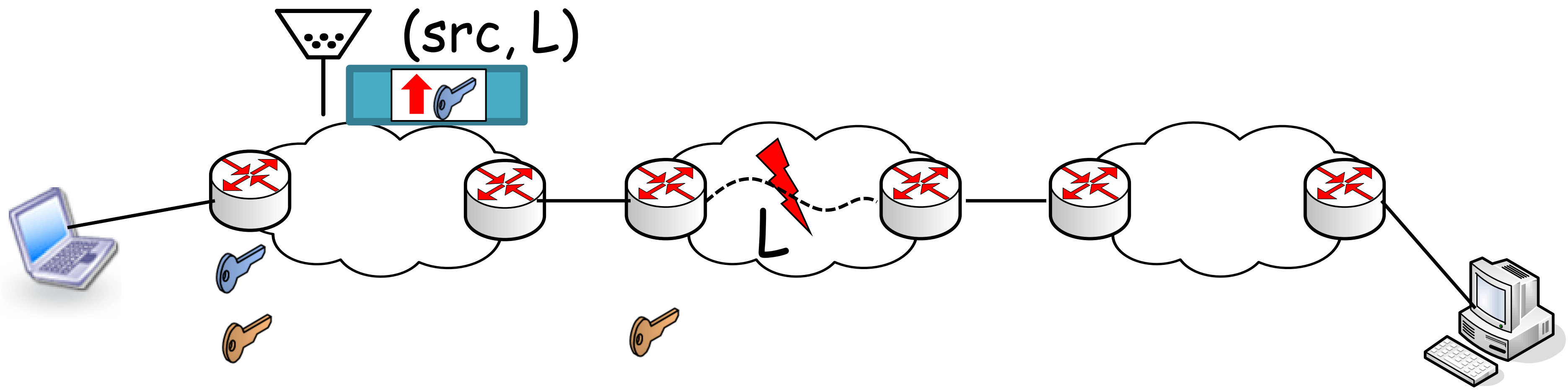


How does NetFence Work?



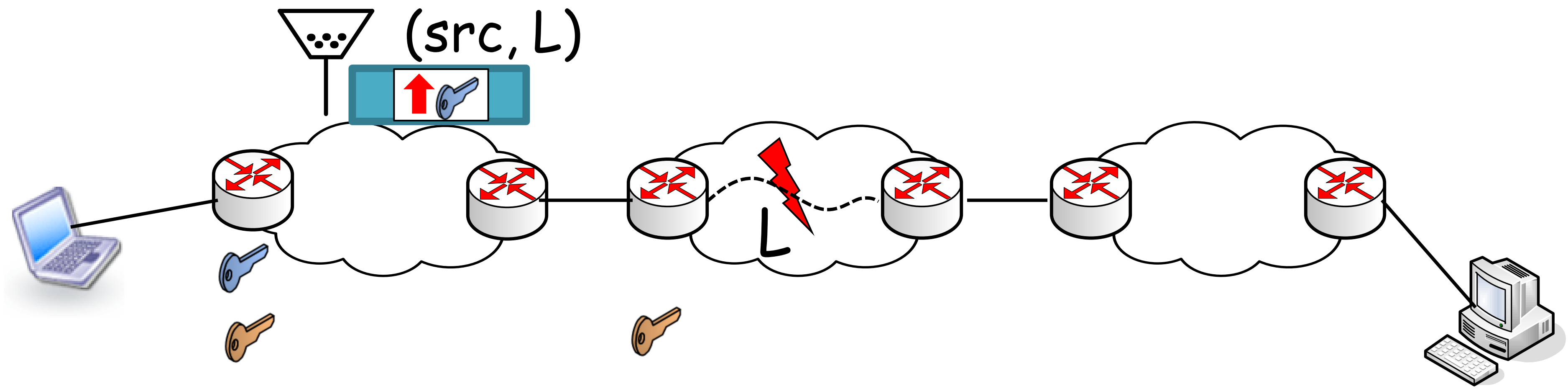
- Establishes a congestion policing loop
  - Bottleneck router signals
    - If congested, L↑ → L↓
    - Otherwise, L↑
  - Access router polices
    - Periodic Additive Increase Multiplicative Decrease (AIMD, TCP-like) for fairness and efficiency

How does NetFence Work?



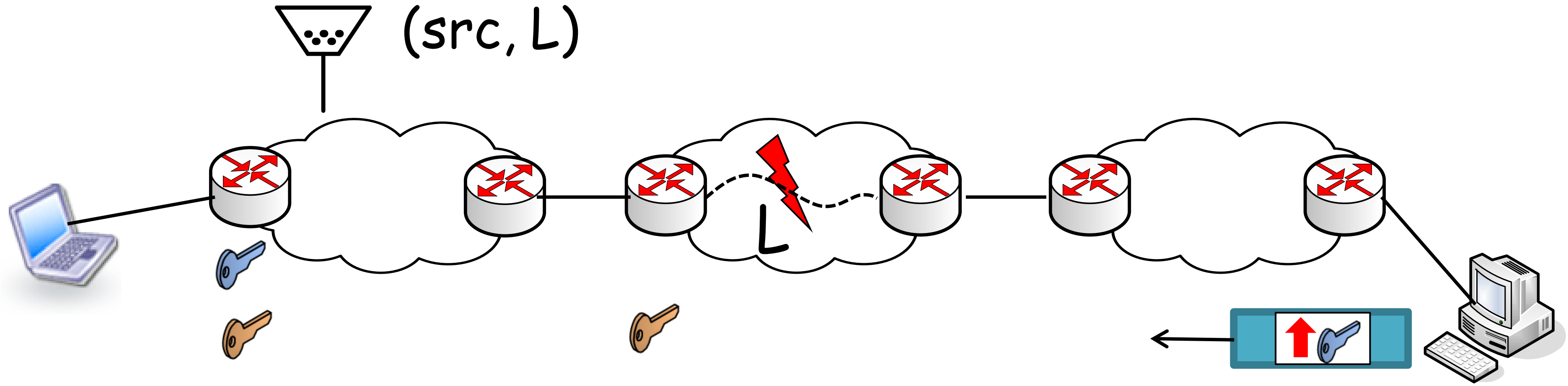
- Establishes a congestion policing loop
  - Bottleneck router signals
    - If congested, L↑ → L↓
    - Otherwise, L↑
  - Access router polices
    - Periodic Additive Increase Multiplicative Decrease (AIMD, TCP-like) for fairness and efficiency

How does NetFence Work?



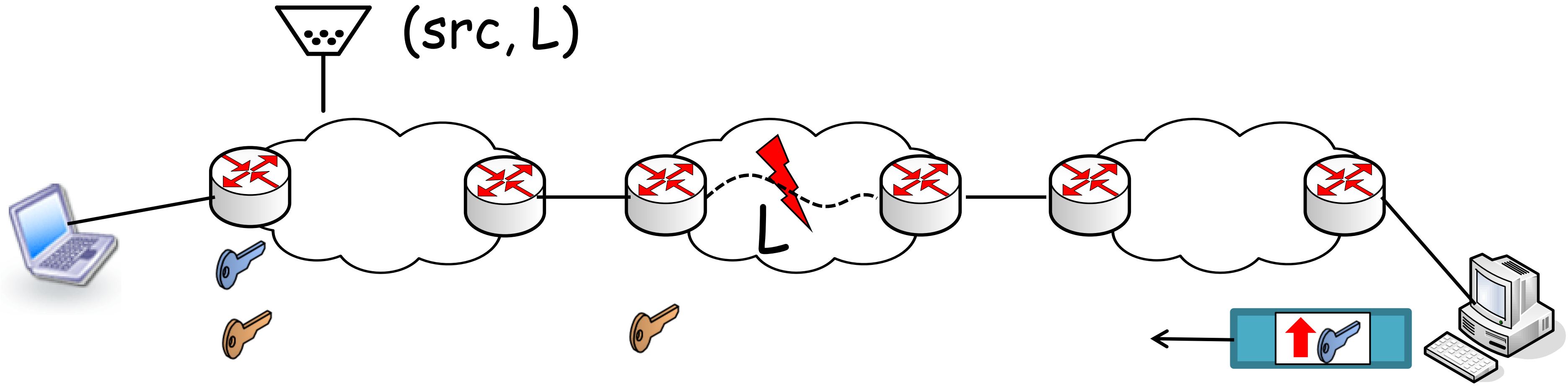
- Establishes a congestion policing loop
  - Bottleneck router signals
    - If congested, L↑ → L↓
    - Otherwise, L↑
  - Access router polices
    - Periodic Additive Increase Multiplicative Decrease (AIMD, TCP-like) for fairness and efficiency

How does NetFence Work?

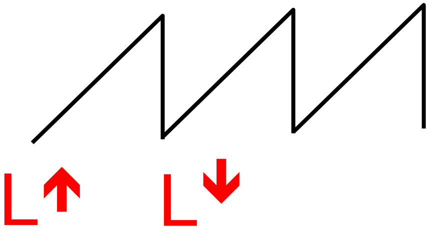


- Establishes a congestion policing loop
  - Bottleneck router signals
    - If congested, L↑ → L↓
    - Otherwise, L↑
  - Access router polices
    - Periodic Additive Increase Multiplicative Decrease (AIMD, TCP-like) for fairness and efficiency

How does NetFence Work?



- Establishes a congestion policing loop
  - Bottleneck router signals
    - If congested, L↑ → L↓
    - Otherwise, L↑
  - Access router polices
    - Periodic Additive Increase Multiplicative Decrease (AIMD, TCP-like) for fairness and efficiency



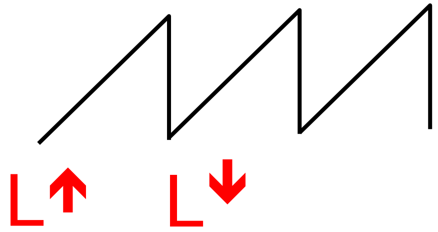
How does NetFence Work?

- Bottleneck router
  1. Detect attack to start a policing cycle
     - Loss or load based
  2. Signal congestion within a cycle
     - Random Early Detection (RED)

Recap: Why It Works

1. Secret keys to secure congestion policing feedback

2. Periodic AIMD based on secure congestion police feedback



3. Secure congestion feedback as network capabilities

Properties

- Provable fairness
  - Denial of Service → Predictable Delay of Service

Theorem: Given *G* good and *B* bad senders sharing a bottleneck link of capacity *C*, regardless of the attack strategies, any good sender *g* with sufficient demand eventually obtains a fair share

$$\frac{v_g \rho C}{G+B}$$

where $\rho \approx 1$ and v_g is a transport efficiency factor.

Properties

- Provable fairness
  - Denial of Service → Predictable Delay of Service

Theorem: Given *G* good and *B* bad senders sharing a bottleneck link of capacity *C*, regardless of the attack strategies, any good sender *g* with sufficient demand eventually obtains a fair share

$$\frac{v_g \rho C}{G+B}$$

where $\rho \approx 1$ and v_g is a transport efficiency factor.

Properties

- Provable fairness
  - Denial of Service → Predictable Delay of Service

Theorem: Given *G* good and *B* bad senders sharing a bottleneck link of capacity *C*, regardless of the attack strategies, any good sender *g* with sufficient demand eventually obtains a fair share

$$\frac{v_g \rho C}{G+B}$$

where $\rho \approx 1$ and v_g is a transport efficiency factor.

Now the Trickier Stuff

More Challenges

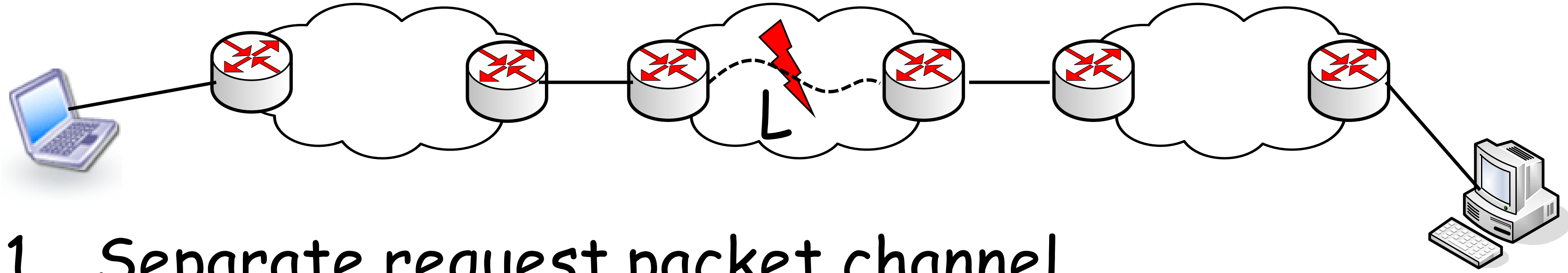
- A broad range of attacks
  - Flood request packets (with no feedback)
  - Hide L↓
  - Evade attack detection
  - On/Off
  - …
- Multiple bottlenecks
- Practical constraints
  - Low overhead
  - Gradual deployment
  - Incentive-compatible adoption

More Challenges

- A broad range of attacks
  - Flood request packets (with no feedback)
  - Hide L↓
  - Evade attack detection
  - On/Off
  - …

