Programmable network monitoring

and what to do with it…

Laurent Vanbever

nsg.ee.ethz.ch

Google Networking Summit

March 12 2019

Programmable network monitoring

and what to do with it…

Stroboscope [NSDI 2018]	Blink [NSDI 2019]
fine-grained network monitoring	data-driven fast rerouting

Programmable network monitoring

and what to do with it…

Stroboscope

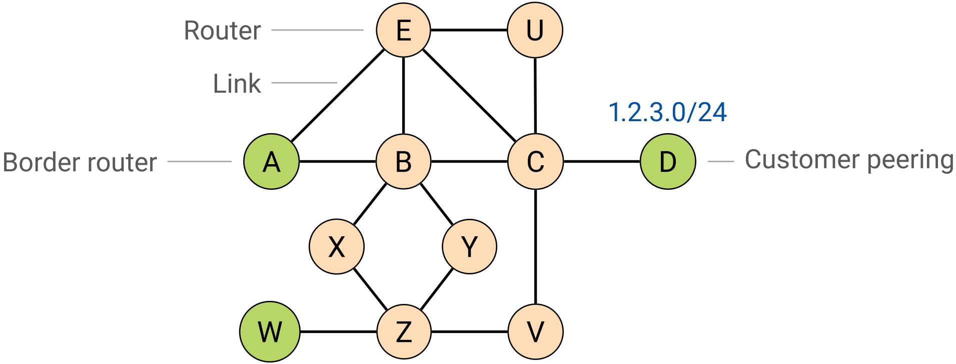
[NSDI 2018]

Blink

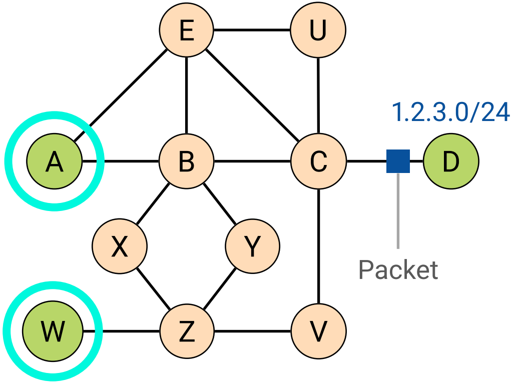
[NSDI 2019]

fine-grained
network monitoring

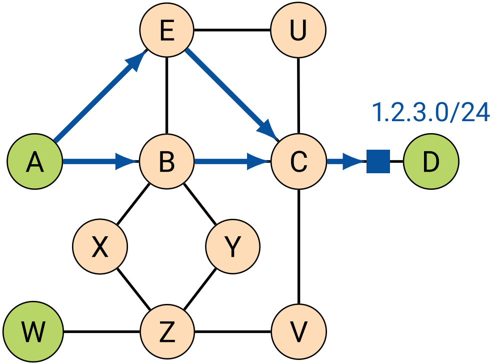
Consider this example ISP network topology



What is the ingress router for this packet arriving at router D?

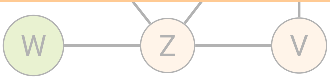


Which paths does the traffic follow?



Which paths does the traffic follow?

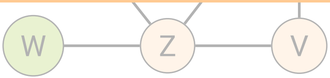
Tracking flows network-wide requires to
match measurements across multiple vantage points



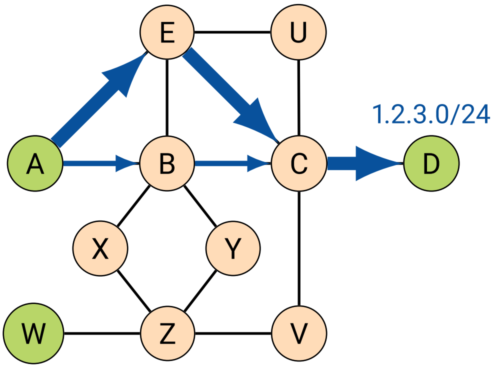
Which paths does the traffic follow?

Tracking flows network-wide requires to
match measurements across multiple vantage points

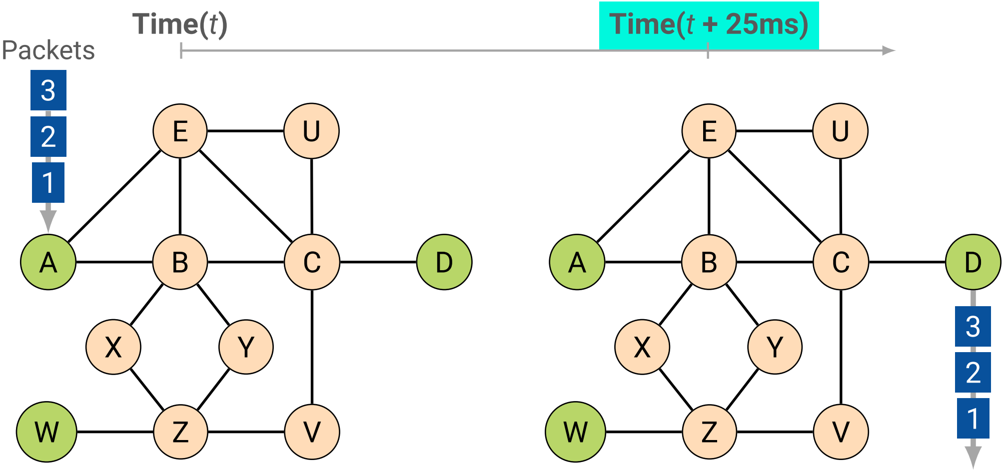
~~NetFlow, ProgME [ToN'11], FlowRadar [NSDI'16]~~



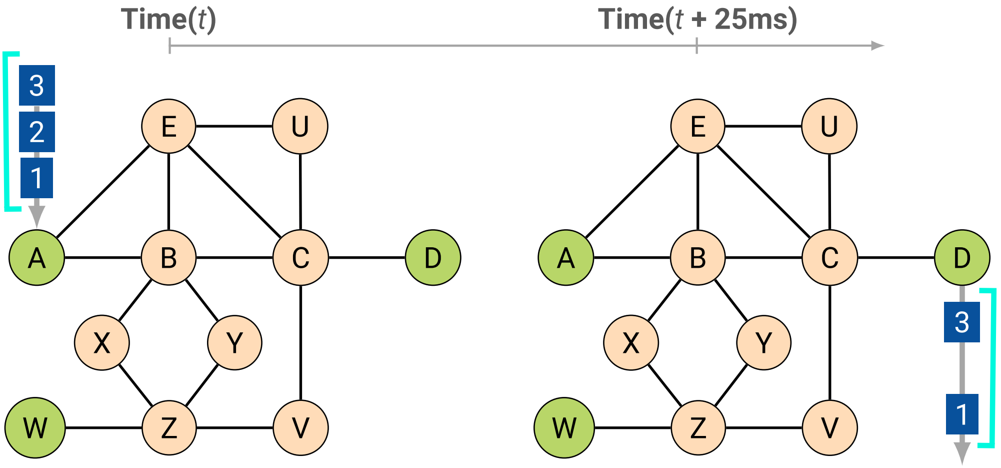
Is traffic load-balanced as expected?



Is the latency acceptable?

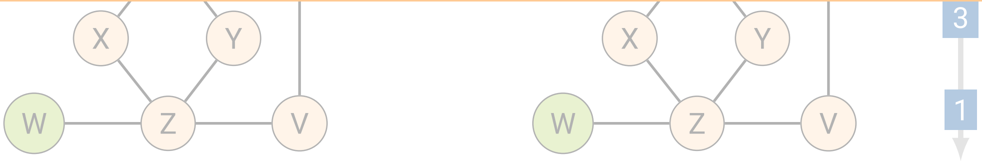


Are there losses?



Are there losses?

Fine-grained data-plane performance metrics require **packet-level visibility** over individual flows



Fined-grained network monitoring is widely researched

Gigascope [SIGMOD’03]

Planck [SIGCOMM’14]

Everflow [SIGCOMM’15]

Compiling Path Queries [NSDI’16]

Trumpet [SIGCOMM’16]

Marple [SIGCOMM’17]

Fined-grained ISP network monitoring poses unique and unmet challenges

- No control over end hosts

Gigascope [SIGMOD’03]

Planck [SIGCOMM’14]

Everflow [SIGCOMM’15]

Compiling Path Queries [NSDI’16]

~~Trumpet [SIGCOMM’16]~~

Marple [SIGCOMM’17]

Fined-grained ISP network monitoring poses unique and unmet challenges

- No control over end hosts
- Limited data-plane flexibility

Gigascope [SIGMOD’03]

Planck [SIGCOMM’14]

Everflow [SIGCOMM’15]

~~Compiling Path Queries [NSDI’16]~~

~~Trumpet [SIGCOMM’16]~~

~~Marple [SIGCOMM’17]~~

Fined-grained ISP network monitoring poses unique and unmet challenges

- No control over end hosts
- Limited data-plane flexibility
- Limited monitoring bandwidth

~~Gigascope [SIGMOD'03]~~

~~Planck [SIGCOMM'14]~~

~~Everflow [SIGCOMM'15]~~

~~Compiling Path Queries [NSDI'16]~~

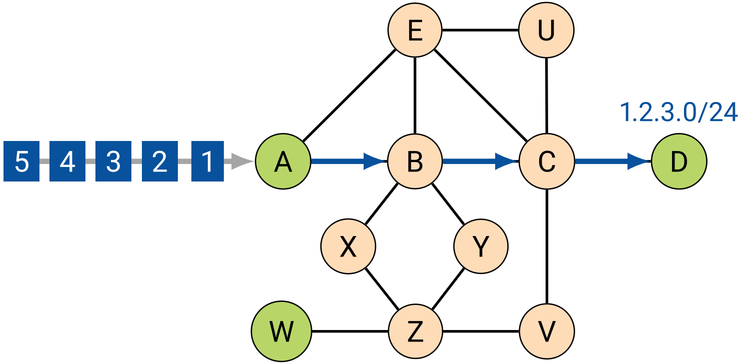
~~Trumpet [SIGCOMM'16]~~

~~Marple [SIGCOMM'17]~~

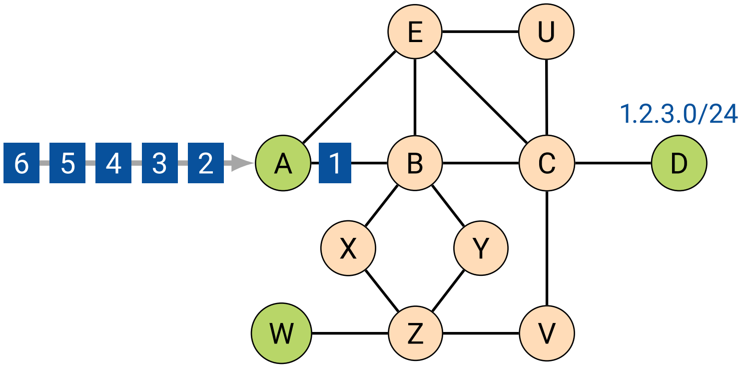
Stroboscope: Declarative Network Monitoring on a Budget

- Collecting traffic slices to monitor networks
- Adhering to a monitoring budget
- Using Stroboscope today

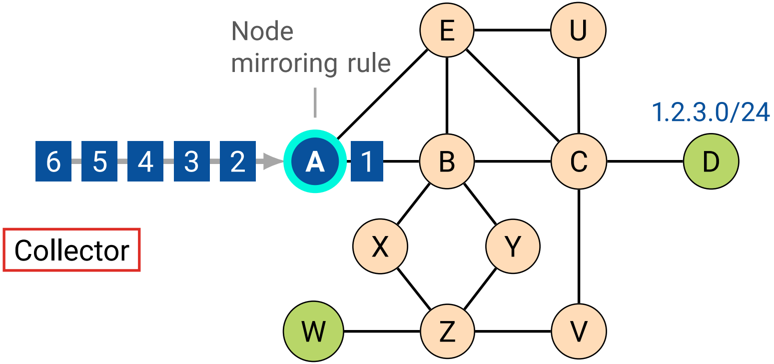
Consider the following flow of packets



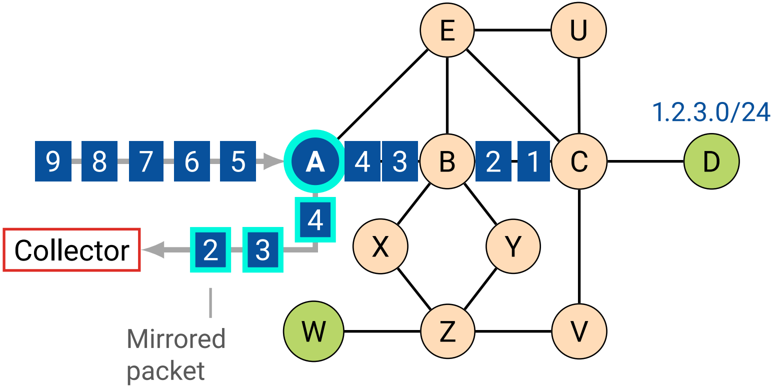
Consider the following flow of packets



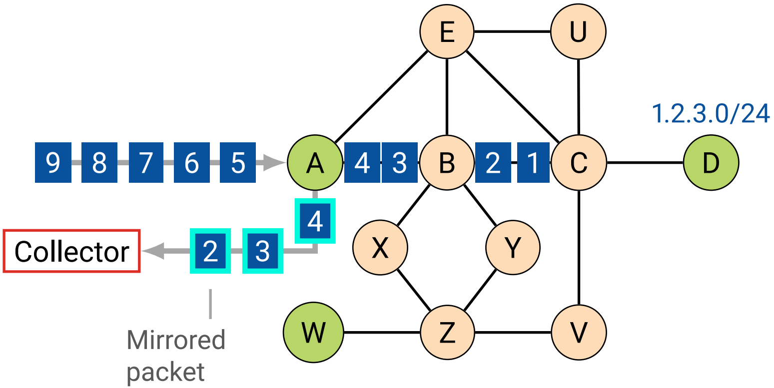
Stroboscope activates mirroring for the flow



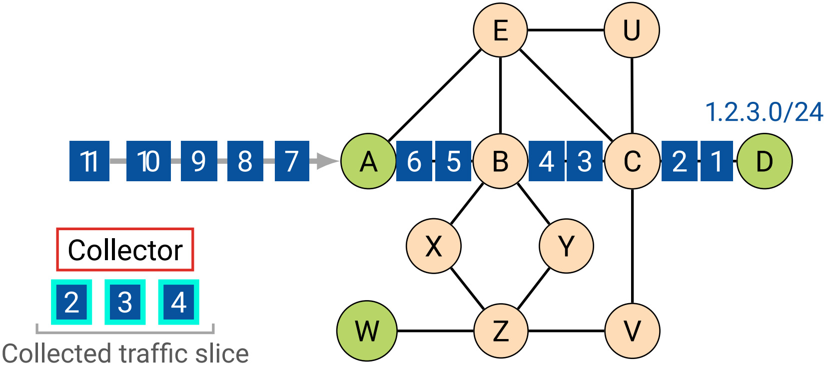
Packets are copied and encapsulated towards the collector



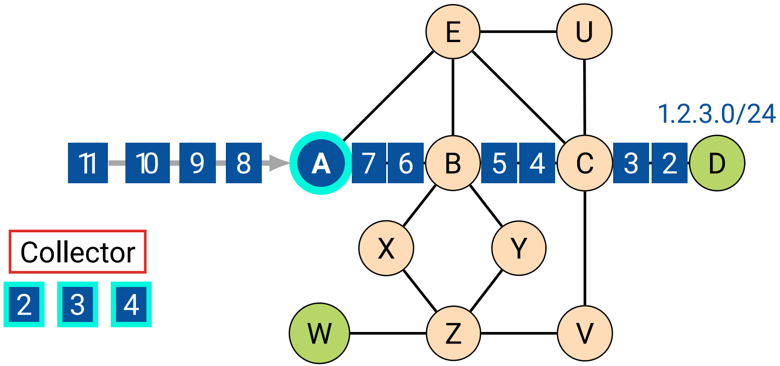
The mirroring rule is deactivated after a preset delay



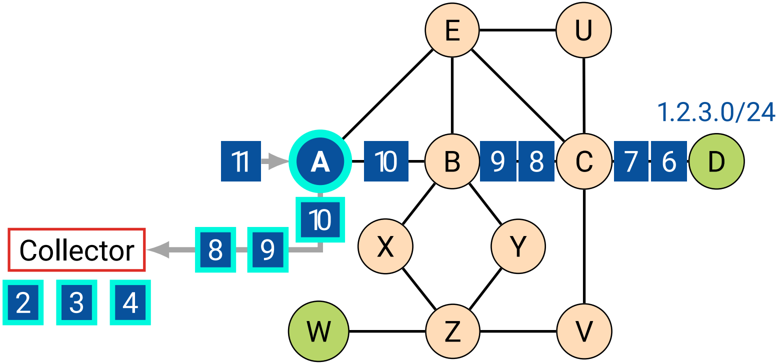
Stroboscope stores the traffic slice for analysis



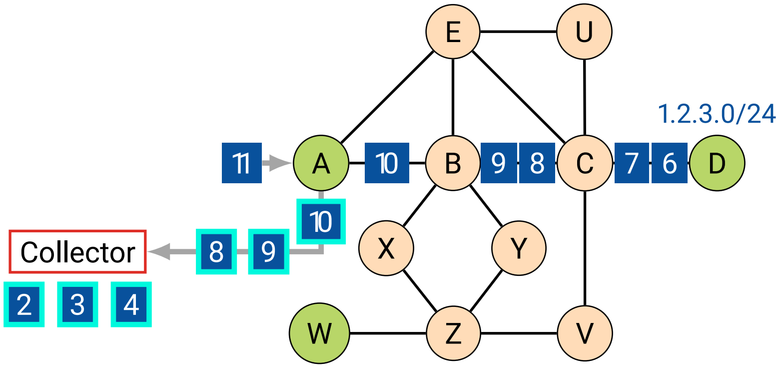
Stroboscope periodically toggles the mirroring rule



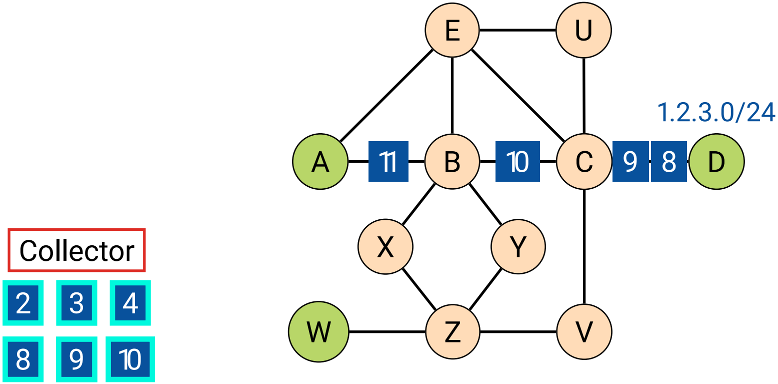
Stroboscope periodically toggles the mirroring rule



Stroboscope periodically toggles the mirroring rule



Stroboscope collects multiples traffic slices over time



Analyzing matching packets across traffic slices enables fine-grained measurements at scale

Analyzing matching packets across traffic slices enables fine-grained measurements at scale

Forwarding paths discovery, timestamp reconstruction, payload inspection, . . .

