DoS Resilience in Ad Hoc Networks

I. Aad, J.-P. Hubaux and E. Knightly

MobiCom 2004, Sept. 29th 2004,

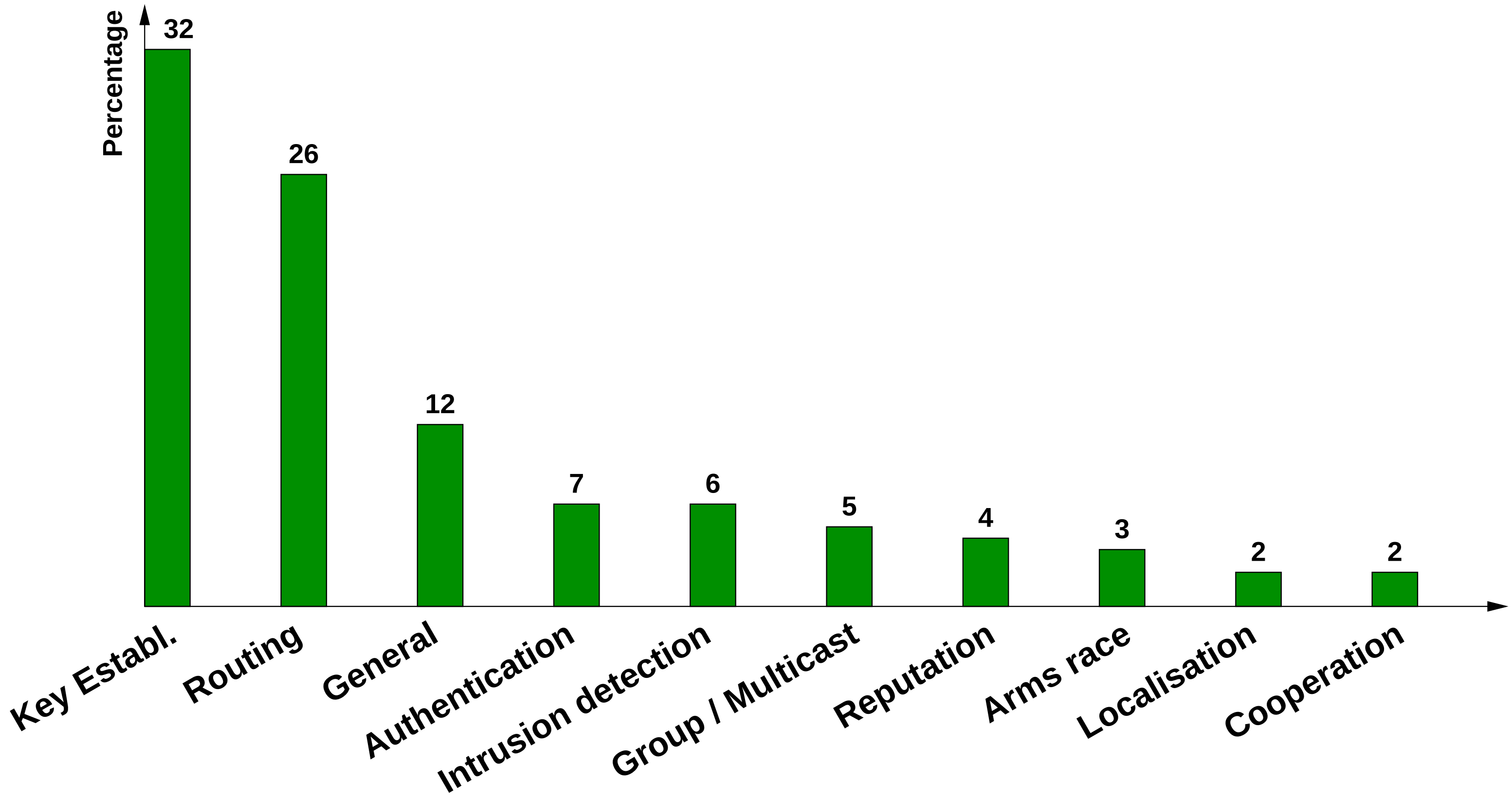
Philadelphia - PA, USA

Outline

- $\boxed{\text{Introduction and system model}}$
- DoS attacks:
  - “Protocol-compliant” attacks: JellyFish
  - BlackHole
- The cost of counter-measures
- Network performance under DoS attacks
- Conclusion

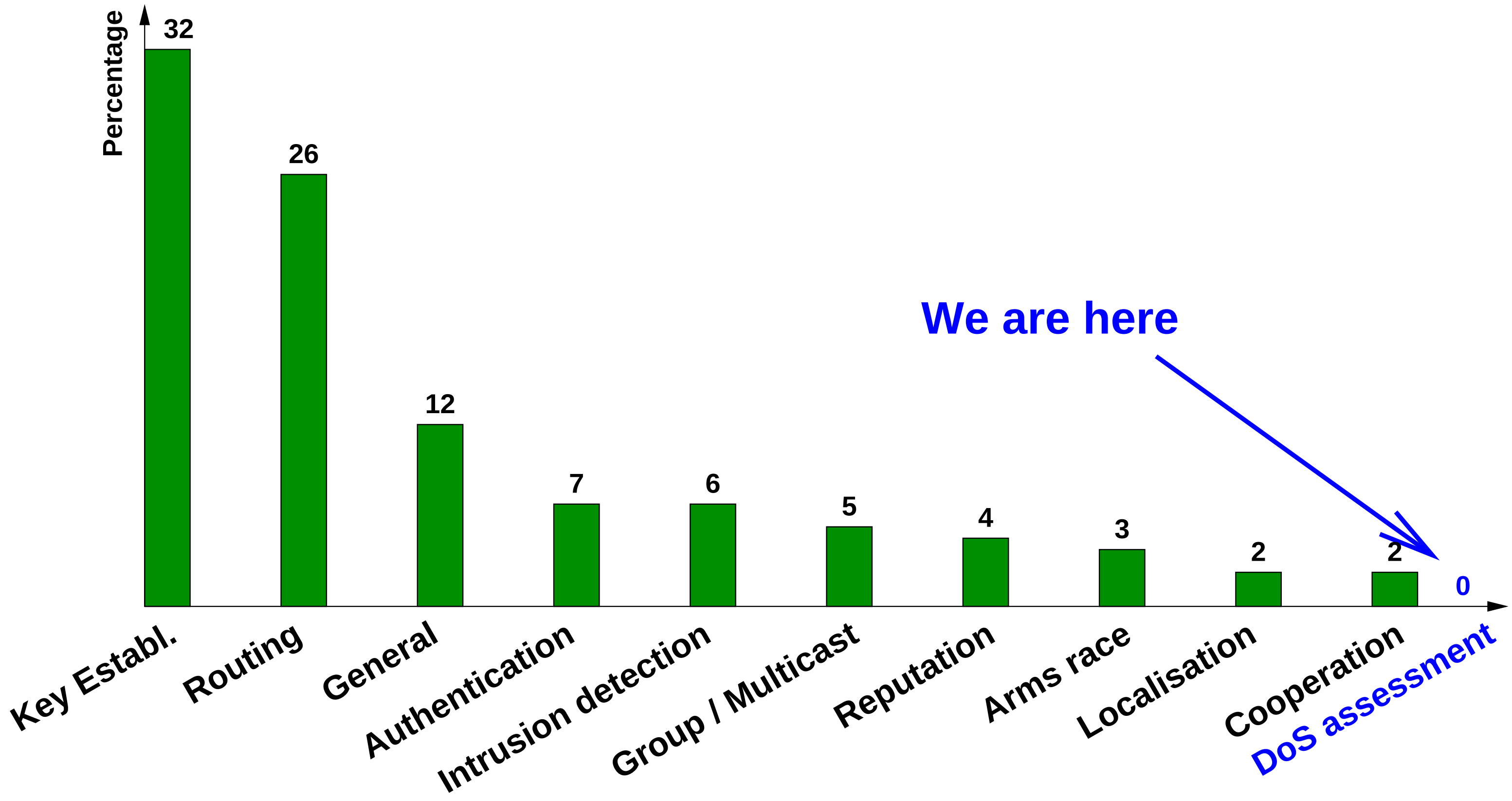
Introduction

Significant work has been made in:



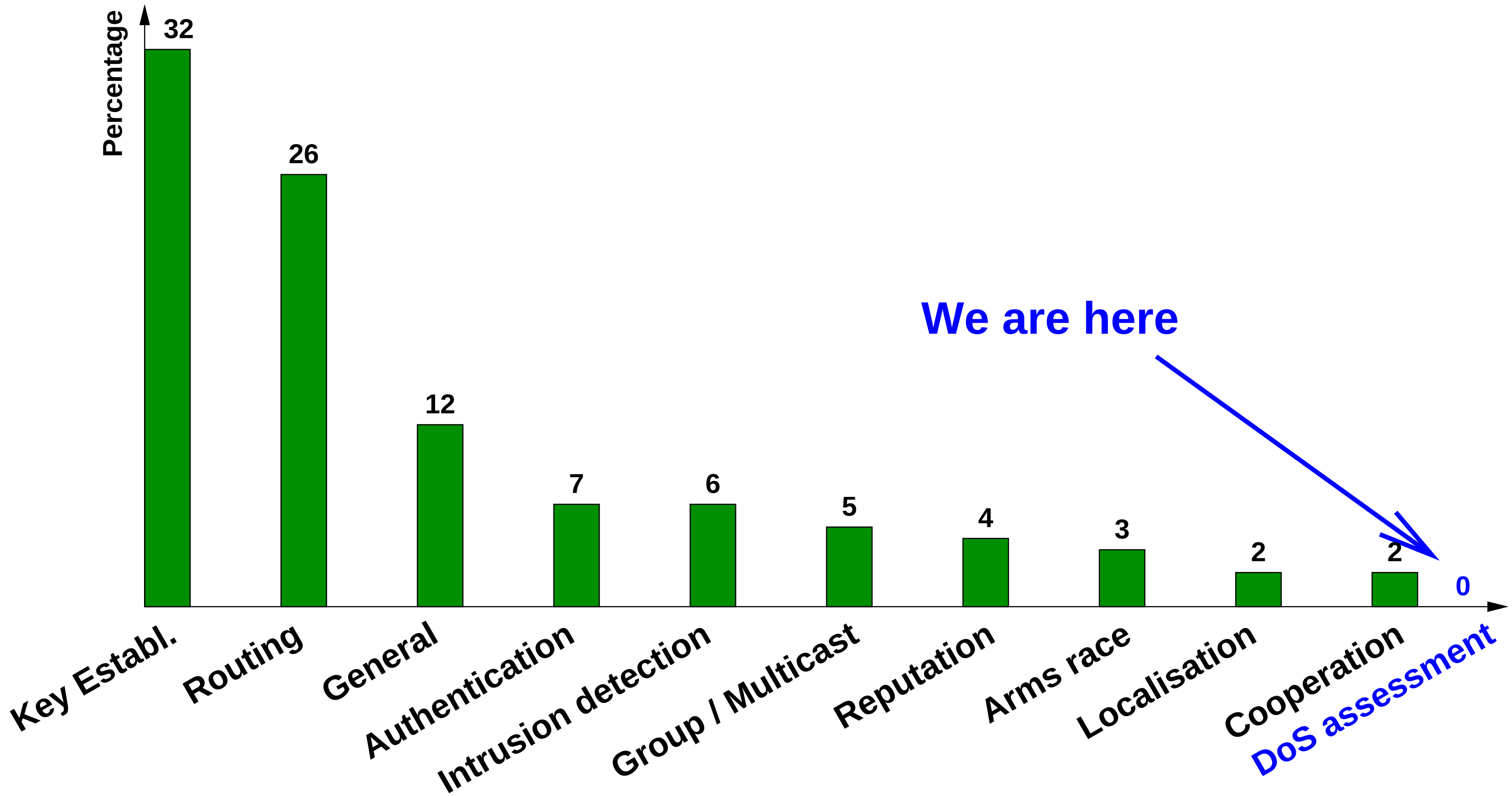
Introduction

Significant work has been made in:



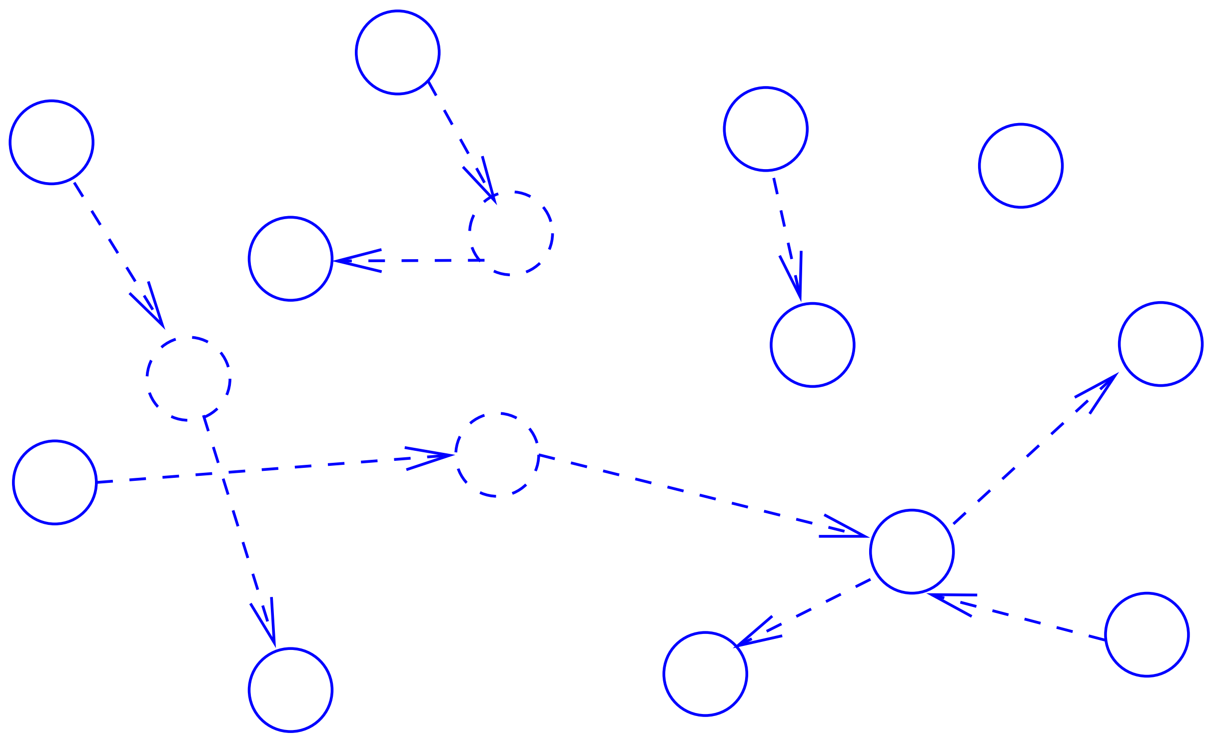
Our goal: quantify the damage of a DoS attack on an ad-hoc network

Significant work has been made in:



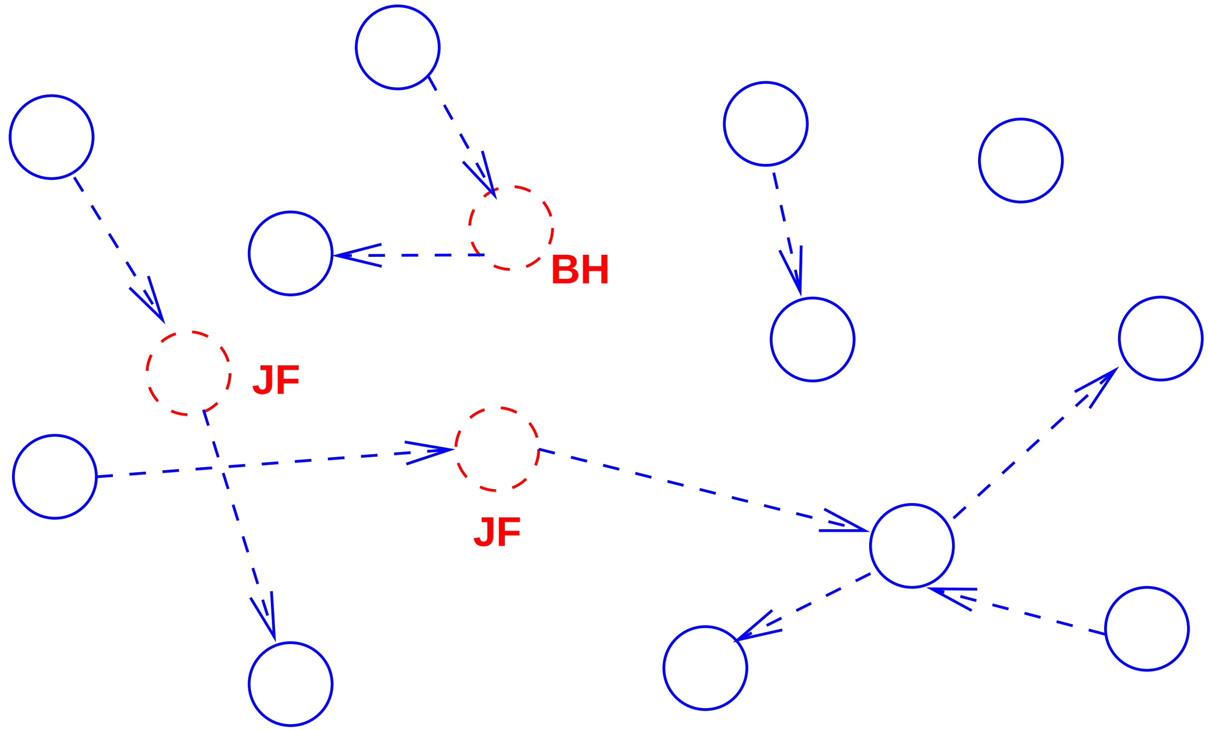
Design (and study) a new class of “protocol-compliant” attacks

System model



Ad-hoc multi-hop network, Mobile nodes, Secure routing, Node Authentication, 1 ID/node, Packet Authentication and Encryption...

System model



JF: JellyFish
BH: BlackHole

The dual role of hosts as routers introduces a critical vulnerability!

Outline

- Introduction and system model
- DoS attacks:
  - “Protocol-compliant” attacks: JellyFish
  - BlackHole
- The cost of counter-measures
- Network performance under DoS attacks
- Conclusion

What is a "protocol-compliant" attack?

Just like any IP service, it can:

- Drop packets
- Reorder packets
- Delay / jitter packets

What is a "protocol-compliant" attack?

Just like any IP service, it can:

- Drop packets
- Reorder packets
- Delay / jitter packets

BUT!
in a MALICIOUS way...

What is a "protocol-compliant" attack?

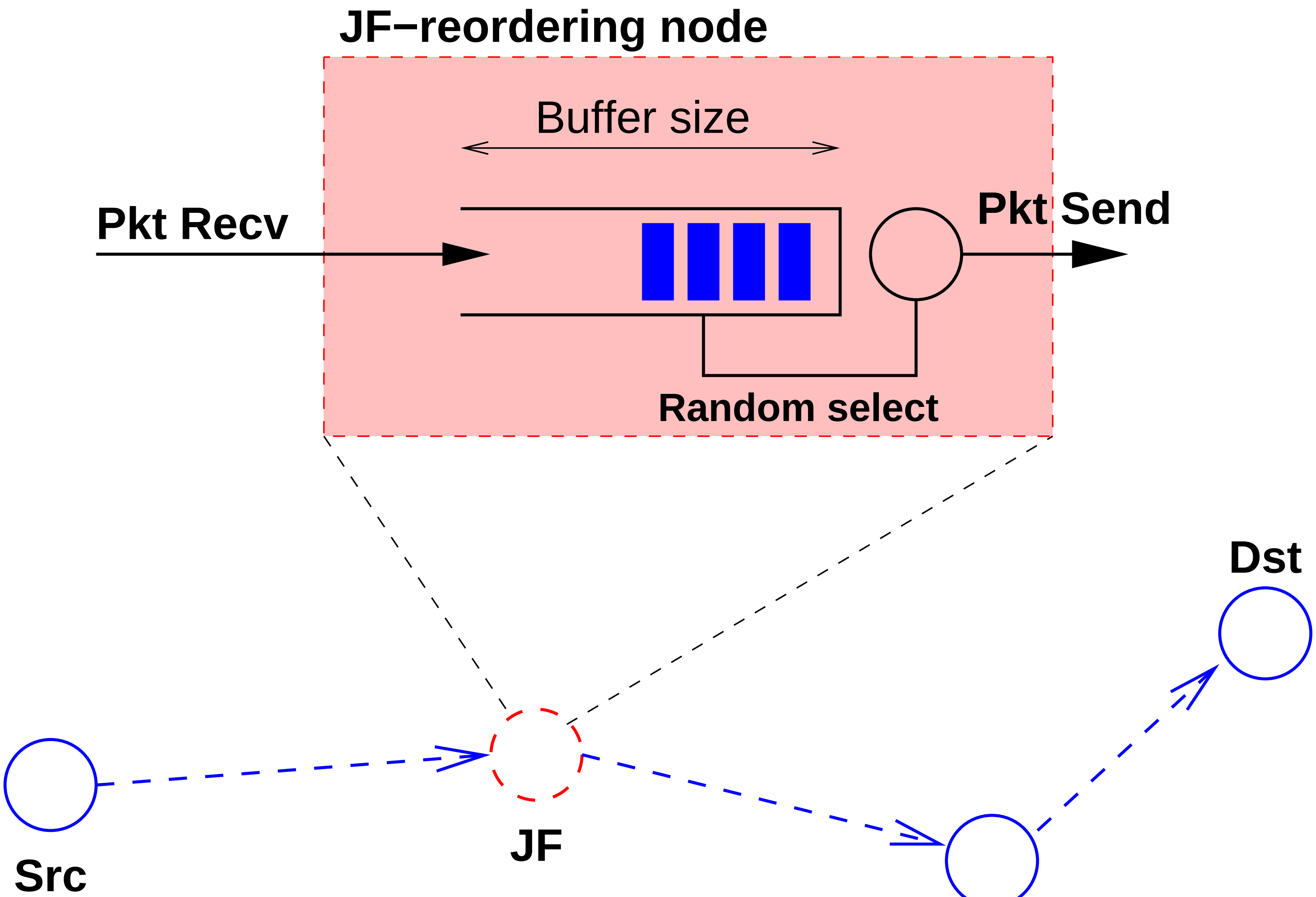
Just like any IP service, it can:

- Drop packets
- Reorder packets
- Delay / jitter packets

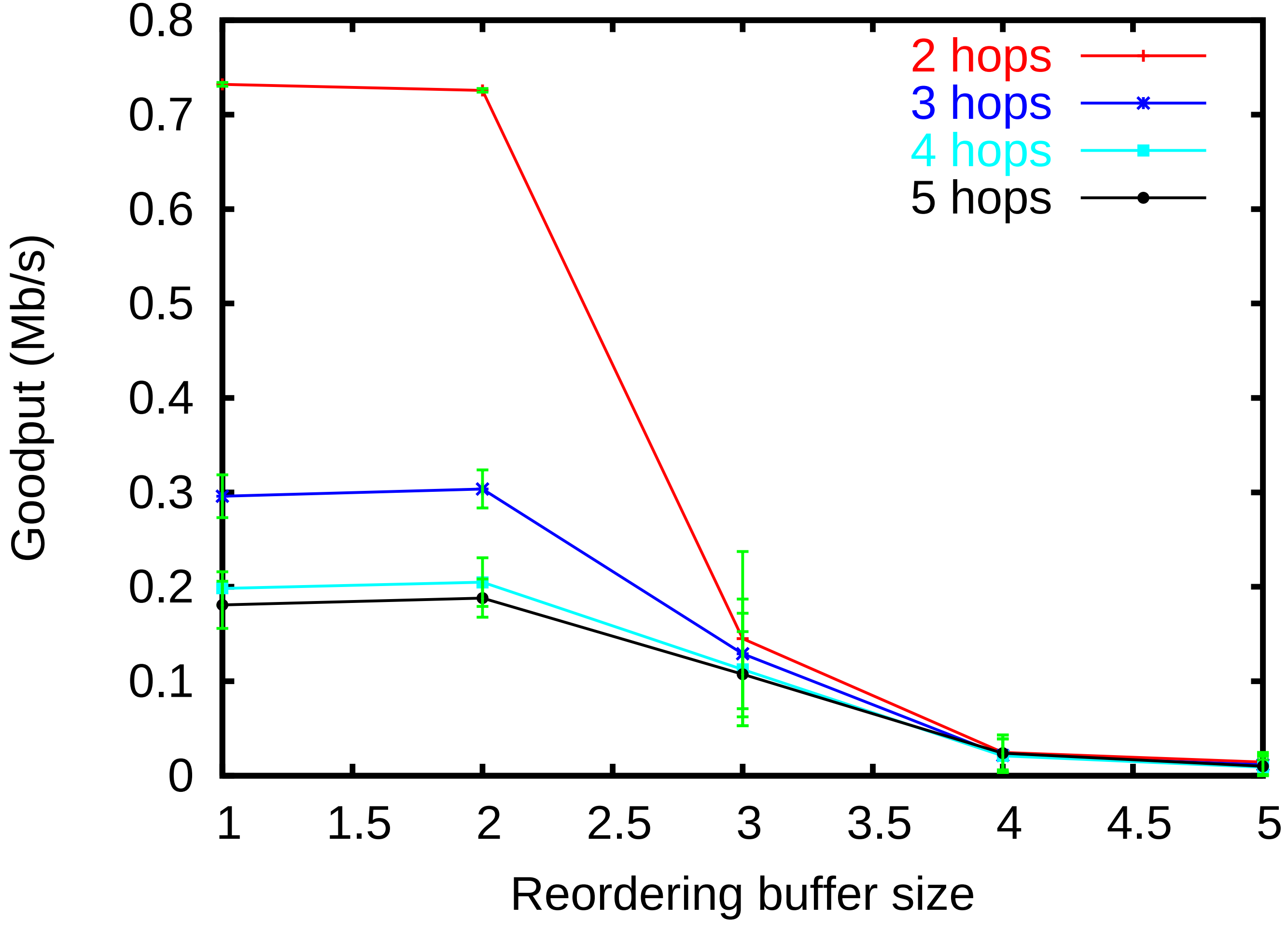
Why use "protocol-compliant" attacks ?