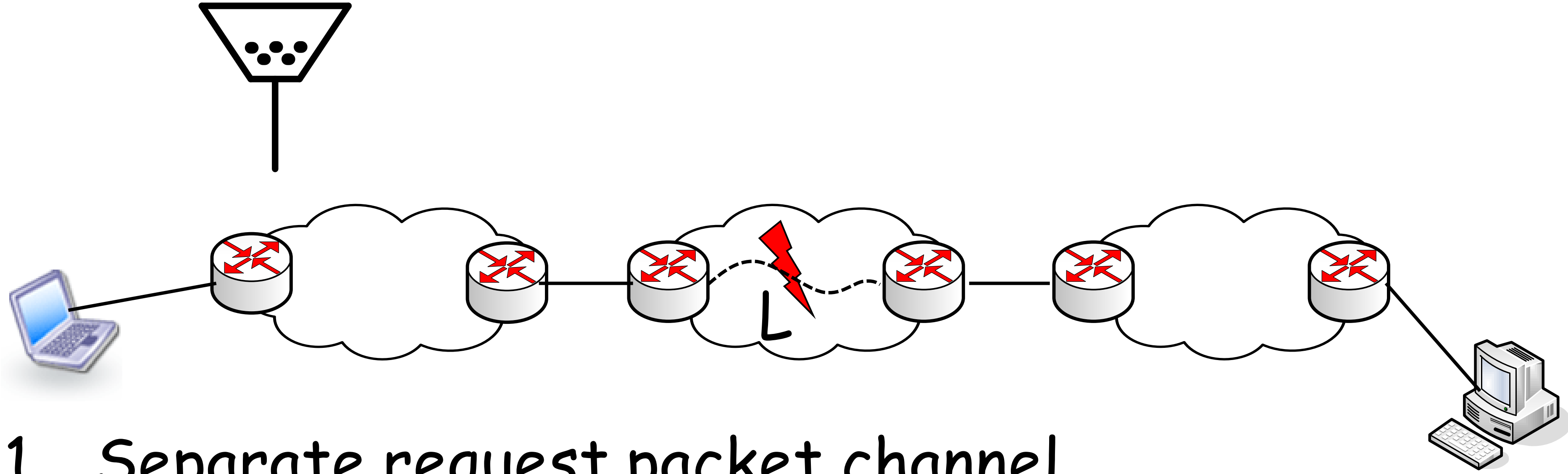
- Multiple bottlenecks
- Practical constraints
  - Low overhead
  - Gradual deployment
  - Incentive-compatible adoption

Limiting Request Packet Floods



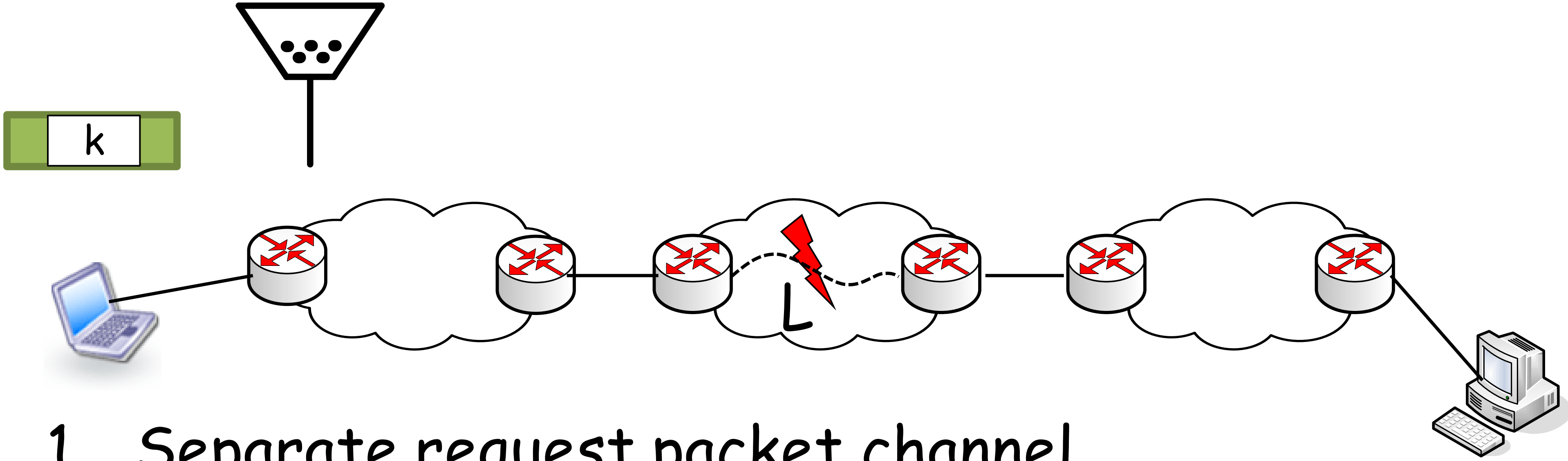
1. Separate request packet channel
2. Per-sender request packet policing
3. Priority-based backoff
   - Emulate computational puzzles

Limiting Request Packet Floods



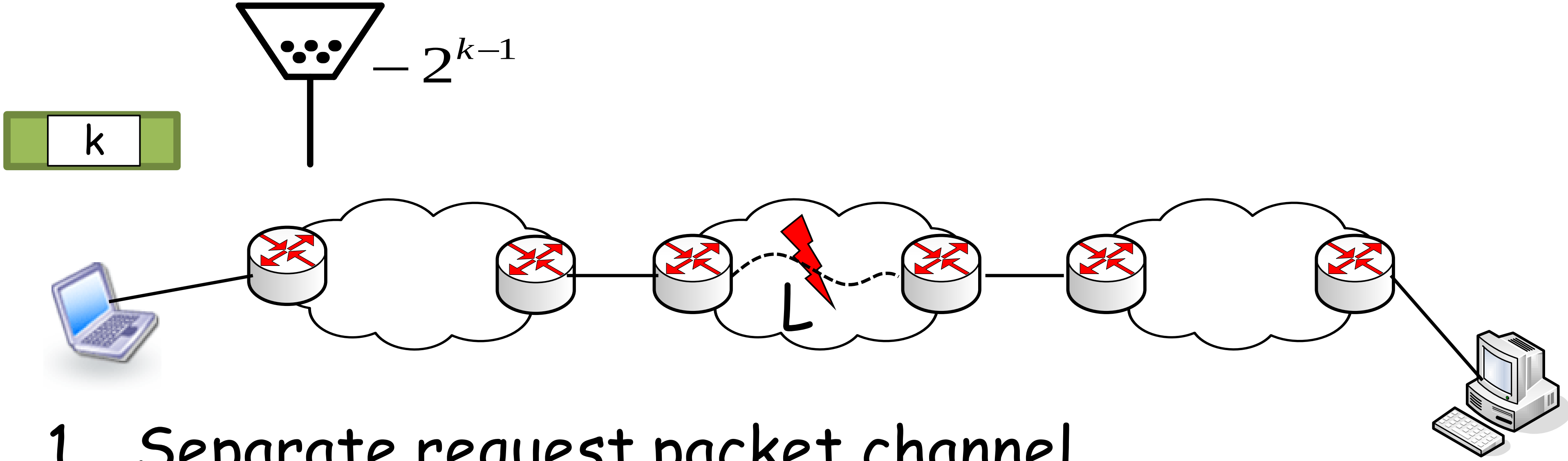
1. Separate request packet channel
2. Per-sender request packet policing
3. Priority-based backoff
   - Emulate computational puzzles

Limiting Request Packet Floods



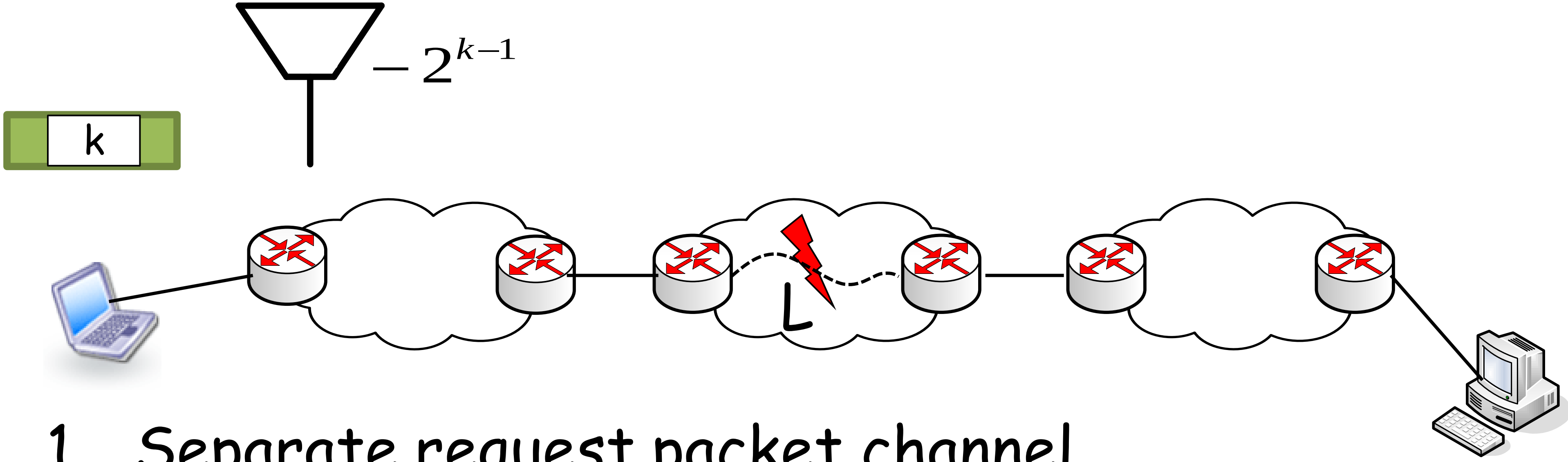
1. Separate request packet channel
2. Per-sender request packet policing
3. Priority-based backoff
   - Emulate computational puzzles

Limiting Request Packet Floods



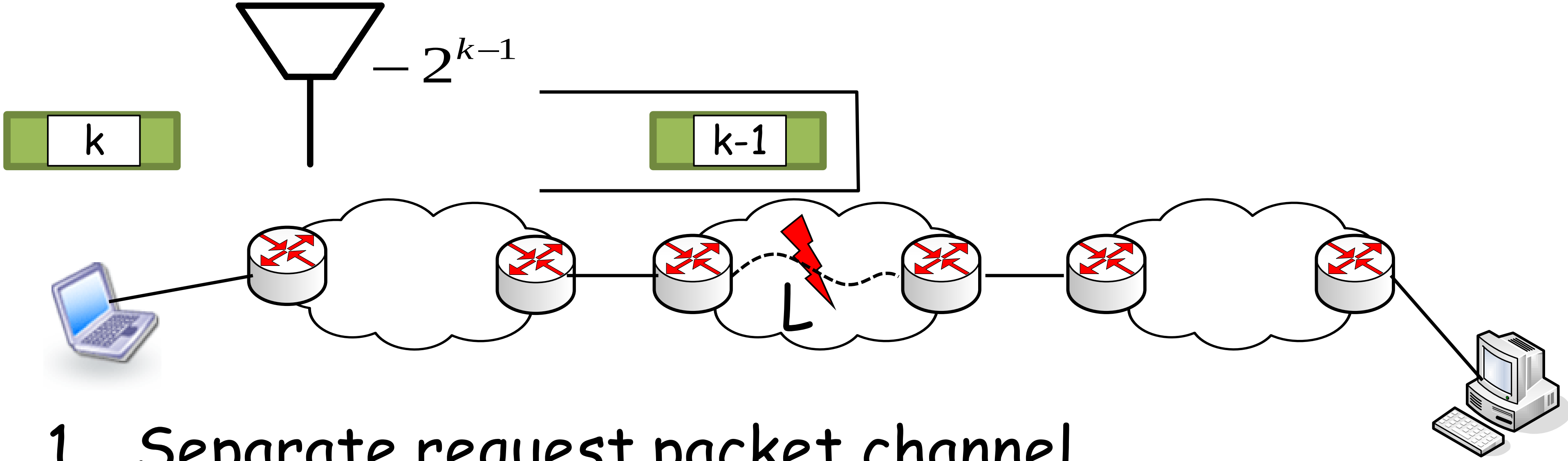
1. Separate request packet channel
2. Per-sender request packet policing
3. Priority-based backoff
   - Emulate computational puzzles

Limiting Request Packet Floods



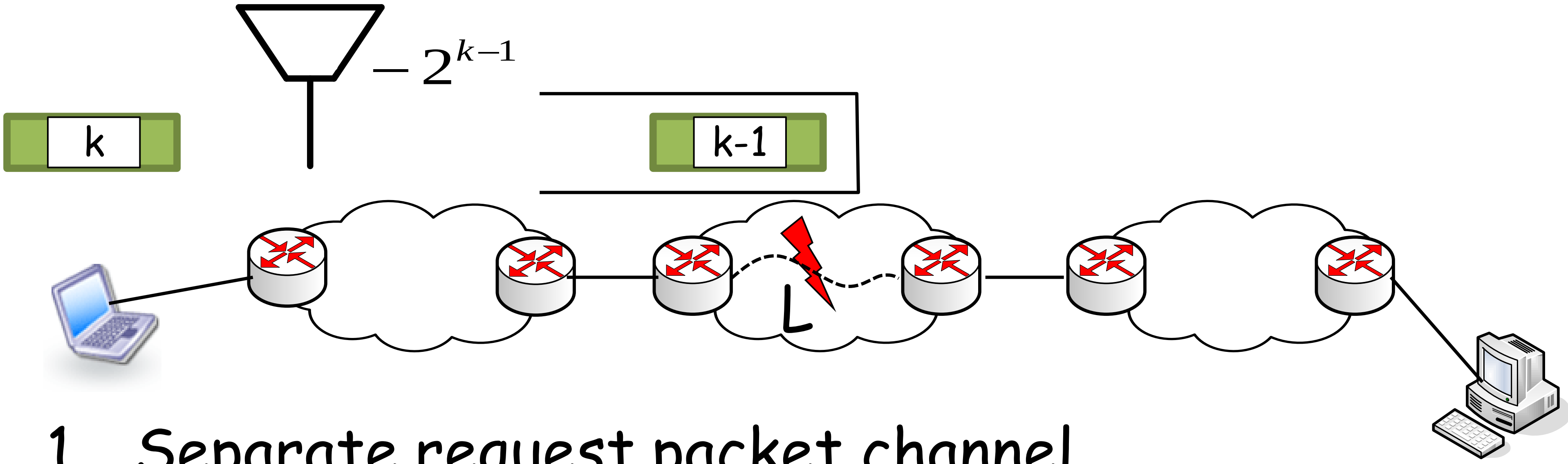
1. Separate request packet channel
2. Per-sender request packet policing
3. Priority-based backoff
   - Emulate computational puzzles