Stroboscope works with currently deployed routers

- Most vendors provide traffic mirroring and encapsulation primitives
- The collector activates mirroring for a flow by updating one ACL
- Routers autonomously deactivate mirroring rules using timers
- Traffic slices can be as small as **23 ms** on our routers (Cisco C7018)

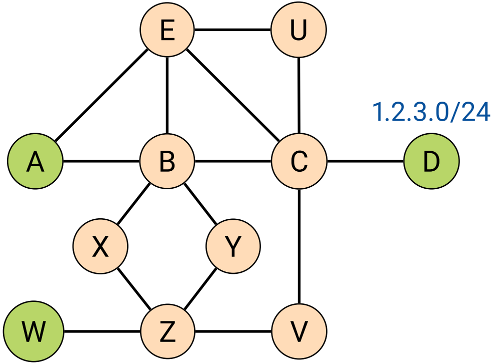
Stroboscope defines a declarative requirement language

```
MIRROR  1.2.3.0/24 ON [A B C D], [A E C D]
MIRROR  1.2.3.0/24 ON [A -> D]

CONFINE 1.2.3.0/24 ON [A B E C D]
CONFINE 1.2.3.0/24 [A -> D]

MIRROR  1.2.3.0/24 ON [ -> D]
CONFINE 1.2.3.0/24 ON [ -> D]

USING 15 Mbps DURING 500 ms EVERY 5 s
```



Stroboscope defines two types of queries

MIRROR

CONFINE

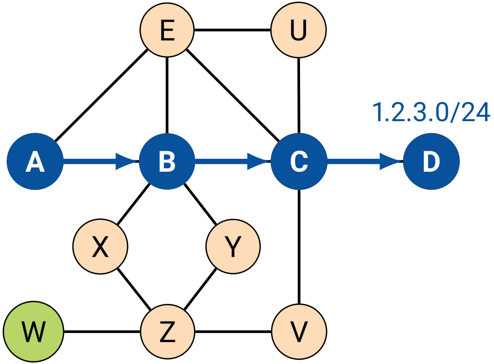
Stroboscope defines two types of queries

MIRROR

CONFINE

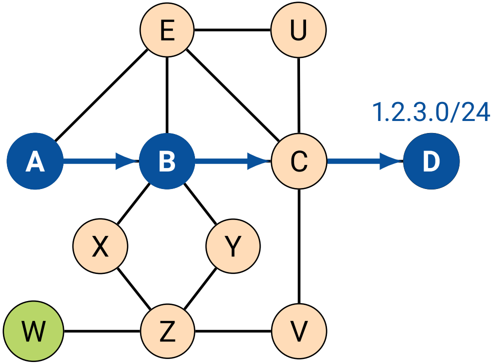
MIRROR queries reconstruct the path taken by packets

`MIRROR` 1.2.3.0/24 `ON` [A B C D]



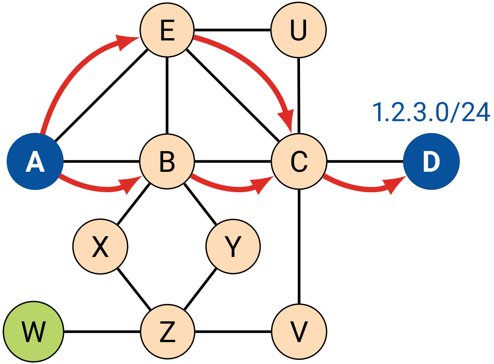
Fewer mirroring rules reduces bandwidth usage

`MIRROR` 1.2.3.0/24 `ON` [A B C D]



Too few mirroring rules creates ambiguity

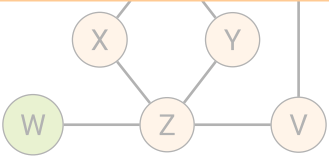
MIRROR 1.2.3.0/24 ON [A B C D]



Too few mirroring rules creates ambiguity

MIRROR 1.2.3.0/24 ON [A B C D]

The **Key-Points Sampling** algorithm minimizes mirroring rules and guarantees non-ambiguous reconstructed paths

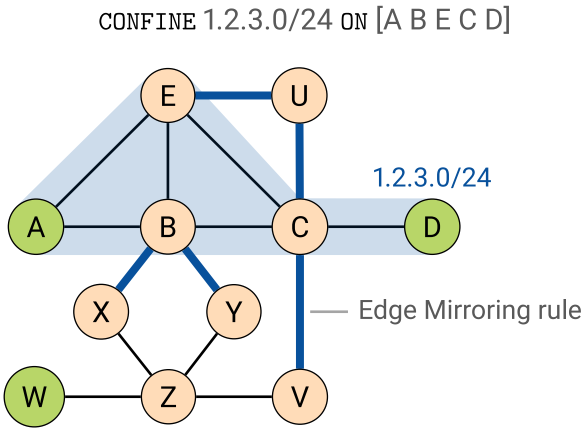


Stroboscope defines two types of queries

MIRROR

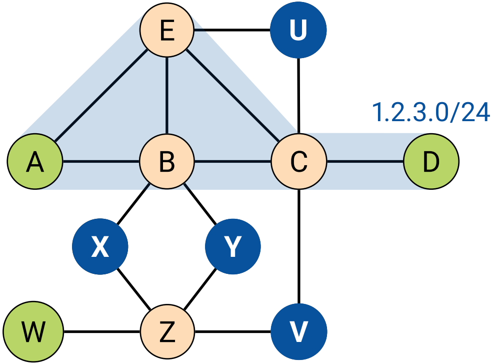
CONFINE

CONFINE queries mirror packets leaving a confinement region



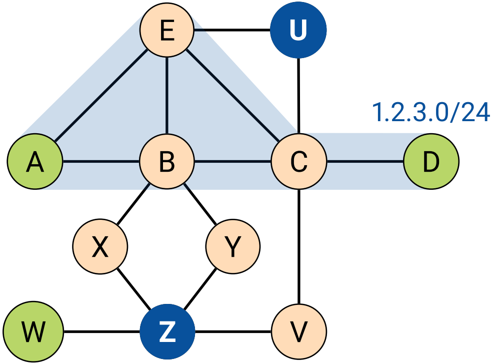
Fewer mirroring rules minimizes control-plane overhead

`CONFINE` 1.2.3.0/24 `ON` [A B E C D]



The lower bound is a multi-terminal node cut

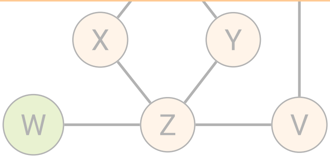
`CONFINE` 1.2.3.0/24 `ON` [A B E C D]



The lower bound is a multi-terminal node cut

CONFINE 1.2.3.0/24 ON [A B E C D]

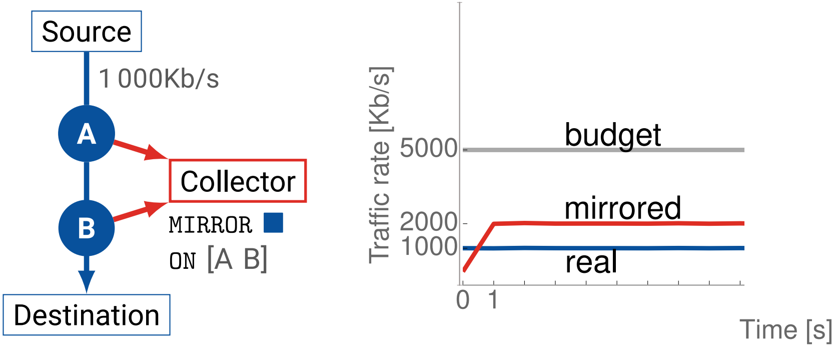
The **Surrounding** algorithm minimizes mirroring rules and guarantees to mirror any packet leaving the confinement region



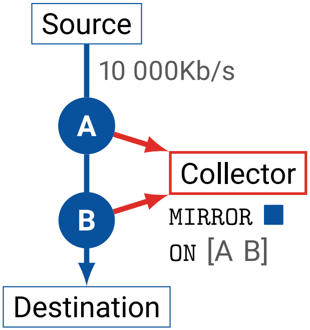
Stroboscope: Declarative Network Monitoring on a Budget

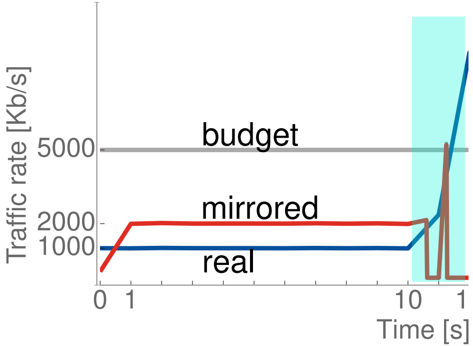
- Collecting traffic slices to monitor networks
- Adhering to a monitoring budget
- Using Stroboscope today

Stroboscope tracks the rate of mirrored traffic in real time

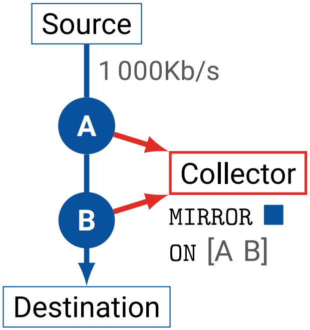


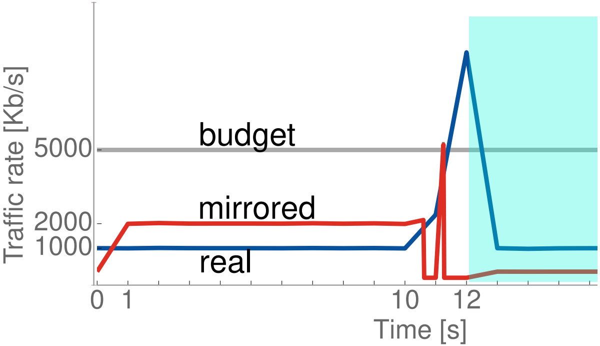
Measurement campaigns are stopped early if the estimated demand are exceeded



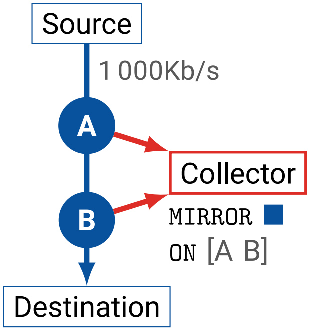


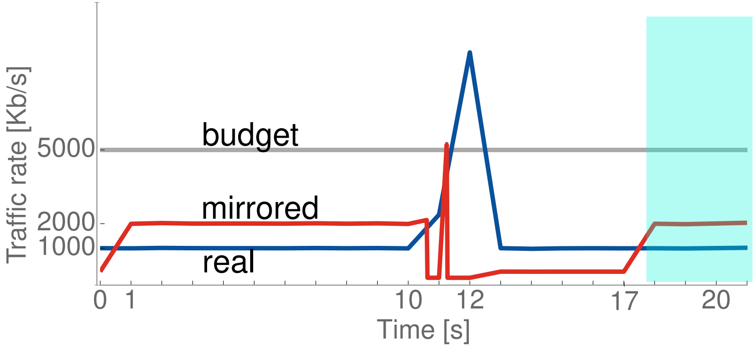
Exceeding the total budget schedules the query once per measurement campaign



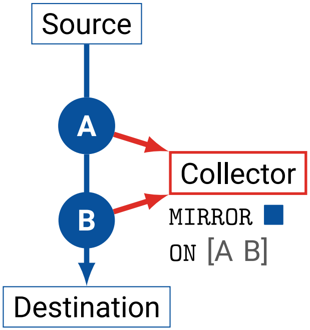


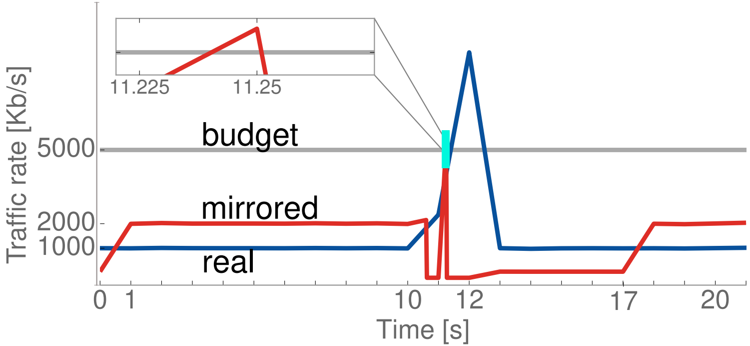
Stable recorded traffic rates are used for future estimations





Stroboscope exceeds the monitoring budget for at most one timeslot





Stroboscope: Declarative Network Monitoring on a Budget

- Traffic slicing as a first-class data-plane primitive
- Strong guarantees on budget compliance and measurement accuracy
- Measurement analysis decoupled from measurement collection

Programmable network monitoring

and what to do with it…

Stroboscope

[NSDI 2018]

Blink

[NSDI 2019]

data-driven
fast rerouting

Upon local failures, connectivity can be quickly restored

Upon local failures, connectivity can be quickly restored

Fast failure detection
using *e.g.*, hardware-generated signals

Fast traffic rerouting
using *e.g.*, Prefix Independent Convergence
or MPLS Fast Reroute

Upon remote failures, the only way to restore connectivity is to wait for the Internet to converge

Upon remote failures, the only way to restore connectivity is to wait for the Internet to converge

… and the Internet converges *very* slowly*

*Holterbach et al. SWIFT: Predictive Fast Reroute ACM SIGCOMM, 2017

Fire at AT&T facility causes widespread outage in North Texas

TELECOM

Nationwide internet outage affects CenturyLink customers

Time Warner Cable comes back from nationwide Internet outage

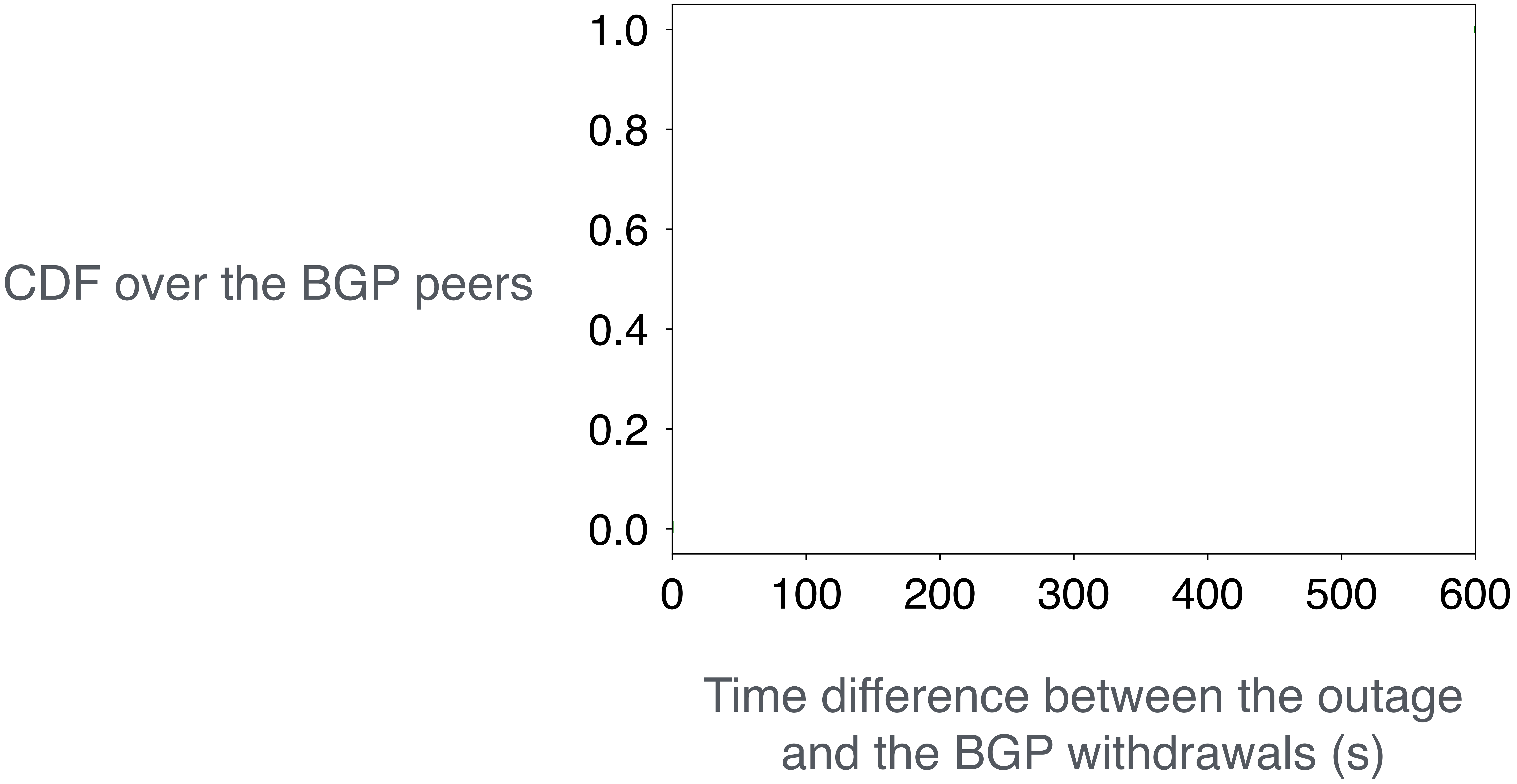
by Brian Stelter @brianstelter

August 27, 2014: 11:07 PM ET

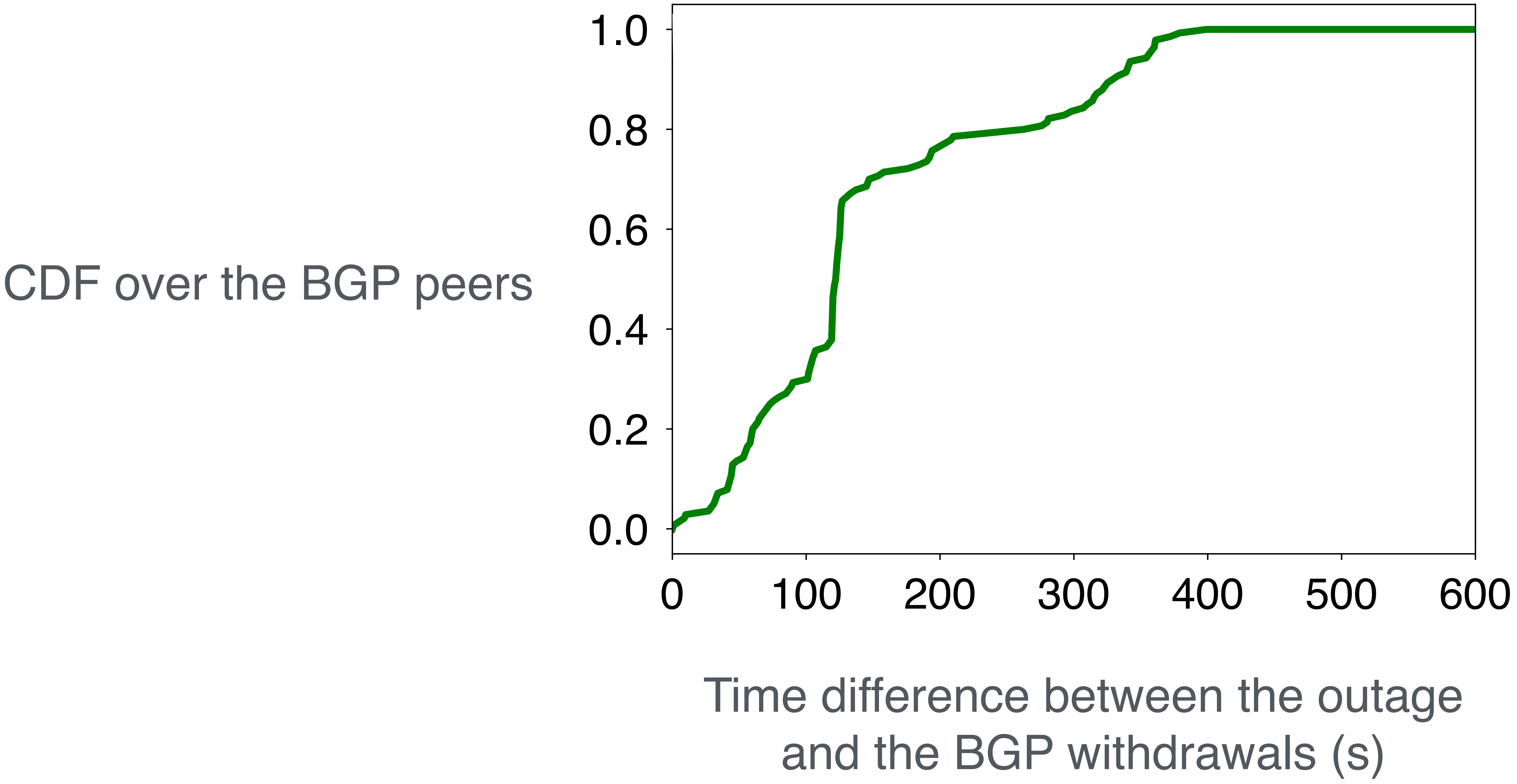
Major internet outage hits the U.S. - Affecting customers of Comcast, Verizon, and AT&T