Detection and diagnosis are time consuming!

Example: the JellyFish

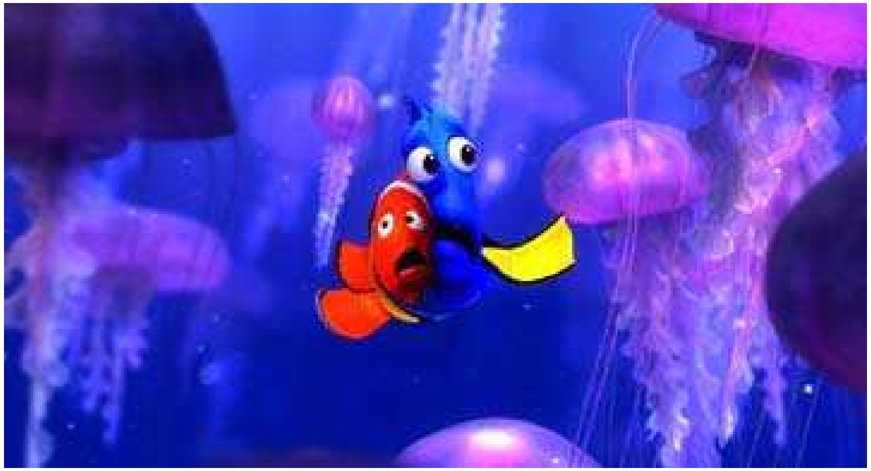


Example: the JellyFish



Reordering >3 packets reduces TCP throughput to ≈zero!

The JellyFish



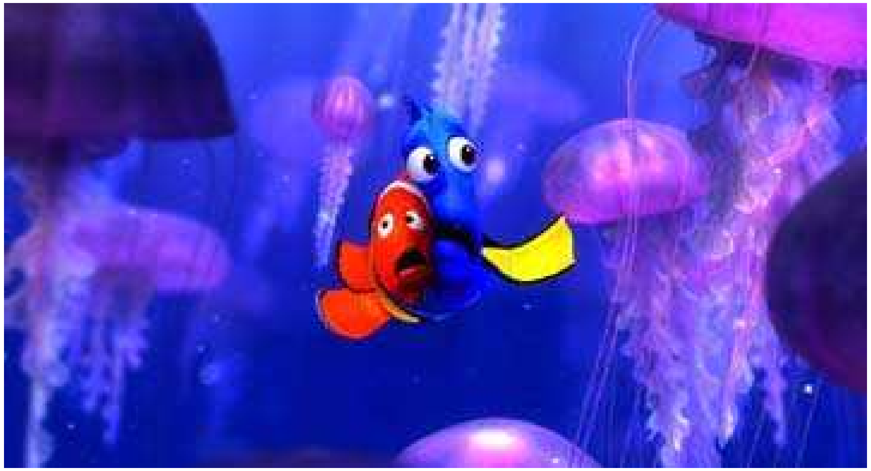
- For closed-loop traffic: TCP, TFRC-like...
- Passive
- Hard to detect...
  ... until after the "sting"

End-to-end control protocols infer network status from feedback measurements.

JF interferes with these measurements...

... to attenuate the traffic flows.

The JellyFish



- For closed-loop traffic: TCP, TFRC-like...
- Passive
- Hard to detect...
  ... until after the "sting"

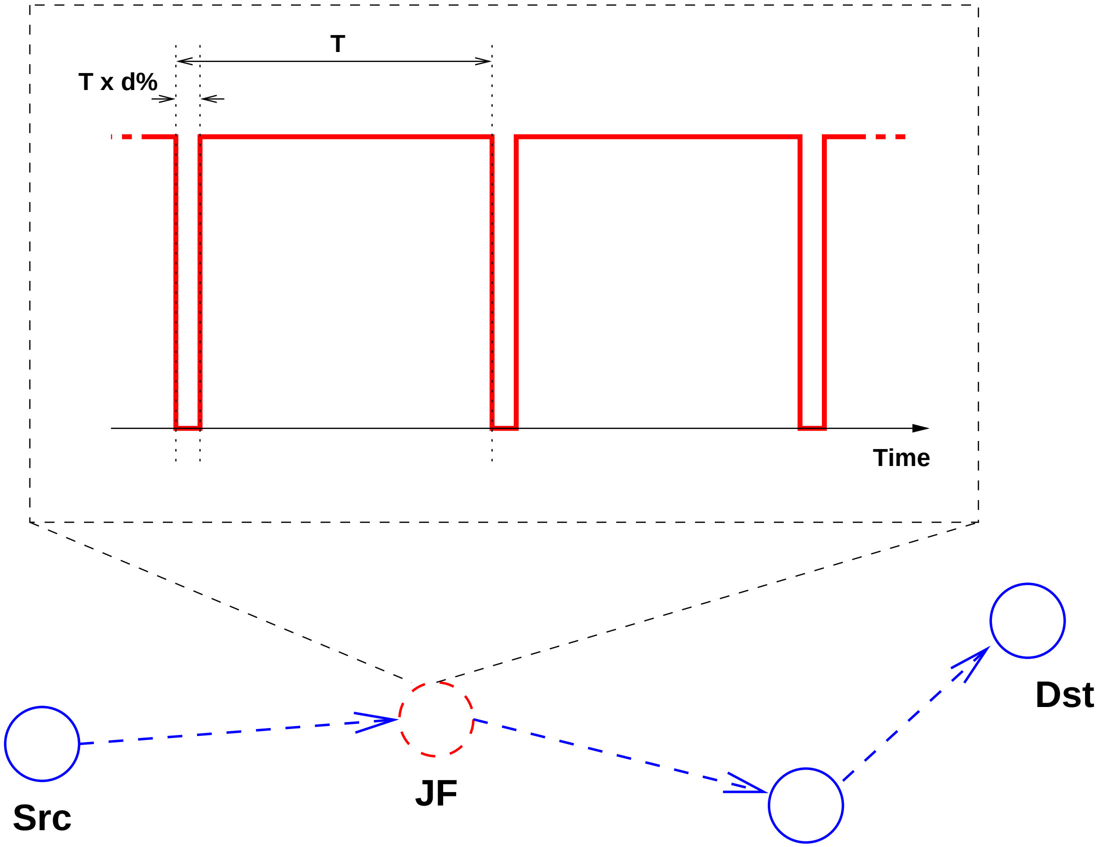
Species:

- JF-Reorder → *“multipath”*
- JF-drop → *“congestion, buffer overflow...”*
- JF-Jitter (variable RTT) → *“variable loads”*

JF-drop

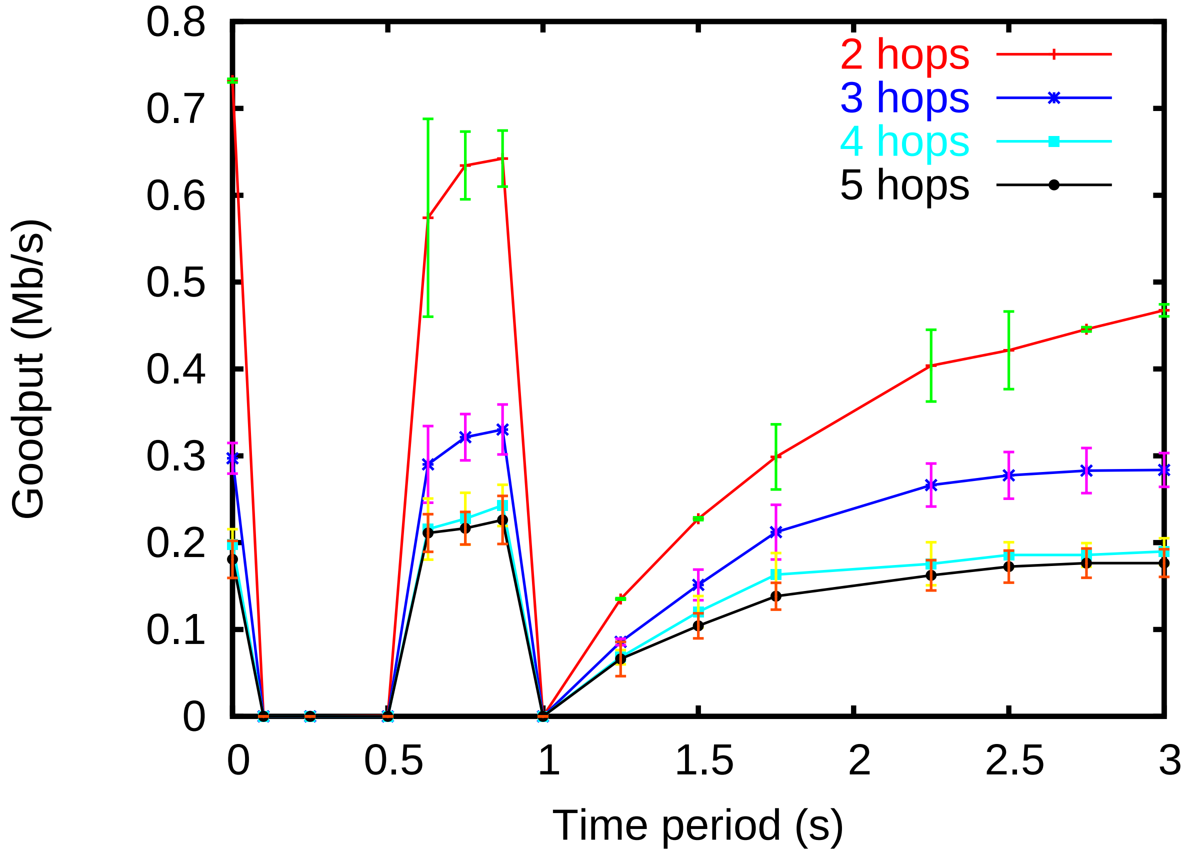
For wired networks: the Shrew [Kuzmanovic & Knightly]

Dropping 5% of the packets periodically (@T = 1sec)



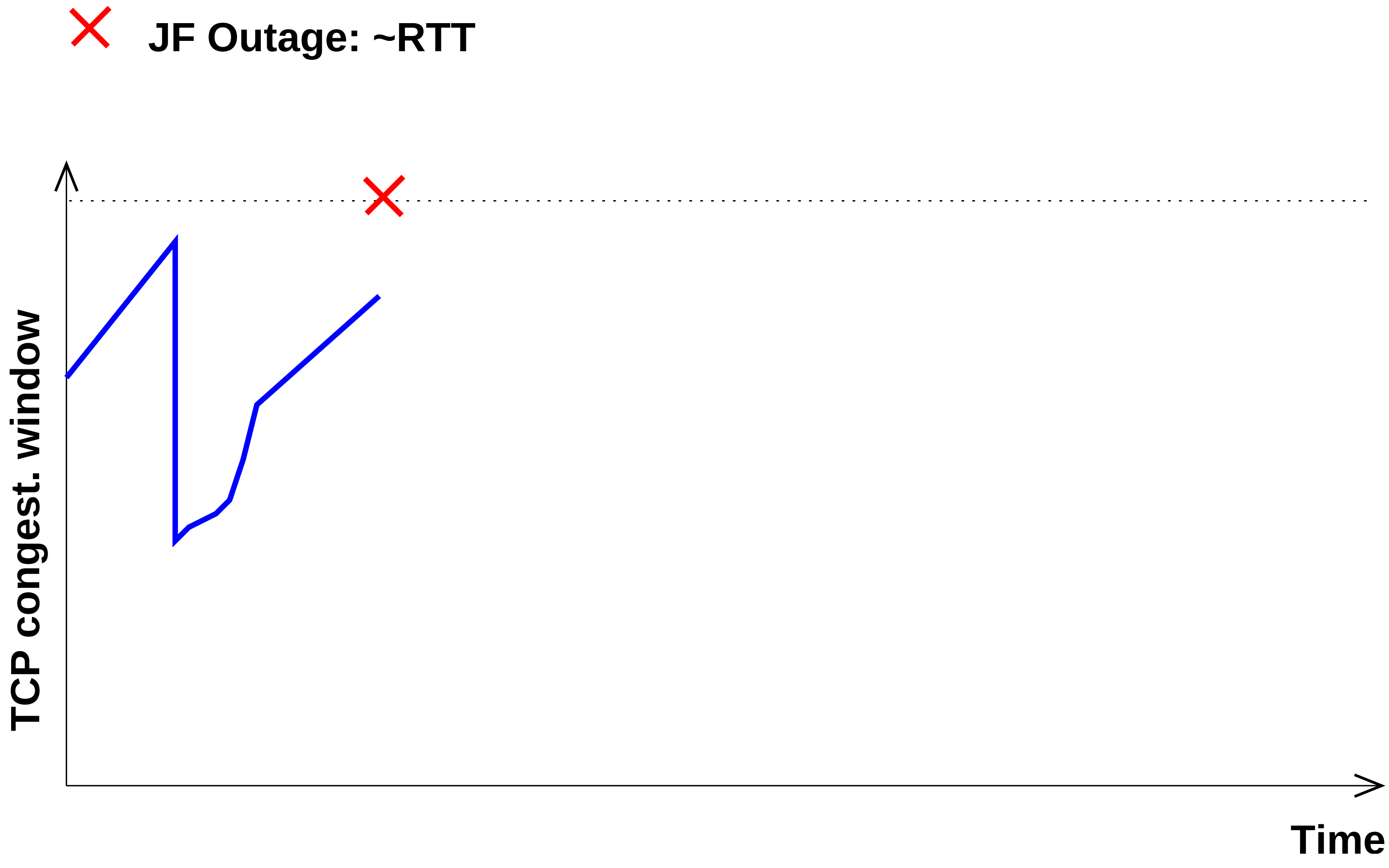
JF-drop

Dropping 5% of the packets periodically (@T = 1sec)

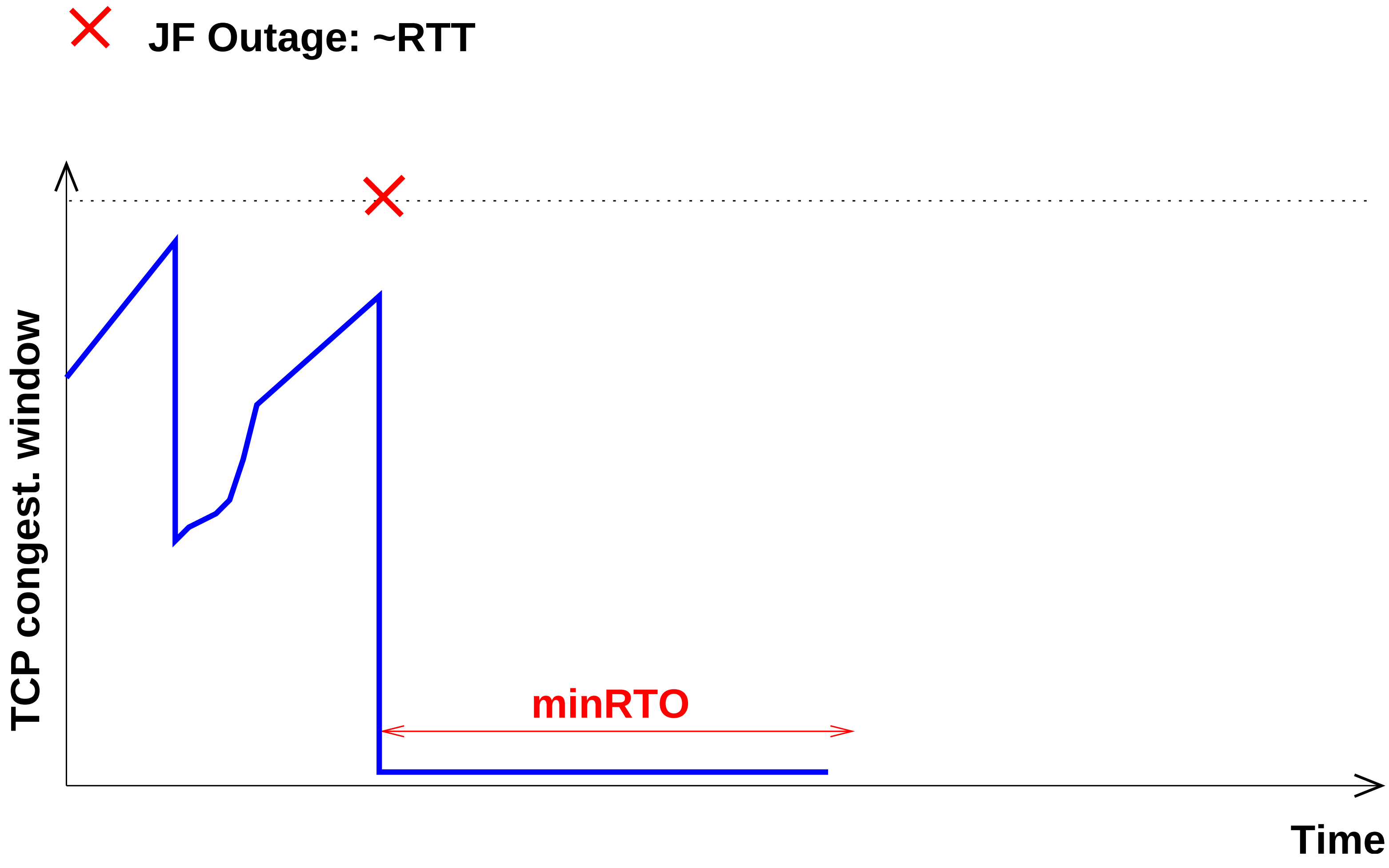


... reduces TCP throughput to zero!