Limiting Request Packet Floods



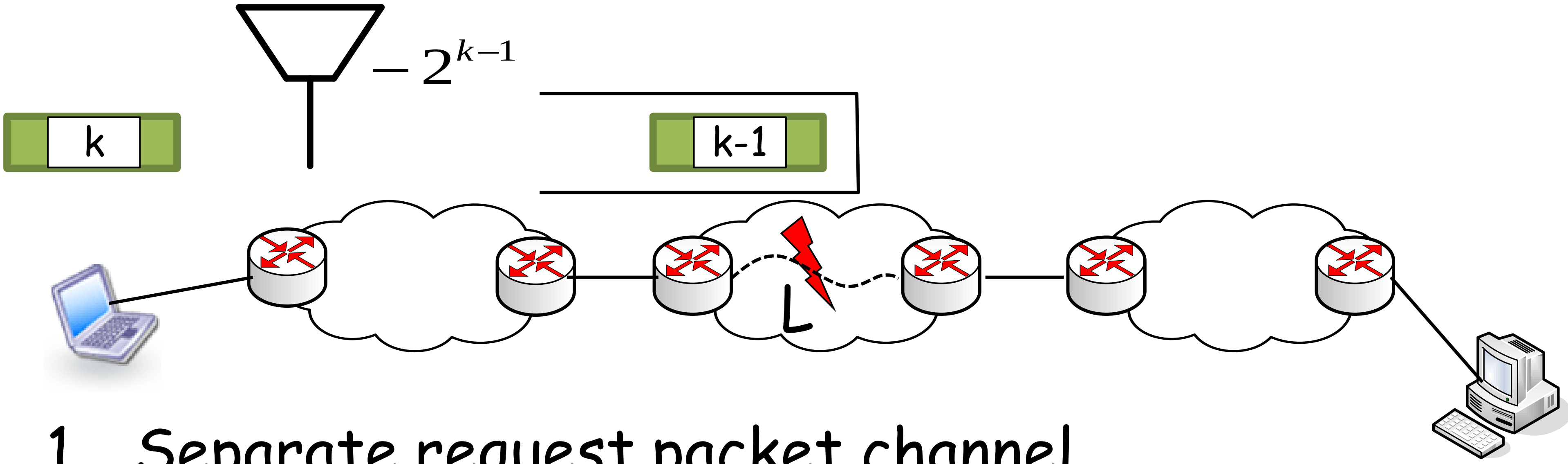
1. Separate request packet channel
2. Per-sender request packet policing
3. Priority-based backoff
   - Emulate computational puzzles

Limiting Request Packet Floods



1. Separate request packet channel
2. Per-sender request packet policing
3. Priority-based backoff
   - Emulate computational puzzles

Limiting Request Packet Floods



1. Separate request packet channel
2. Per-sender request packet policing
3. Priority-based backoff
   - Emulate computational puzzles

> 1. Eventual success
> 2. Efficient: waiting replaces proof of work

Making hiding L↓ ineffective

- Robust signaling rate increase with L↑
  1. Treating the absence of L↑ as L↓
  2. Stamping no L↑ for sufficiently long after congestion ends

Making hiding L↓ ineffective

t_1 t_2

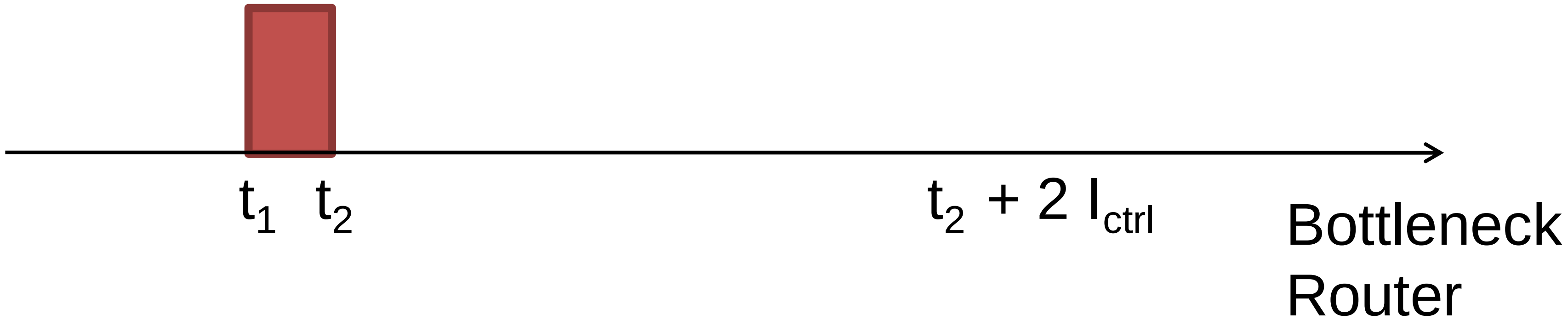
$t_2 + 2\,I_{ctrl}$

Bottleneck Router

- Robust signaling rate increase with L↑

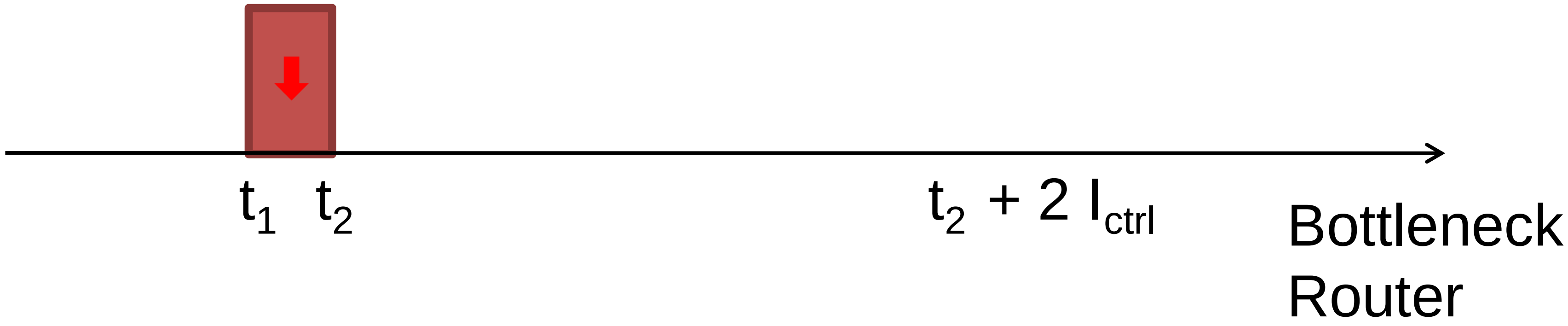
1. Treating the absence of L↑ as L↓
2. Stamping no L↑ for sufficiently long after congestion ends

Making hiding L↓ ineffective



- Robust signaling rate increase with L↑
  1. Treating the absence of L↑ as L↓
  2. Stamping no L↑ for sufficiently long after congestion ends

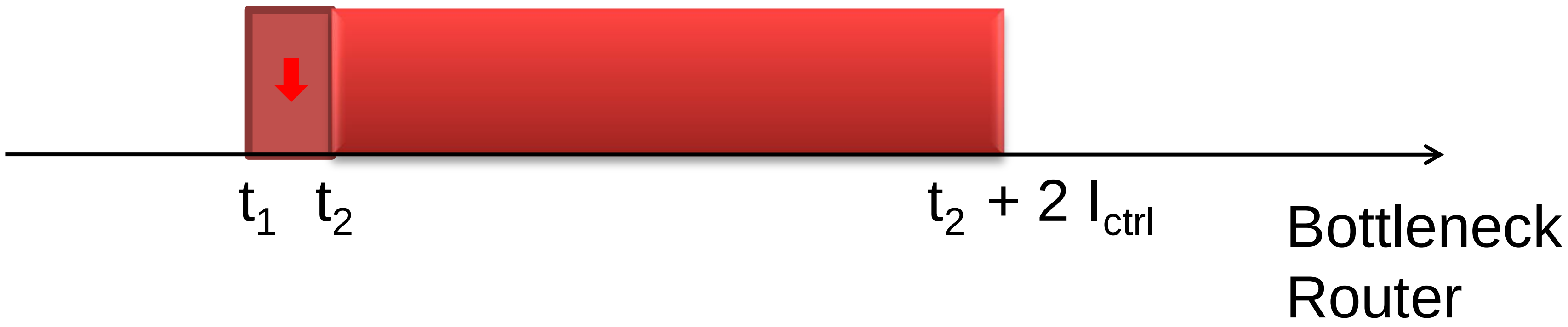
Making hiding L↓ ineffective



- Robust signaling rate increase with L↑

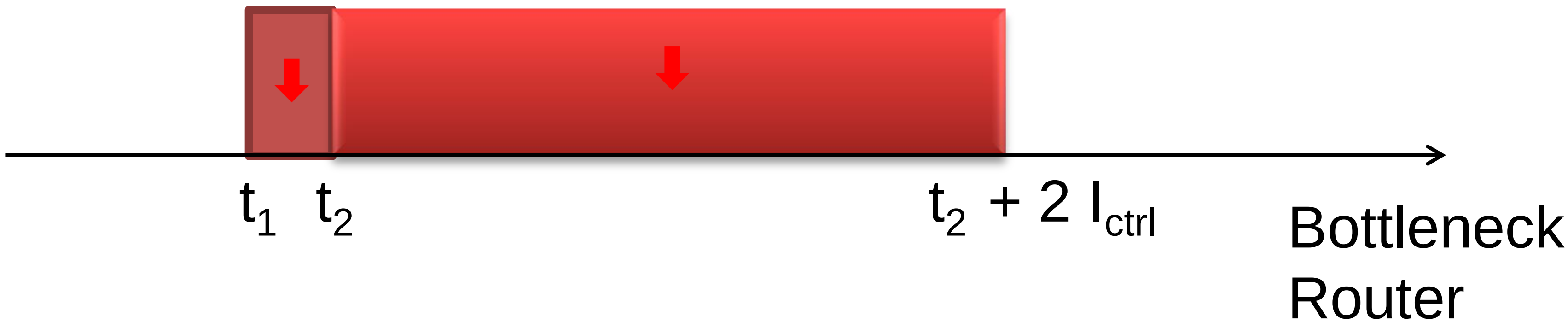
  1. Treating the absence of L↑ as L↓
  2. Stamping no L↑ for sufficiently long after congestion ends

Making hiding L↓ ineffective



- Robust signaling rate increase with L↑
  1. Treating the absence of L↑ as L↓
  2. Stamping no L↑ for sufficiently long after congestion ends