November 6, 2017 | Emerging Threats

BGP took minutes to converge upon the Time Warner Cable outage in 2014



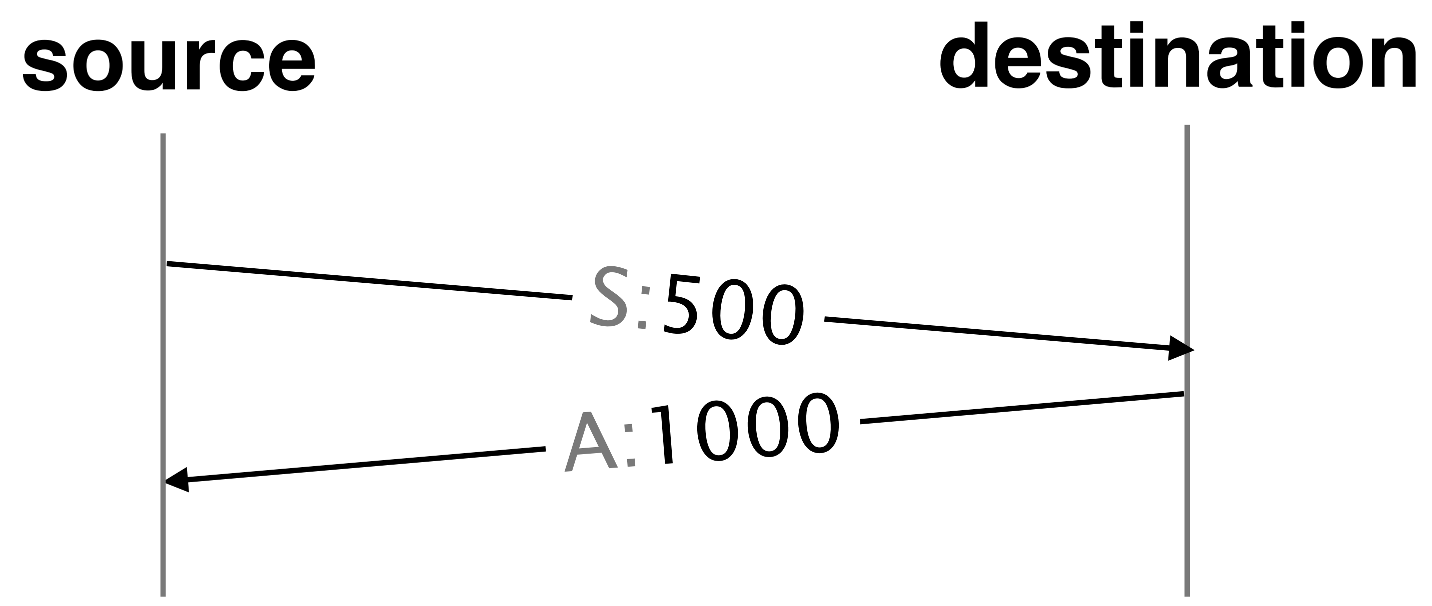
BGP took minutes to converge upon the Time Warner Cable outage in 2014



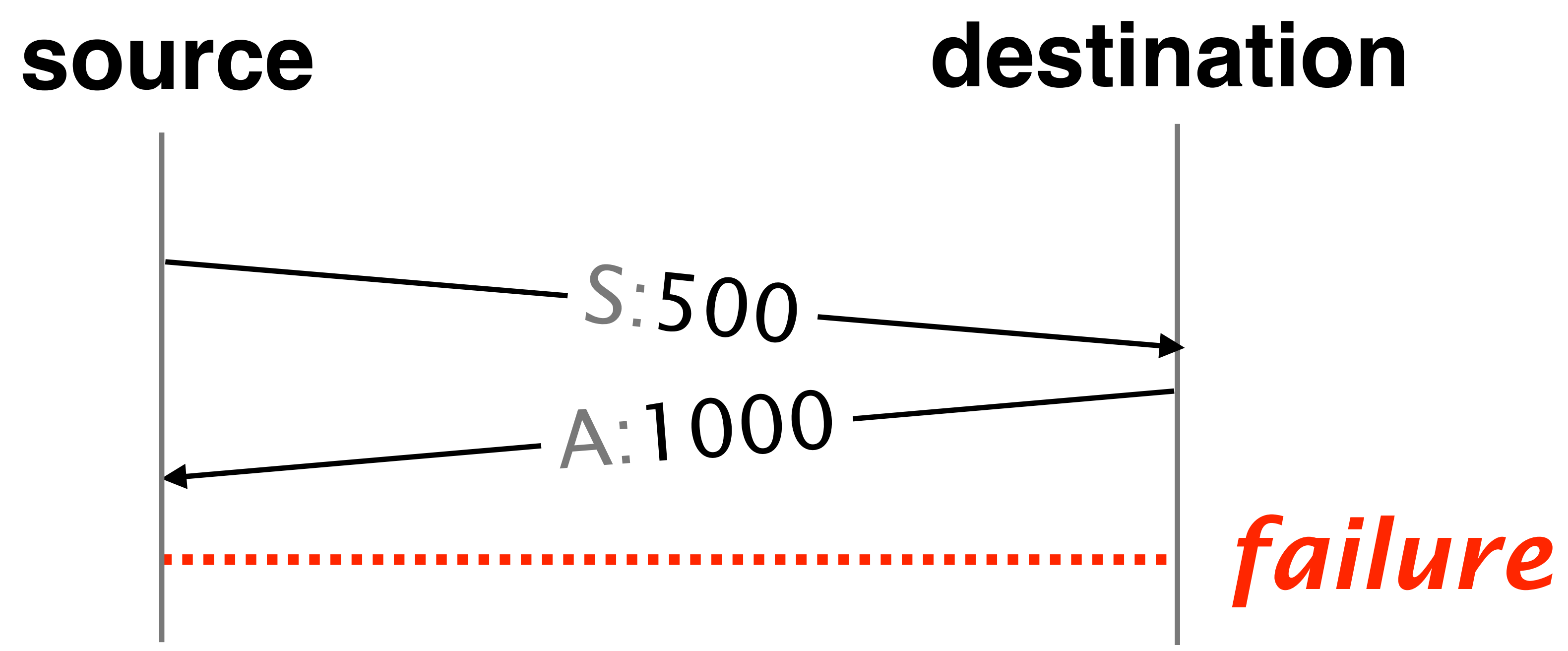
What about using **data plane signals** for fast rerouting?

TCP flows exhibit the same behavior upon failures

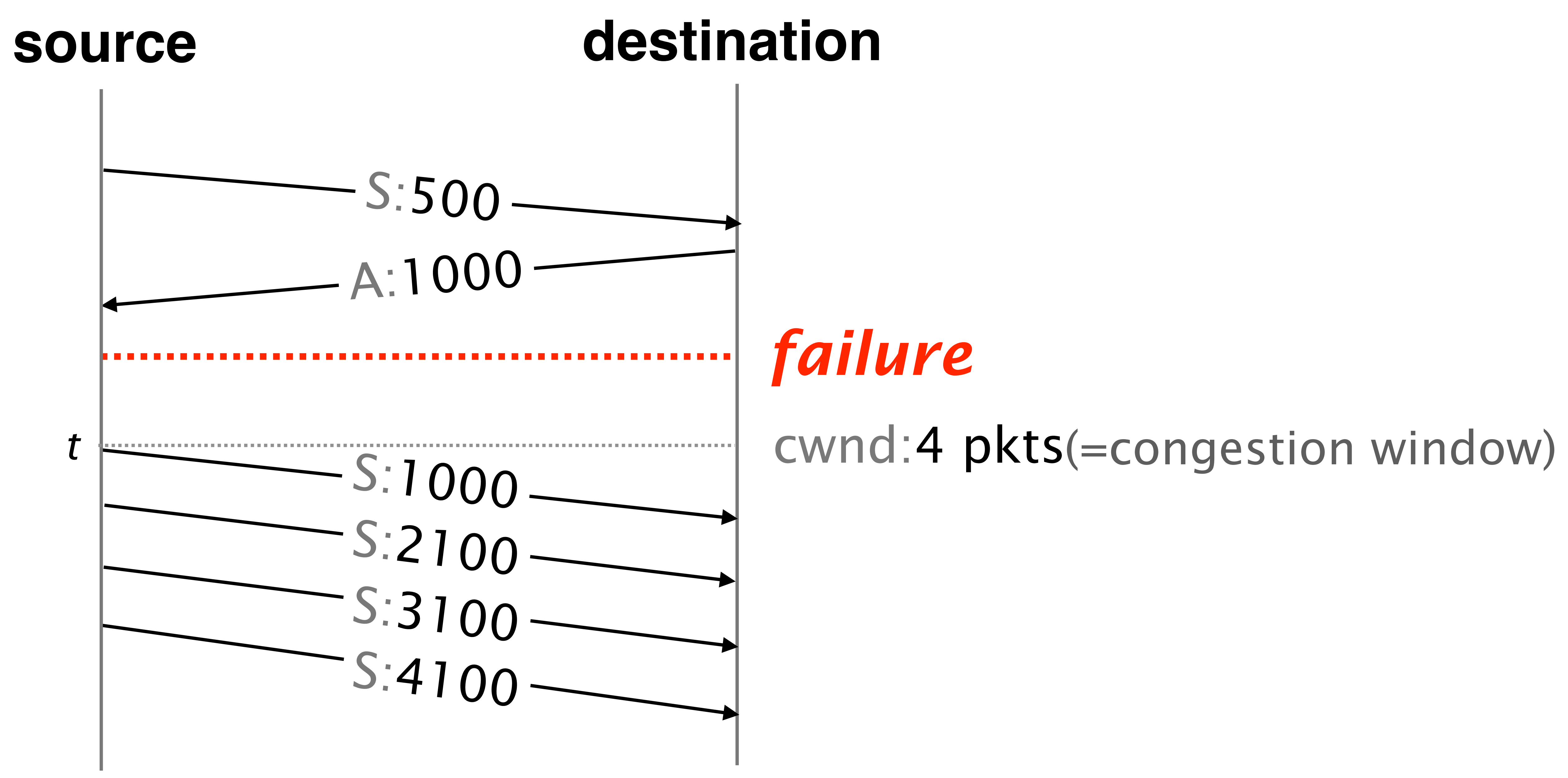
TCP flows exhibit the same behavior upon failures



TCP flows exhibit the same behavior upon failures



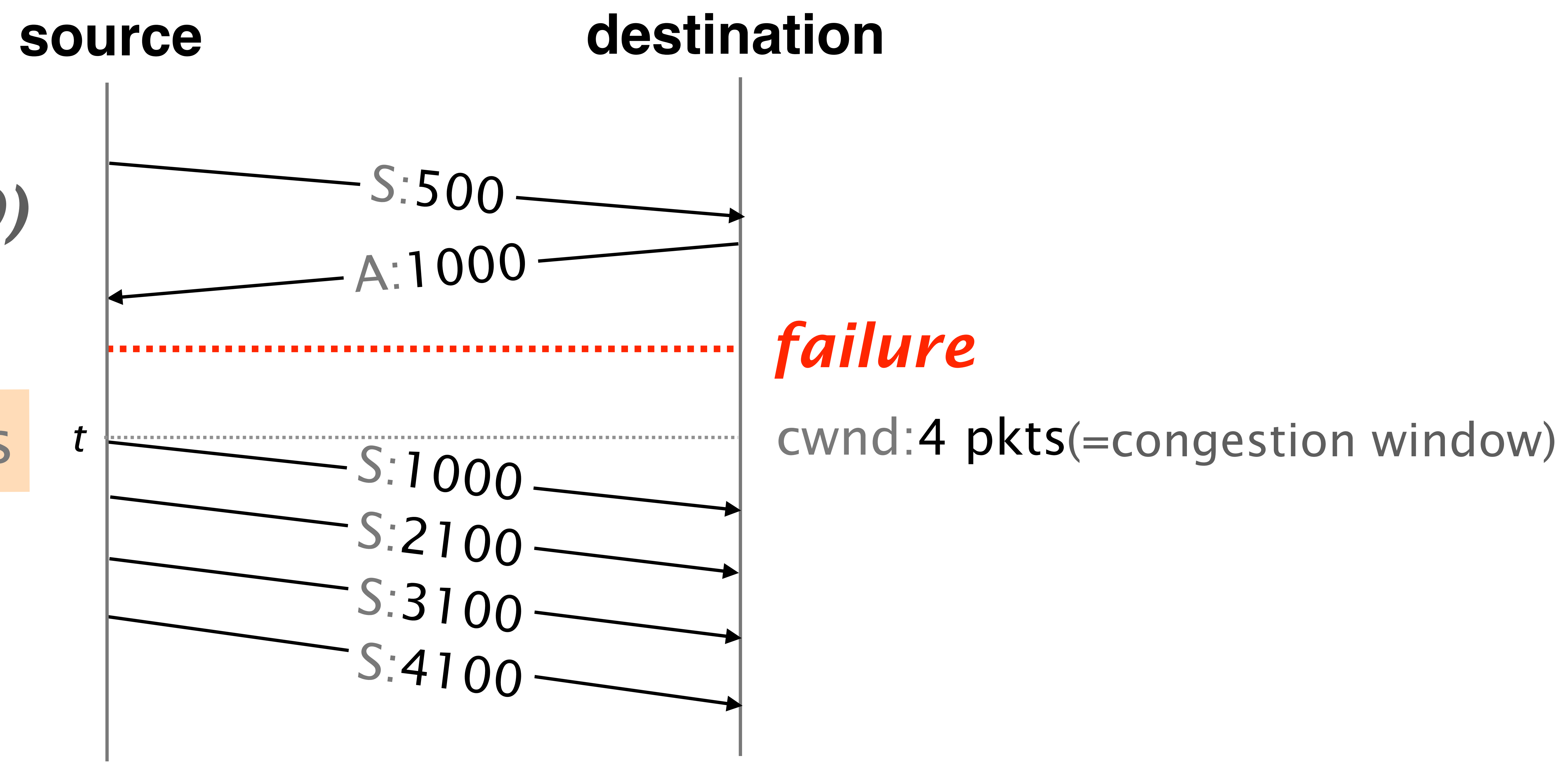
TCP flows exhibit the same behavior upon failures



TCP flows exhibit the same behavior upon failures

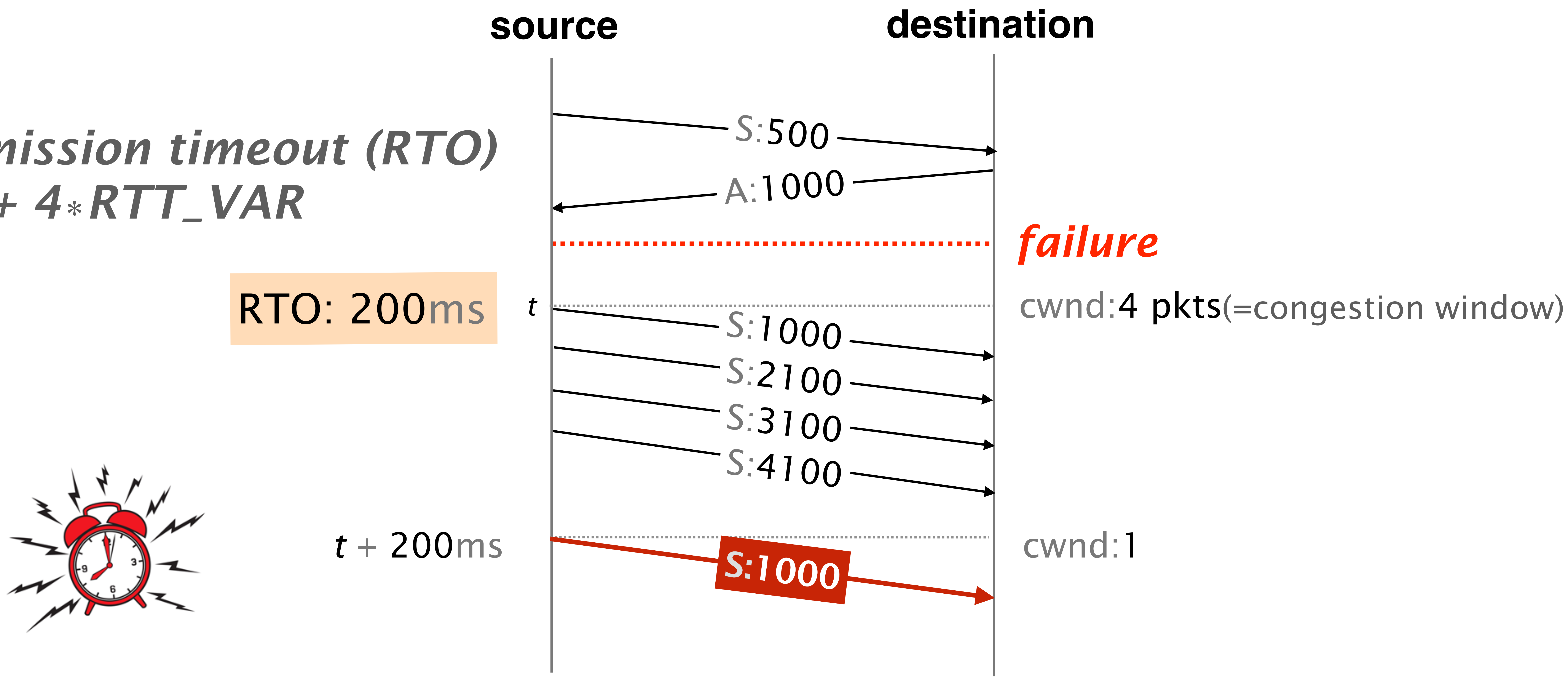
Retransmission timeout (RTO)
= SRTT + 4∗RTT_VAR

RTO: 200ms



TCP flows exhibit the same behavior upon failures

Retransmission timeout (RTO)
= SRTT + 4∗RTT_VAR



TCP flows exhibit the same behavior upon failures

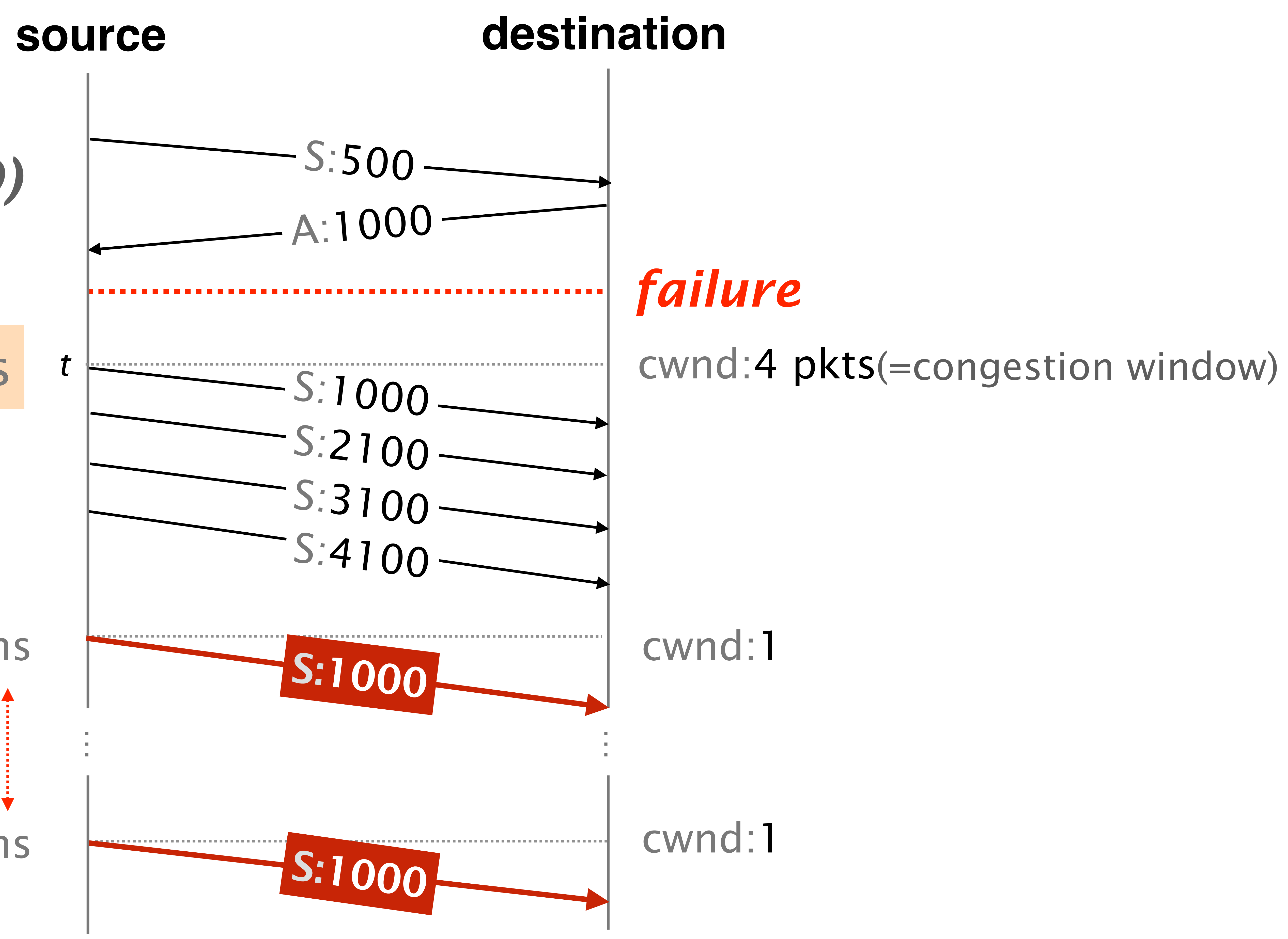
Retransmission timeout (RTO)
= SRTT + 4∗RTT_VAR