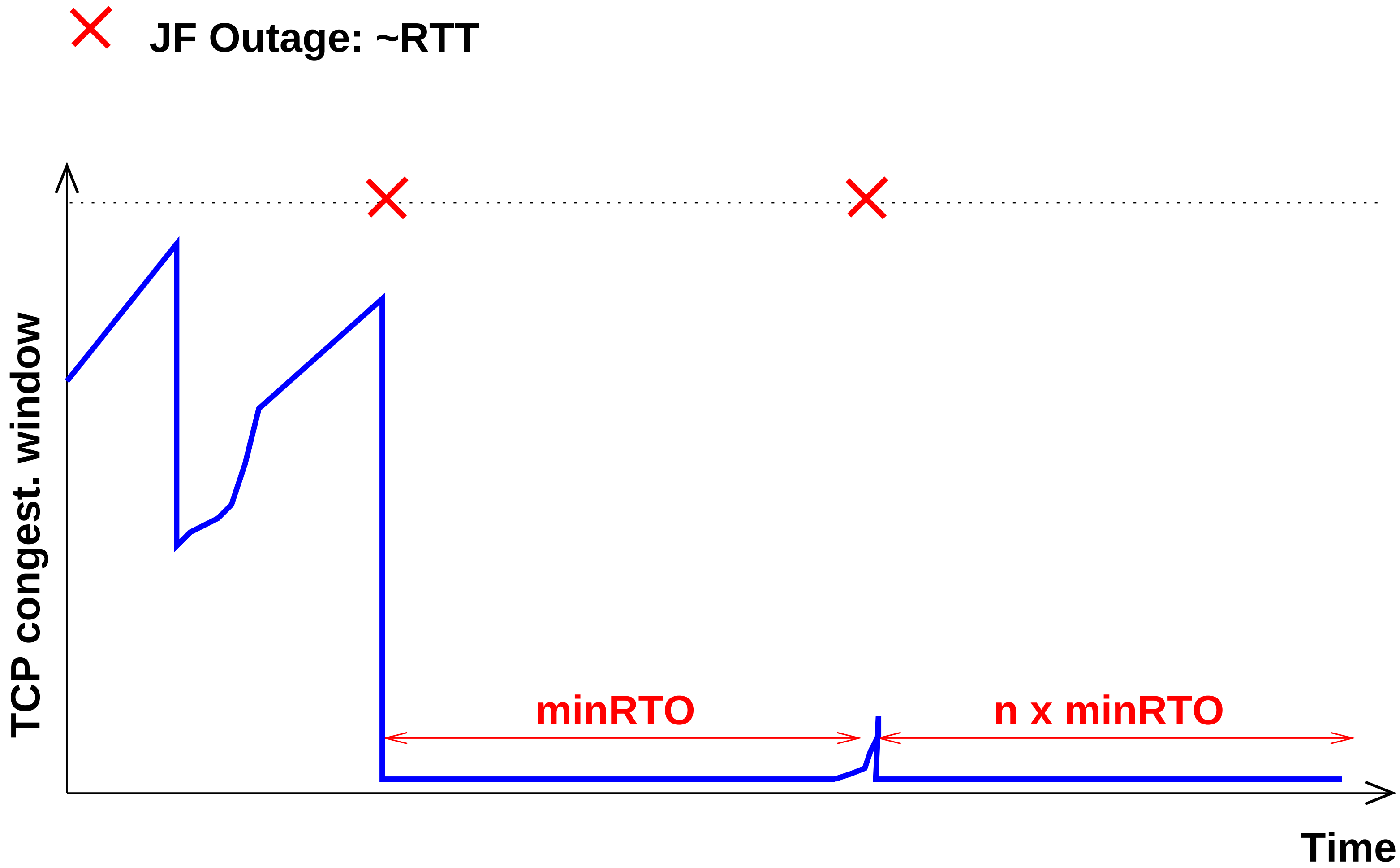
JF-drop



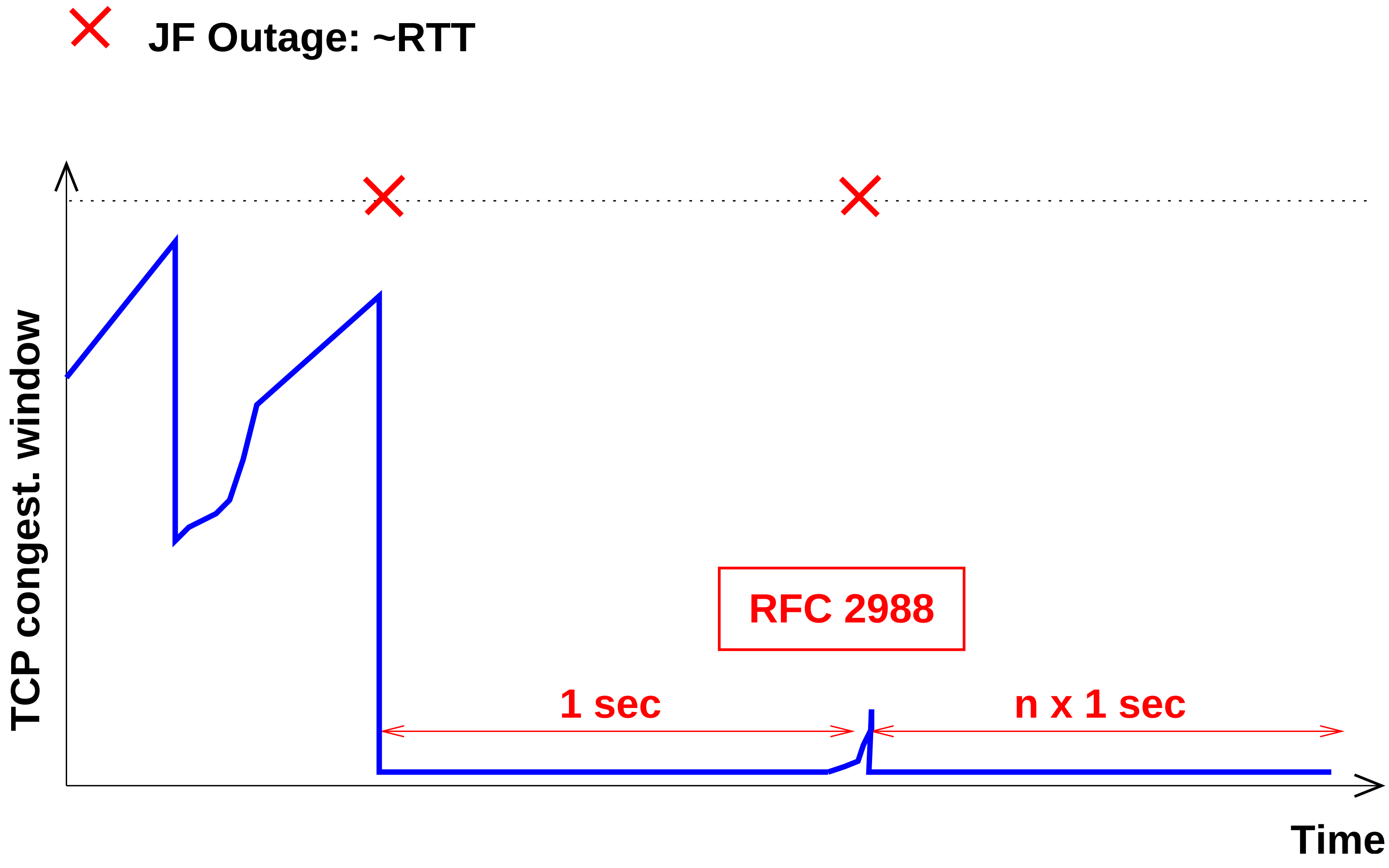
JF-drop



JF-drop

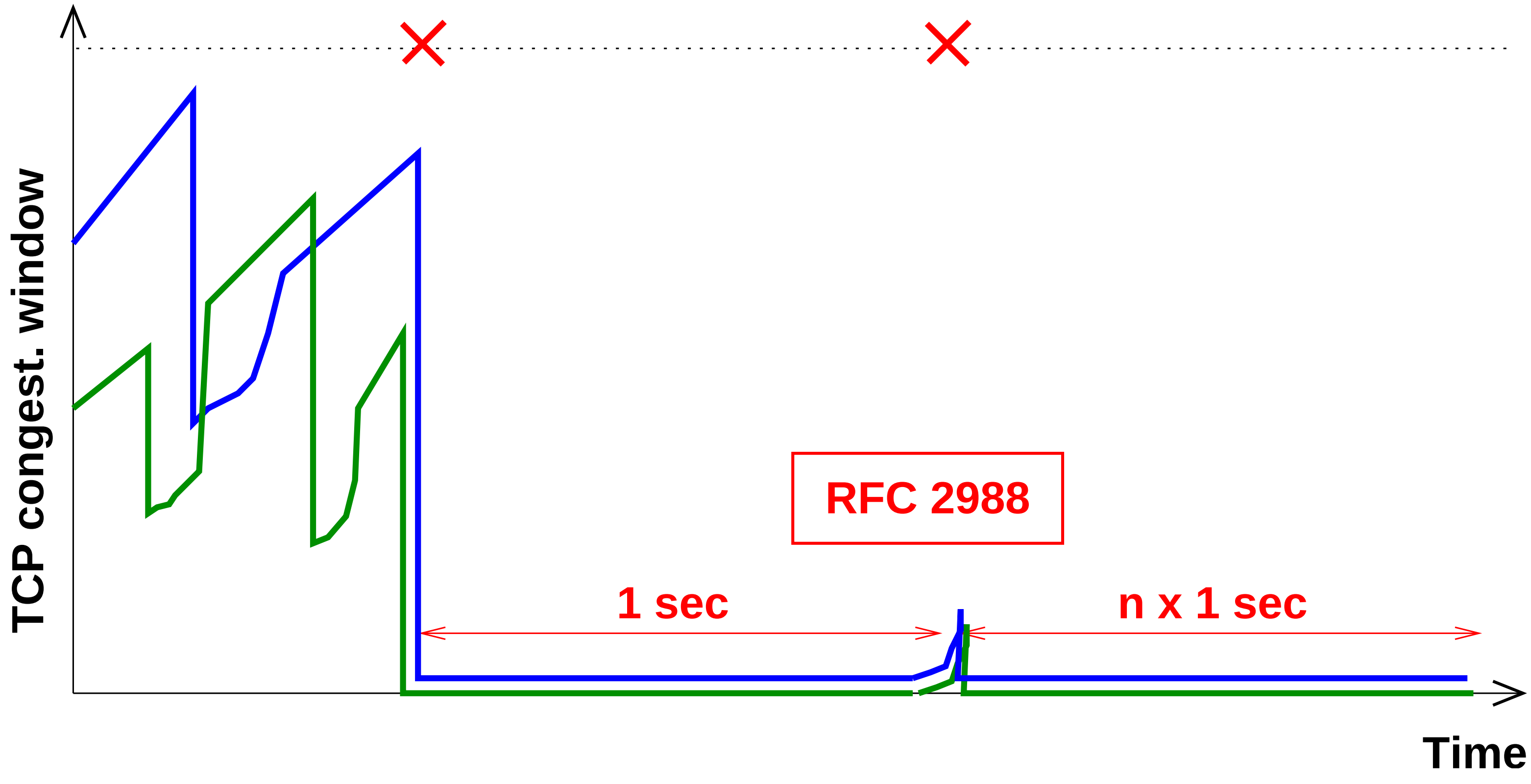


JF-drop

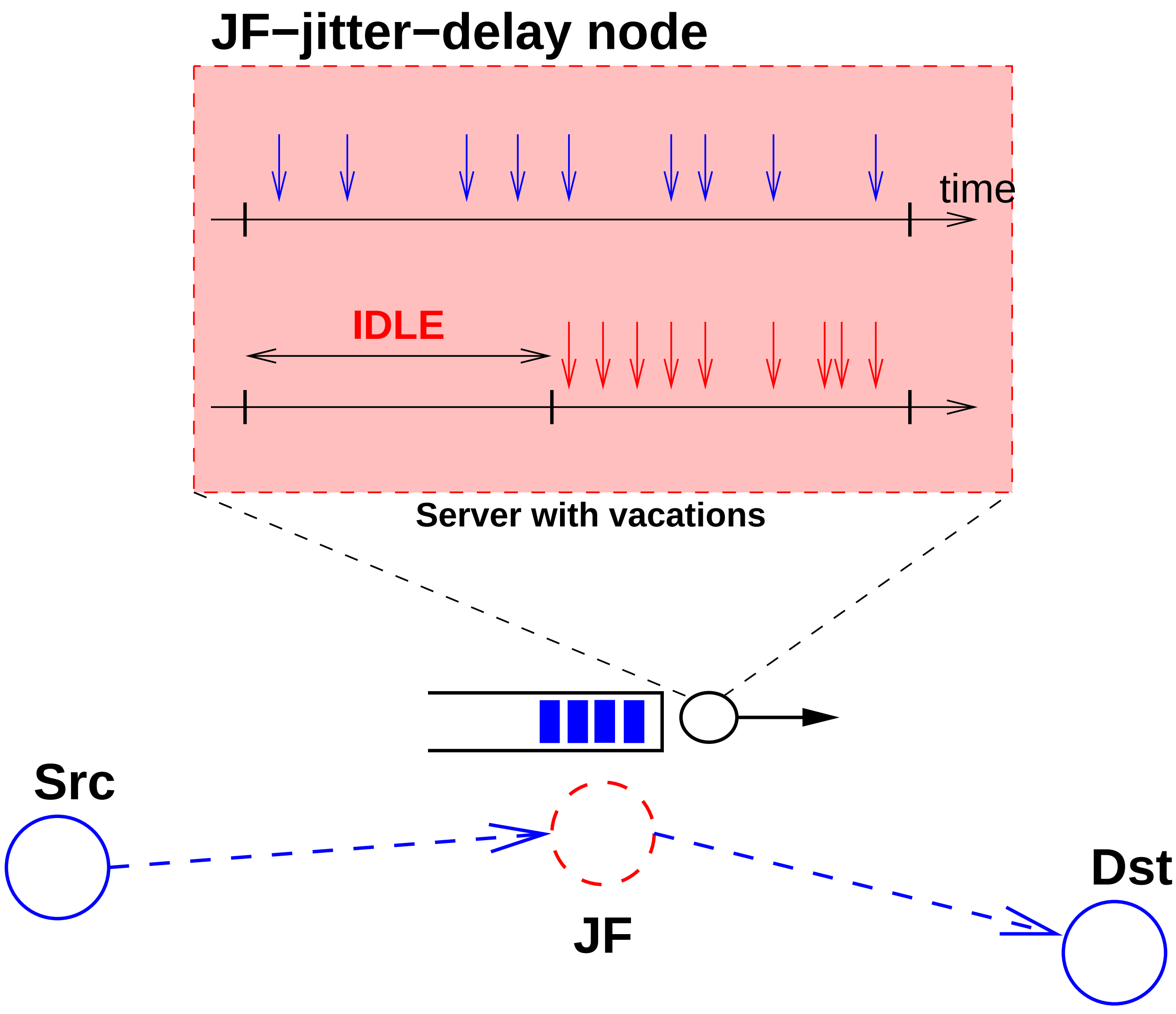


JF-drop

JF Outage: ~RTT



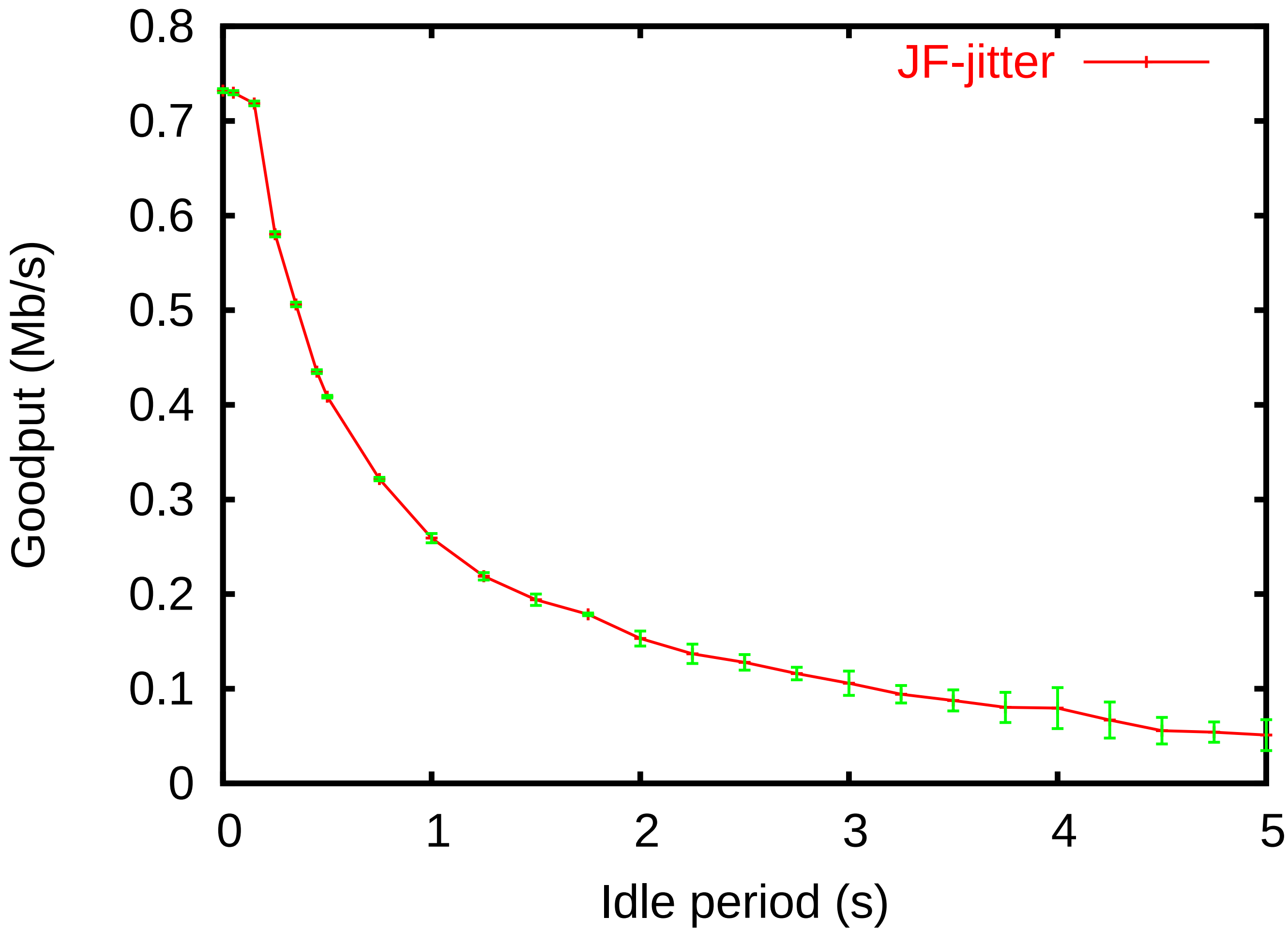
JF-jitter



JF−jitter−delay node
time
IDLE
Server with vacations
Src
JF
Dst

JF-jitter

TCP infers network/congestion status using RTT...



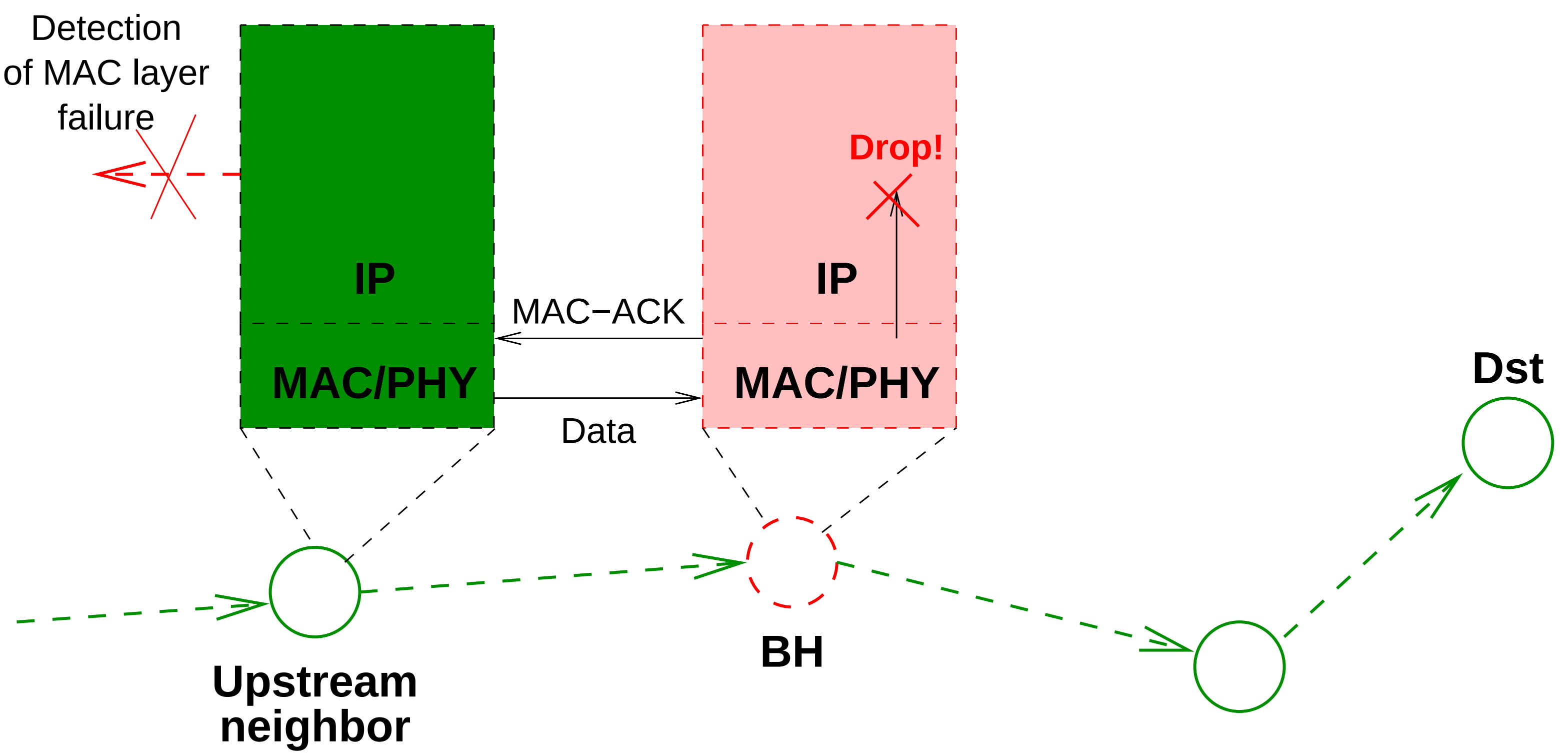
JF interferes with RTT to attenuate the TCP flow!

The BlackHole

For non-responsive / open-loop traffic...

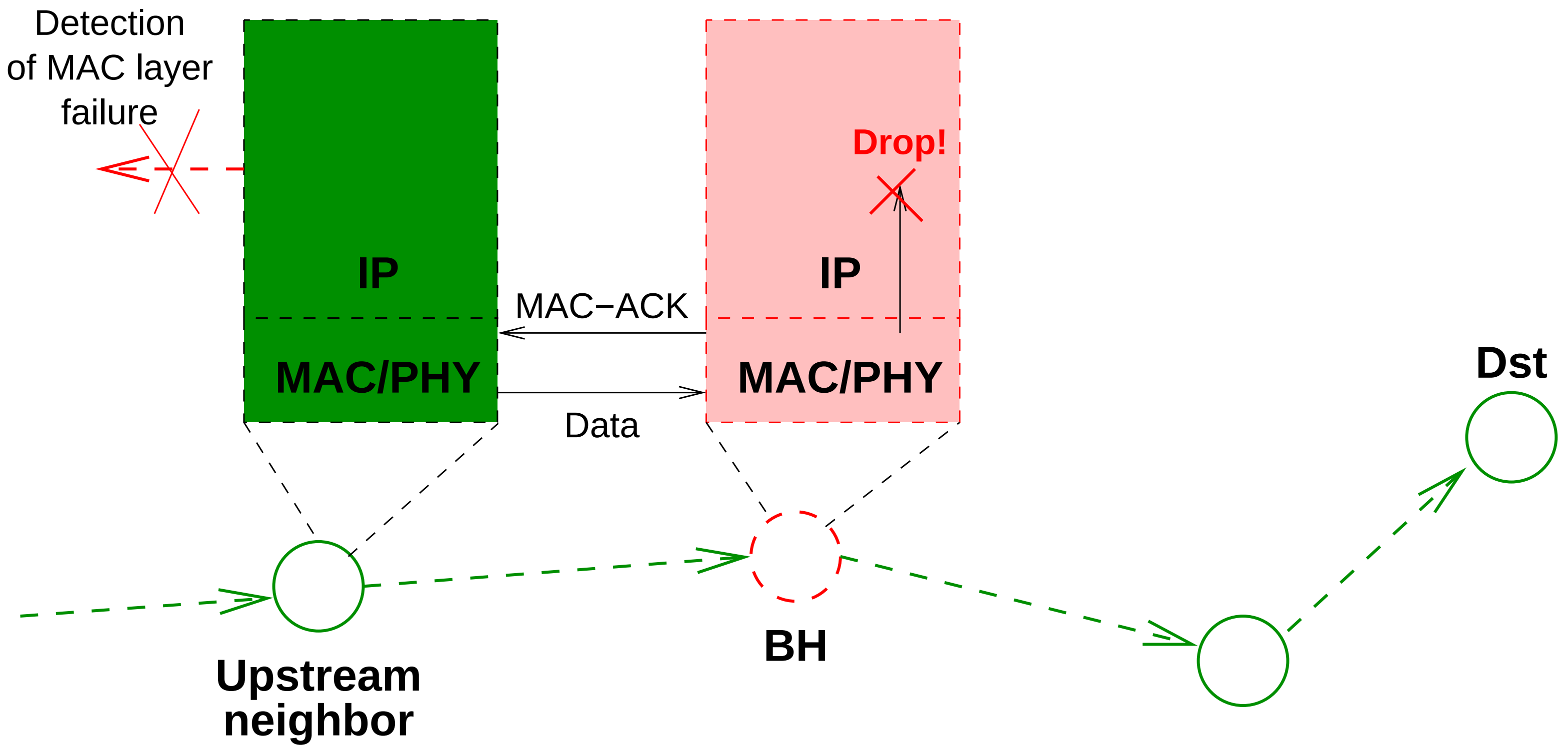
- Passive
- Forwards routing packets
- "Absorbs" all data packets
- Hard to detect...