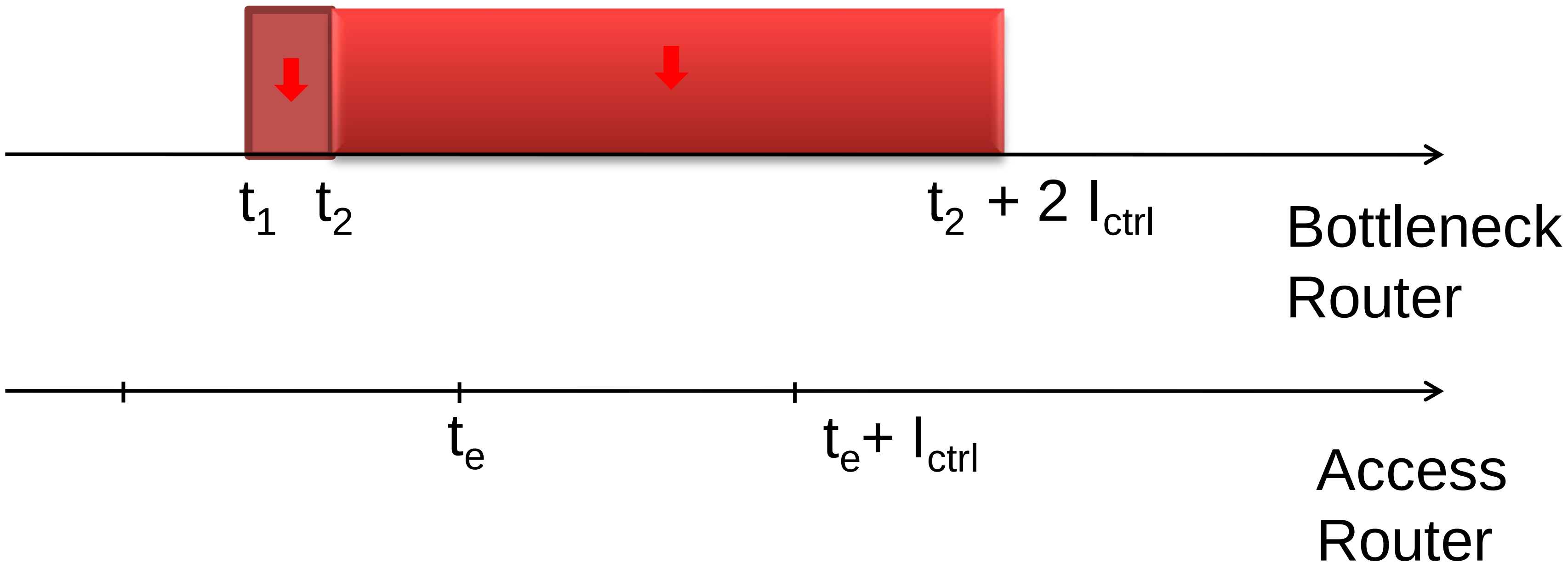
Making hiding L↓ ineffective



- Robust signaling rate increase with L↑

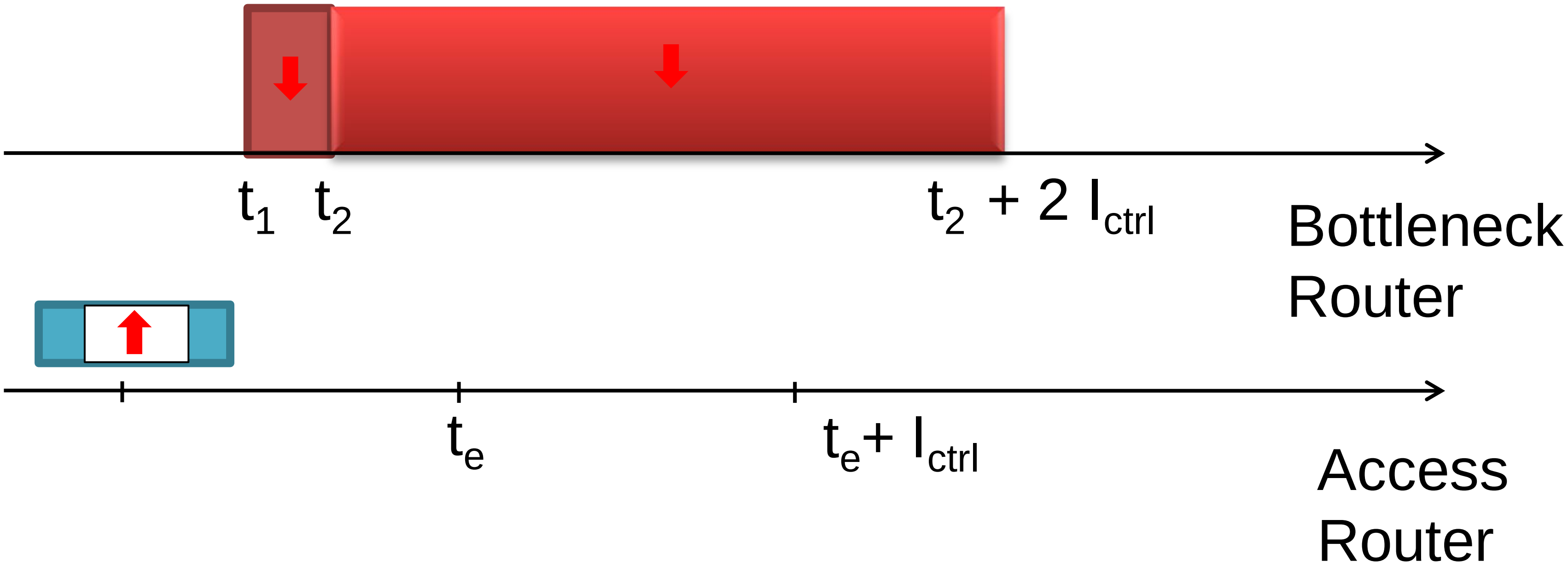
1. Treating the absence of L↑ as L↓
2. Stamping no L↑ for sufficiently long after congestion ends

Making hiding L↓ ineffective



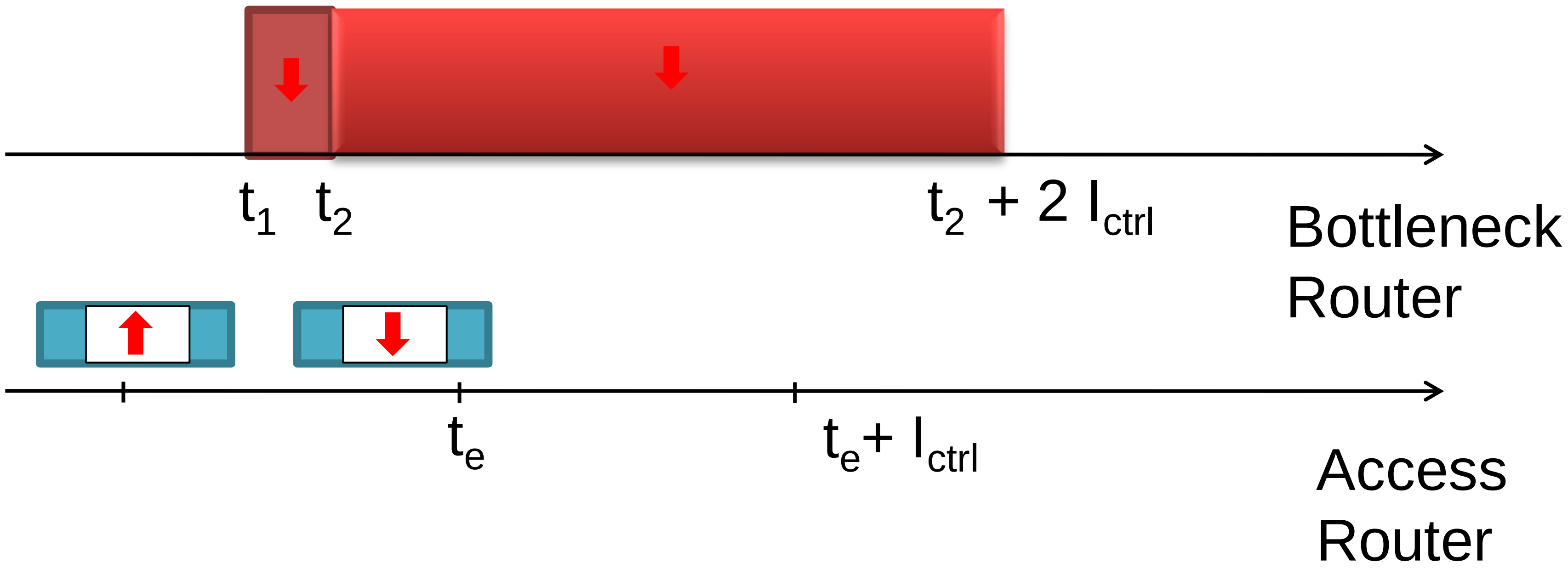
- Robust signaling rate increase with L↑
  1. Treating the absence of L↑ as L↓
  2. Stamping no L↑ for sufficiently long after congestion ends

Making hiding L↓ ineffective



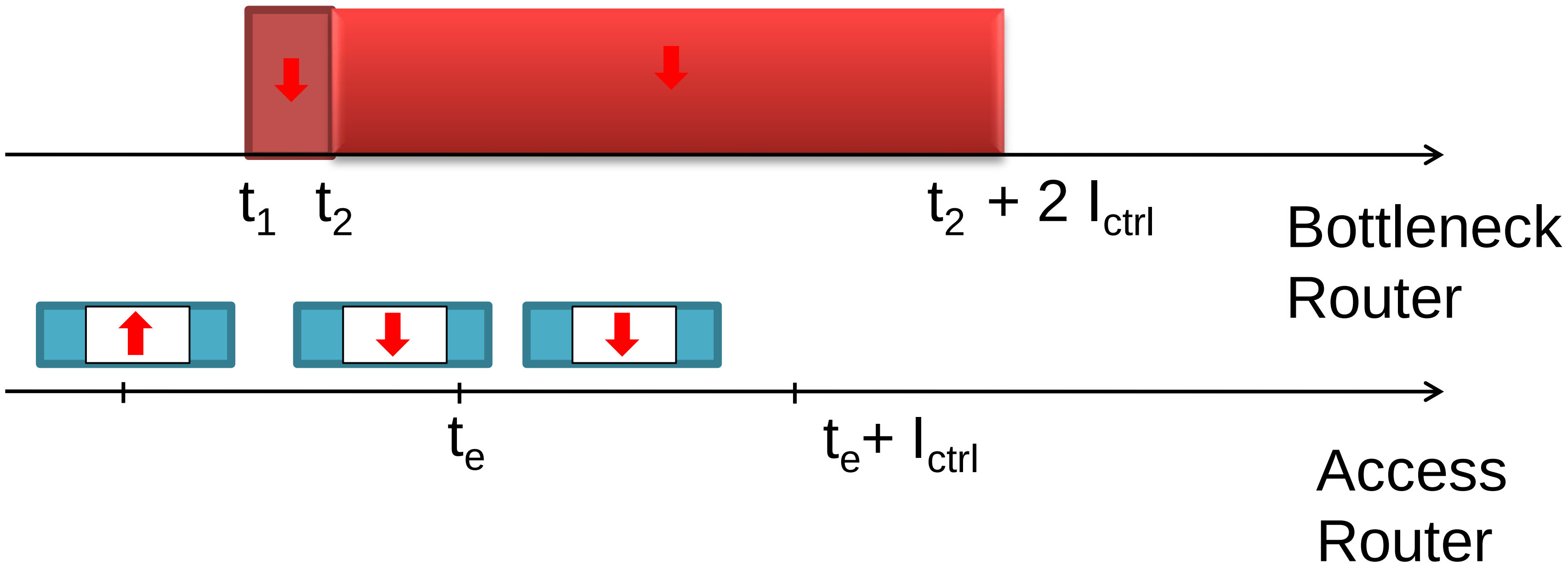
- Robust signaling rate increase with L↑
  1. Treating the absence of L↑ as L↓
  2. Stamping no L↑ for sufficiently long after congestion ends

Making hiding L↓ ineffective



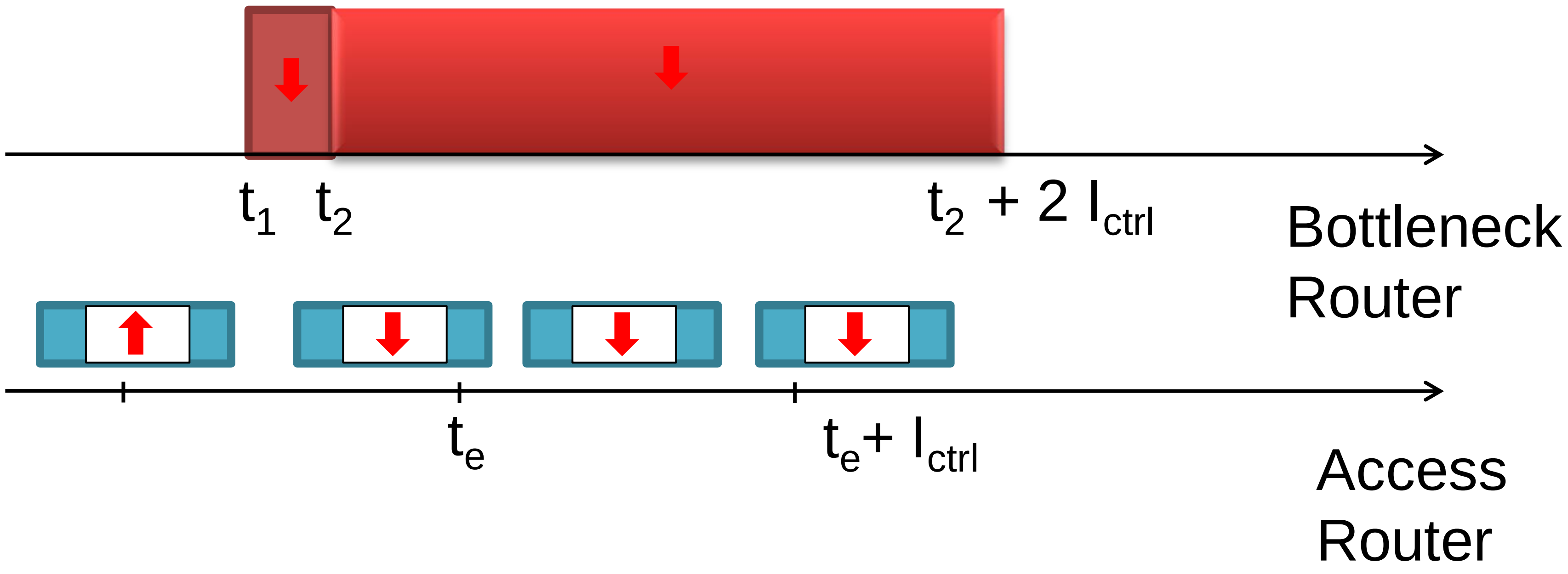
- Robust signaling rate increase with L↑
  1. Treating the absence of L↑ as L↓
  2. Stamping no L↑ for sufficiently long after congestion ends

Making hiding L↓ ineffective



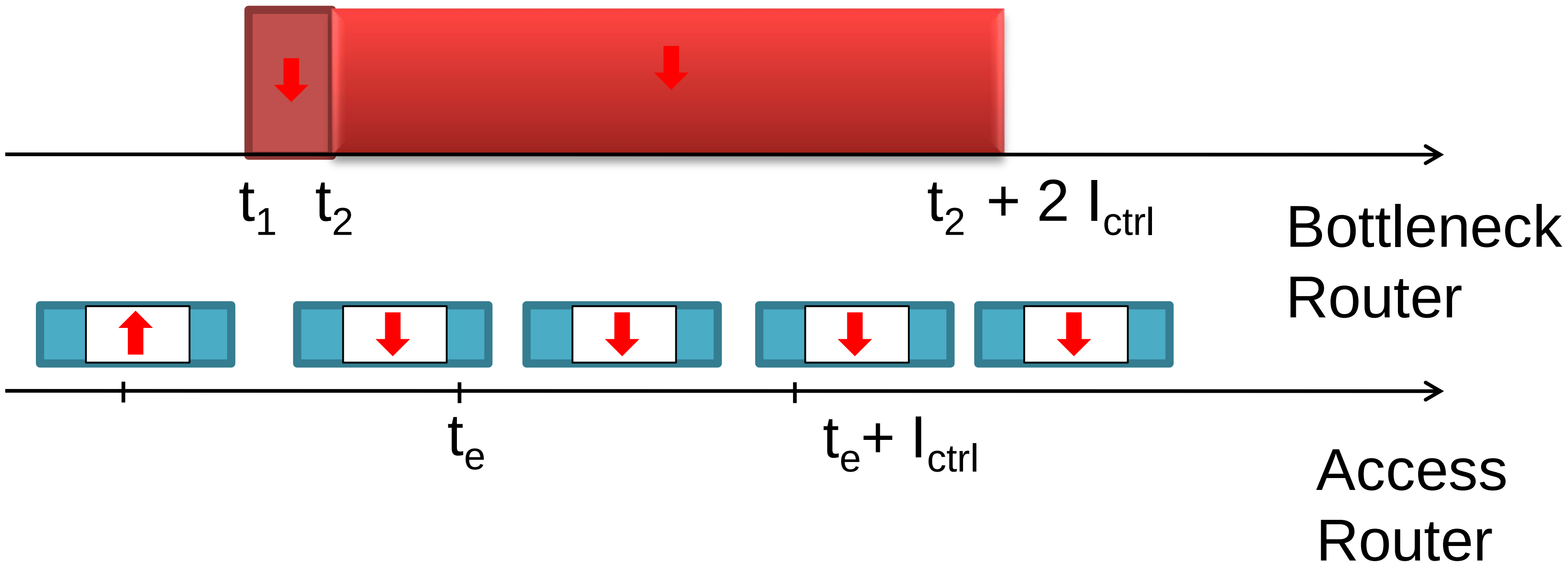
- Robust signaling rate increase with L↑
  1. Treating the absence of L↑ as L↓
  2. Stamping no L↑ for sufficiently long after congestion ends