RTO: 200ms

t + 200ms

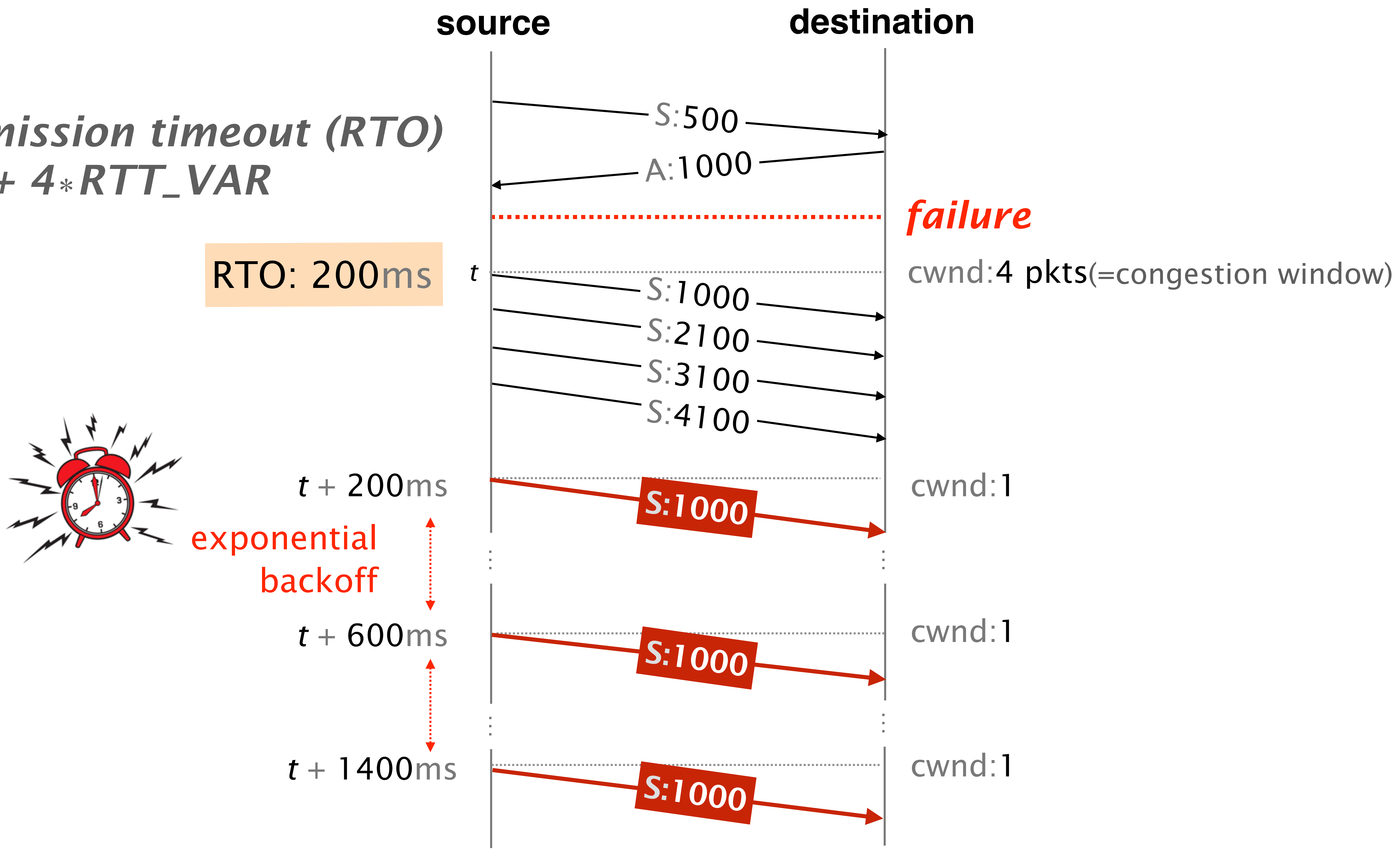
exponential backoff

t + 600ms



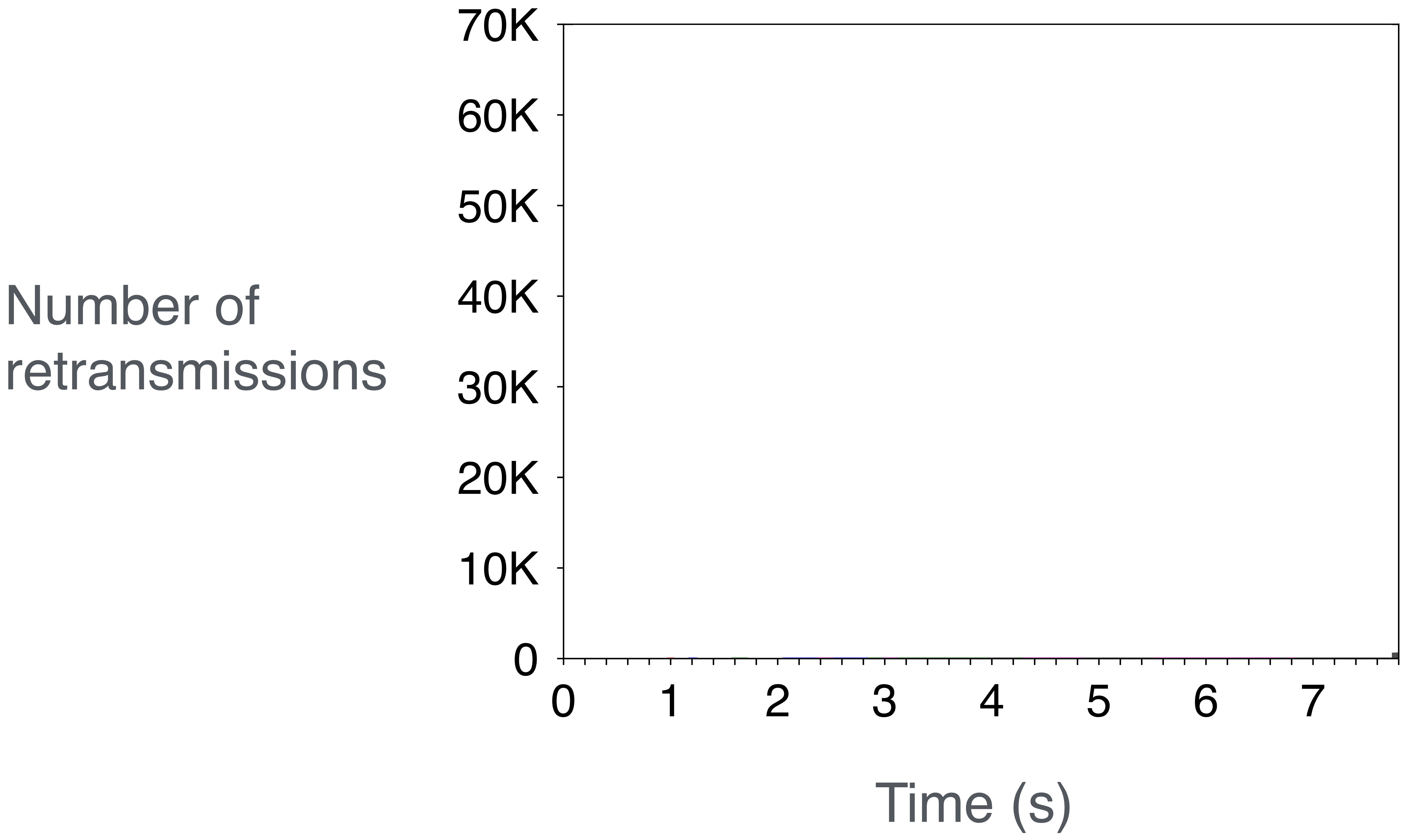
TCP flows exhibit the same behavior upon failures

Retransmission timeout (RTO)
= SRTT + 4∗RTT_VAR

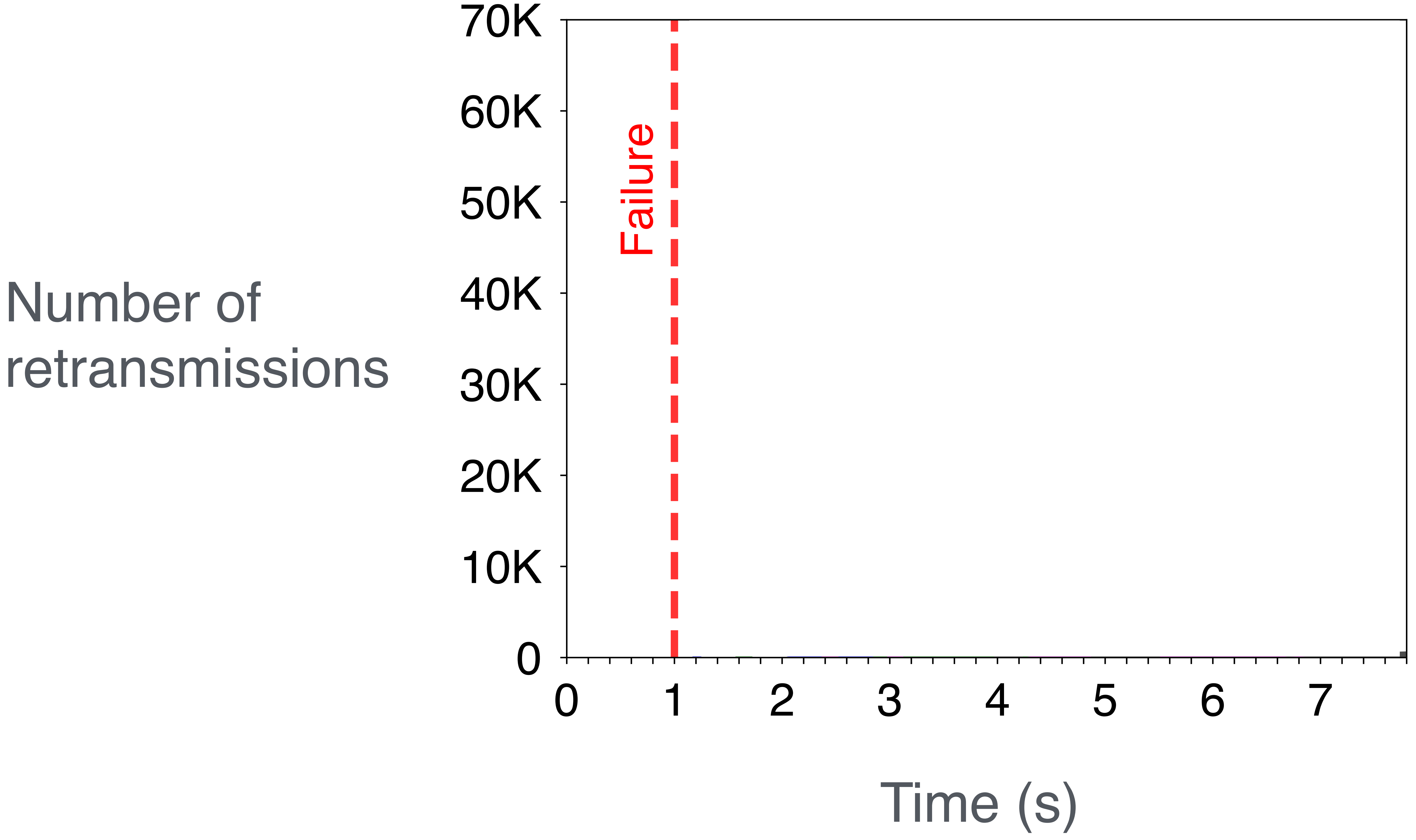


When multiple flows experience the same failure
the signal is a wave of retransmissions

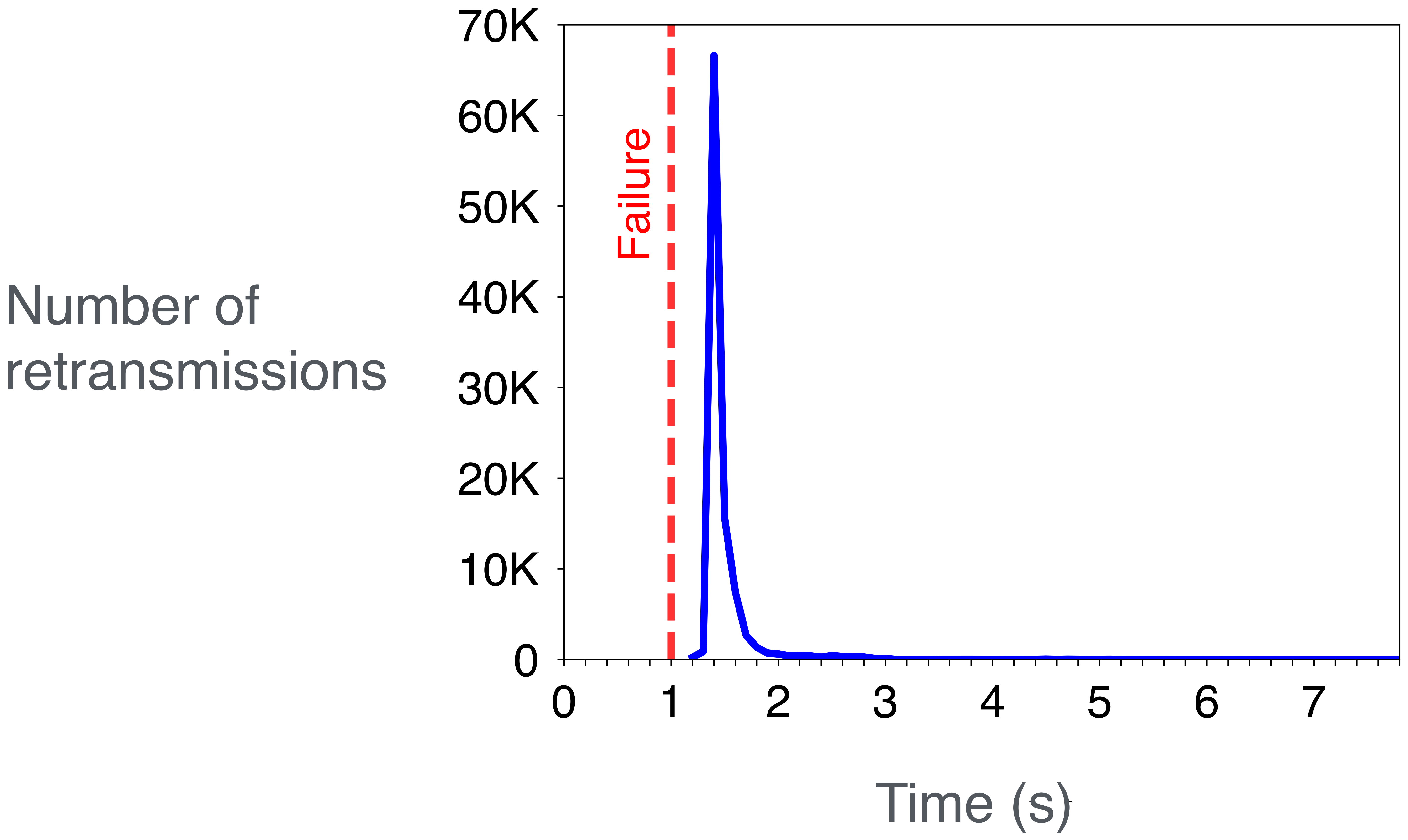
When multiple flows experience the same failure the signal is a wave of retransmissions



When multiple flows experience the same failure the signal is a wave of retransmissions

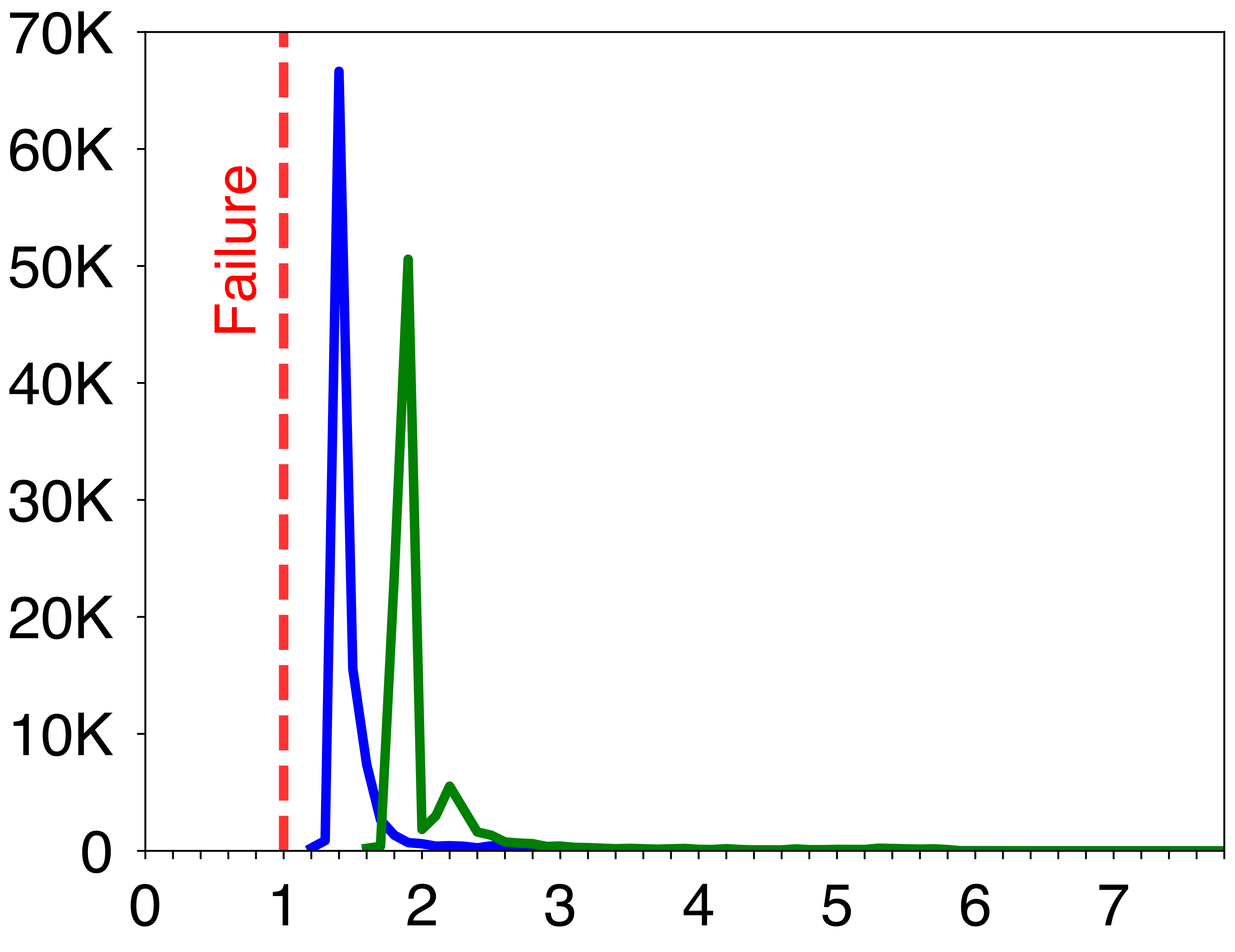


When multiple flows experience the same failure the signal is a wave of retransmissions