The BlackHole



MAC ACK avoids immediate diagnosing

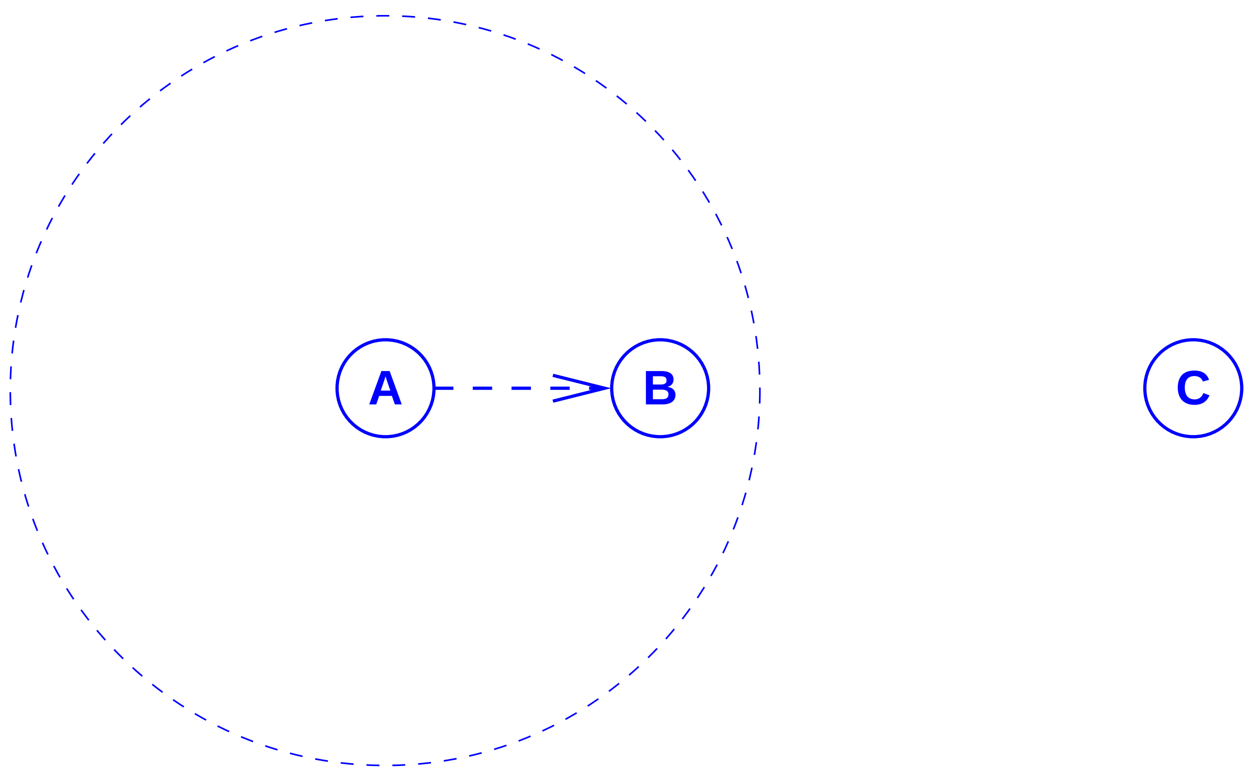
The BlackHole



(zero throughput)

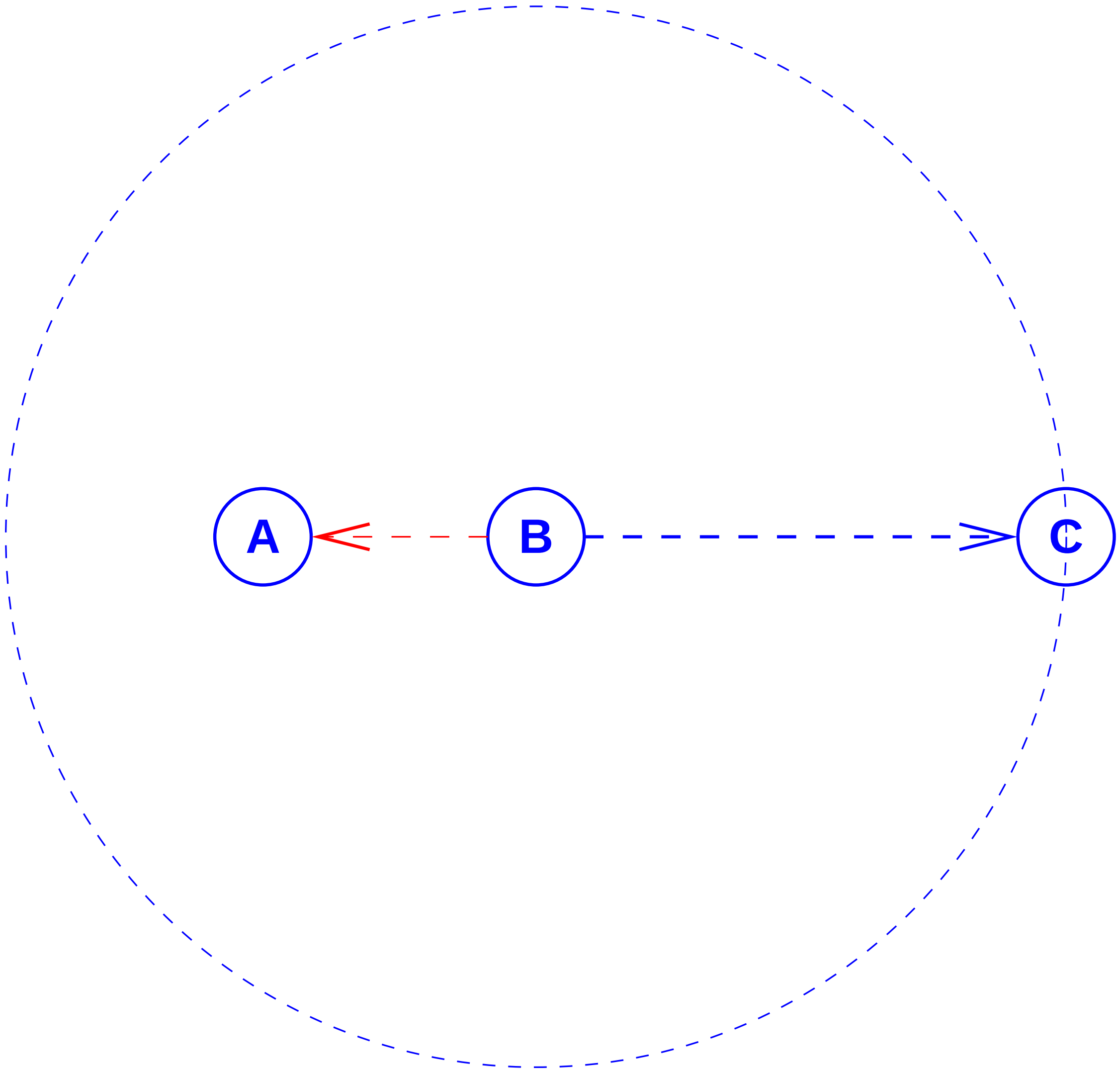
Passive ACK (PACK) [DSR]

A is sending a packet to C via B



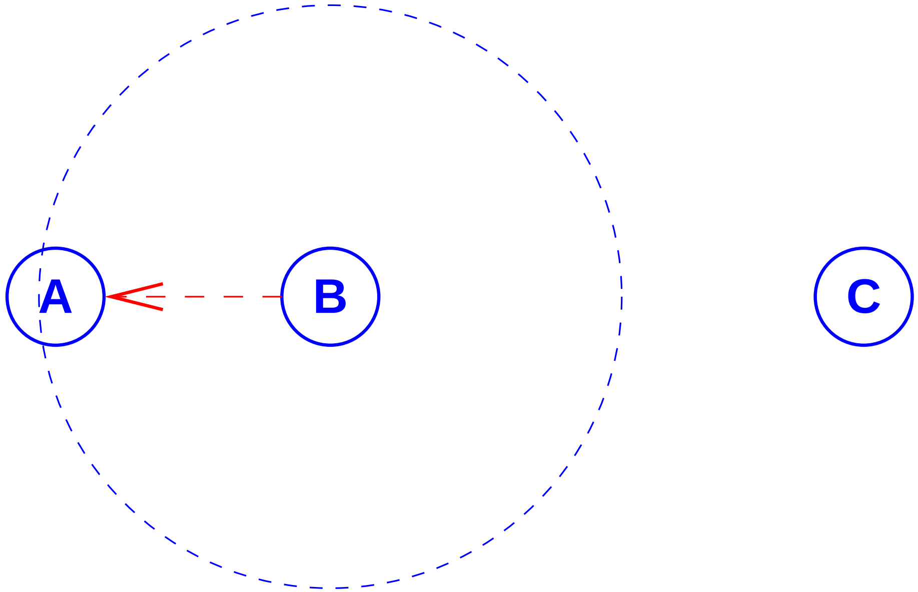
Passive ACK (PACK) [DSR]

A overhears B’s transmission/forward to C



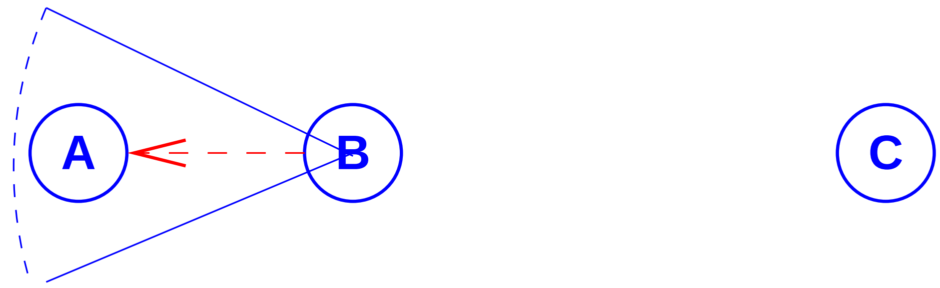
Passive ACK (PACK) [DSR]

PACK can be fooled by low-power transmissions...



Passive ACK (PACK) [DSR]

... Or by using directional antennas!



Outline

- Introduction and system model
- DoS attacks:
  - “Protocol-compliant” attacks: JellyFish
  - BlackHole
- The cost of counter-measures
- Network performance under DoS attacks
- Conclusion

Non-goal: escalating the "arms race"

- Diagnosis are inevitable
  - Locally ?
  - End-to-end ?
- Our goal: how do they perform ?

The cost of counter-measures

Counter-measure parameters:

- Diagnosis time $\rightarrow E(T^n_{diag})$
- (re)Route request $\rightarrow E(T^n_{RR})$

Routing protocol limitations:

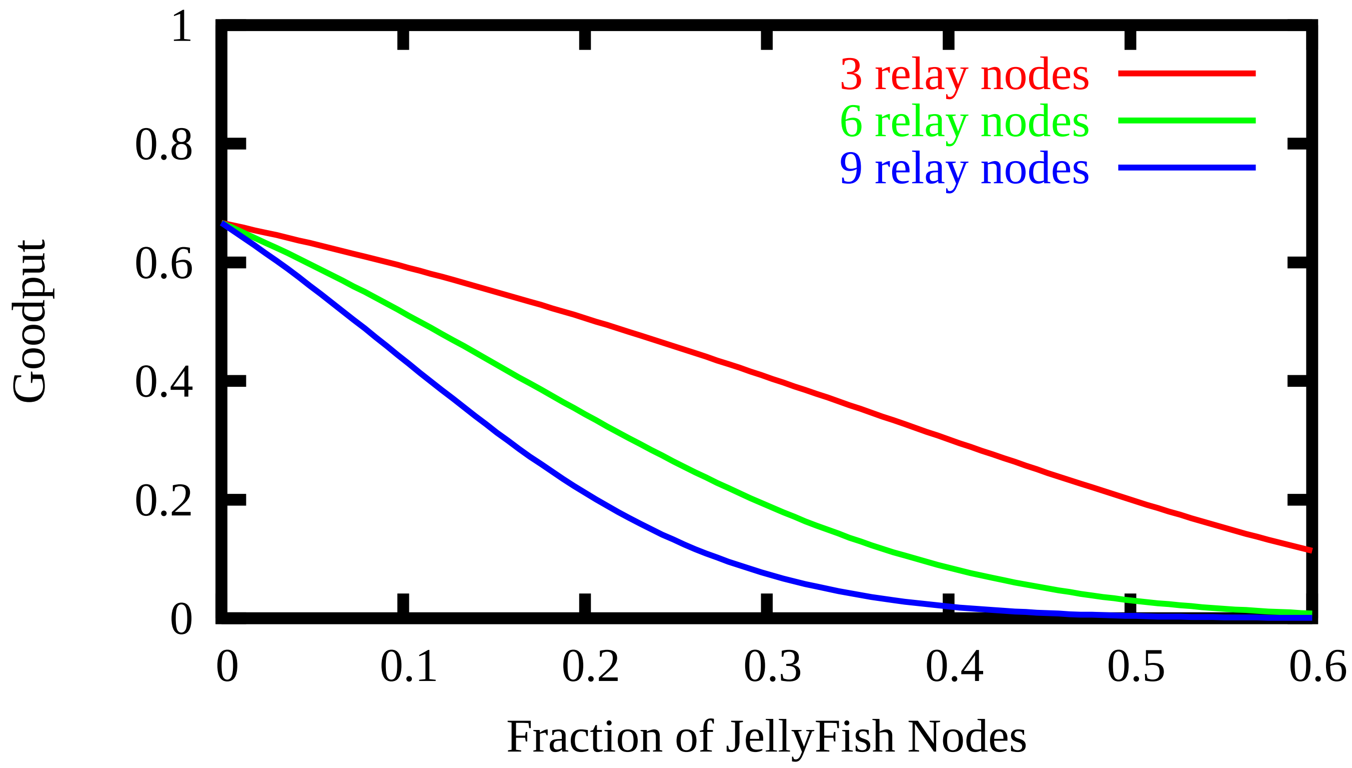
- Rate limiter $\rightarrow E(T^n_{RL})$

Let:

- Flow lifetime $\rightarrow E(T_L)$
- Proportion of JF $\rightarrow p$
- Path length (for recvd. pkts.) $\rightarrow h$

The cost of counter-measures

$$Goodput = \frac{E(T_L)}{E(T_L)+\left(E(T^n_{diag})+E(T^n_{RL})+E(T^n_{RR})\right)(1-p)^{-h}}$$



Diagnosis and rerouting times get magnified by $(1-p)^{-h}$.

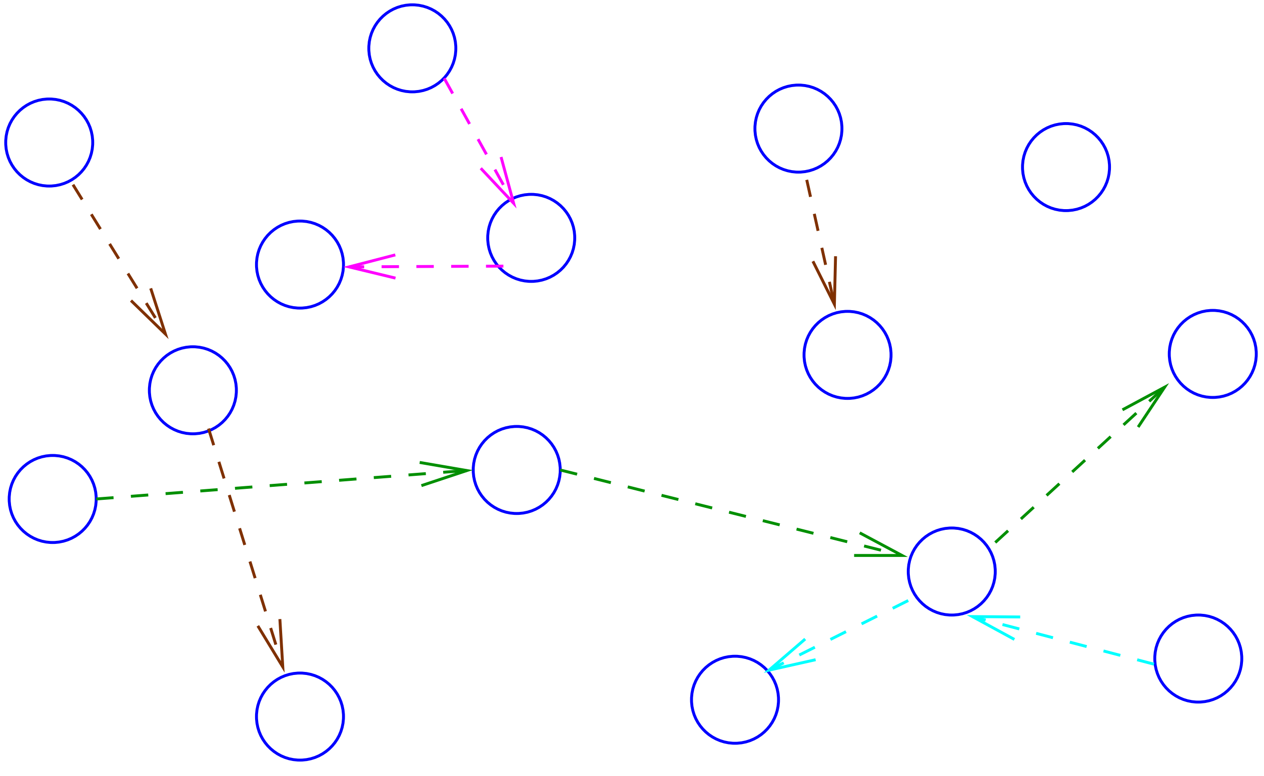
(h: average hop-count, p: proportion of JF)

The cost of counter-measures

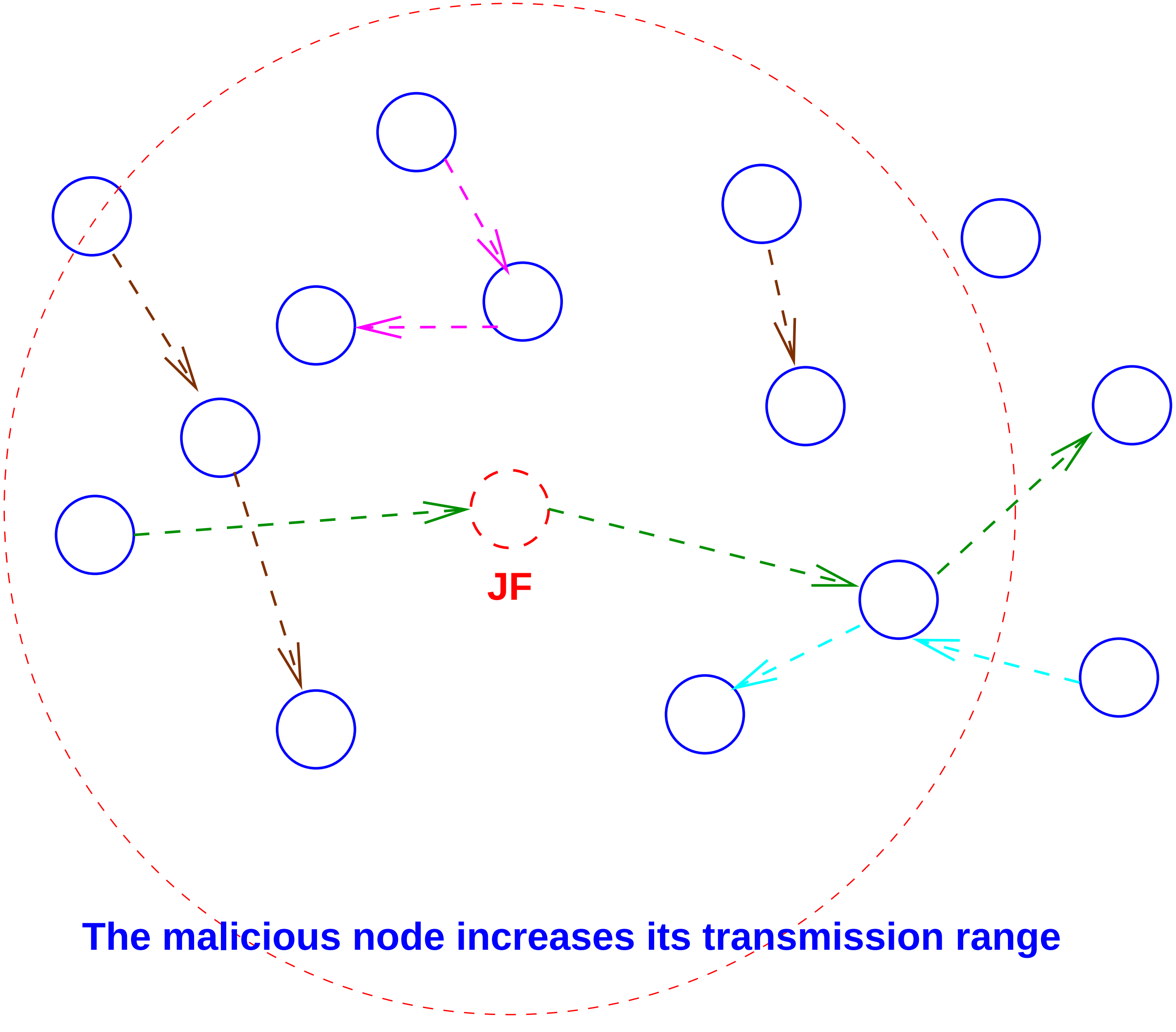
$$Goodput = \frac{E(T_L)}{E(T_L)+\left(E(T^n_{diag})+E(T^n_{RL})+E(T^n_{RR})\right)(1-p)^{-h}}$$

- Mobility
- Network size
- “PACK++”
- Watchdog, path-rater [Marti et al.]
- Identifying “Byzantine nodes” [Awerbuch et al.]
- Reputation systems [Buchegger et al., Michiardi et al.]
- Rushing attack [Hu et al.]

Rushing attack [Hu et al.]

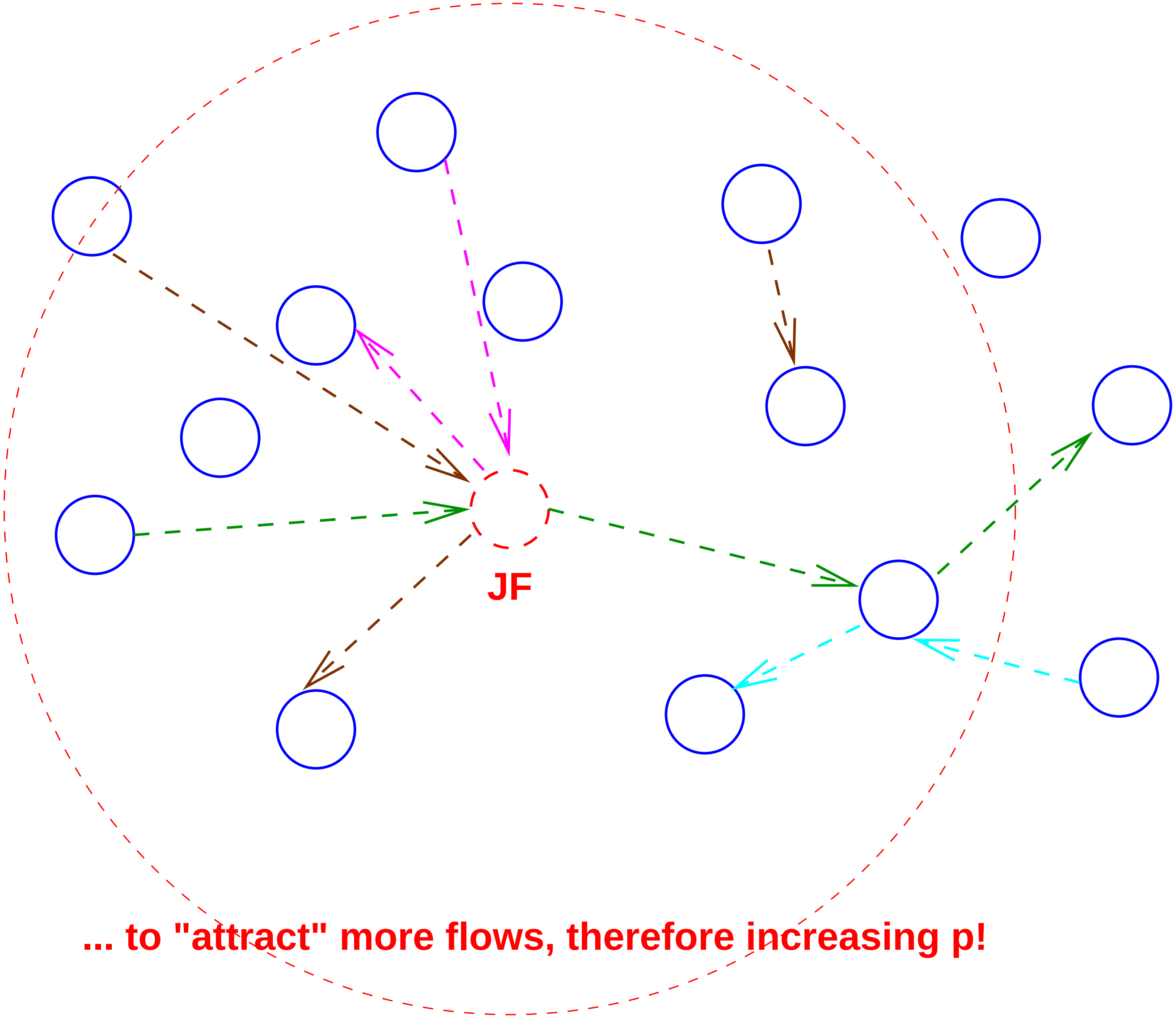


Rushing attack [Hu et al.]