Making hiding $L^{\downarrow}$ ineffective



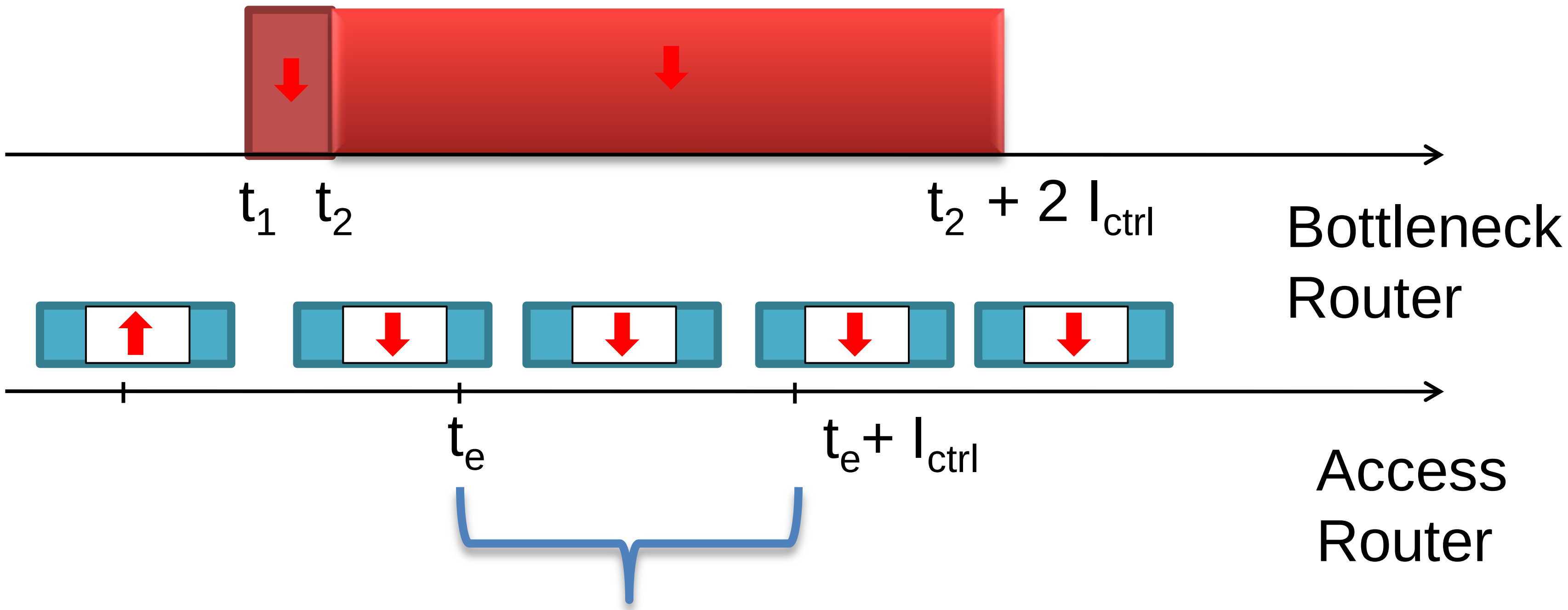
- Robust signaling rate increase with $L^{\uparrow}$
  1. Treating the absence of $L^{\uparrow}$ as $L^{\downarrow}$
  2. Stamping no $L^{\uparrow}$ for sufficiently long after congestion ends

Making hiding L↓ ineffective



- Robust signaling rate increase with L↑
  1. Treating the absence of L↑ as L↓
  2. Stamping no L↑ for sufficiently long after congestion ends

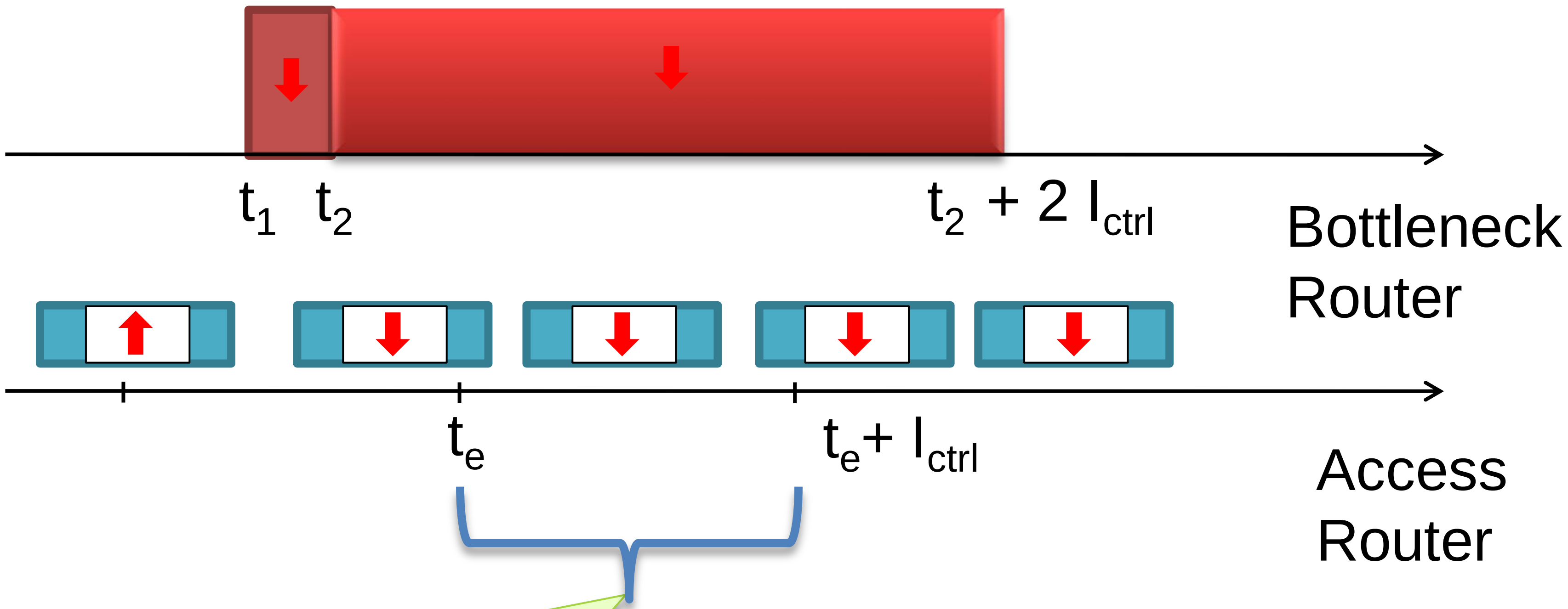
Making hiding L↓ ineffective



- Robust signaling rate increase with L↑

1. Treating the absence of L↑ as L↓
2. Stamping no L↑ for sufficiently long after congestion ends

Making hiding $L^{\downarrow}$ ineffective



➔ $t_e + I_{ctrl} \le t_2 + 2I_{ctrl}$

➔ A sender can't present $L^{\uparrow}$

➔ Rate limit is reduced

- Rob … ith $L^{\uparrow}$

1. T
2. S … ong after congestion ends

Performance

Implementation

- A software implementation in Linux
  - XORP and Click
  - AES-128 as the MAC function

- DeterLab experiments
  - Dual-core Intel Xeon 3GHz CPUs
  - 2GB memory

Implementation

- A software implementation in Linux
  - XORP and Click
  - AES-128 as the MAC function

Encrypting the Internet!

- DeterLab
  - Dual-core Intel Xeon 3GHz CPUs
  - 2GB memory

Implementation

- A software implementation in Linux
  - XORP and Click
  - AES-128 as the MAC function

- DeterLab experiments
  - Dual-core Intel Xeon 3GHz CPUs
  - 2GB memory

Processing overhead

	Packet type	Access router	Bottleneck router
No Attack	Request	546 ns/pkt	0
	Regular	781 ns/pkt	0
Attack	Request	546 ns/pkt	492 ns/pkt
	Regular	1267 ns/pkt	554 ns/pkt

Processing overhead

	Packet type	Access router	Bottleneck router
No Attack	Request	546 ns/pkt	0
	Regular	781 ns/pkt	0
Attack	Request	546 ns/pkt	492 ns/pkt
	Regular	1267 ns/pkt	554 ns/pkt

Processing overhead

	Packet type	Access router	Bottleneck router
No Attack	Request	546 ns/pkt	0
	Regular	781 ns/pkt	0
Attack	Request	546 ns/pkt	492 ns/pkt
	Regular	1267 ns/pkt	554 ns/pkt

Processing overhead

	Packet type	Access router	Bottleneck router
No Attack	Request	546 ns/pkt	0
	Regular	781 ns/pkt	0
Attack	Request	546 ns/pkt	492 ns/pkt
	Regular	1267 ns/pkt	554 ns/pkt

One AES computation
Tput ~ 2mpps

Processing overhead

	Packet type	Access router	Bottleneck router
No Attack	Request	546 ns/pkt	0
	Regular	781 ns/pkt	0
Attack	Request	546 ns/pkt	492 ns/pkt
	Regular	1267 ns/pkt	554 ns/pkt

One AES computation
Tput ~ 2mpps

Processing overhead

	Packet type	Access router	Bottleneck router
No Attack	Request	546 ns/pkt	0
	Regular	781 ns/pkt	0
Attack	Request	546 ns/pkt	492 ns/pkt
	Regular	1267 ns/pkt	554 ns/pkt

≤ 3AES computation.
Parallelizable

One AES
computation
Tput ~ 2mpps

Processing overhead

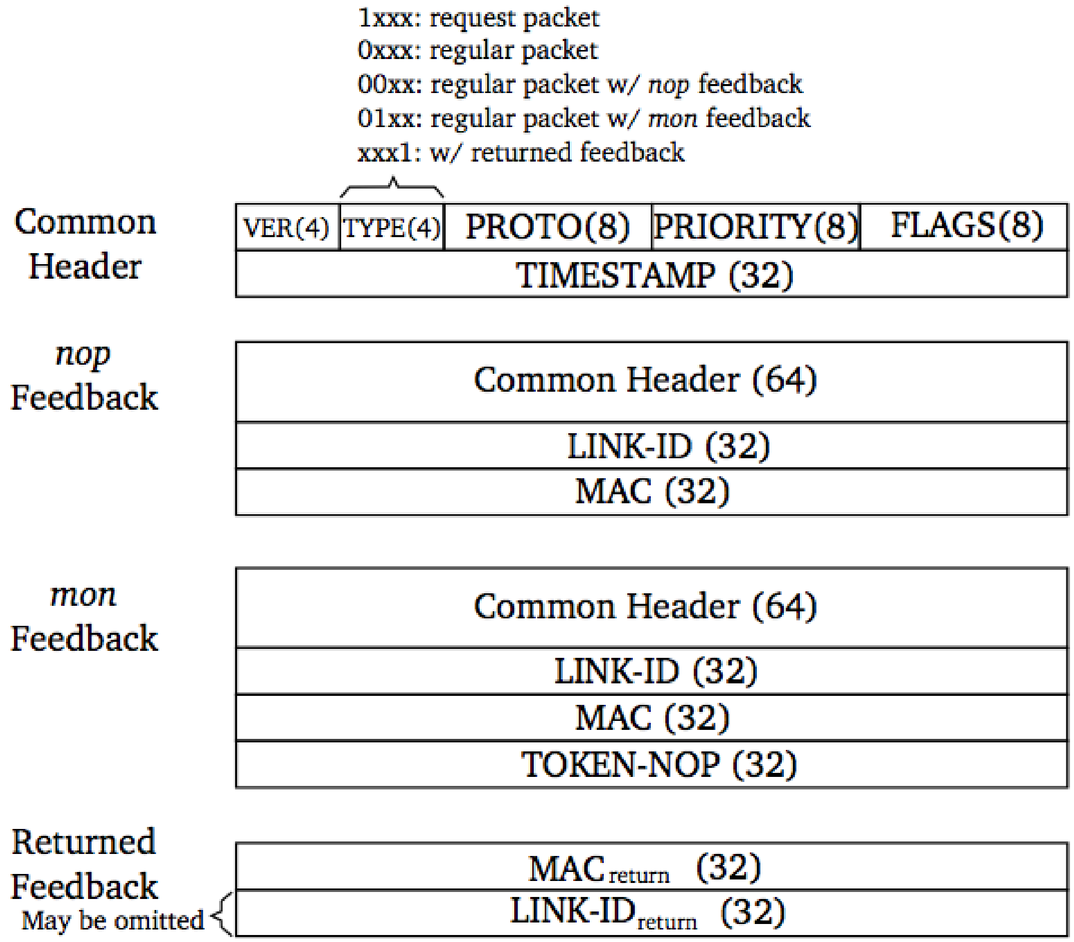
	Packet type	Access router	Bottleneck router
No Attack	Request	546 ns/pkt	0
	Regular	781 ns/pkt	0
Attack	Request	546 ns/pkt	492 ns/pkt
	Regular	1267 ns/pkt	554 ns/pkt

≤ 3AES computation.
Parallelizable

One AES computation
Tput ~ 2mpps

NetFence is suitable for high-speed implementation

Header overhead



1xxx: request packet
0xxx: regular packet
00xx: regular packet w/ *nop* feedback
01xx: regular packet w/ *mon* feedback
xxx1: w/ returned feedback