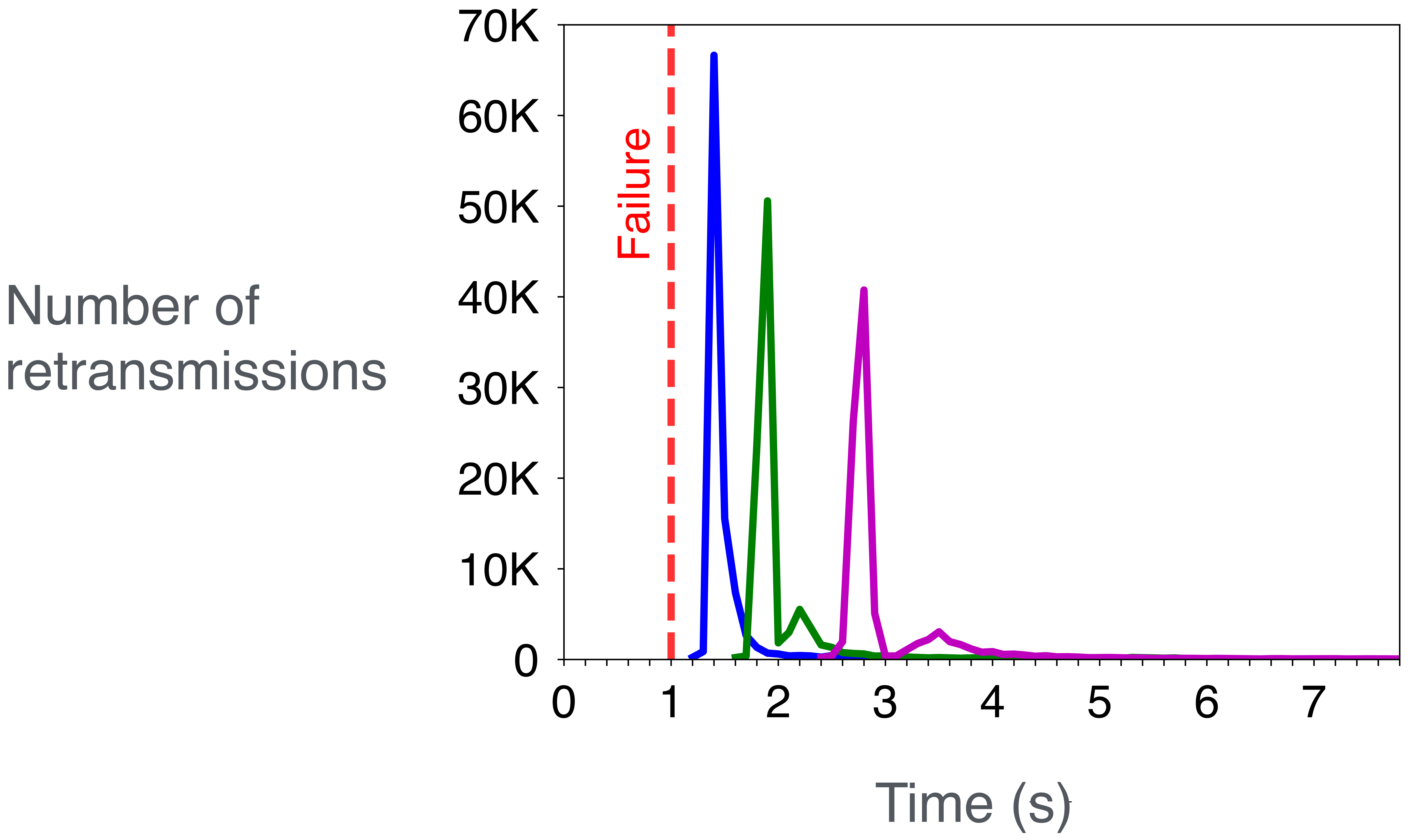
When multiple flows experience the same failure the signal is a wave of retransmissions

Number of retransmissions

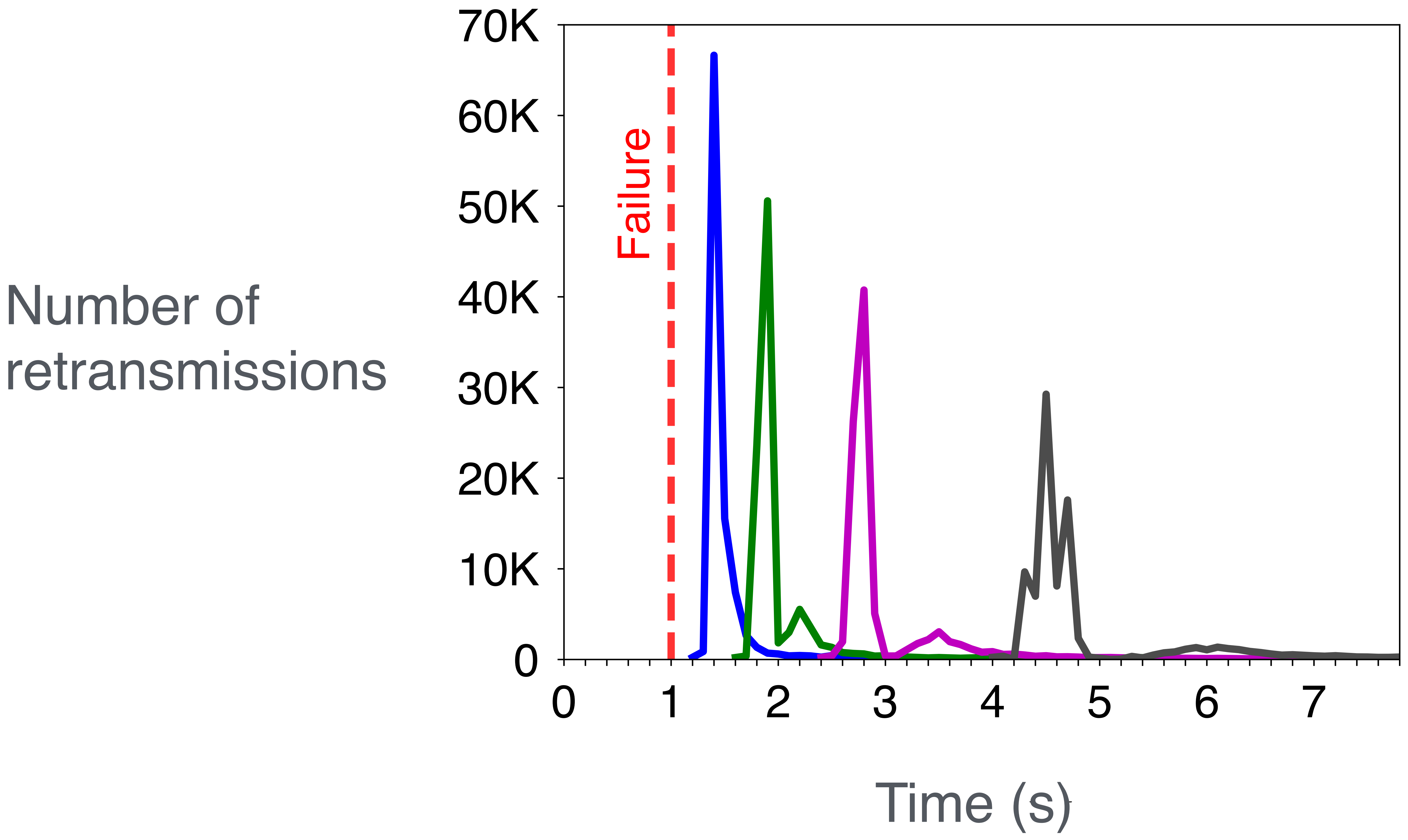


Time (s)

When multiple flows experience the same failure
the signal is a wave of retransmissions



When multiple flows experience the same failure
the signal is a wave of retransmissions



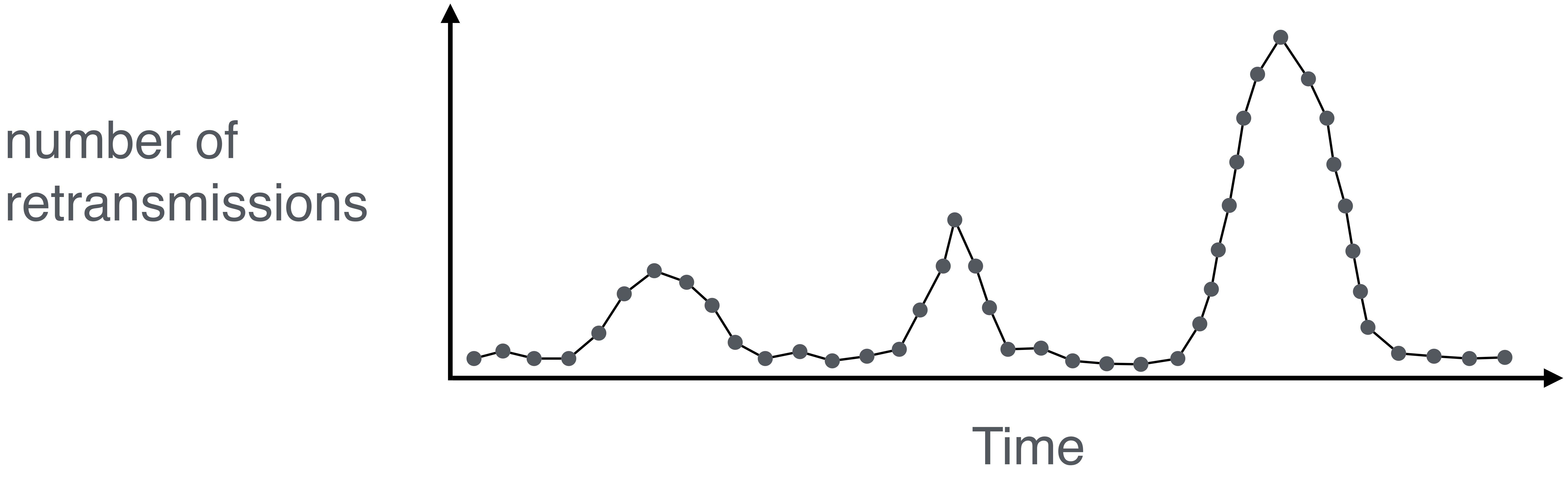
To detect failures, ***Blink*** looks at TCP retransmissions

To detect failures, ***Blink*** looks at TCP retransmissions

Problem: TCP retransmissions can be unrelated to a failure (*i.e.,* noise)

To detect failures, ***Blink*** looks at TCP retransmissions

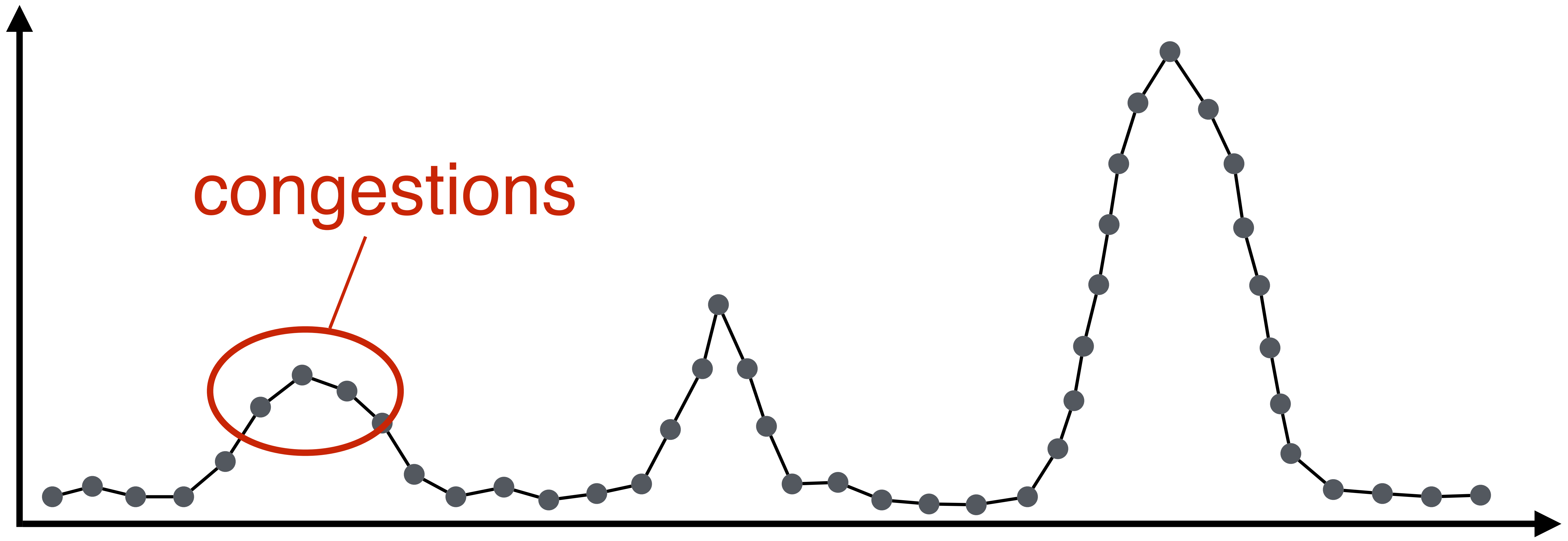
Problem: TCP retransmissions can be unrelated to a failure (*i.e.,* noise)



To detect failures, ***Blink*** looks at TCP retransmissions

Problem: TCP retransmissions can be unrelated to a failure (*i.e.*, noise)

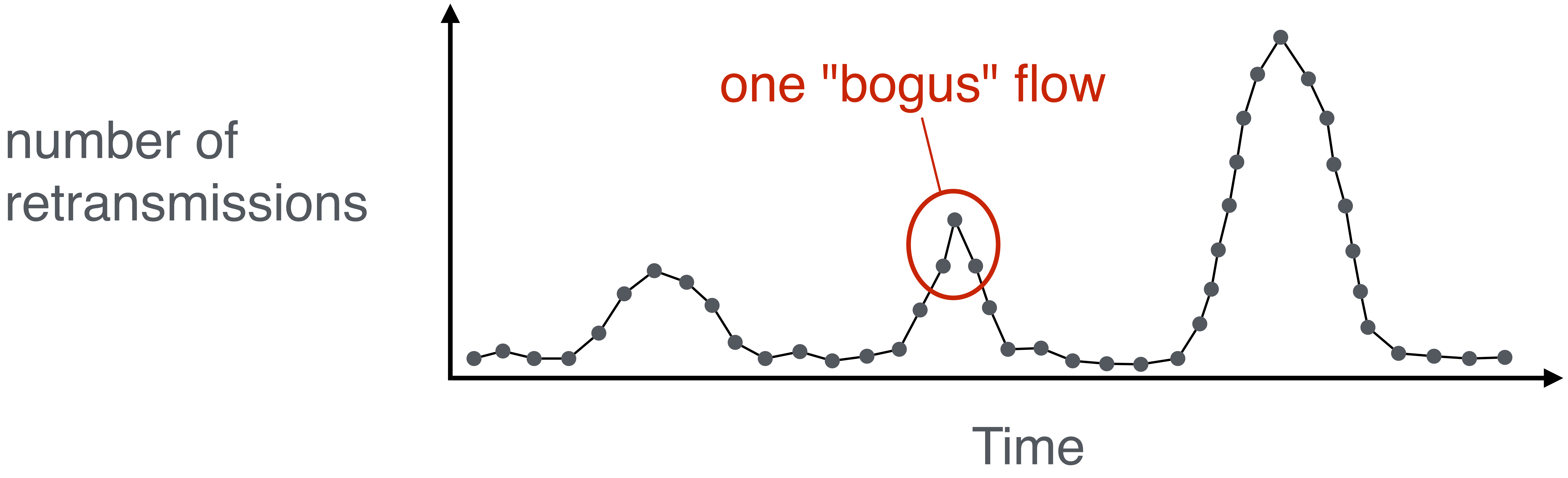
number of retransmissions



Time

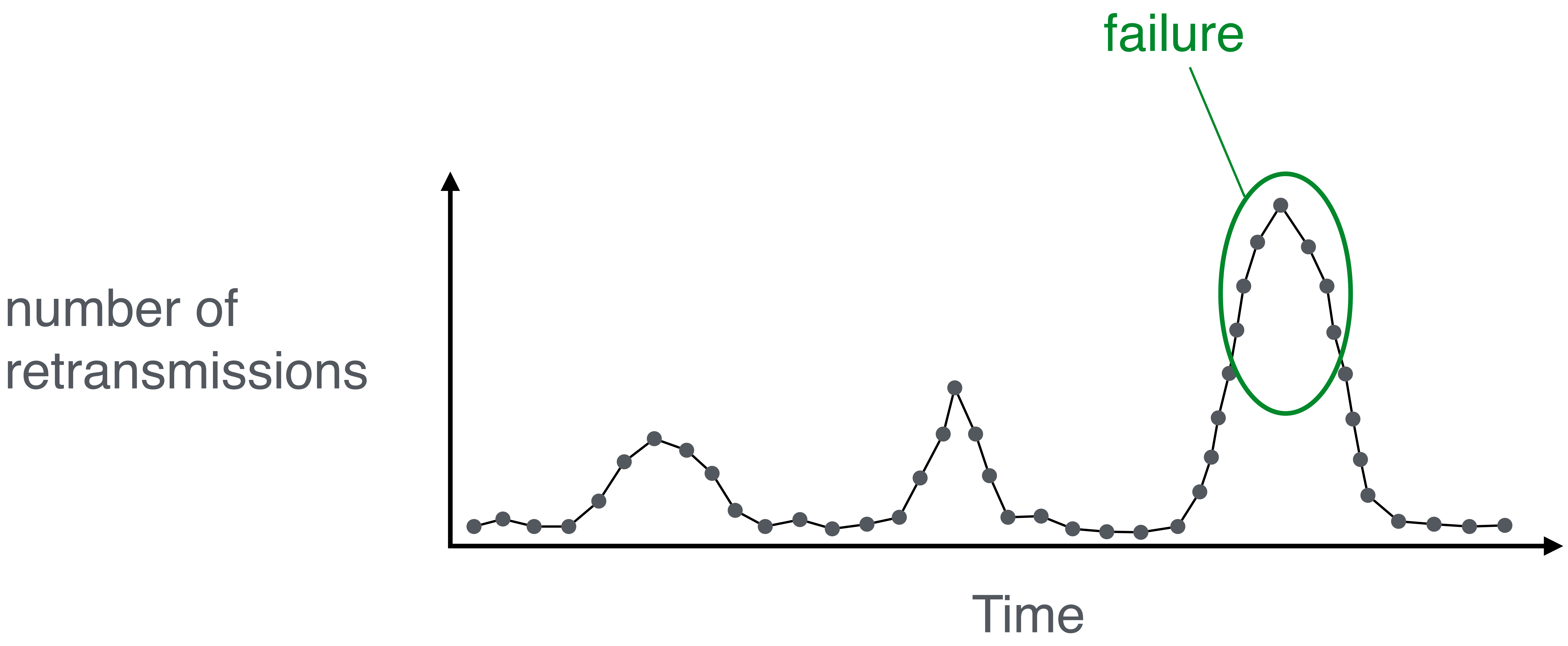
To detect failures, ***Blink*** looks at TCP retransmissions

Problem: TCP retransmissions can be unrelated to a failure (*i.e.*, noise)



To detect failures, ***Blink*** looks at TCP retransmissions

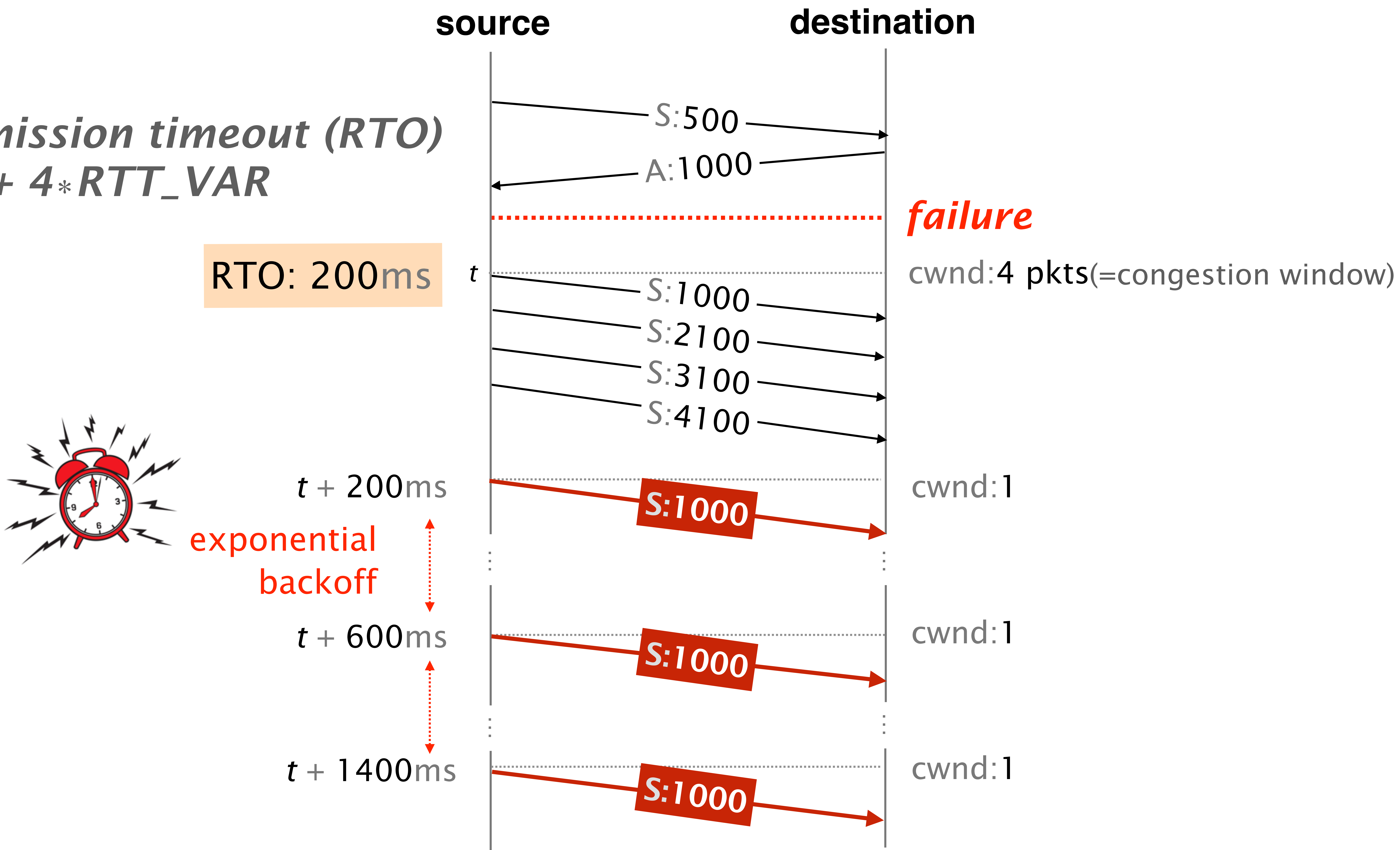
Problem: TCP retransmissions can be unrelated to a failure (*i.e.*, noise)