The malicious node increases its transmission range

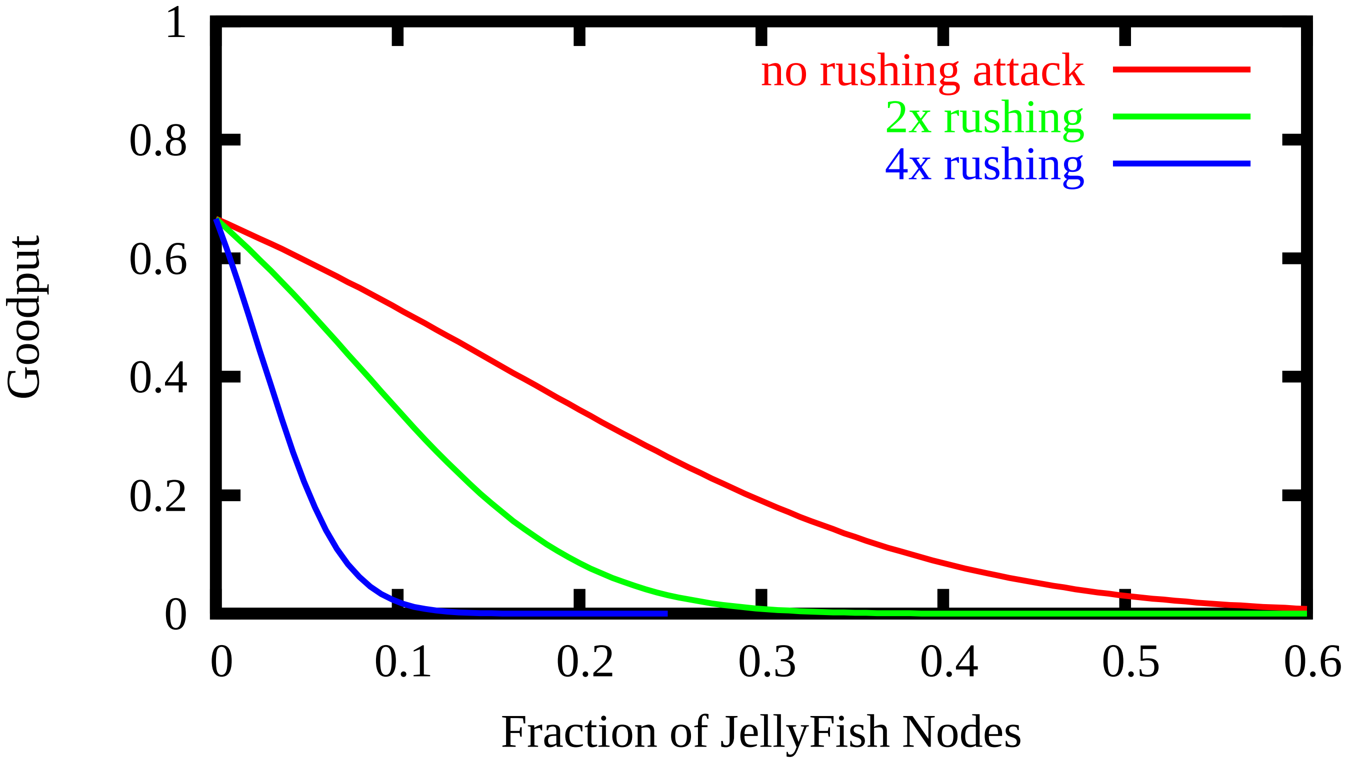
Rushing attack [Hu et al.]



... to "attract" more flows, therefore increasing p!

Rushing attack [Hu et al.]

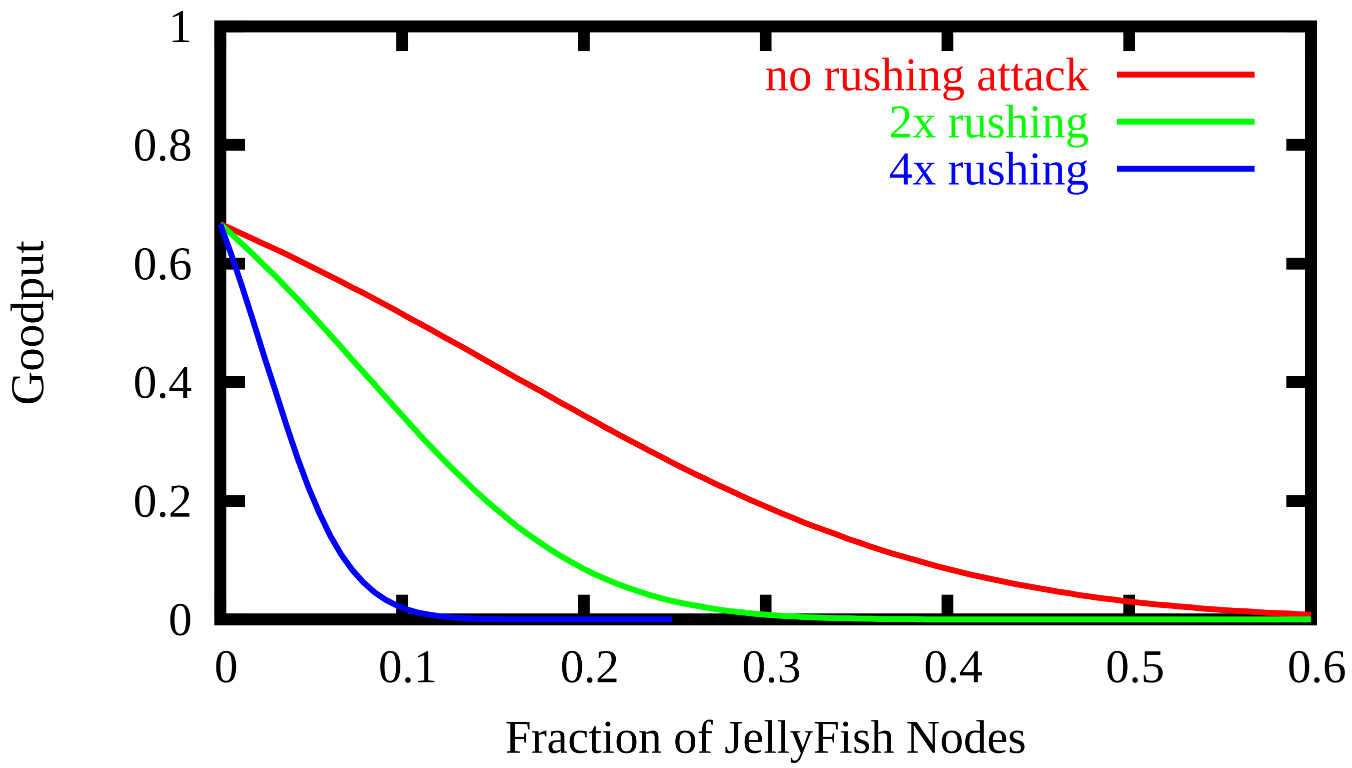
$$Goodput = \frac{E(T_L)}{E(T_L)+\left(E(T^n_{diag})+E(T^n_{RL})+E(T^n_{RR})\right)(1-p)^{-h}}$$



The rushing attack makes things even worse, exponentiating the effect with hop length!
(h: average hop-count, p: proportion of JF)

Rushing attack [Hu et al.]

$$Goodput = \frac{E(T_L)}{E(T_L)+\left(E(T^n_{diag})+E(T^n_{RL})+E(T^n_{RR})\right)(1-p)^{-h}}$$



The goodput collapses under 10% of attackers!

Outline

- Introduction and system model
- DoS attacks:
  - "Protocol-compliant" attacks: JellyFish
  - BlackHole
- The cost of counter-measures
- Network performance under DoS attacks
- Conclusion

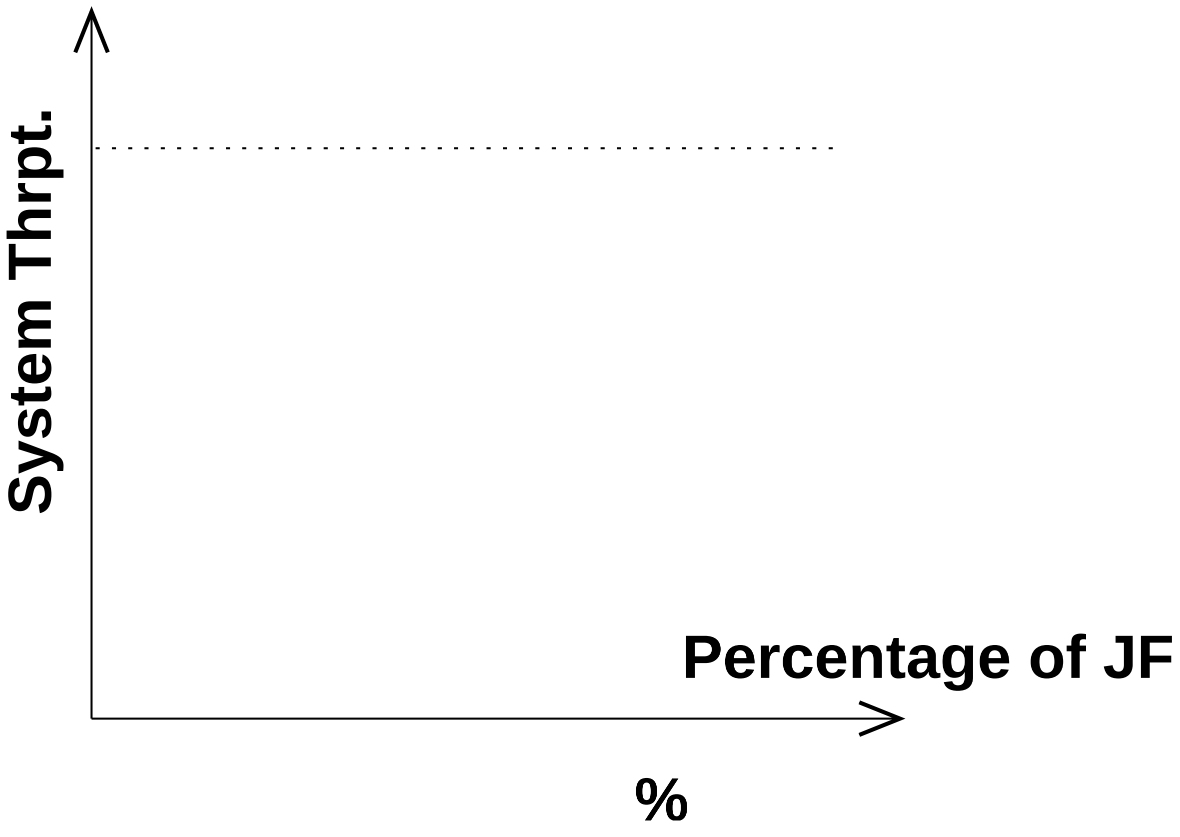
What about the network resistance?

Simulation setup:

- 2000m $\times$ 2000m topology
- 200 mobile nodes
- Velocity: 0 to 10m/s
- Average pause time: 10s
- 50 UDP flows: 500B packets / 5s, (800b/s)
- Clear non-fading channel
- Simulation: 100s warmup + 500s simulation
- (50 simulations, 18 topologies) / point, 95% conf. intervals

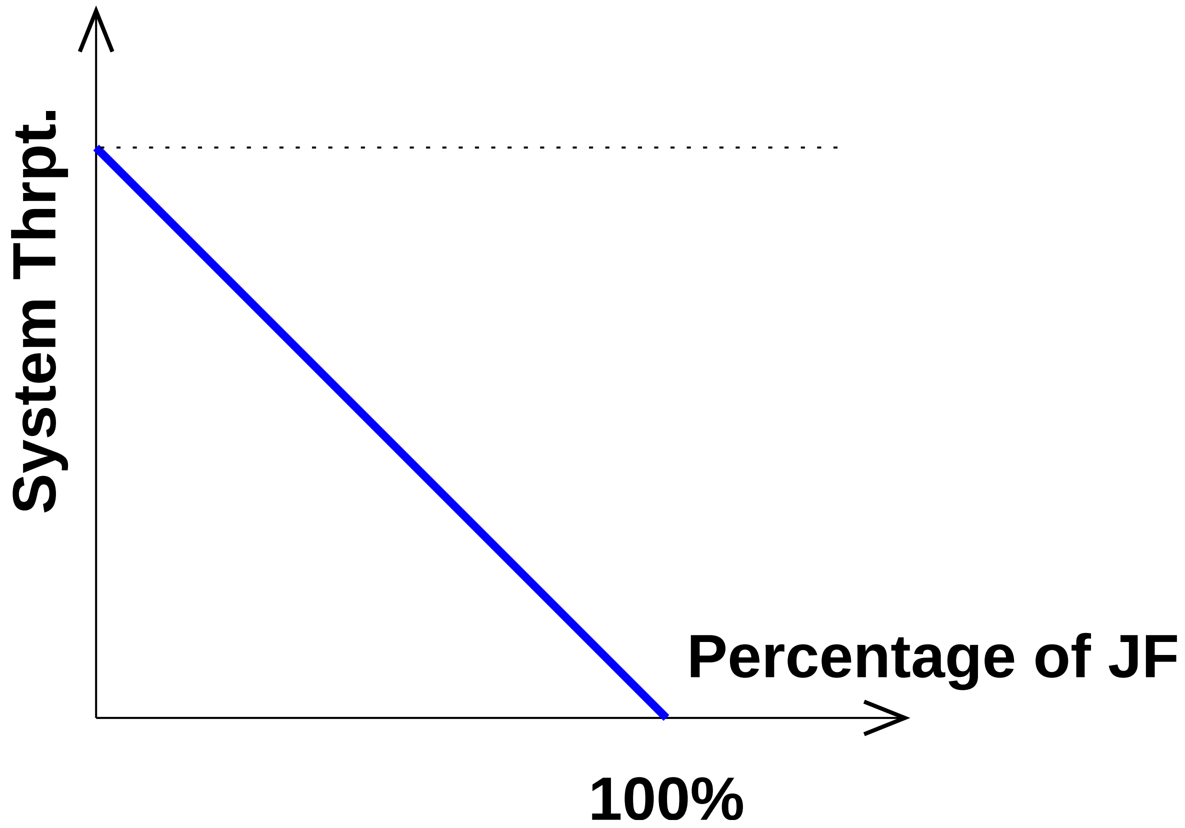
What about the network resistance?

System-wide total throughput = sum of E-2-E throughputs:



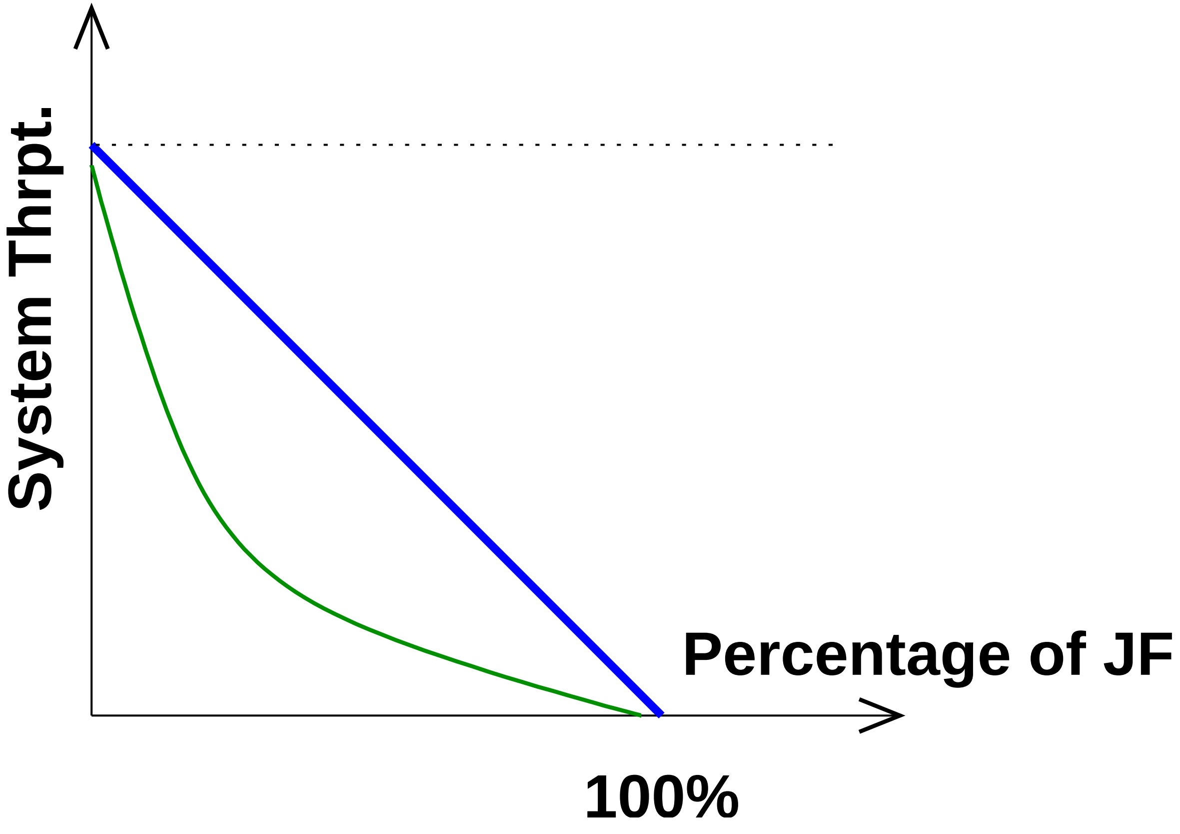
What about the network resistance?

System-wide total throughput = sum of E-2-E throughputs:



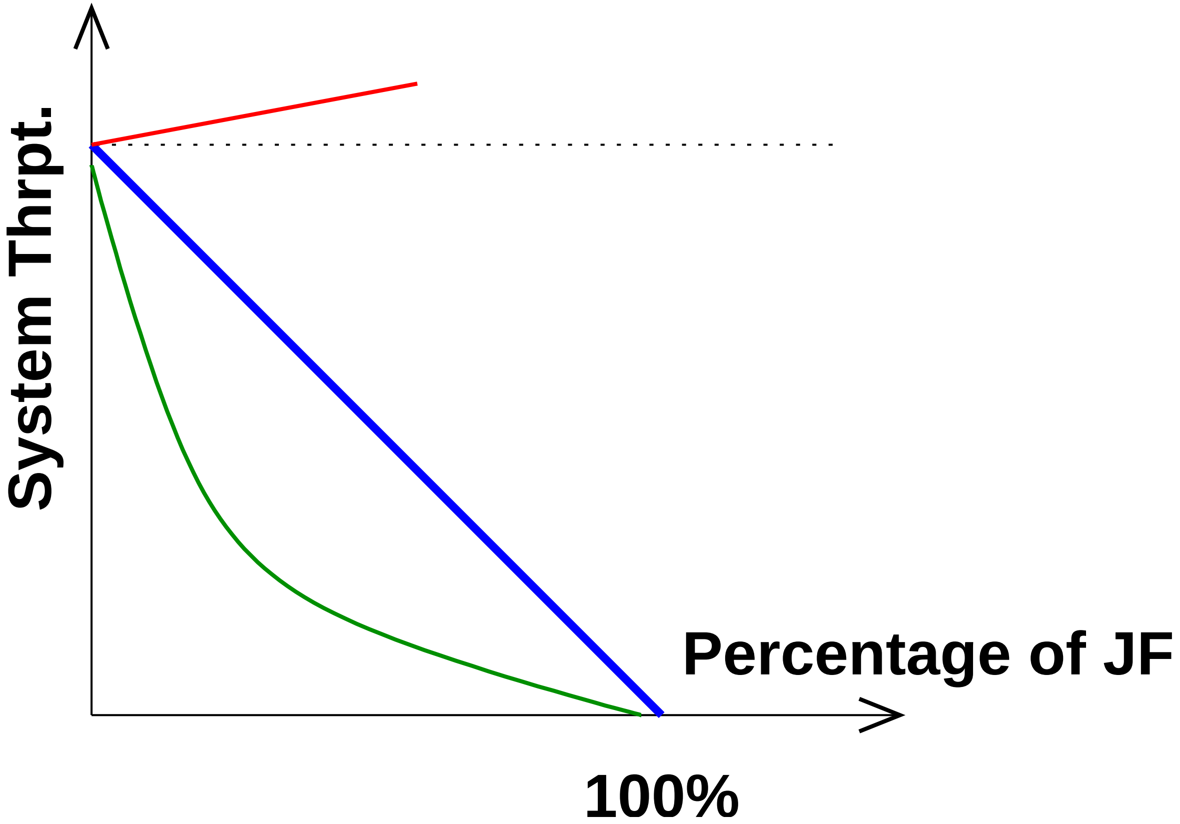
What about the network resistance?

System-wide total throughput = sum of E-2-E throughputs:



What about the network resistance?

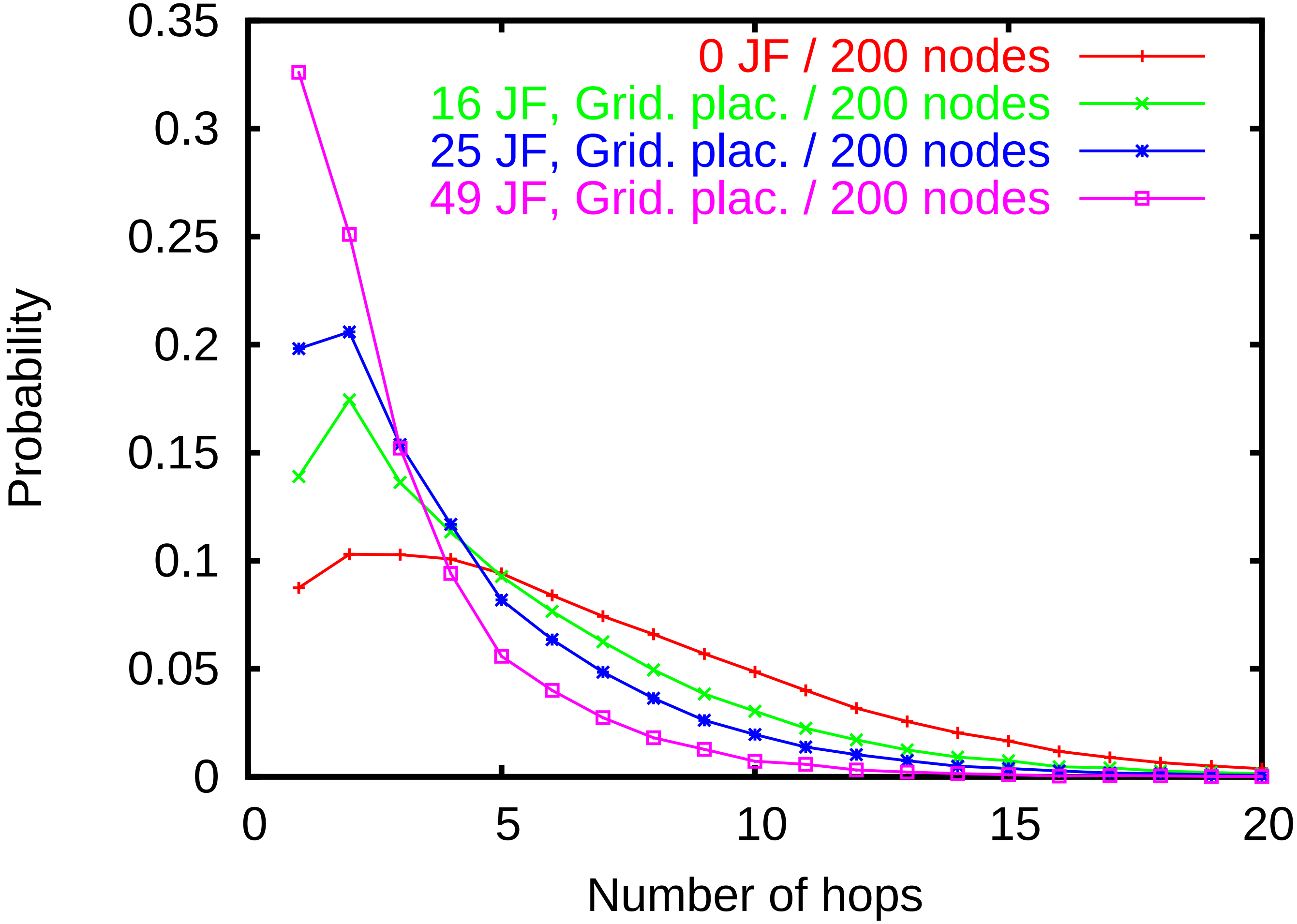
System-wide total throughput = sum of E-2-E throughputs:



again ?