Common Header

VER(4)	TYPE(4)	PROTO(8)	PRIORITY(8)	FLAGS(8)
TIMESTAMP (32)				

nop Feedback

Common Header (64)
LINK-ID (32)
MAC (32)

mon Feedback

Common Header (64)
LINK-ID (32)
MAC (32)
TOKEN-NOP (32)

Returned Feedback

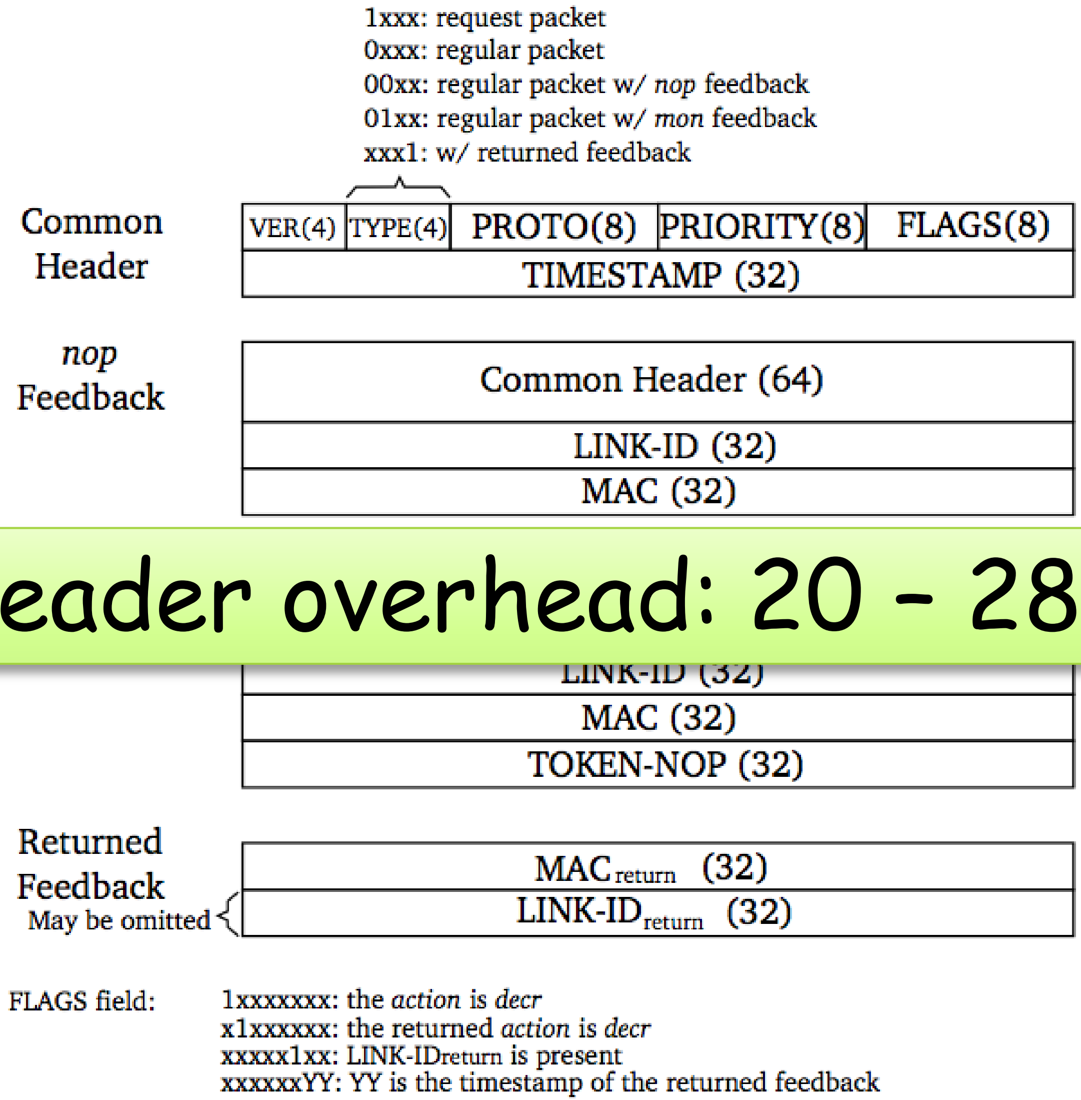
$\text{MAC}_{\text{return}}$ (32)
$\text{LINK-ID}_{\text{return}}$ (32)

May be omitted

FLAGS field:

1xxxxxxx: the *action* is *decr*
x1xxxxxx: the returned *action* is *decr*
xxxxx1xx: LINK-IDreturn is present
xxxxxxYY: YY is the timestamp of the returned feedback

Header overhead



1xxx: request packet
0xxx: regular packet
00xx: regular packet w/ *nop* feedback
01xx: regular packet w/ *mon* feedback
xxx1: w/ returned feedback

Common Header

VER(4)	TYPE(4)	PROTO(8)	PRIORITY(8)	FLAGS(8)
TIMESTAMP (32)				

nop Feedback

Common Header (64)
LINK-ID (32)
MAC (32)

Header overhead: 20 - 28 bytes

LINK-ID (32)
MAC (32)
TOKEN-NOP (32)

Returned Feedback

MAC$_{return}$ (32)
LINK-ID$_{return}$ (32)

May be omitted

FLAGS field:

1xxxxxxx: the *action* is *decr*
x1xxxxxx: the returned *action* is *decr*
xxxxx1xx: LINK-IDreturn is present
xxxxxxYY: YY is the timestamp of the returned feedback

Simulations

- Extensive ns-2 simulations
- Systems compared: more state in core
  - Per-sender Fair Queuing (FQ)
  - TVA+: capability + per-sender/receiver FQ
  - StopIt: filter + per-sender FQ

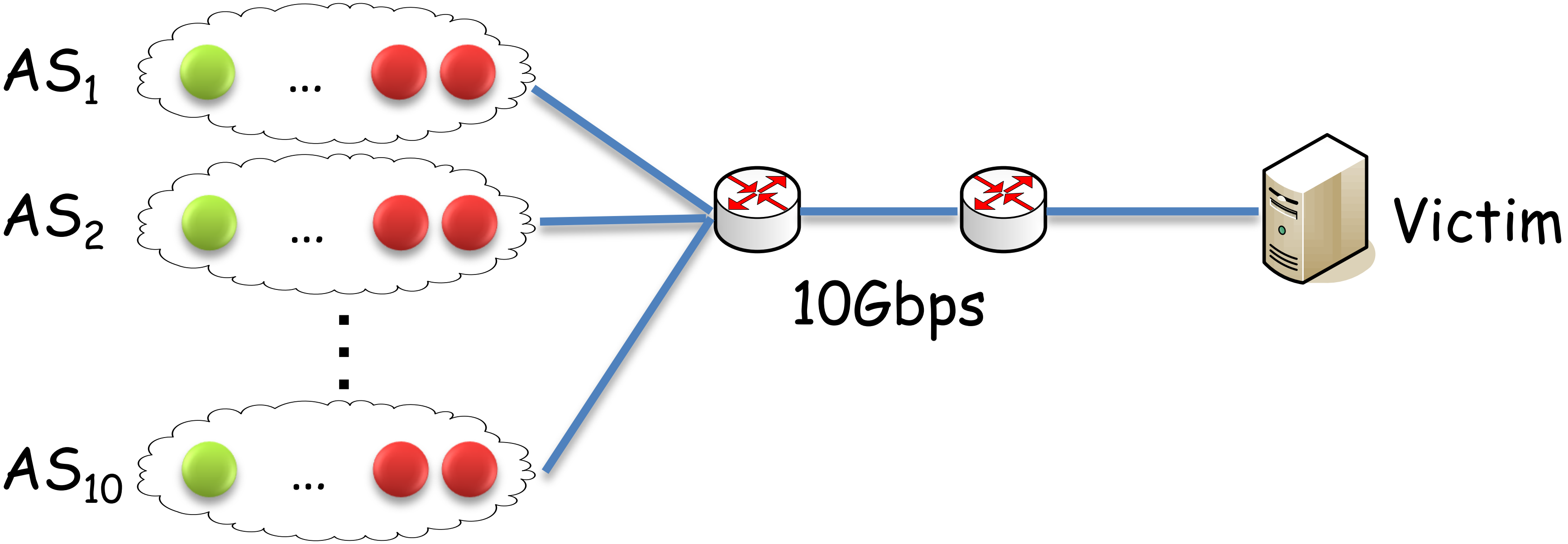
NetFence

- Enables receivers to suppress unwanted traffic
- Effectively polices malicious flows