Solution #1: ***Blink*** looks at consecutive packets with the same sequence number

Solution #1: ***Blink*** looks at consecutive packets with the same sequence number

Retransmission timeout (RTO)
= SRTT + 4∗RTT_VAR



Solution #2: ***Blink*** monitors the number of flows experiencing retransmissions over time using a sliding window

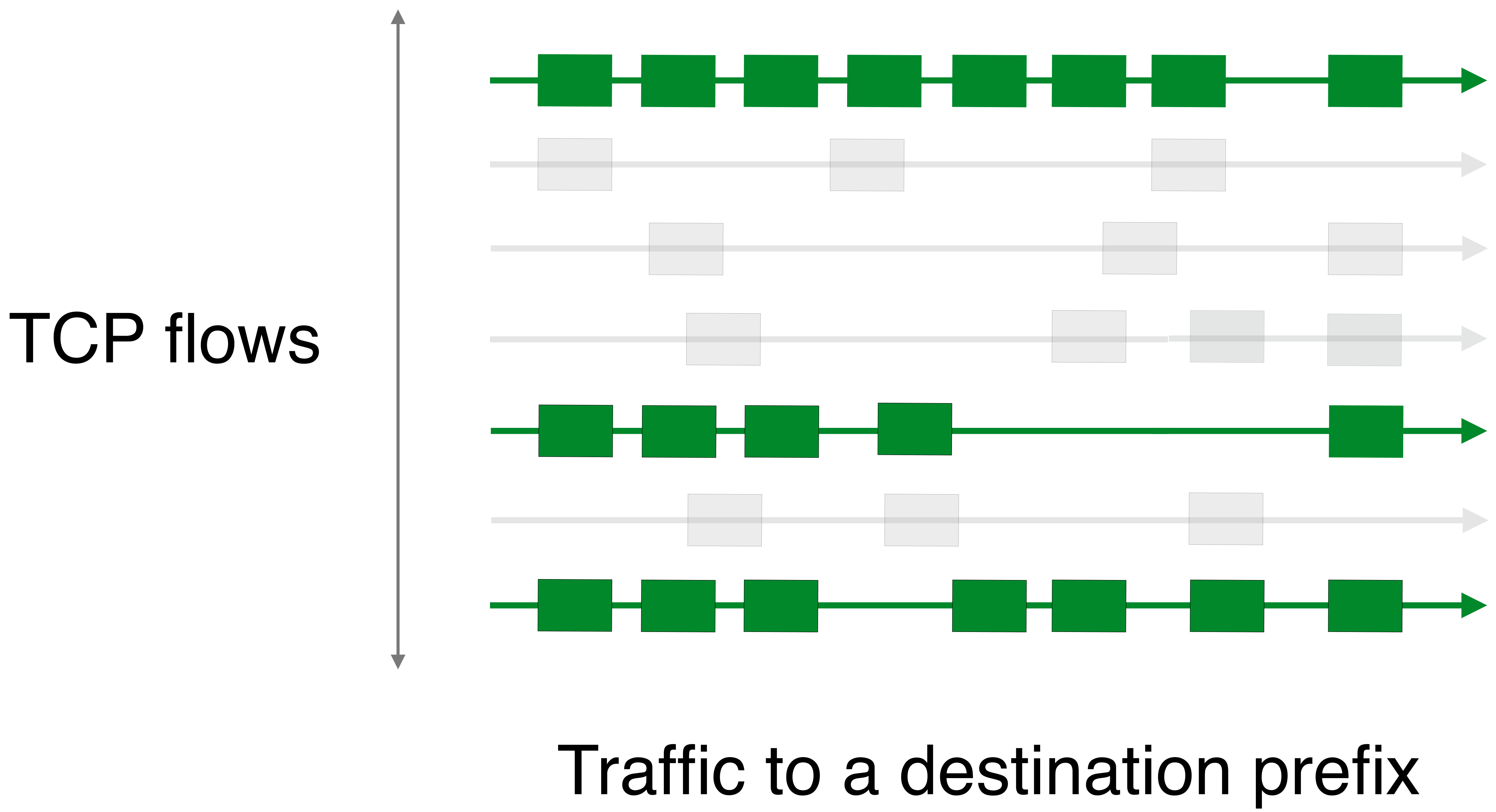
Blink is intended to run in programmable switches

Blink is intended to run in programmable switches

Problem: those switches have very limited resources

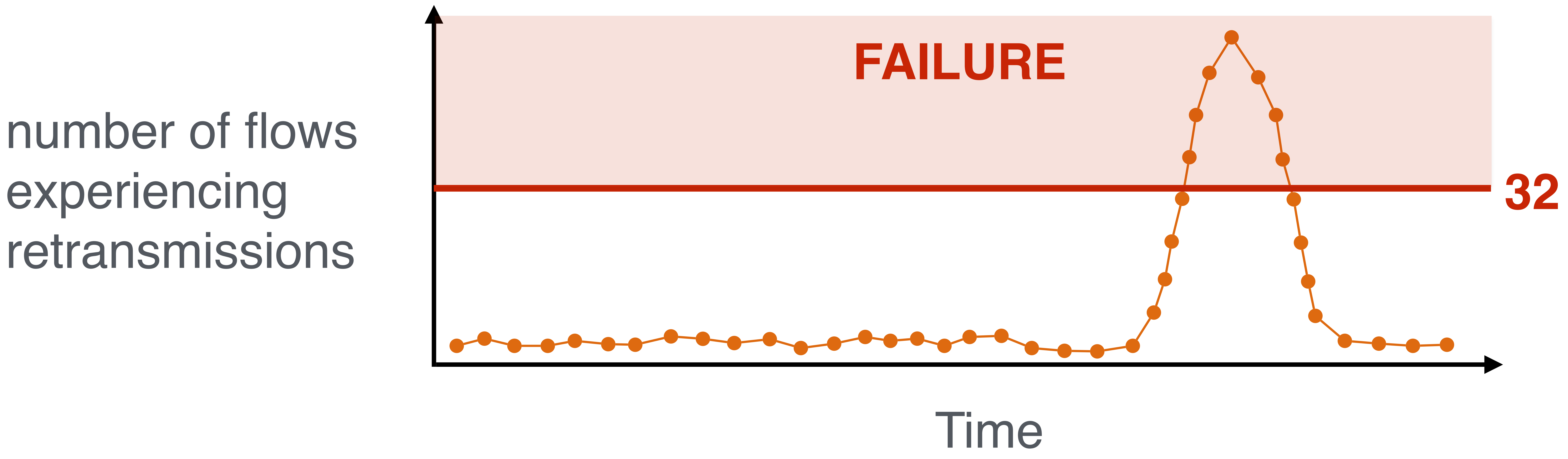
Solution #1: ***Blink*** focuses on the popular prefixes, *i.e.*, the ones that attract data traffic

Solution #2: ***Blink*** monitors a sample of the **active** flows for each monitored prefix



default 64 flows monitored

Blink infers a failure for a prefix when the majority of the monitored flows experience retransmissions



We evaluated ***Blink*** failure inference using synthetic traces

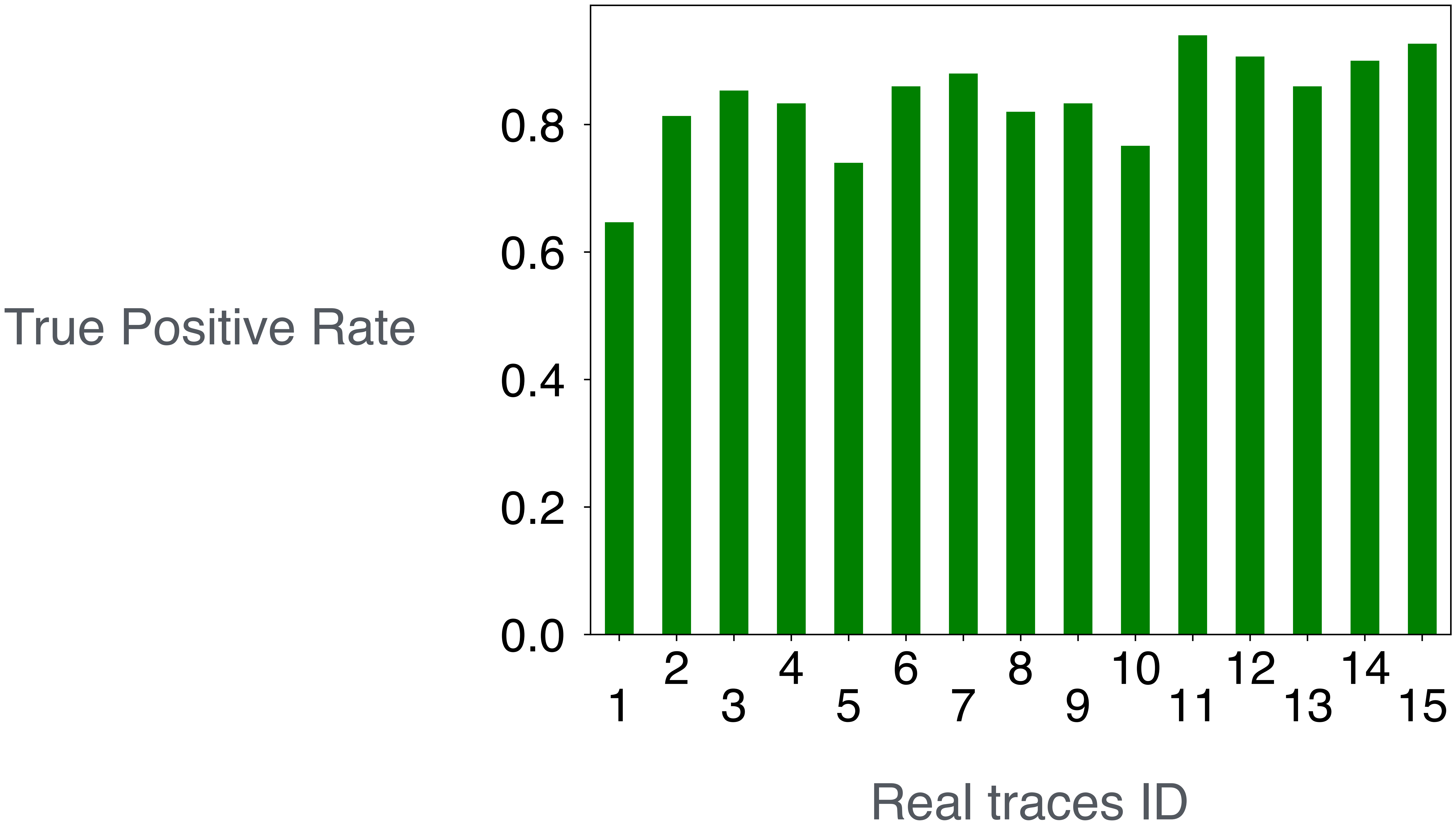
following the traffic characteristics extracted from the real traces

We are interested in:

Accuracy: True Positive Rate vs False Positive Rate

Speed: How long does Blink take to infer failures

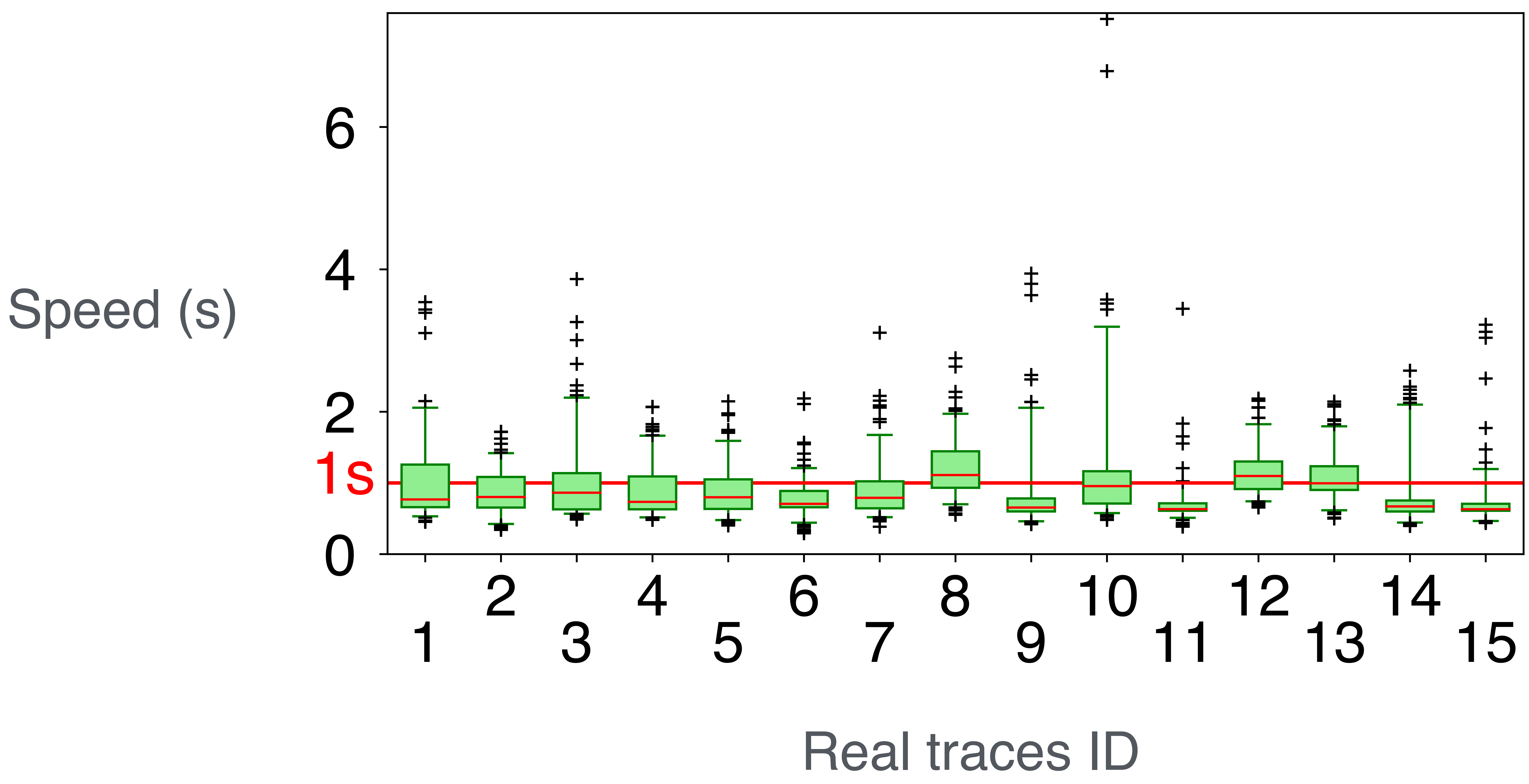
Blink failure inference accuracy is above 80% for 13 real traces out of 15



Blink avoids incorrectly inferring failures when packet loss is below 4%

packet loss %	1	2	3	4	5	...	8	9
False Positive Rate	**0**	**0**	**0**	0.67	0.67	...	1.3	2.7

Blink infers a failure within 1s for the majority of the cases



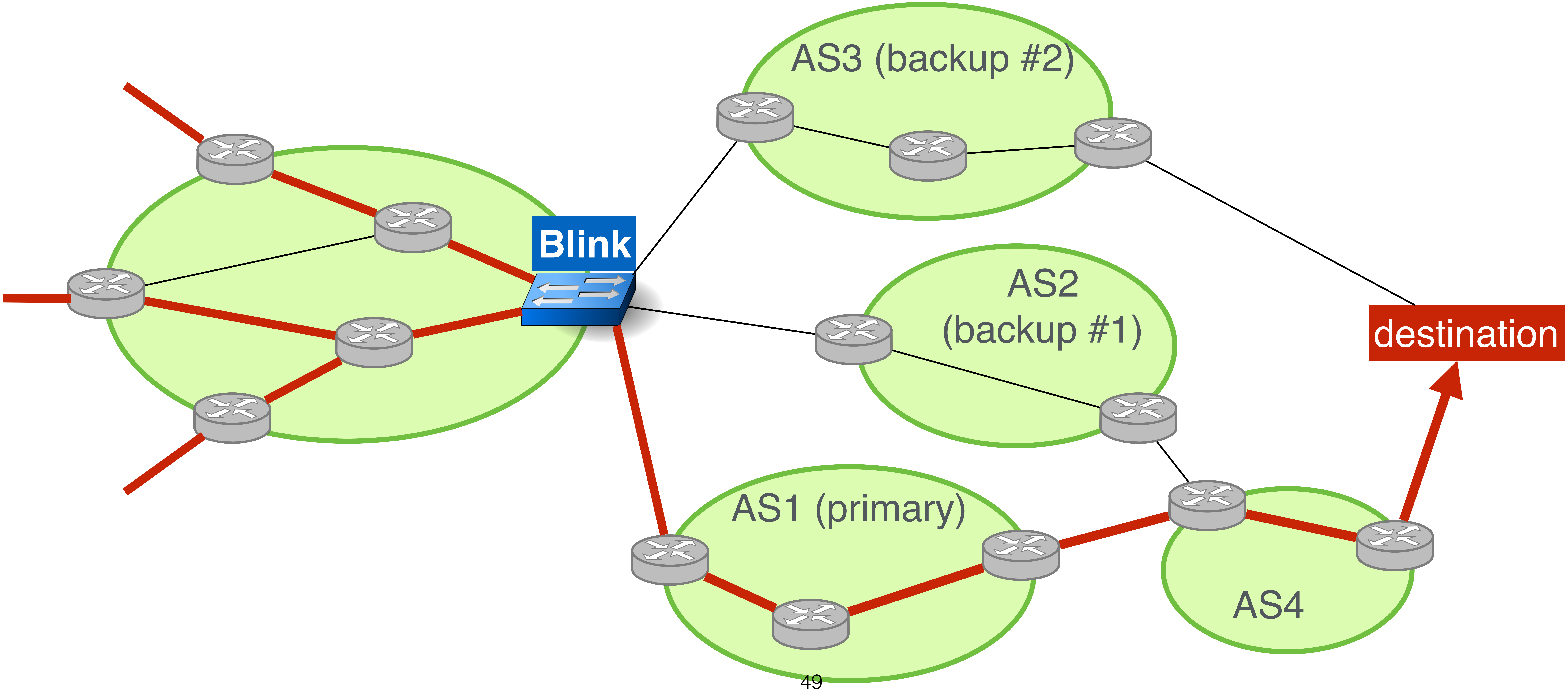
Outline

1. Why and how to use data-plane signals for fast rerouting
2. ***Blink*** infers more than 80% of the failures, often within 1s
3. ***Blink*** quickly reroutes traffic to working backup paths
4. ***Blink*** works in practice, on existing devices