DoS increases the capacity of ad-hoc networks!

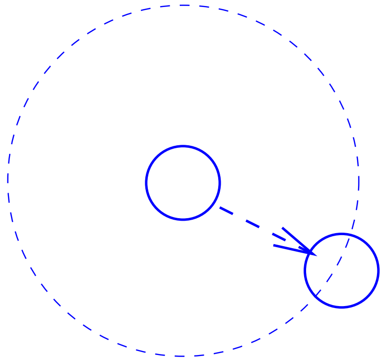
Path length for received packets



After DoS: $\longrightarrow$ Long paths are extinguished...

$\longrightarrow$ Short paths will survive...

Path length for received packets

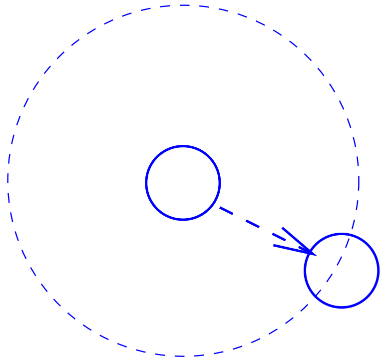


- End−to−End throughput = channel capacity
- Less interference
- More channel reuse

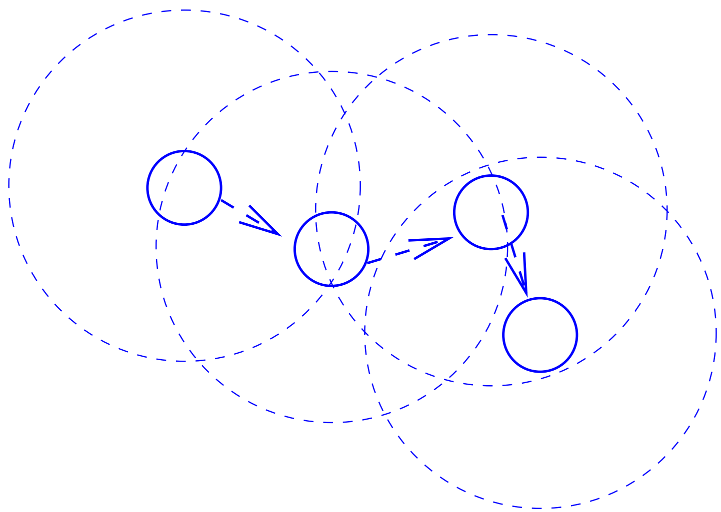
After DoS: $\longrightarrow$ Long paths are extinguished...

$\longrightarrow$ Short paths will survive...

Path length for received packets



- End−to−End throughput = channel capacity
- Less interference
- More channel reuse

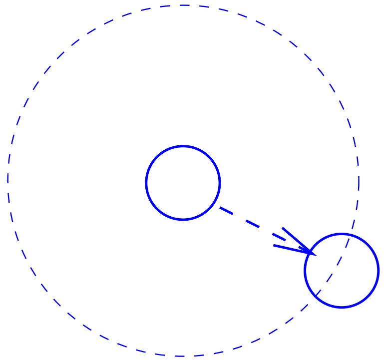


- E2E throughput = ch. capacity / 3
- More interference
- Less channel reuse

After DoS: $\longrightarrow$ Long paths are extinguished...

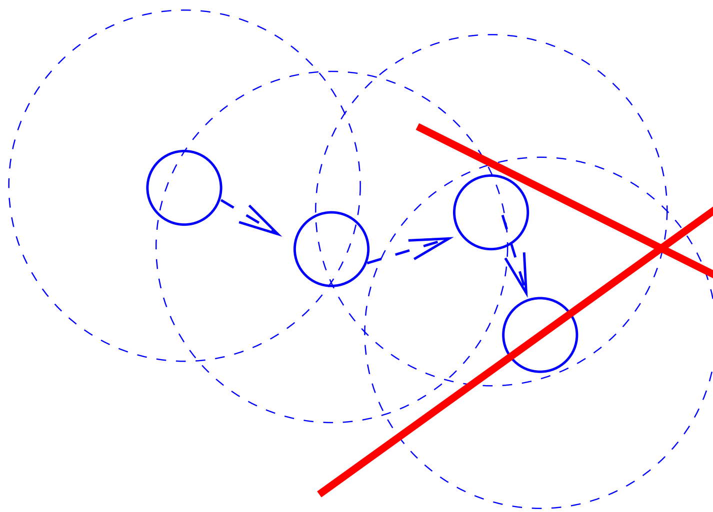
$\longrightarrow$ Short paths will survive...

Path length for received packets



– End–to–End throughput = channel capacity
– Less interference
– More channel reuse

System throughput maximizer

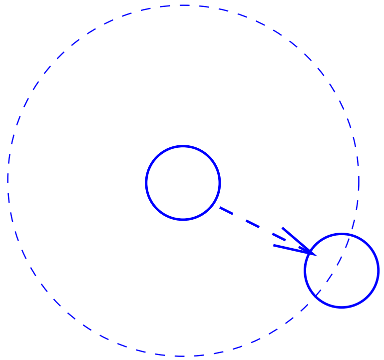


– E2E throughput = ch. capacity / 3
– More interference
– Less channel reuse

After DoS: $\longrightarrow$ Long paths are extinguished...

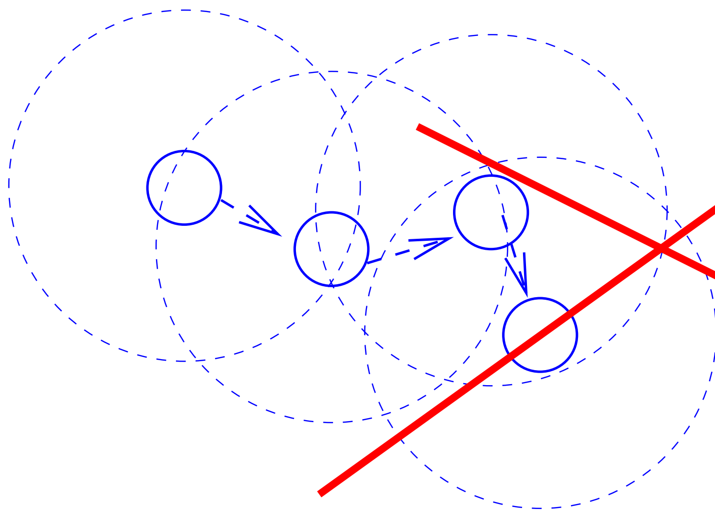
$\longrightarrow$ Short paths will survive...

Path length for received packets



- End−to−End throughput = channel capacity
- Less interference
- More channel reuse

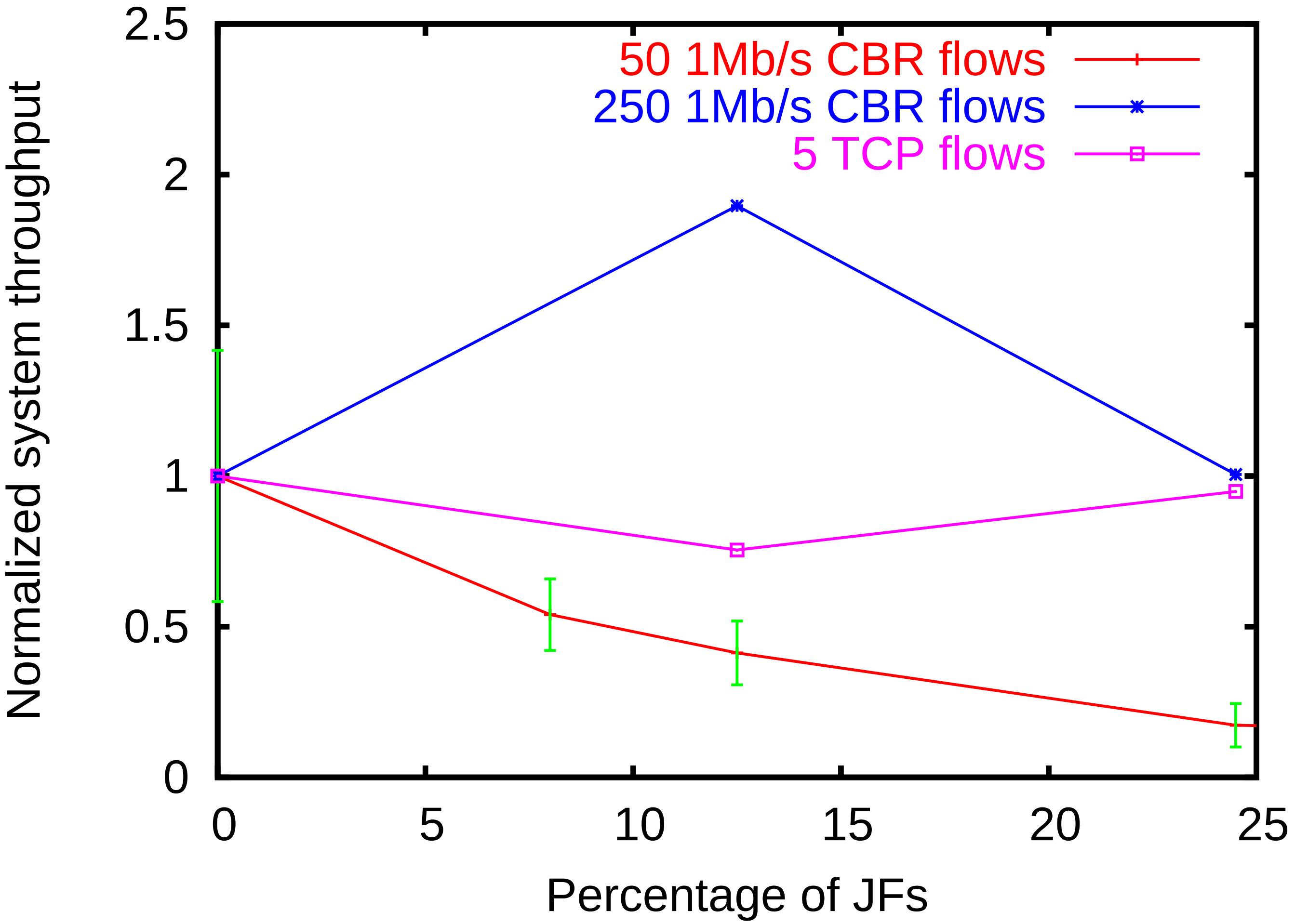
System throughput maximizer



- E2E throughput = ch. capacity / 3
- More interference
- Less channel reuse

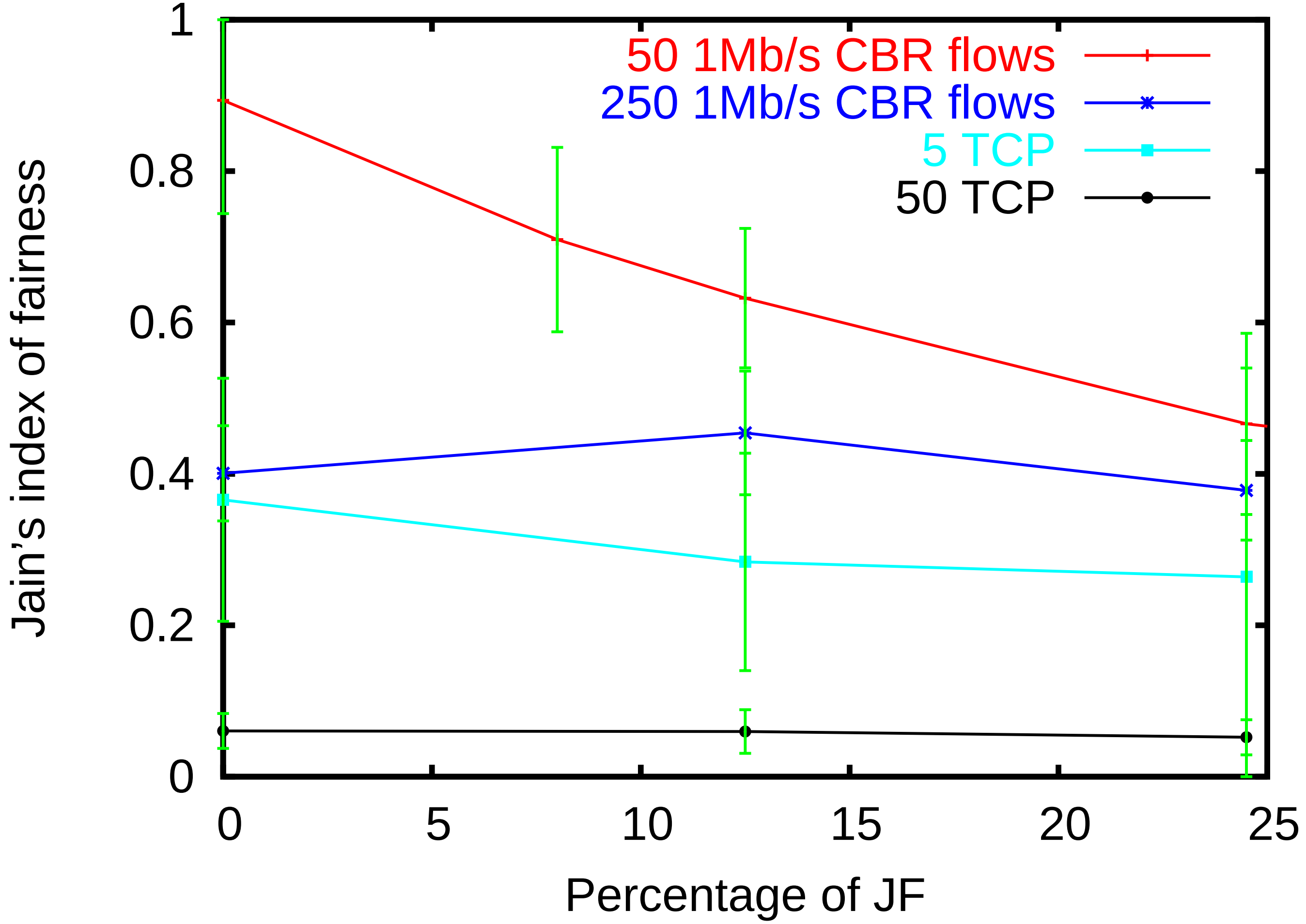
and this is what JF and BlackHoles are doing!

System throughput



System throughput often increases after DoS!

BUT!



System becomes unfair, in favor of short paths.

After DoS...

- Network gets severely partitioned
- Short flows survive
- Long flows are attenuated
- Aggregated system throughput may increase!

More in the paper...

We analyze the performance of the system when varying the:

- Offered load
- Network size
- Node density
- Node mobility
- JF placement strategy

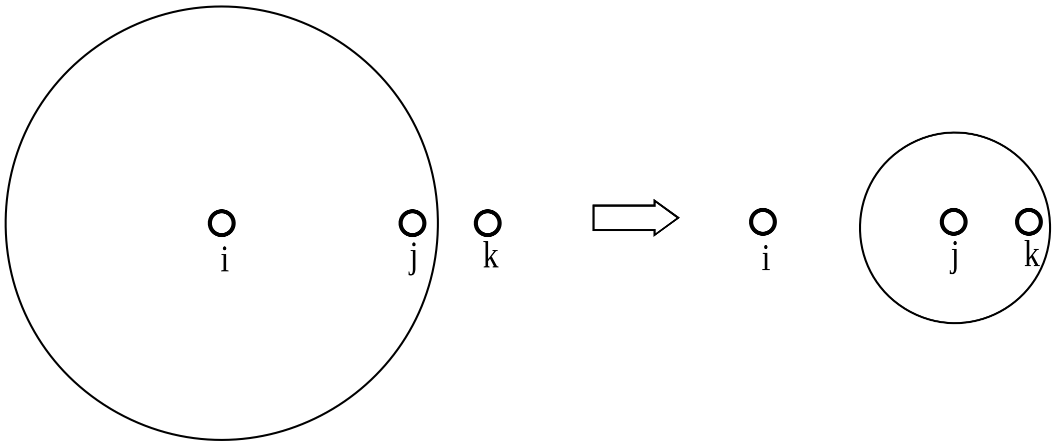
Outline

- Introduction and system model
- DoS attacks:
  - “Protocol-compliant” attacks: JellyFish
  - BlackHole
- The cost of counter-measures
- Network performance under DoS attacks
- Conclusion

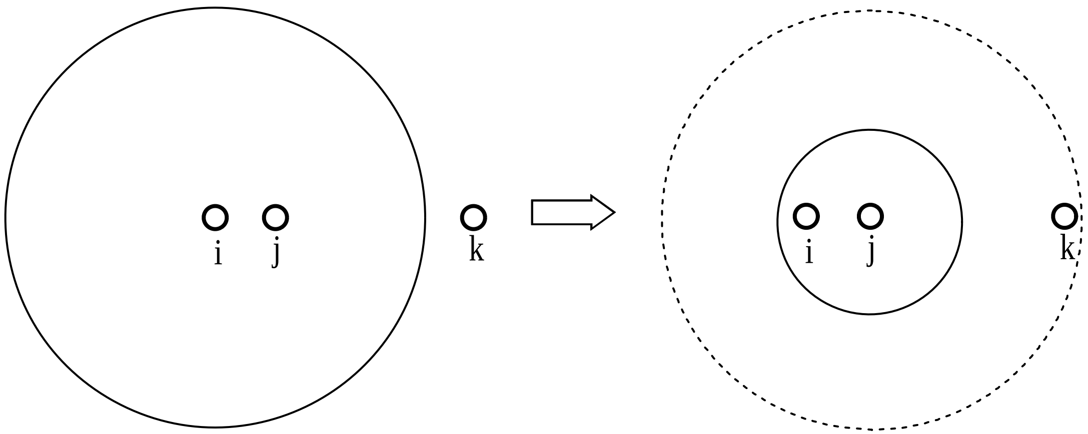
Conclusion

- TCP collapses with malicious:
  - Dropping, reordering, jitter ...
- More generally, all closed-loop mechanisms are vulnerable to malicious tampering
- “Protocol-compliance” makes defense more problematic
- First paper to quantify DoS effects on ad-hoc networks:
  - DoS increases capacity! BUT!
  - Network gets partitioned
  - Fairness decreases
  - → System throughput, alone, is not enough to measure DoS impacts

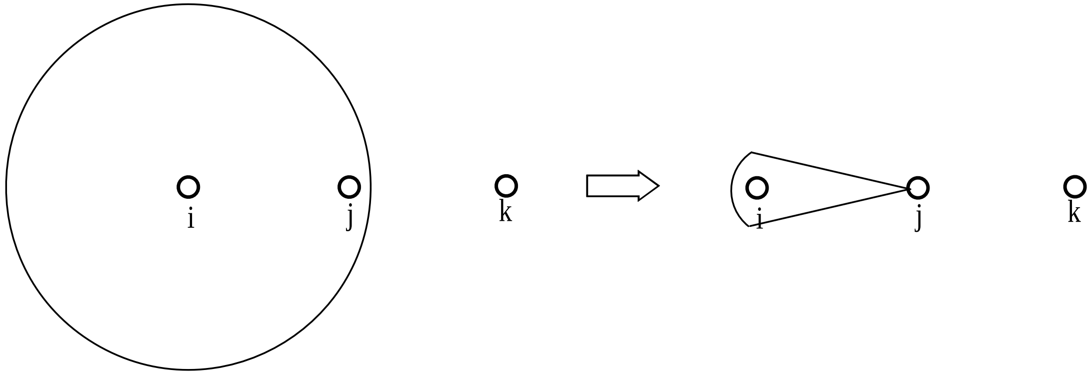
PACK power



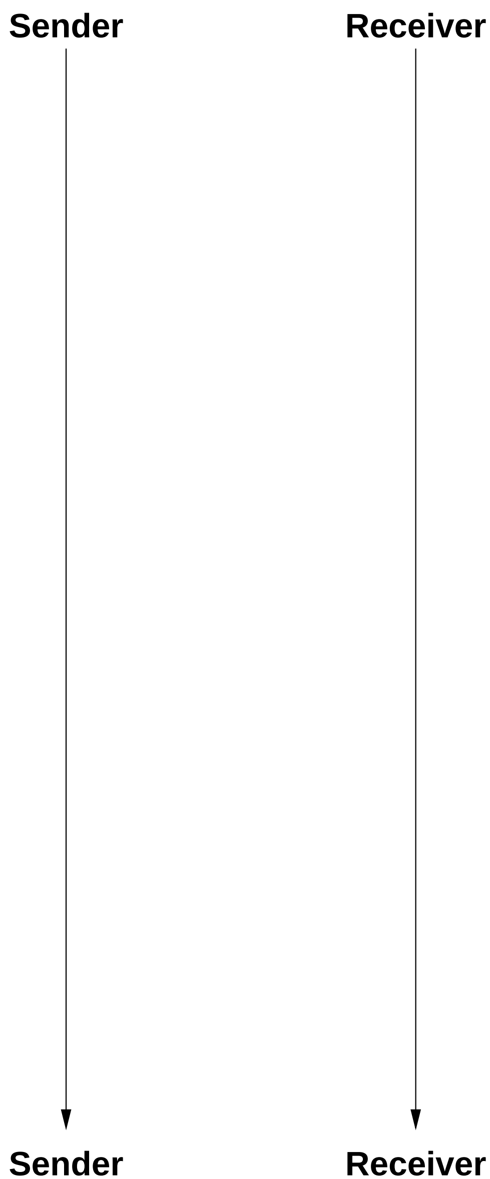
PACK fool



PACK directional antenna



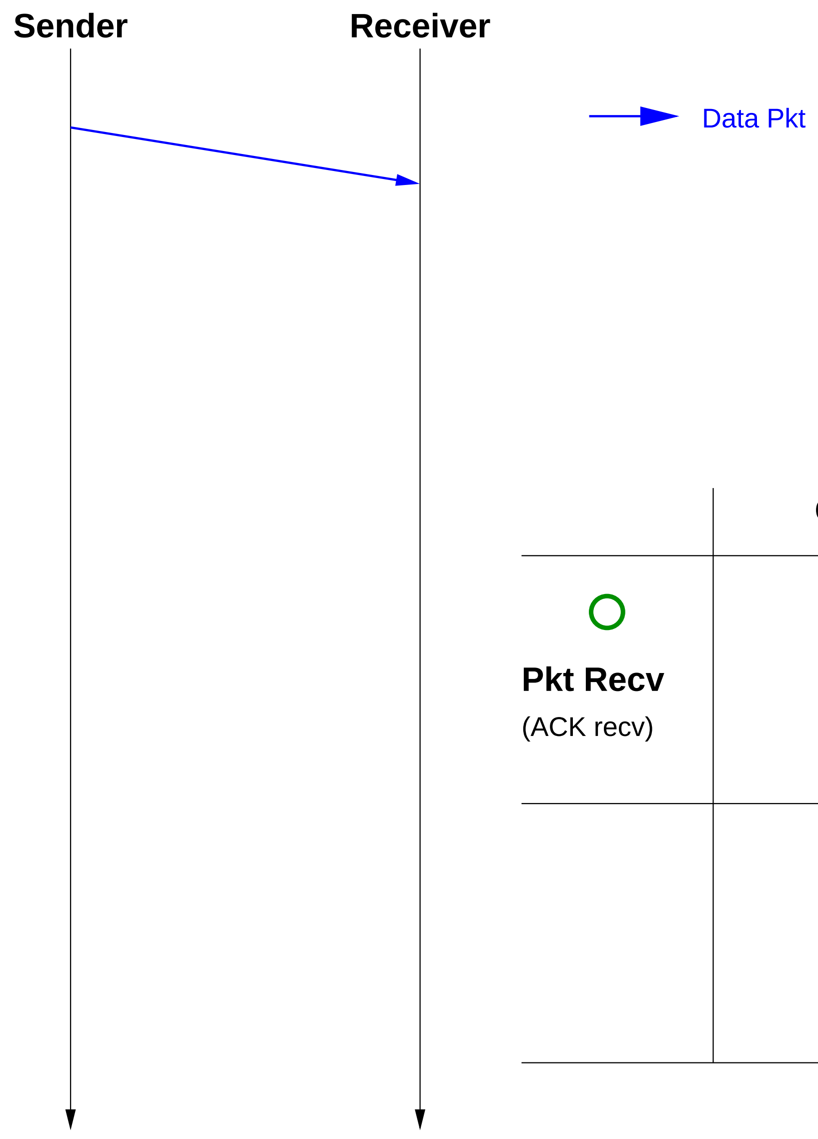
Reminder on TCP



	Cong. Window	Timers
Pkt Recv (ACK recv)		