→ A robust and scalable DoS solution

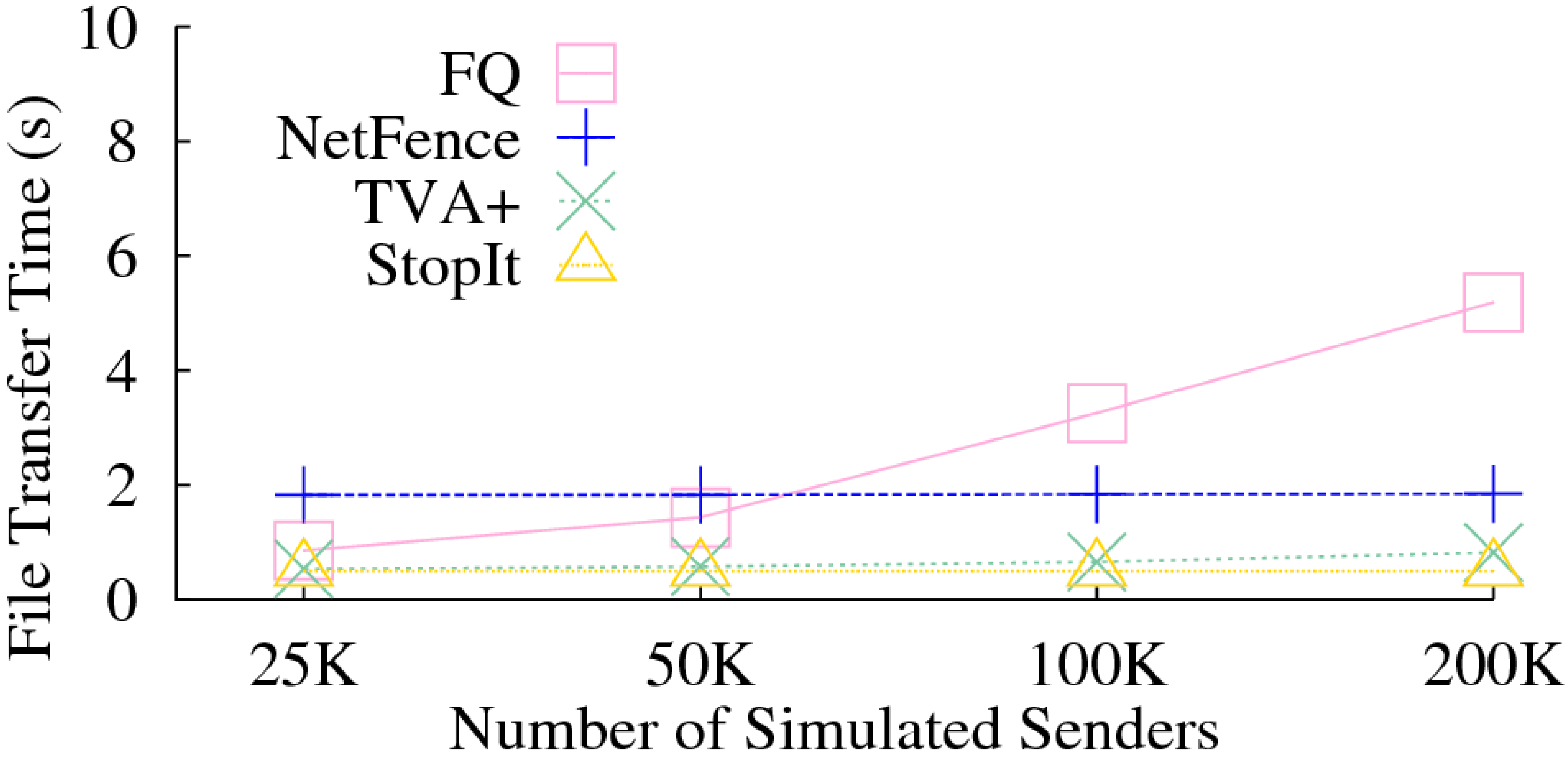
A Subset of Results

Expr 1: DoES Attacks



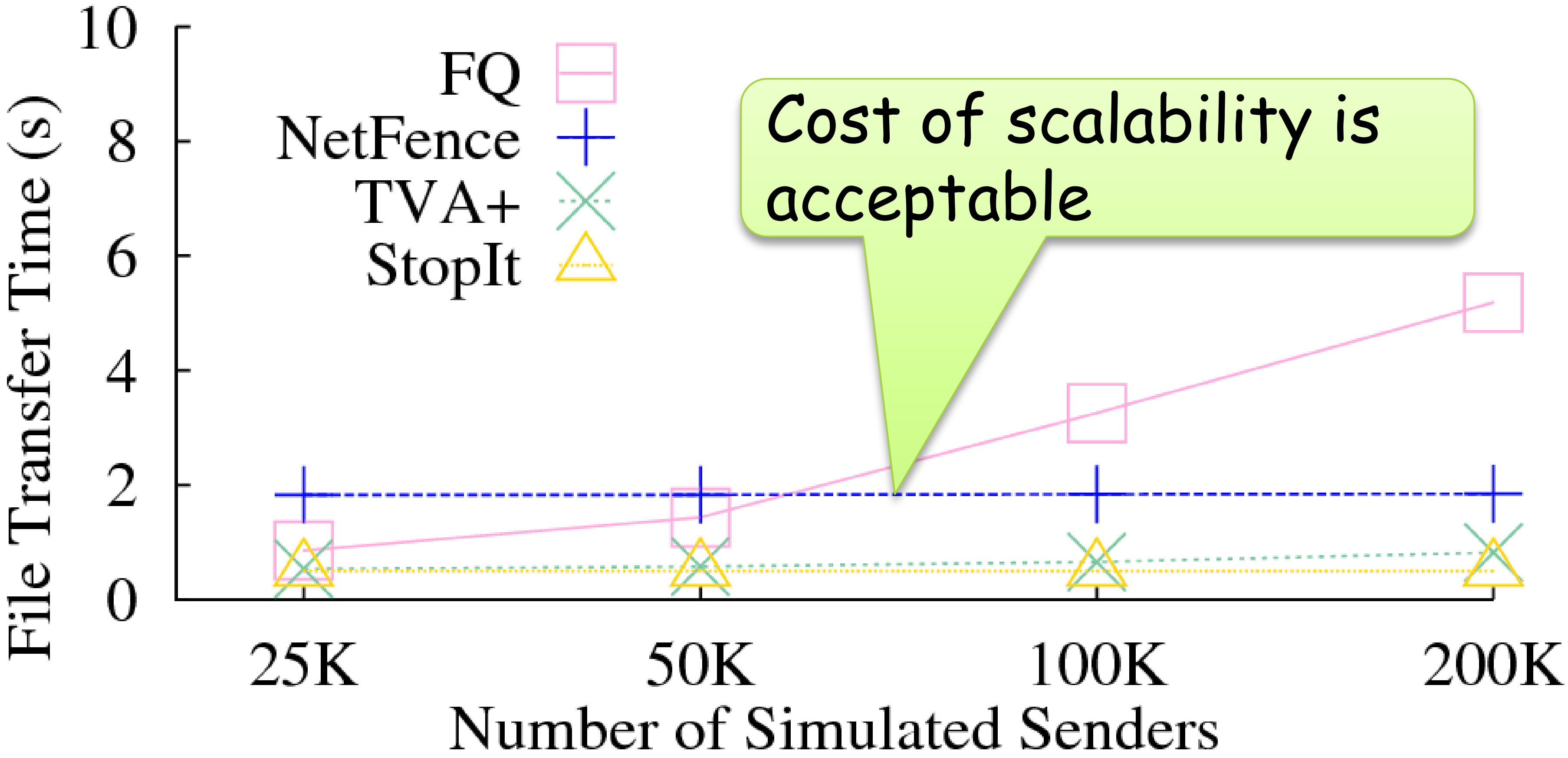
- In each source AS
  - 1 user sends a 20KB file to a victim via TCP
  - 99 attackers each send 1Mbps UDP traffic to the victim

NetFence Limits DoES



- All transfer finishes despite attackers >> users
- No per-sender queues

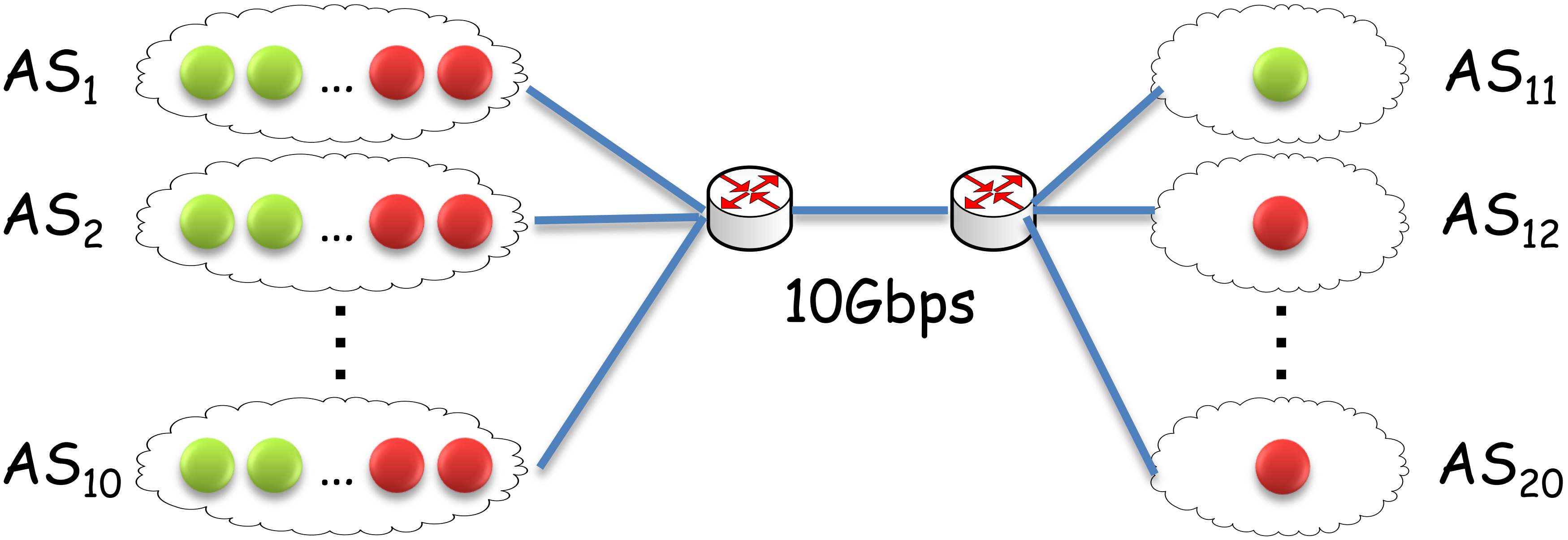
NetFence Limits DoES



Cost of scalability is acceptable

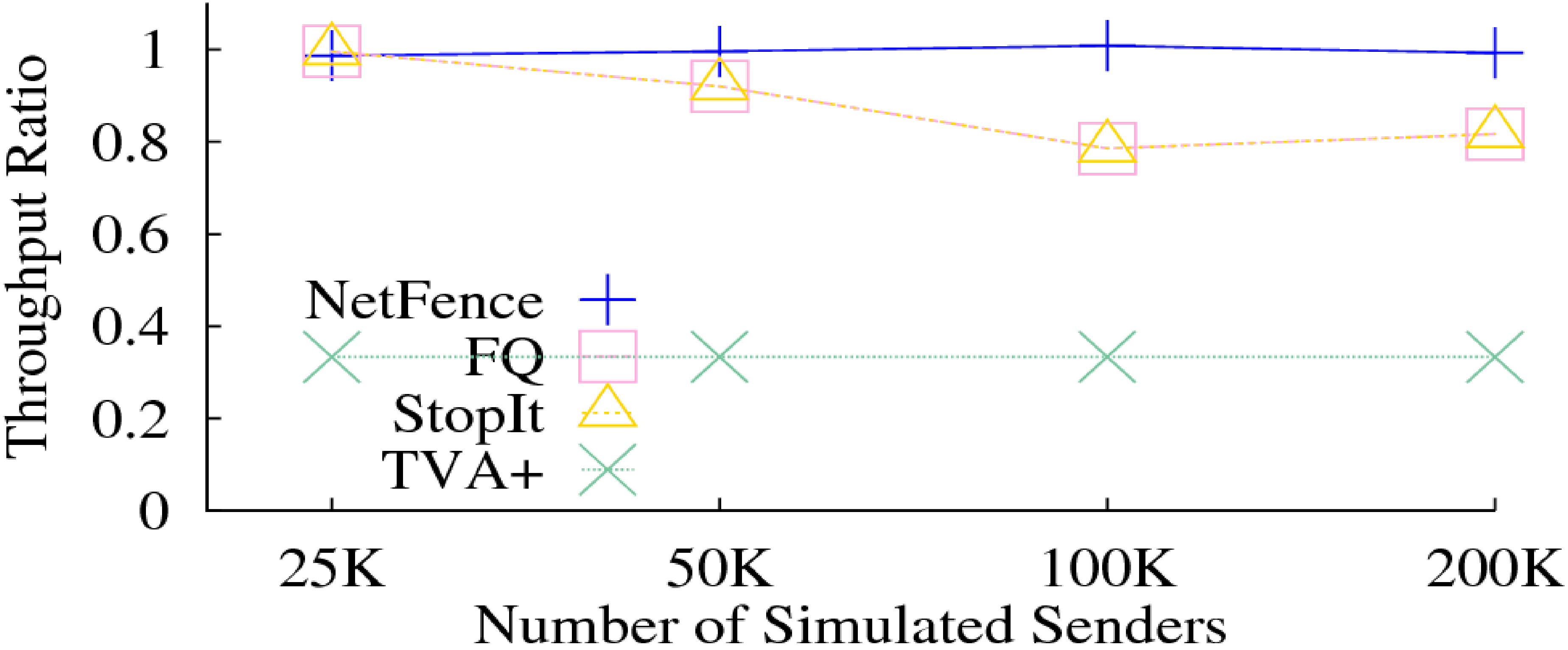
- All transfer finishes despite attackers >> users
- No per-sender queues

Expr 2: DoNS Attacks



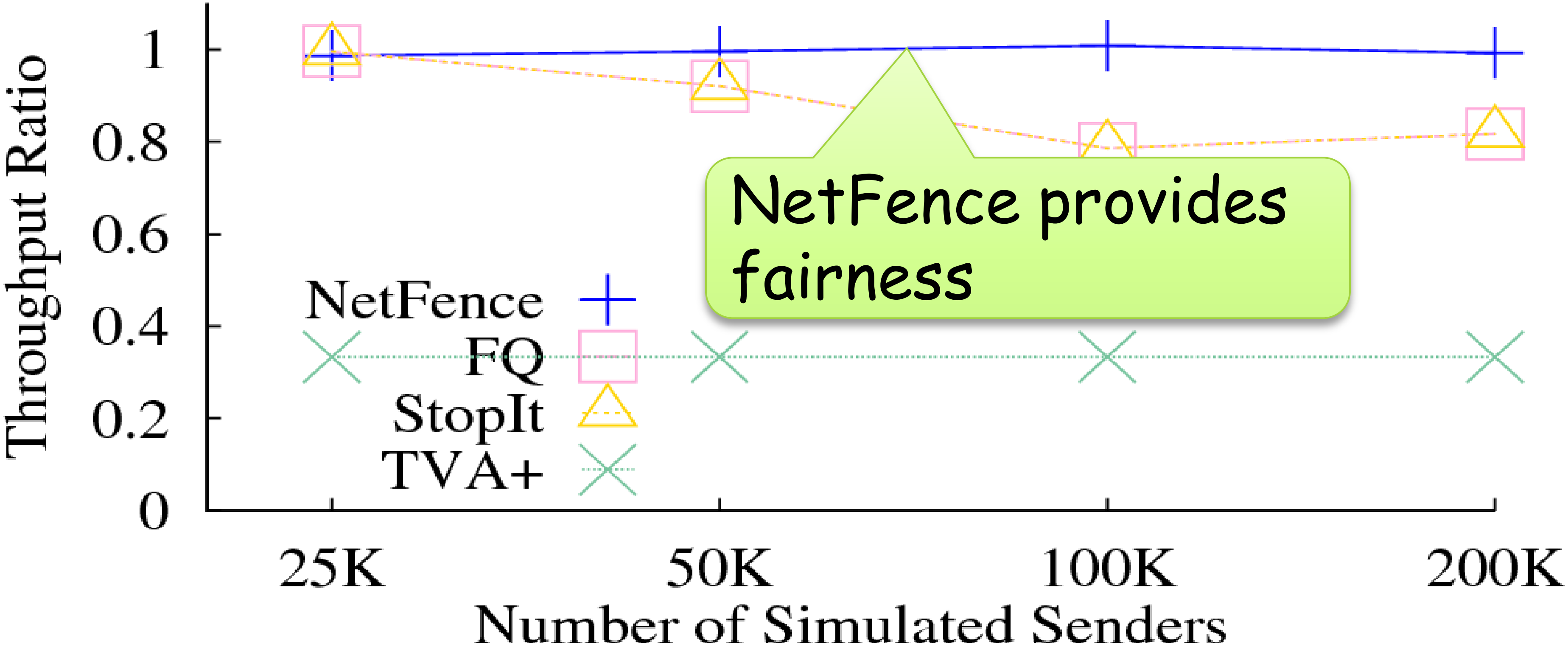
- In each source AS
  - 25% legitimate users and 75% attackers
- In each destination AS
  - One legitimate receiver or one colluding attacker

NetFence Limits DoNS



- Throughput ratio = avg(user)/avg(attacker)