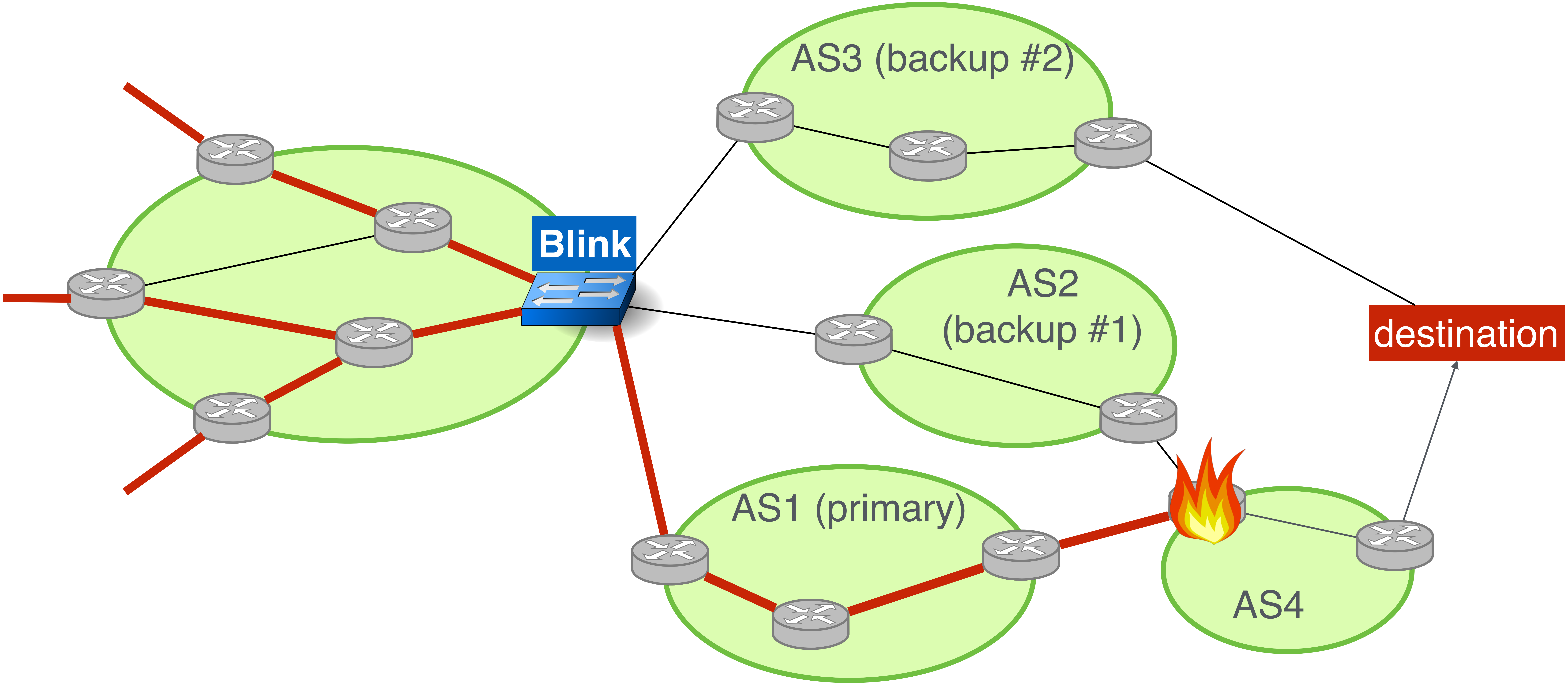
Upon detection of a failure, ***Blink*** immediately activates backup paths pre-populated by the control-plane

Problem: since the rerouting is done entirely in the data-plane,
Blink cannot prevent forwarding issues

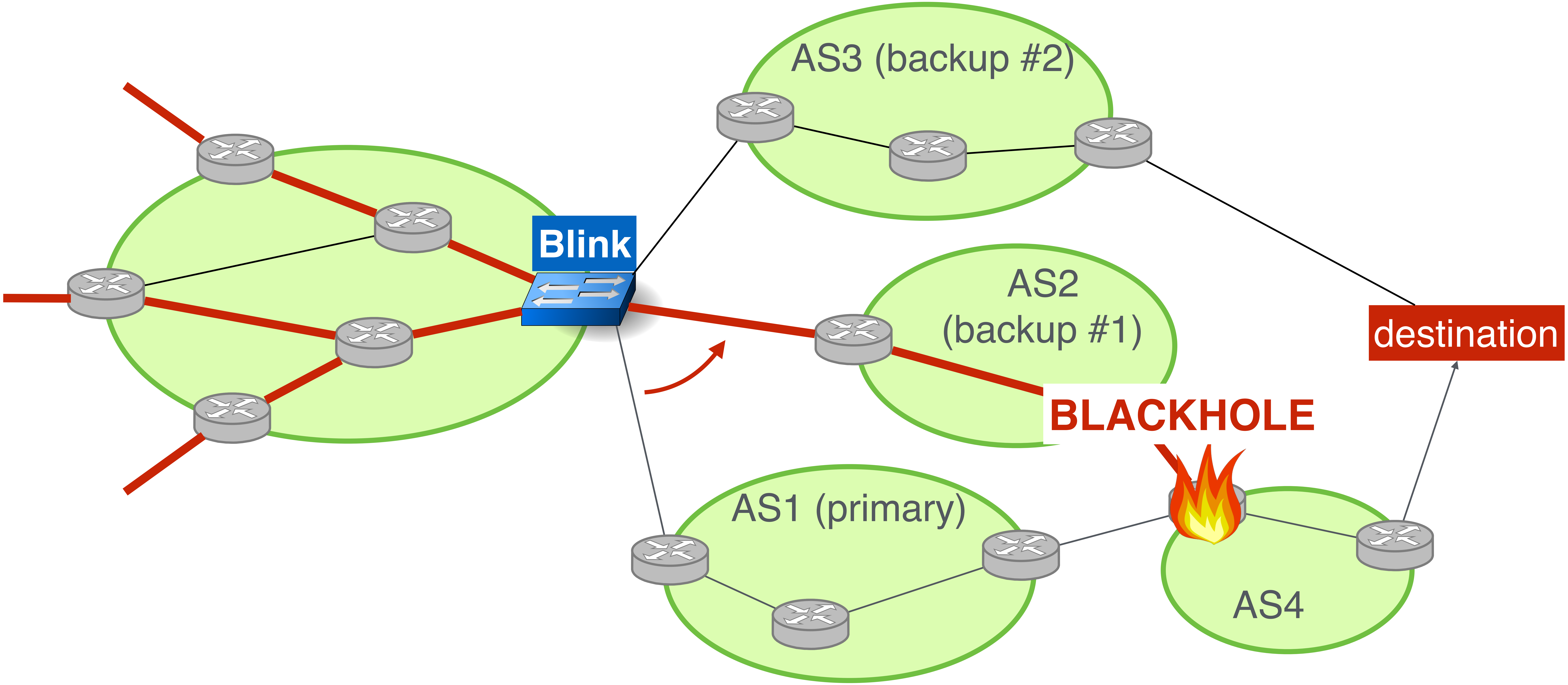
Problem: since the rerouting is done entirely in the data-plane, ***Blink*** cannot prevent forwarding issues



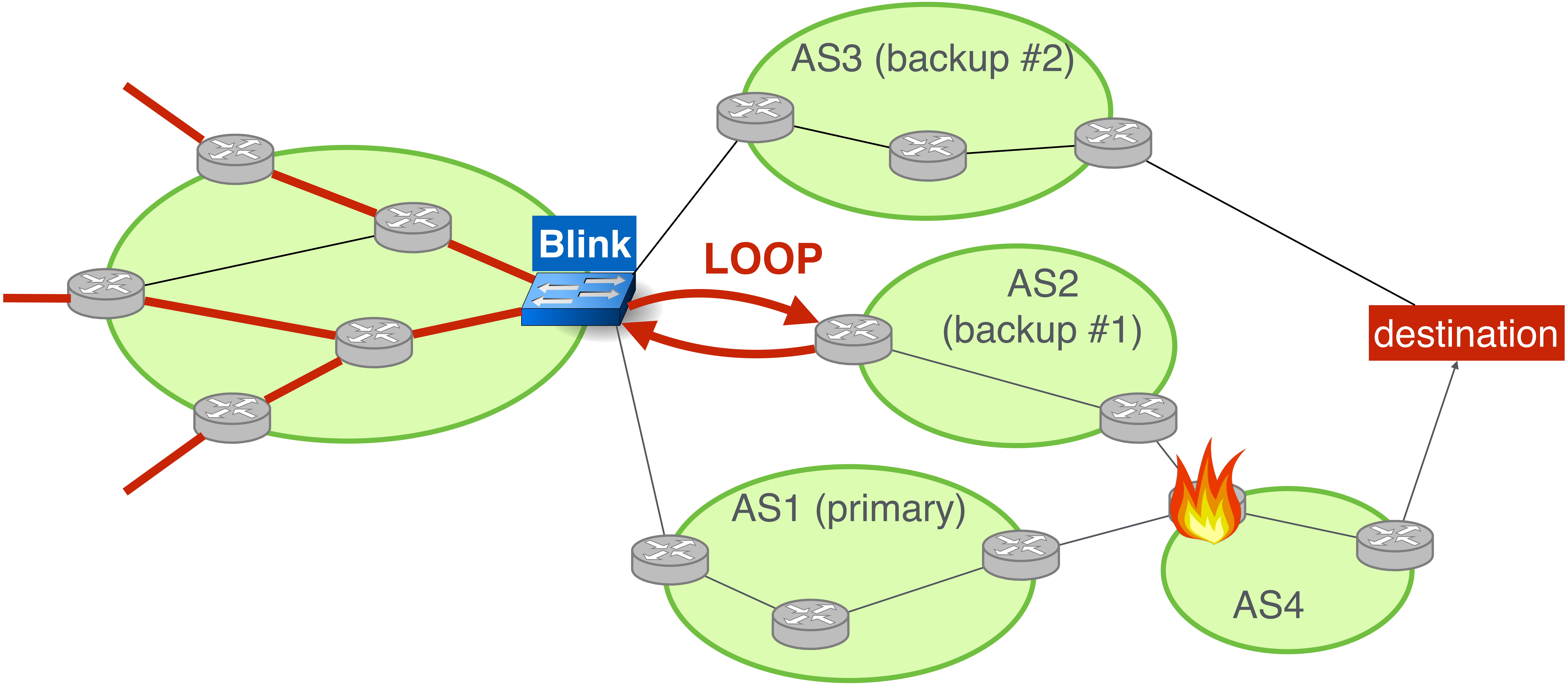
Problem: since the rerouting is done entirely in the data-plane, ***Blink*** cannot prevent forwarding issues



Problem: since the rerouting is done entirely in the data-plane, ***Blink*** cannot prevent forwarding issues

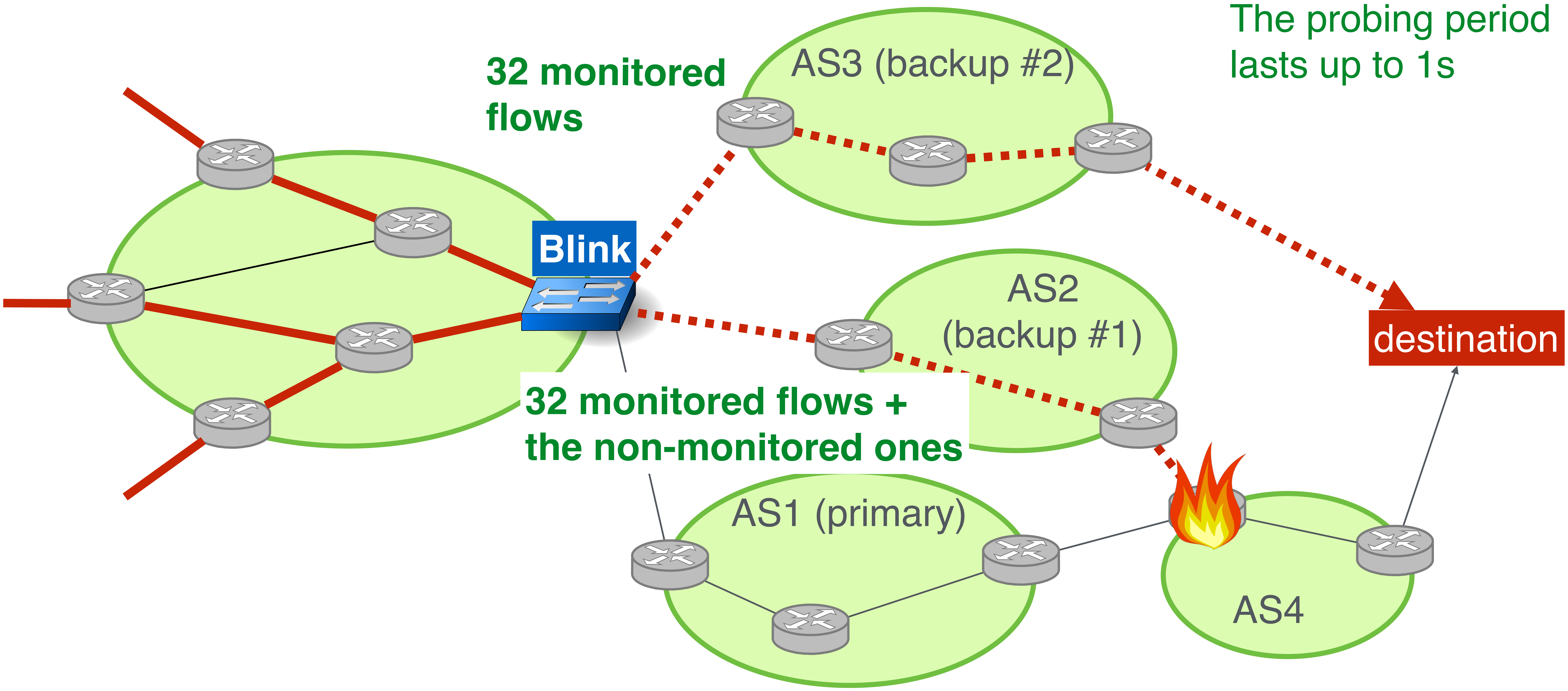


Problem: since the rerouting is done entirely in the data-plane,
Blink cannot prevent forwarding issues

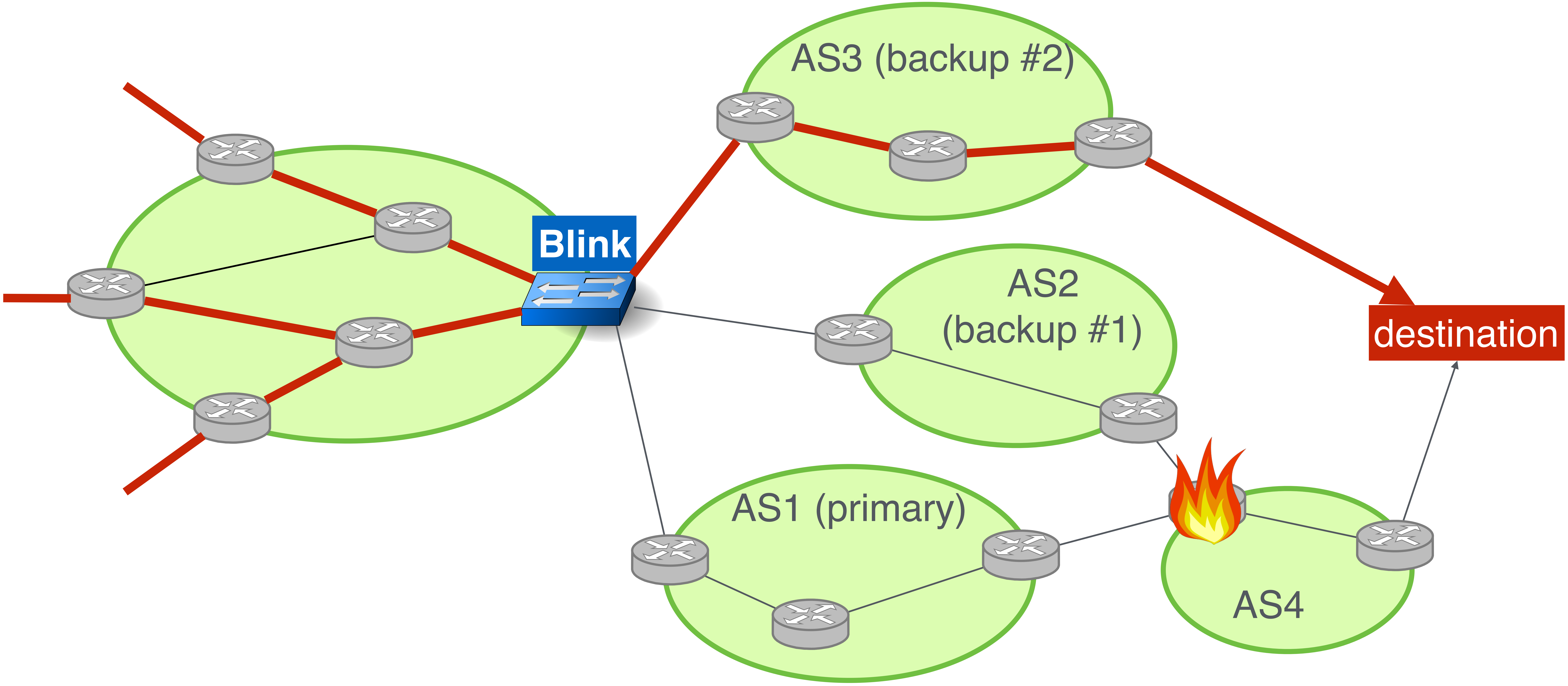


Solution: As for failures, ***Blink*** uses data-plane signals to pick a working backup path

Solution: As for failures, ***Blink*** uses data-plane signals to pick a working backup path



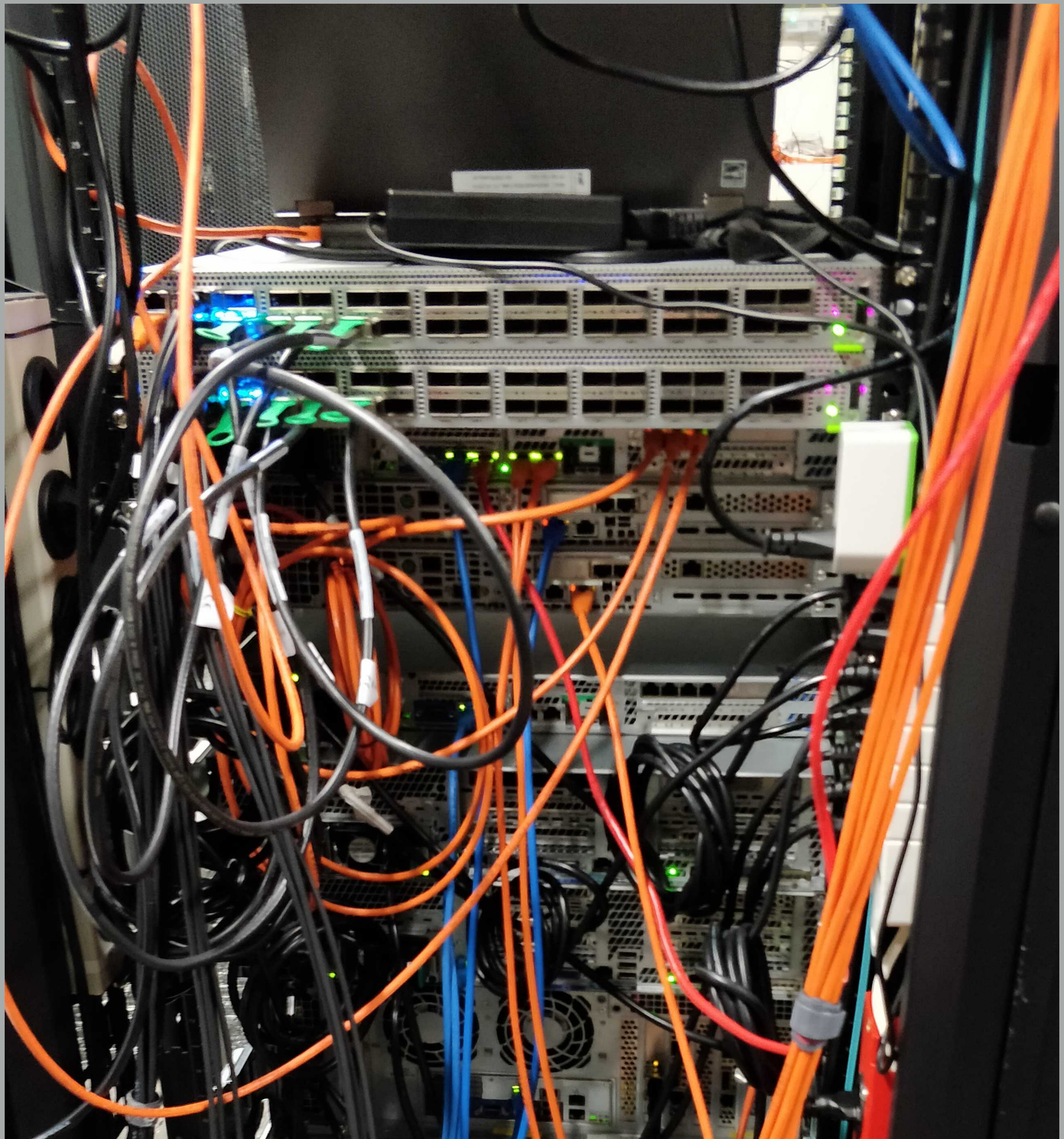
Solution: As for failures, ***Blink*** uses data-plane signals to pick a working backup path