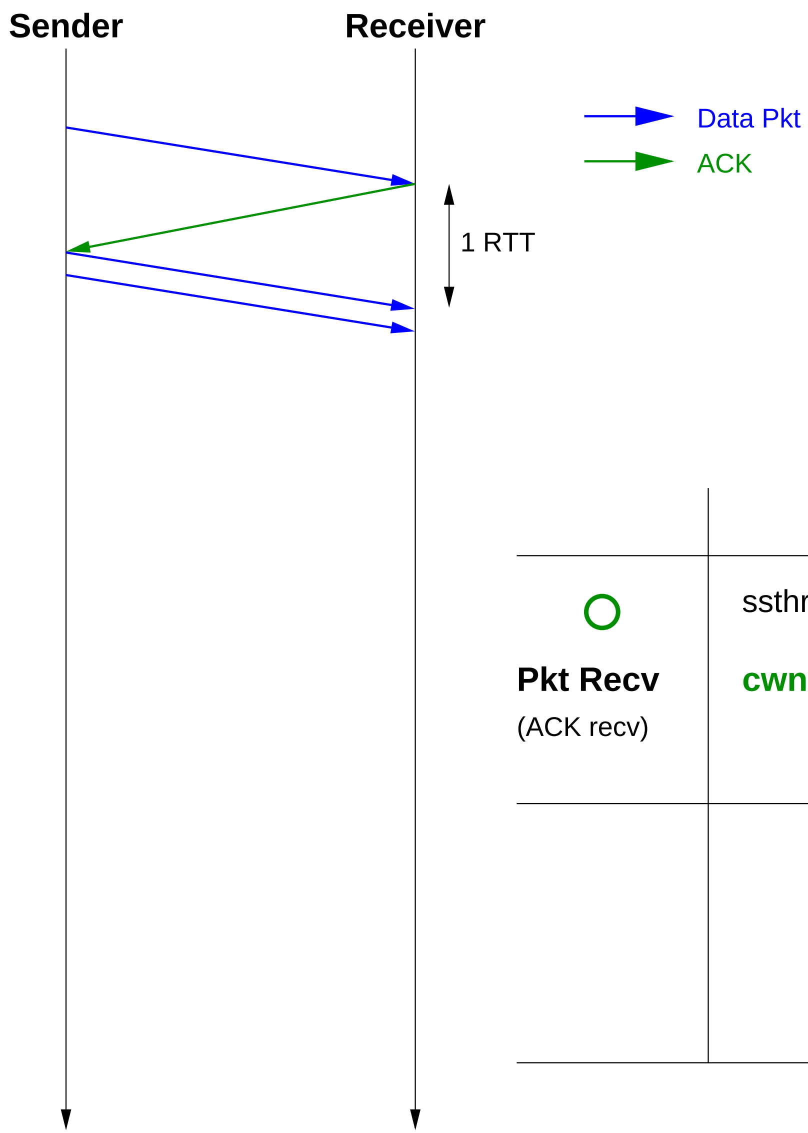
Slow Start (SS)

Reminder on TCP



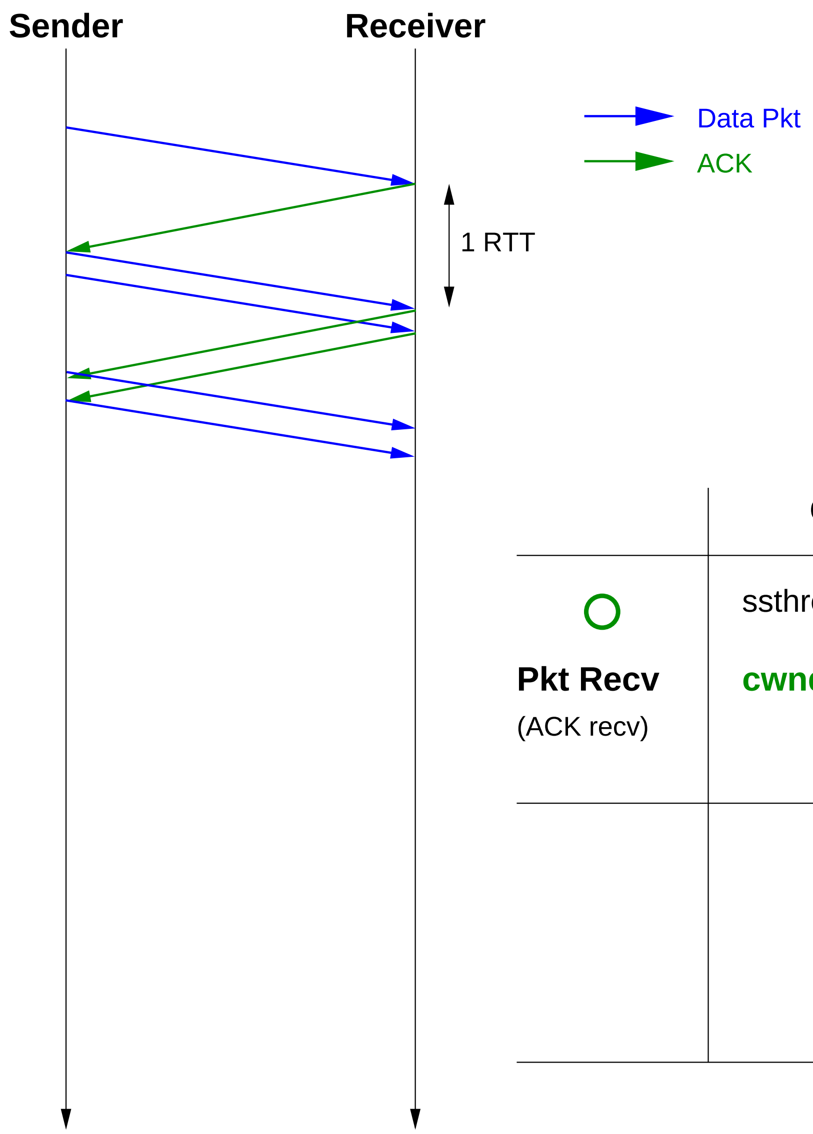
	Cong. Window	Timers
Pkt Recv (ACK recv)		

Reminder on TCP



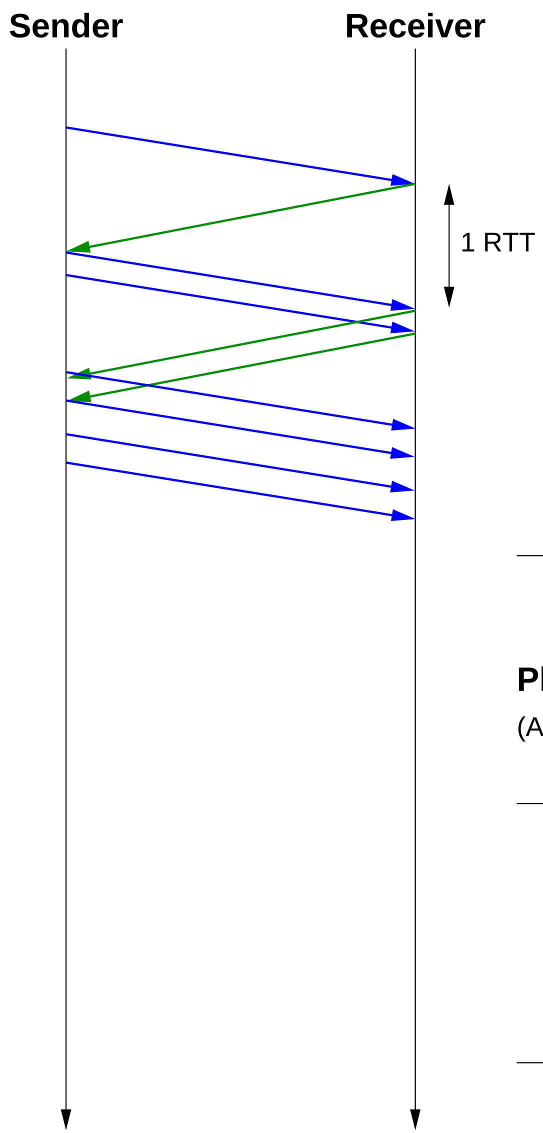
	Cong. Window	Timers
Pkt Recv (ACK recv)	ssthresh **cwnd += 1 (SS)**	RTTVAR = (1−b) RTTVAR + b |SRTT−RTT| SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO ,** SRTT+ max(G, 4 RTTVAR))

Reminder on TCP



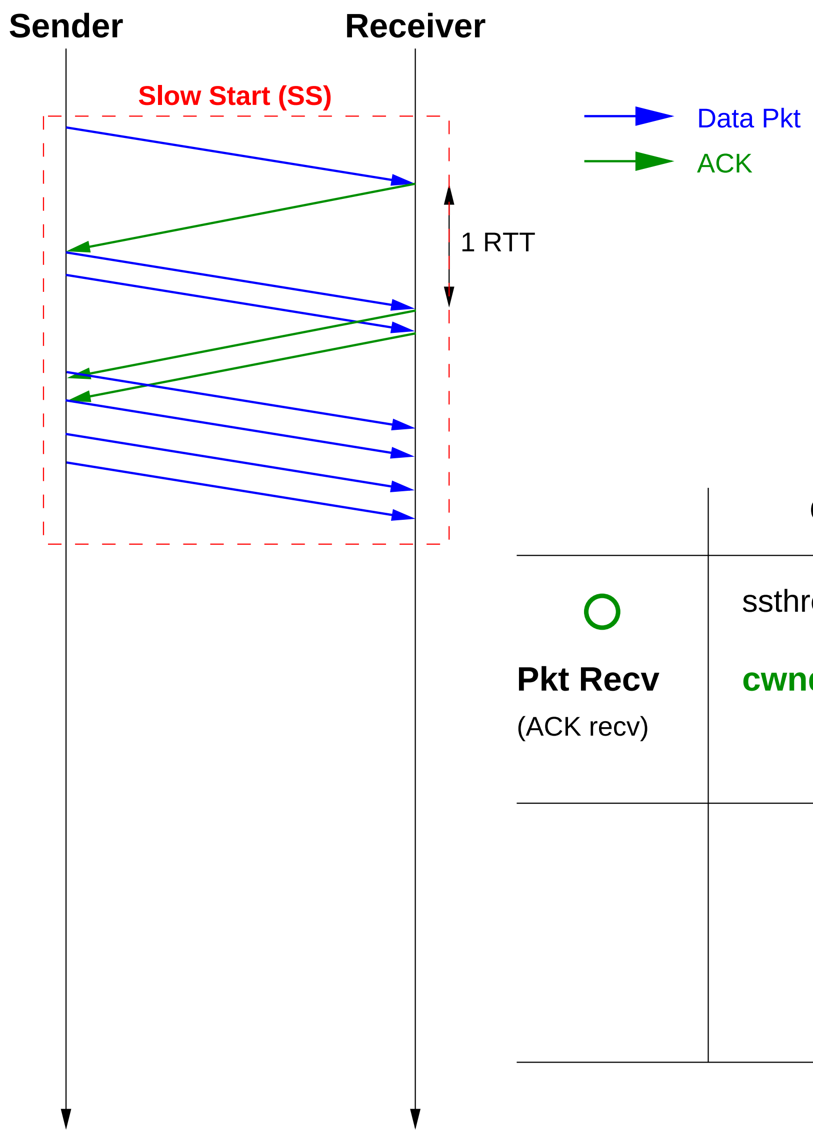
	Cong. Window	Timers
Pkt Recv (ACK recv)	ssthresh **cwnd += 1 (SS)**	RTTVAR = (1−b) RTTVAR + b |SRTT−RTT| SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO ,** SRTT+ max(G, 4 RTTVAR))

Reminder on TCP



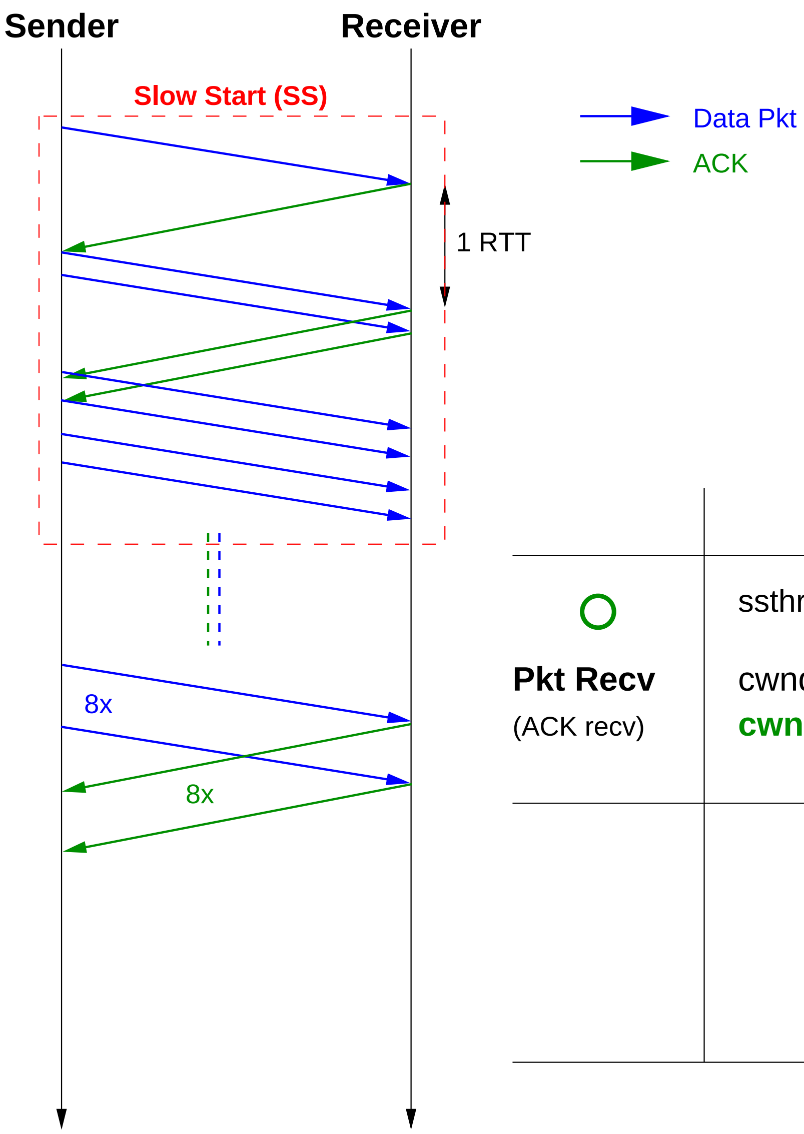
	Cong. Window	Timers
○ **Pkt Recv** (ACK recv)	ssthresh **cwnd += 1 (SS)**	RTTVAR = (1−b) RTTVAR + b |SRTT−RTT| SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO ,** SRTT+ max(G, 4 RTTVAR))

Reminder on TCP



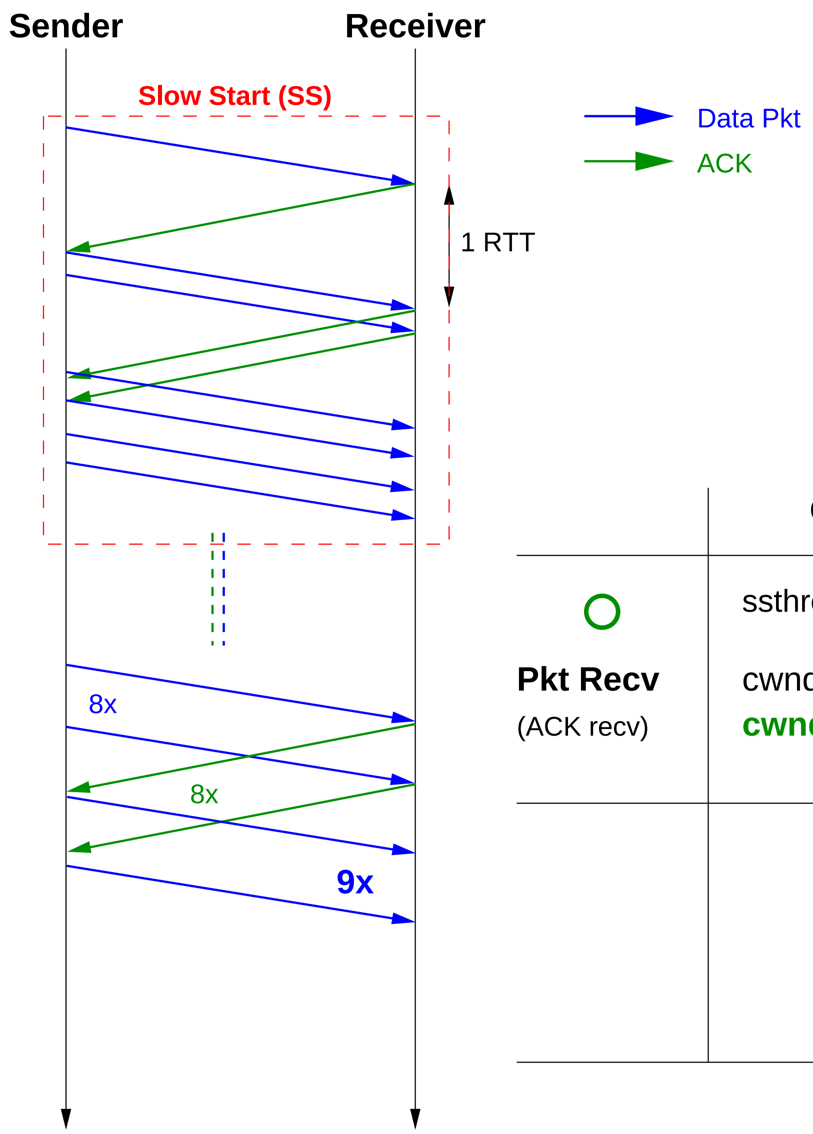
	Cong. Window	Timers
Pkt Recv (ACK recv)	ssthresh **cwnd += 1 (SS)**	RTTVAR = (1−b) RTTVAR + b |SRTT−RTT| SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO ,** SRTT+ max(G, 4 RTTVAR))

Reminder on TCP



	Cong. Window	Timers
Pkt Recv (ACK recv)	ssthresh cwnd += 1 (SS) **cwnd += 1/cwnd (CA)**	RTTVAR = (1−b) RTTVAR + b |SRTT−RTT| SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO ,** SRTT+ max(G, 4 RTTVAR))

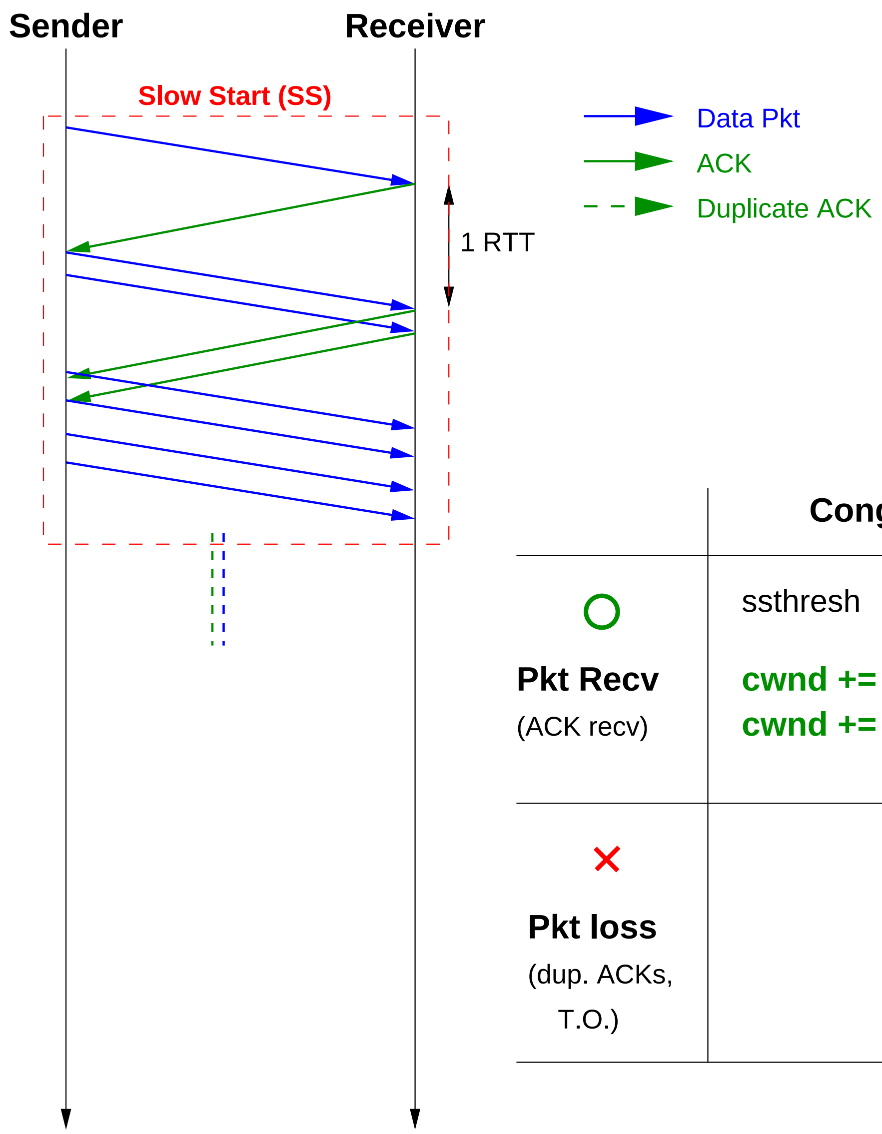
Reminder on TCP



	Cong. Window	Timers
Pkt Recv (ACK recv)	ssthresh cwnd += 1 (SS) **cwnd += 1/cwnd (CA)**	RTTVAR = (1−b) RTTVAR + b |SRTT−RTT| SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO ,** SRTT+ max(G, 4 RTTVAR))