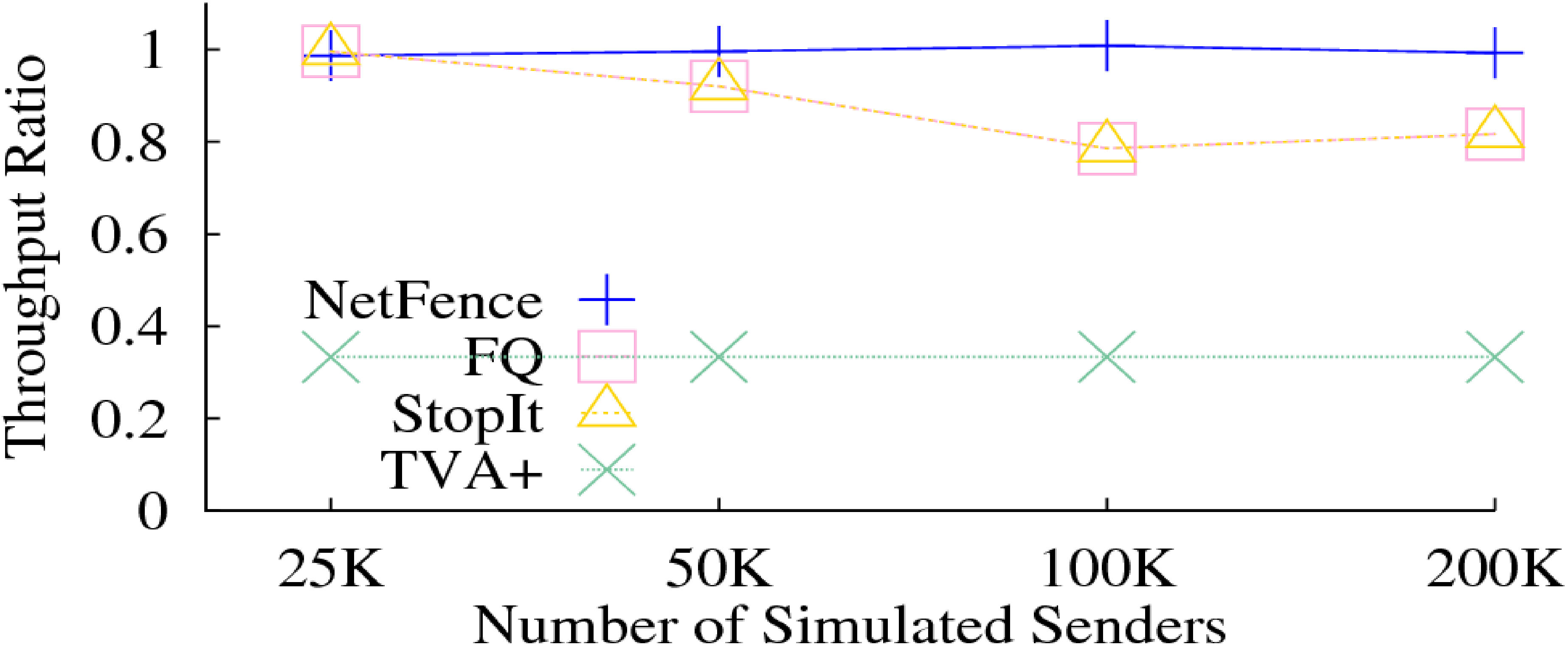
NetFence Limits DoNS



NetFence provides fairness

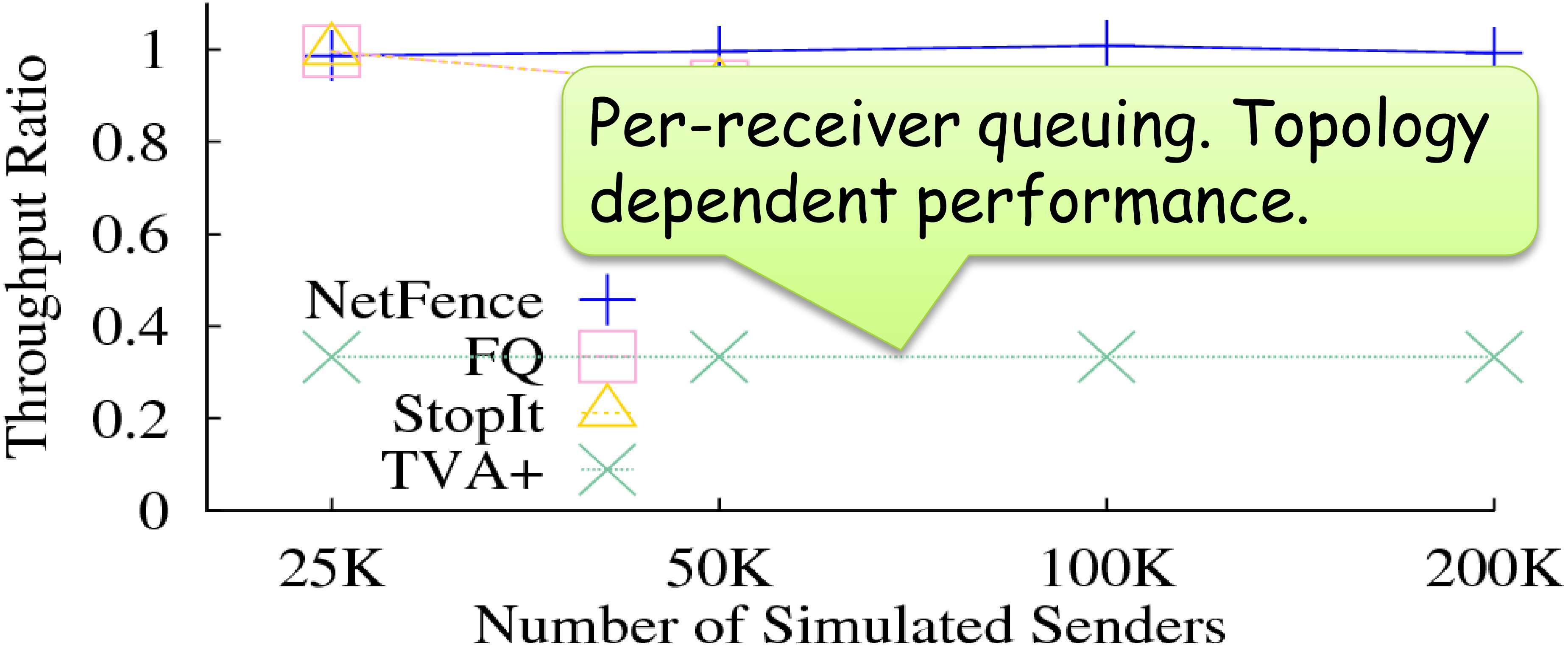
- Throughput ratio = avg(user)/avg(attacker)

NetFence Limits DoNS



- Throughput ratio = avg(user)/avg(attacker)

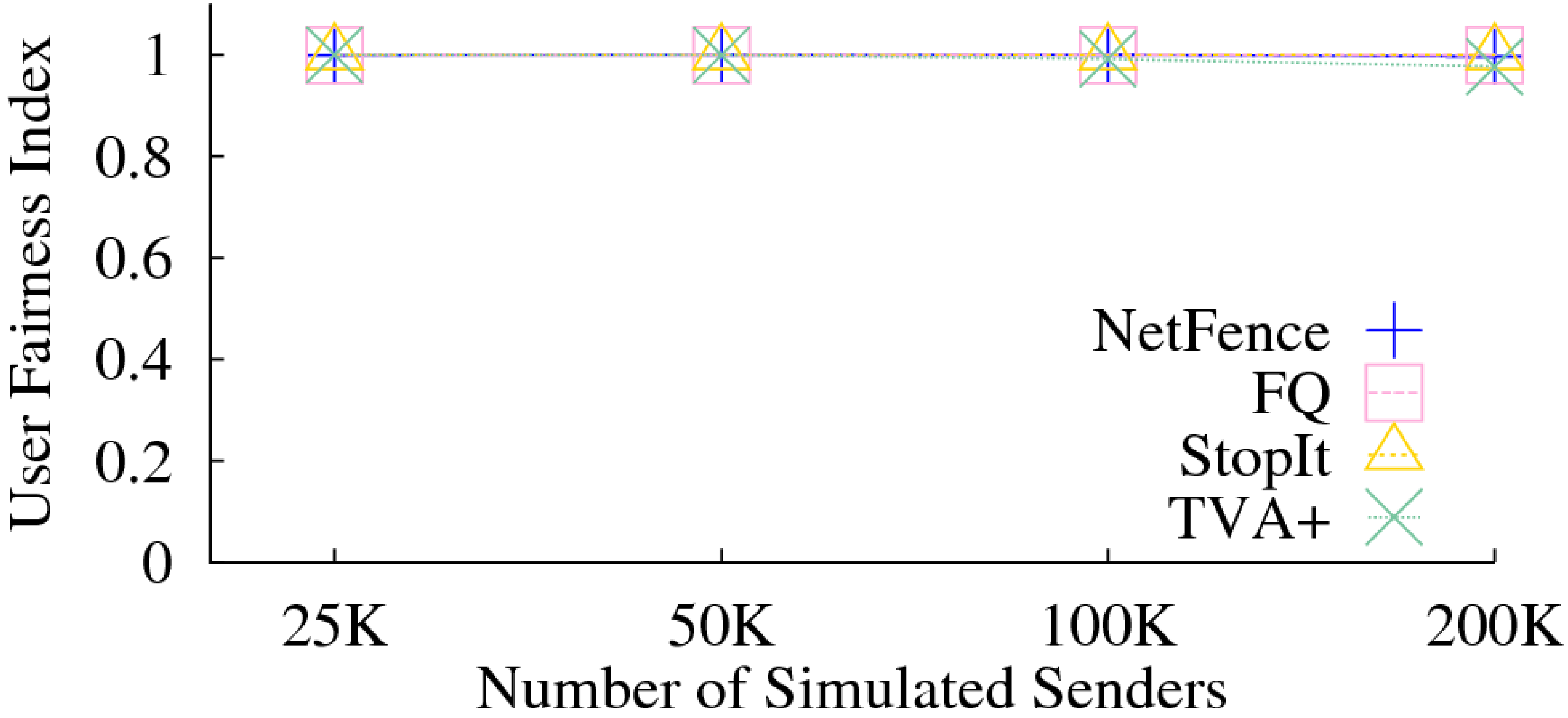
NetFence Limits DoNS



Per-receiver queuing. Topology dependent performance.

- Throughput ratio = avg(user)/avg(attacker)

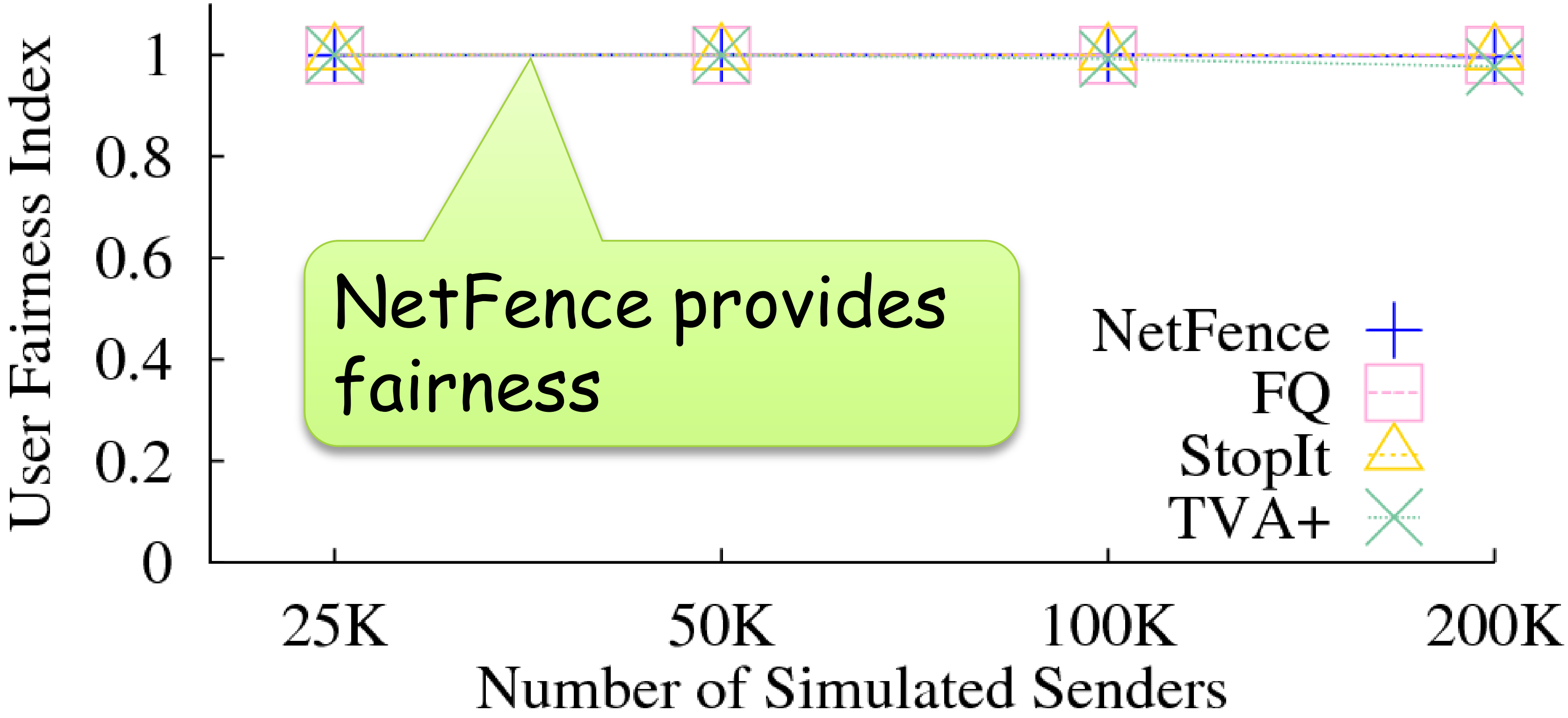
NetFence Limits DoNS



- Fairness index among legitimate users

$$(\sum x_i)^2 / n \sum x_i^2$$

NetFence Limits DoNS

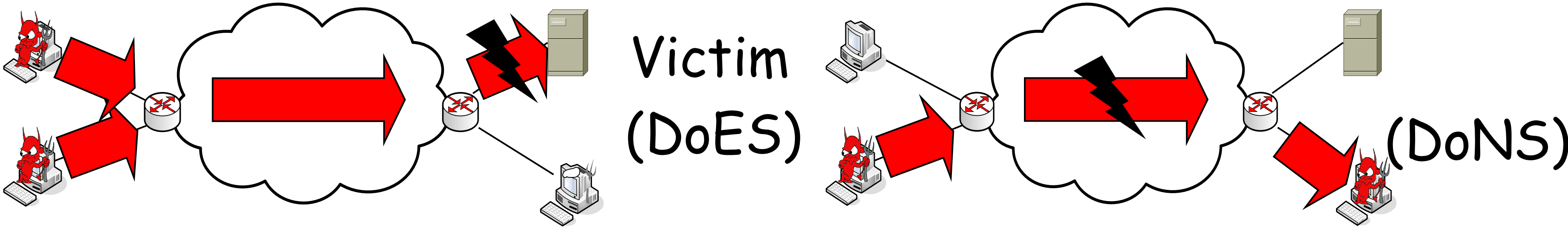


NetFence provides fairness

- Fairness index among legitimate users

$$(\sum x_i)^2 / n \sum x_i^2$$

Conclusion



Victim

(DoES)

(DoNS)

- NetFence
  - First comprehensive solution combating DoES and DoNS attacks scalably
  - Design principle: inside-out, network-host joint lines of defense
  - Goals: Scalable, robust, and open
  - Key idea: Hierarchical, secure congestion policing coupled with network capabilities

Thank you!

- Questions
  - xwy@cs.duke.edu
  - xinl@cs.duke.edu
  - xia_yong@nec.cn