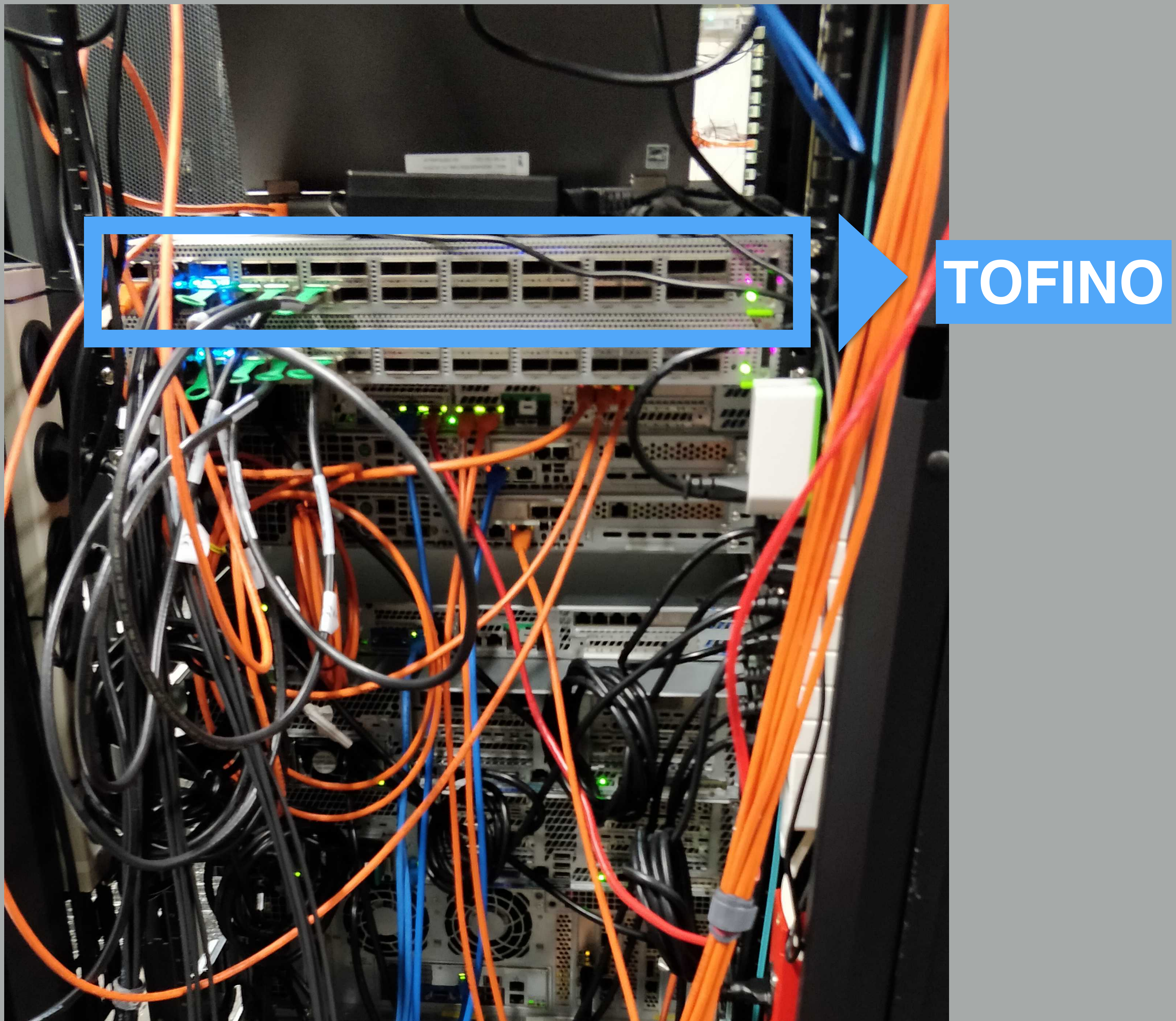
Outline

1. Why and how to use data-plane signals for fast rerouting
2. ***Blink*** infers more than 80% of the failures, often within 1s
3. ***Blink*** quickly reroutes traffic to working backup paths
4. ***Blink*** works in practice, on existing devices

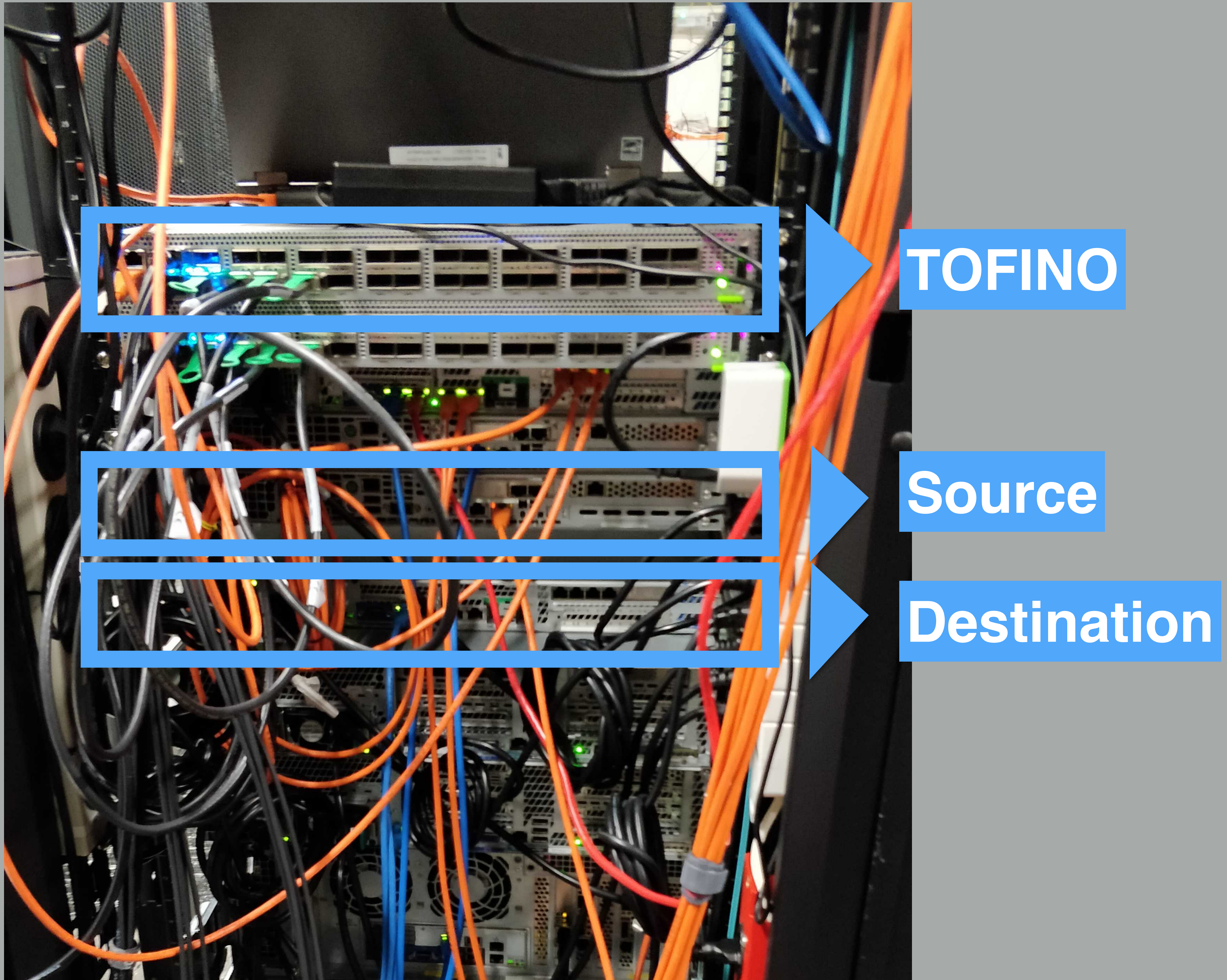
Blink works on a Barefoot Tofino switch



Blink works on a Barefoot Tofino switch



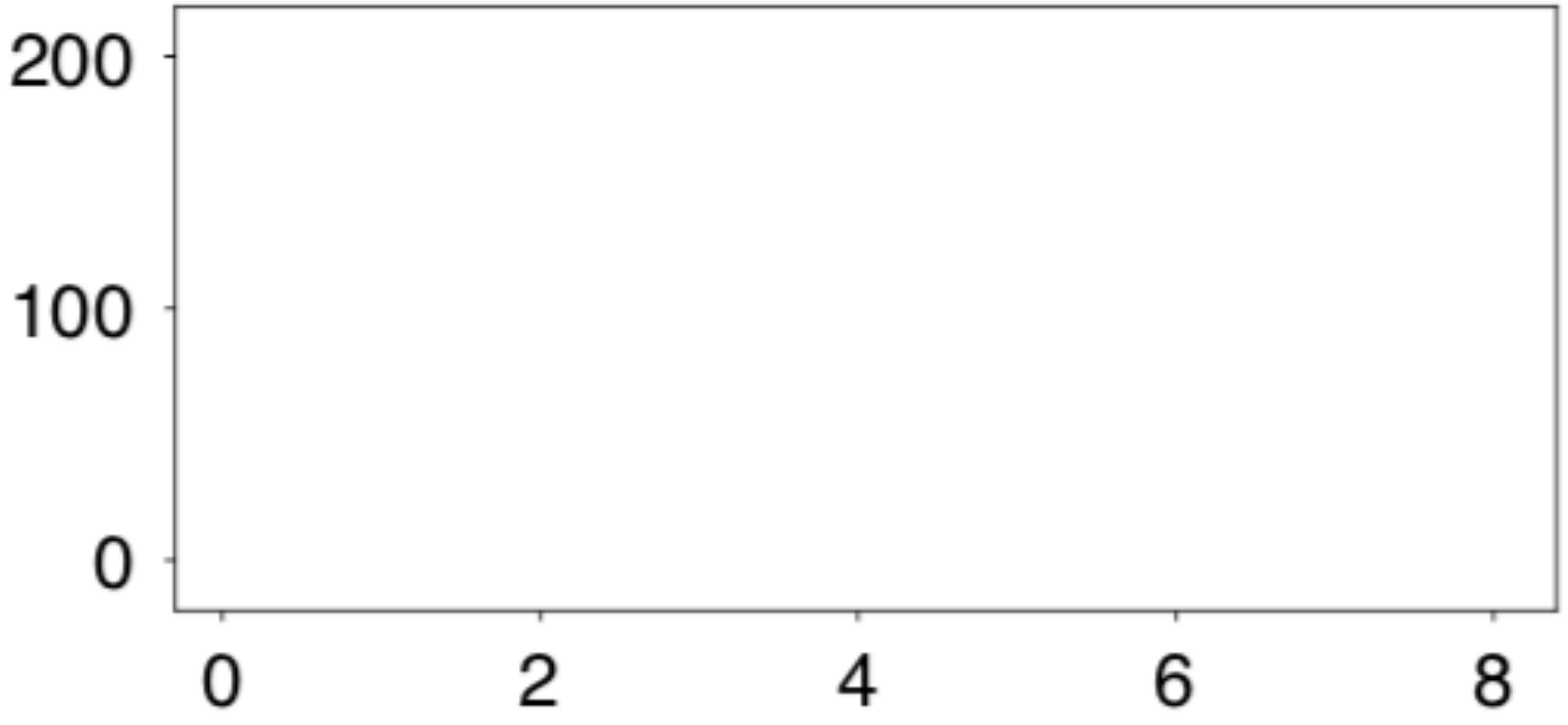
Blink works on a Barefoot Tofino switch



Blink works on a Barefoot Tofino switch

RTTs in [10ms; 300ms]

Number of packets every 100ms

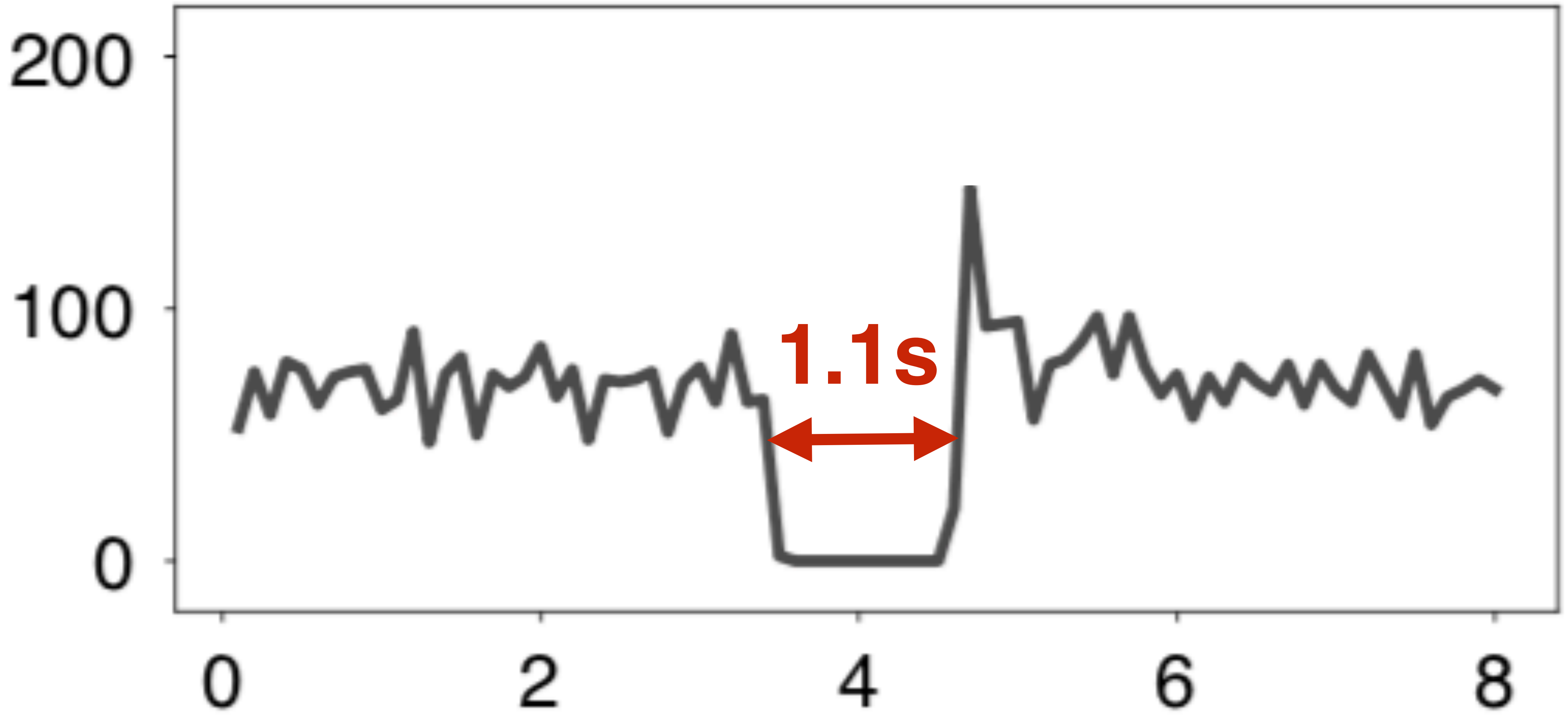


Time (s)

Blink works on a Barefoot Tofino switch

RTTs in [10ms; 300ms]

Number of packets every 100ms



Time (s)

Programmable network monitoring

and what to do with it…

Stroboscope

[NSDI 2018]

fine-grained
network monitoring

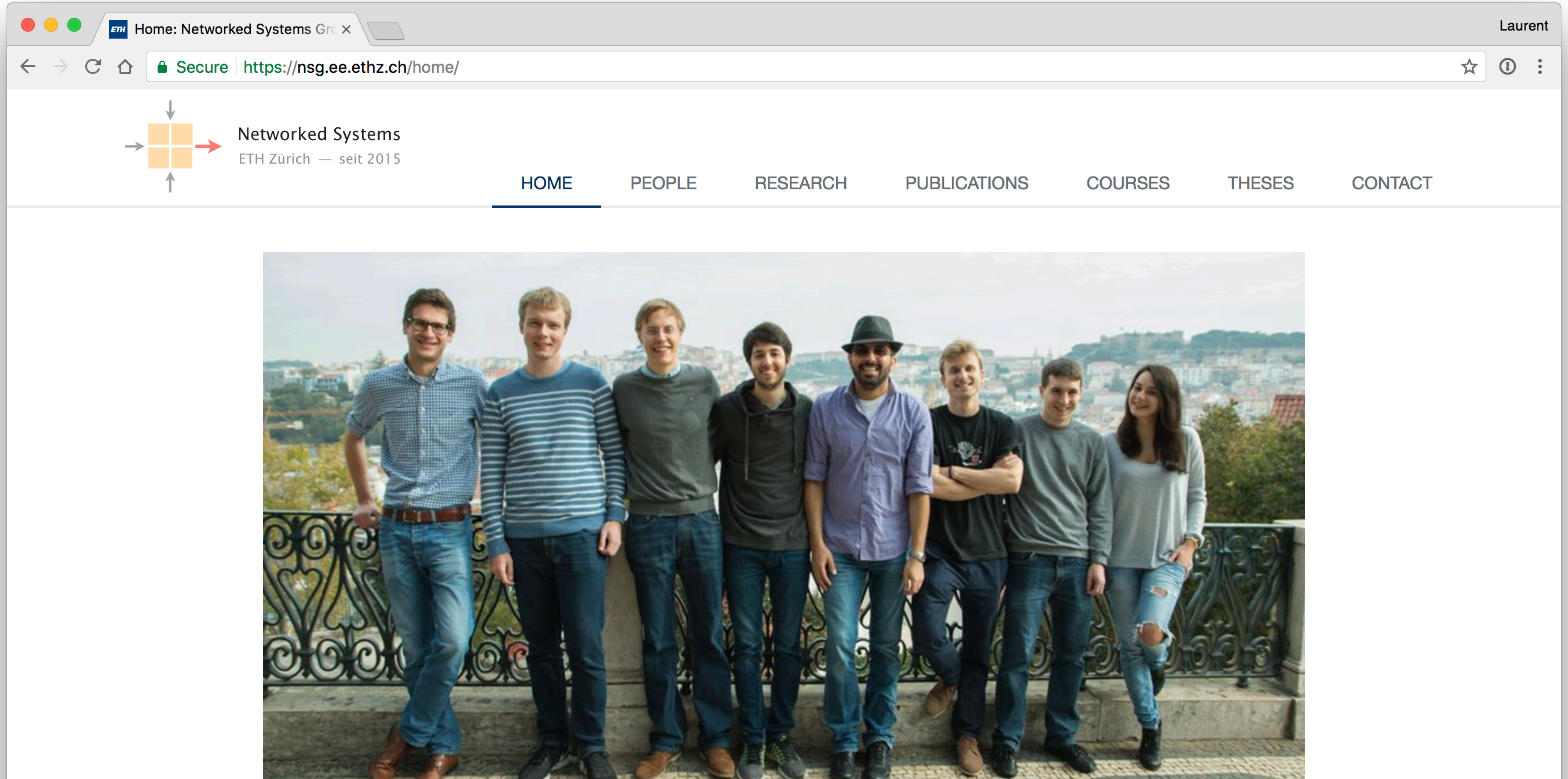
Blink

[NSDI 2019]

data-driven
fast rerouting

Check our website for more results!

https://nsg.ee.ethz.ch



Network monitoring at scale

and what to do with it…

Laurent Vanbever

nsg.ee.ethz.ch

Google Networking Summit

March 12 2019