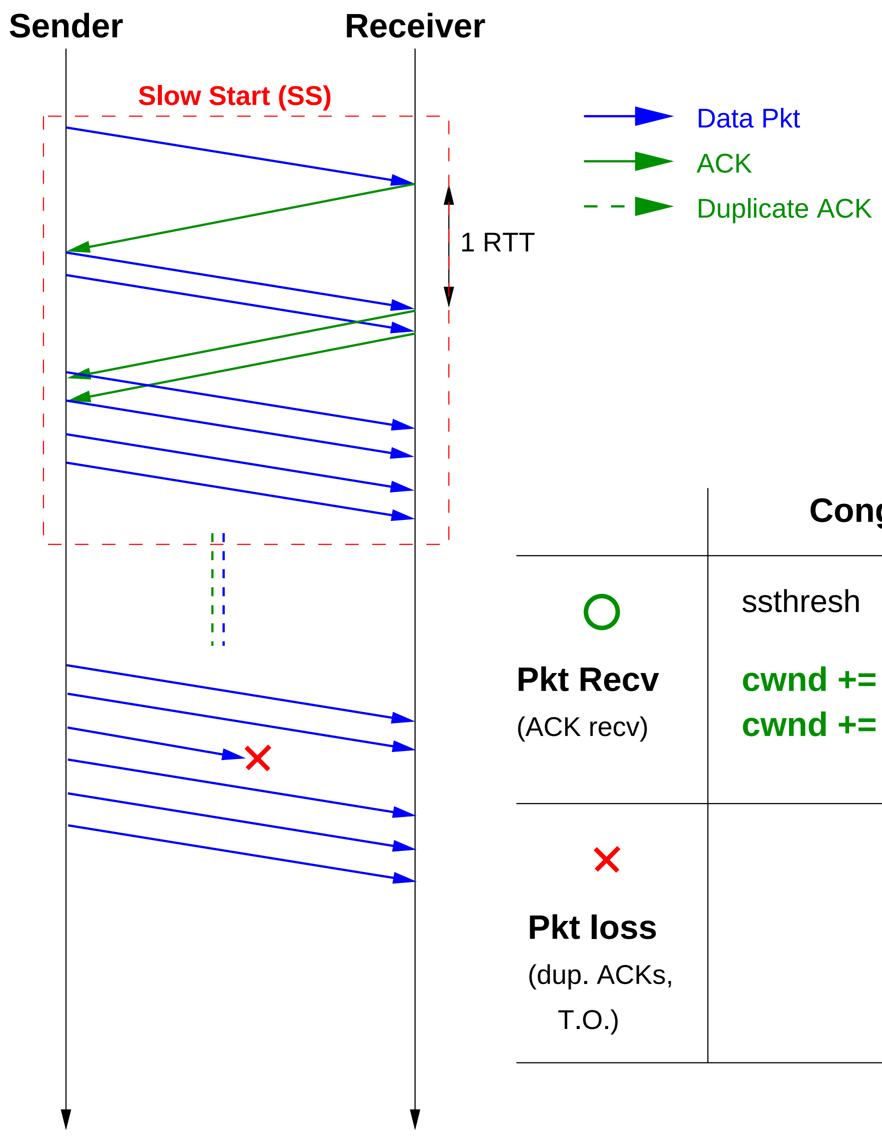
Reminder on TCP

Reminder on TCP



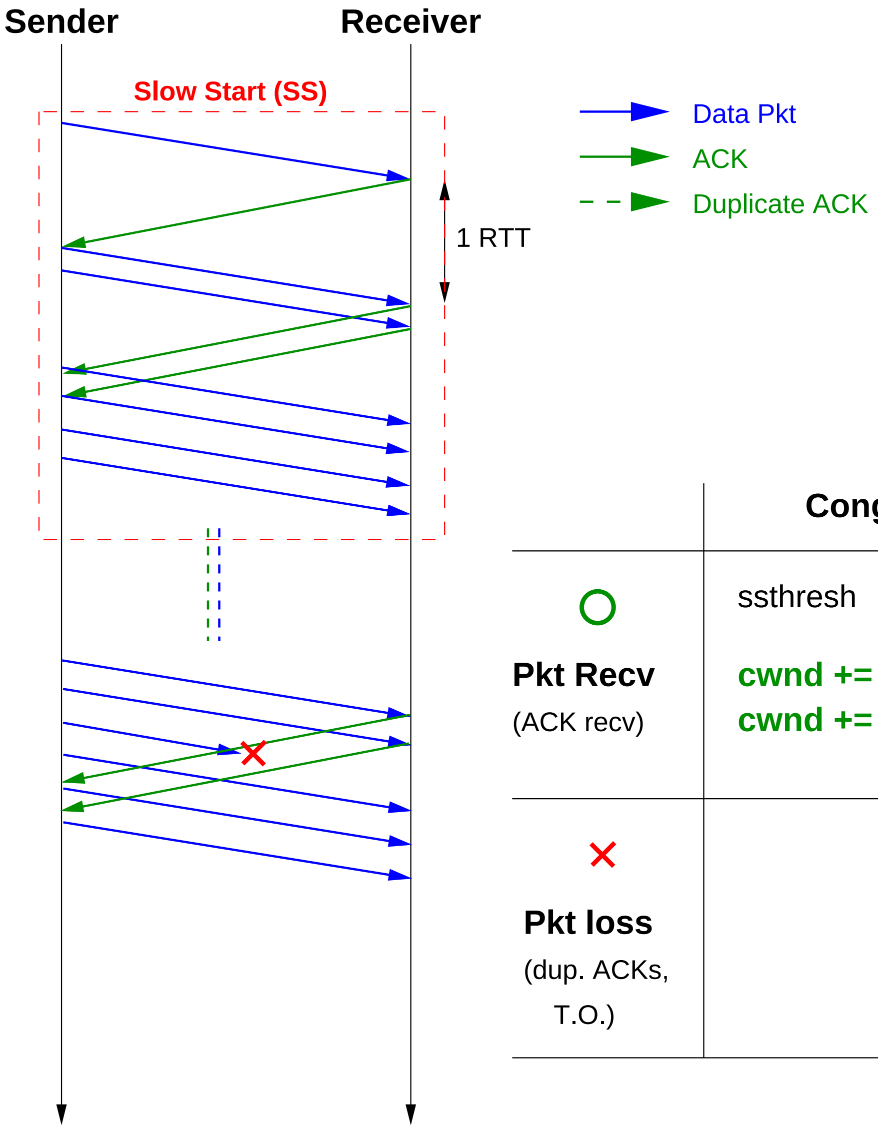
	Cong. Window	Timers
Pkt Recv (ACK recv)	ssthresh **cwnd += 1 (SS)** **cwnd += 1/cwnd (CA)**	RTTVAR = (1−b) RTTVAR + b |SRTT−RTT| SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO ,** SRTT+ max(G, 4 RTTVAR))
Pkt loss (dup. ACKs, T.O.)		

Reminder on TCP



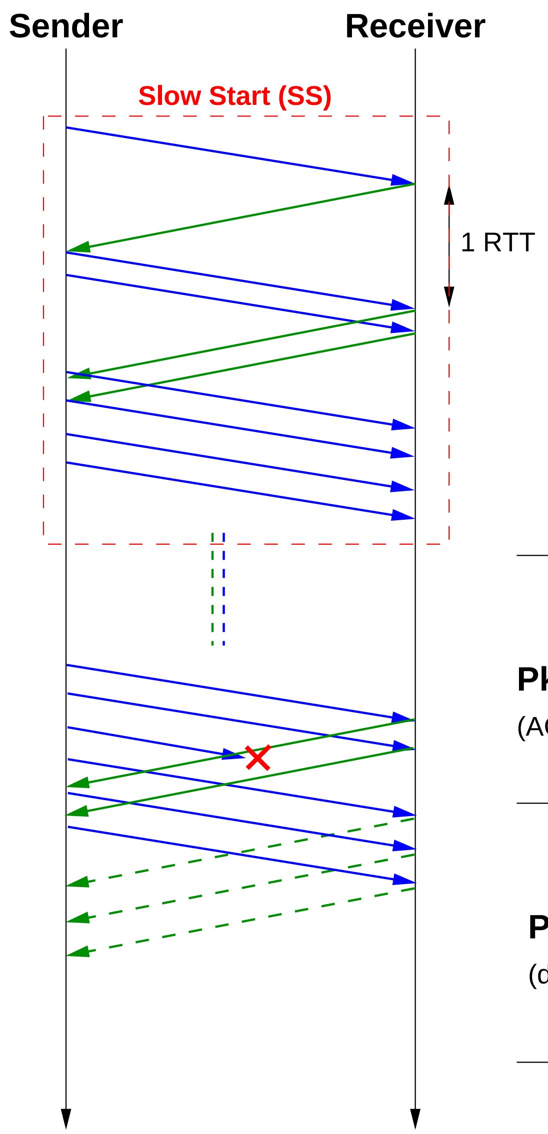
	Cong. Window	Timers
Pkt Recv (ACK recv)	ssthresh **cwnd += 1 (SS)** **cwnd += 1/cwnd (CA)**	RTTVAR = (1−b) RTTVAR + b $\lvert$SRTT−RTT$\rvert$ SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO ,** SRTT+ max(G, 4 RTTVAR))
Pkt loss (dup. ACKs, T.O.)		

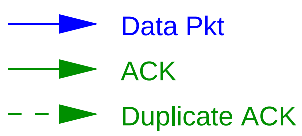
Reminder on TCP



	Cong. Window	Timers
Pkt Recv (ACK recv)	ssthresh **cwnd += 1 (SS)** **cwnd += 1/cwnd (CA)**	RTTVAR = (1−b) RTTVAR + b |SRTT−RTT| SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO , SRTT+ max(G, 4 RTTVAR))**
Pkt loss (dup. ACKs, T.O.)		

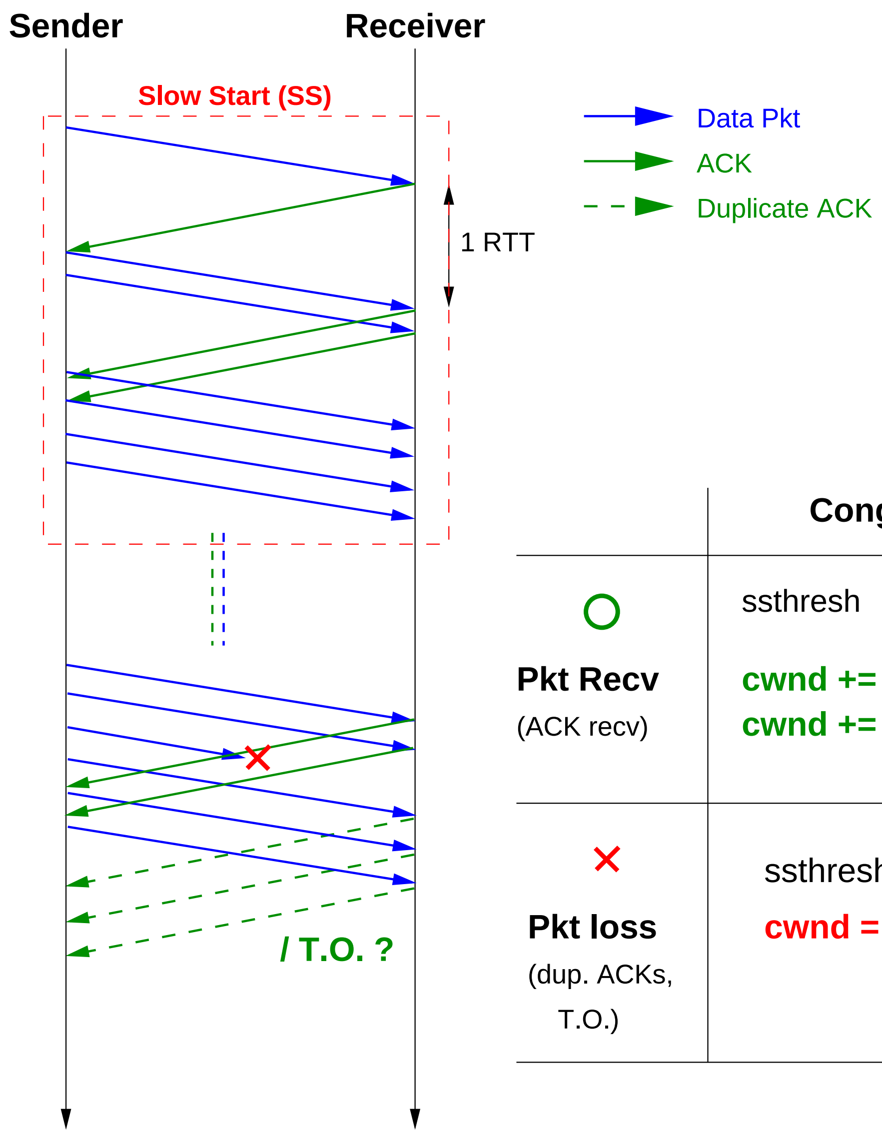
Reminder on TCP





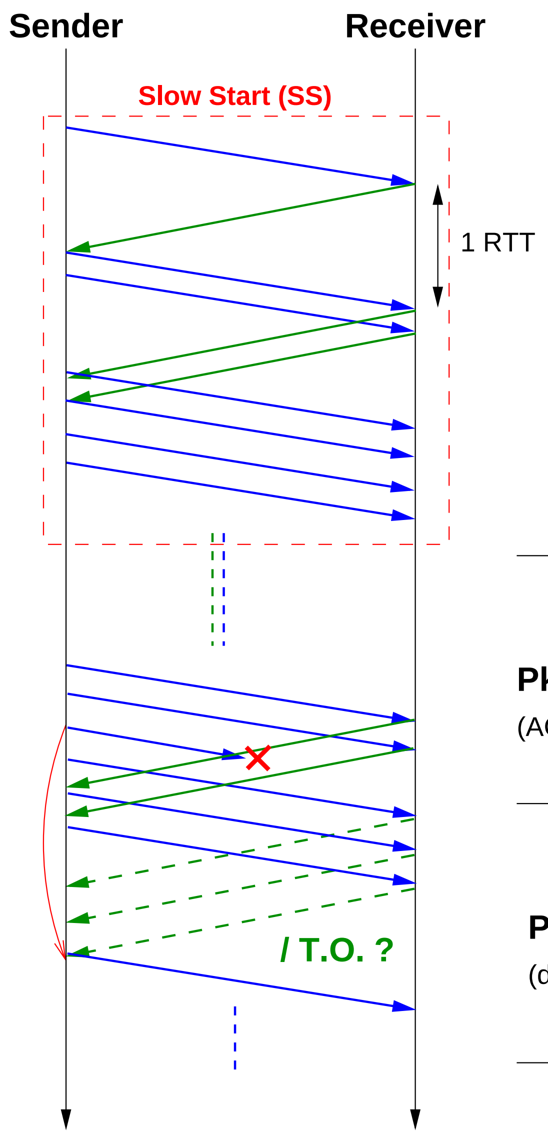
	Cong. Window	Timers
○ **Pkt Recv** (ACK recv)	ssthresh **cwnd += 1 (SS)** **cwnd += 1/cwnd (CA)**	RTTVAR = (1−b) RTTVAR + b |SRTT−RTT| SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO ,** SRTT+ max(G, 4 RTTVAR))
✕ **Pkt loss** (dup. ACKs, T.O.)	ssthresh = cwnd / 2 **cwnd = 1**	**RTO = RTO x 2**

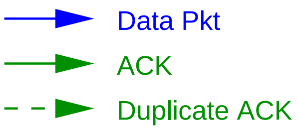
Reminder on TCP



	Cong. Window	Timers
O **Pkt Recv** (ACK recv)	ssthresh **cwnd += 1 (SS)** **cwnd += 1/cwnd (CA)**	RTTVAR = (1−b) RTTVAR + b |SRTT−RTT| SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO ,** SRTT+ max(G, 4 RTTVAR))
× **Pkt loss** (dup. ACKs, T.O.)	ssthresh = cwnd / 2 **cwnd = 1**	**RTO = RTO x 2**

Reminder on TCP





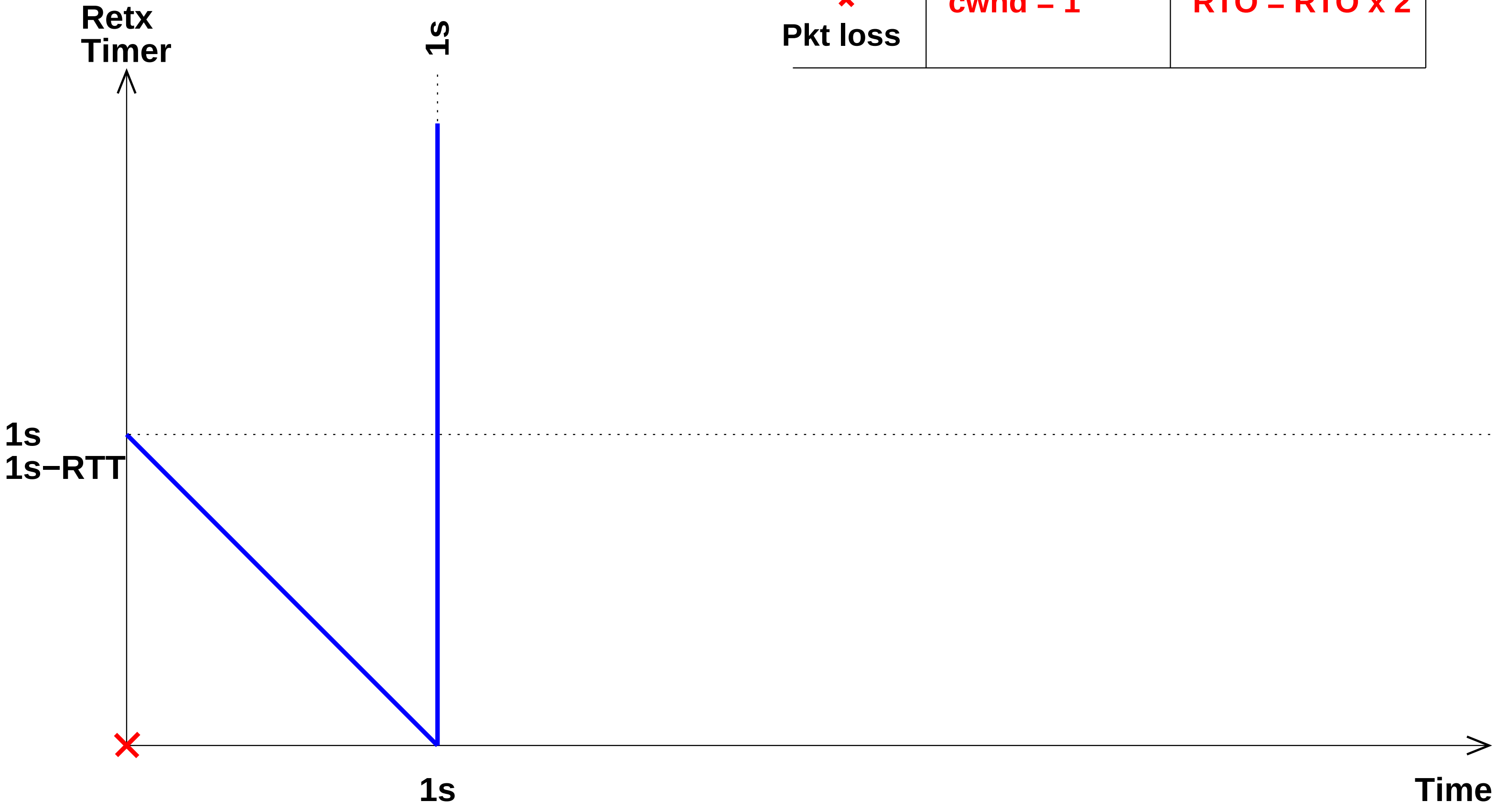
	Cong. Window	Timers
Pkt Recv (ACK recv)	ssthresh **cwnd += 1 (SS)** **cwnd += 1/cwnd (CA)**	RTTVAR = (1−b) RTTVAR + b |SRTT−RTT| SRTT = (1−a) SRTT + a RTT **RTO = max(minRTO ,** SRTT+ max(G, 4 RTTVAR))
Pkt loss (dup. ACKs, T.O.)	ssthresh = cwnd / 2 **cwnd = 1**	**RTO = RTO x 2**

JF-drop

	Cong. Window	Timers
Pkt Recv		
Pkt loss		

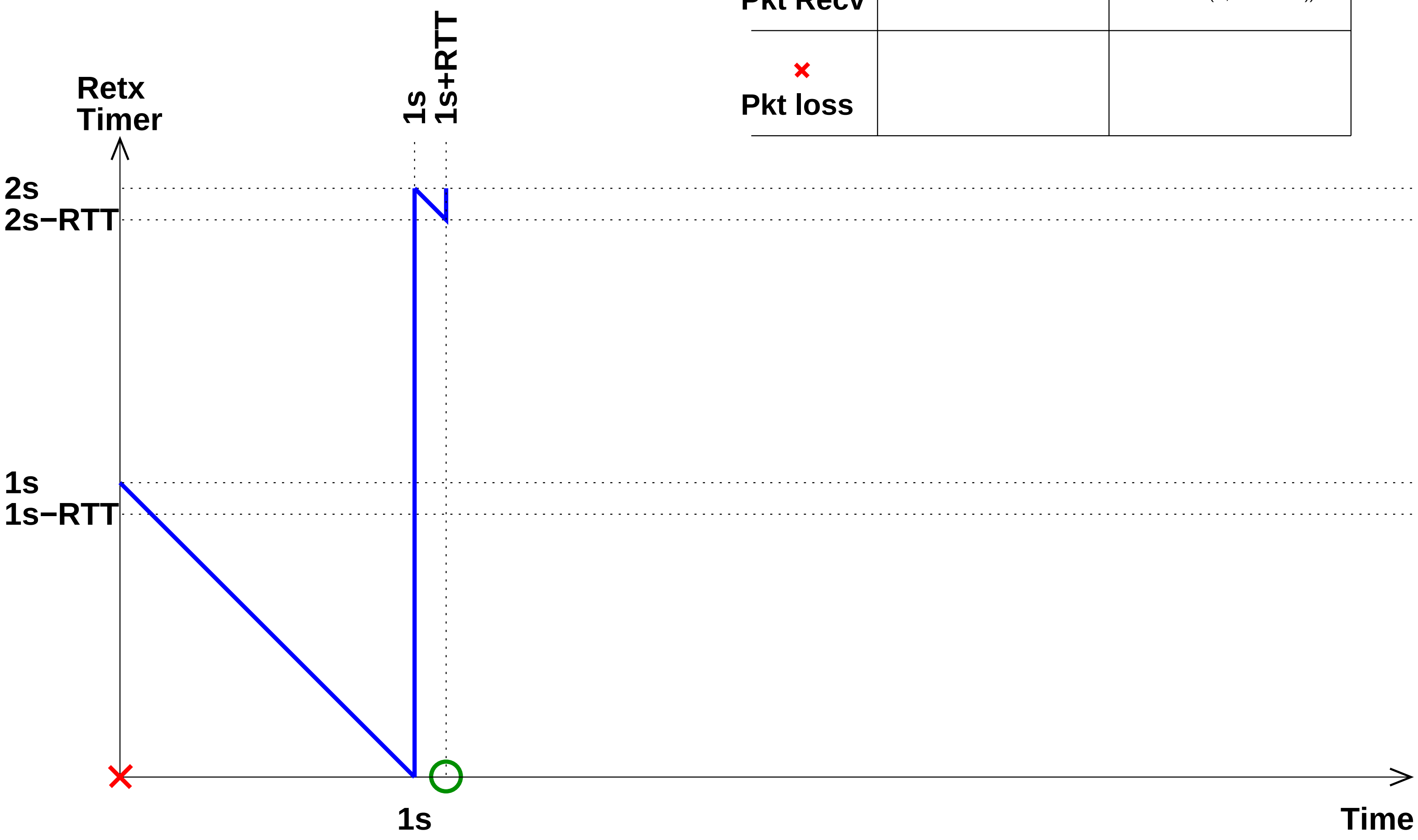
JF-drop

	Cong. Window	Timers
Pkt Recv		
Pkt loss	cwnd = 1	RTO = RTO x 2



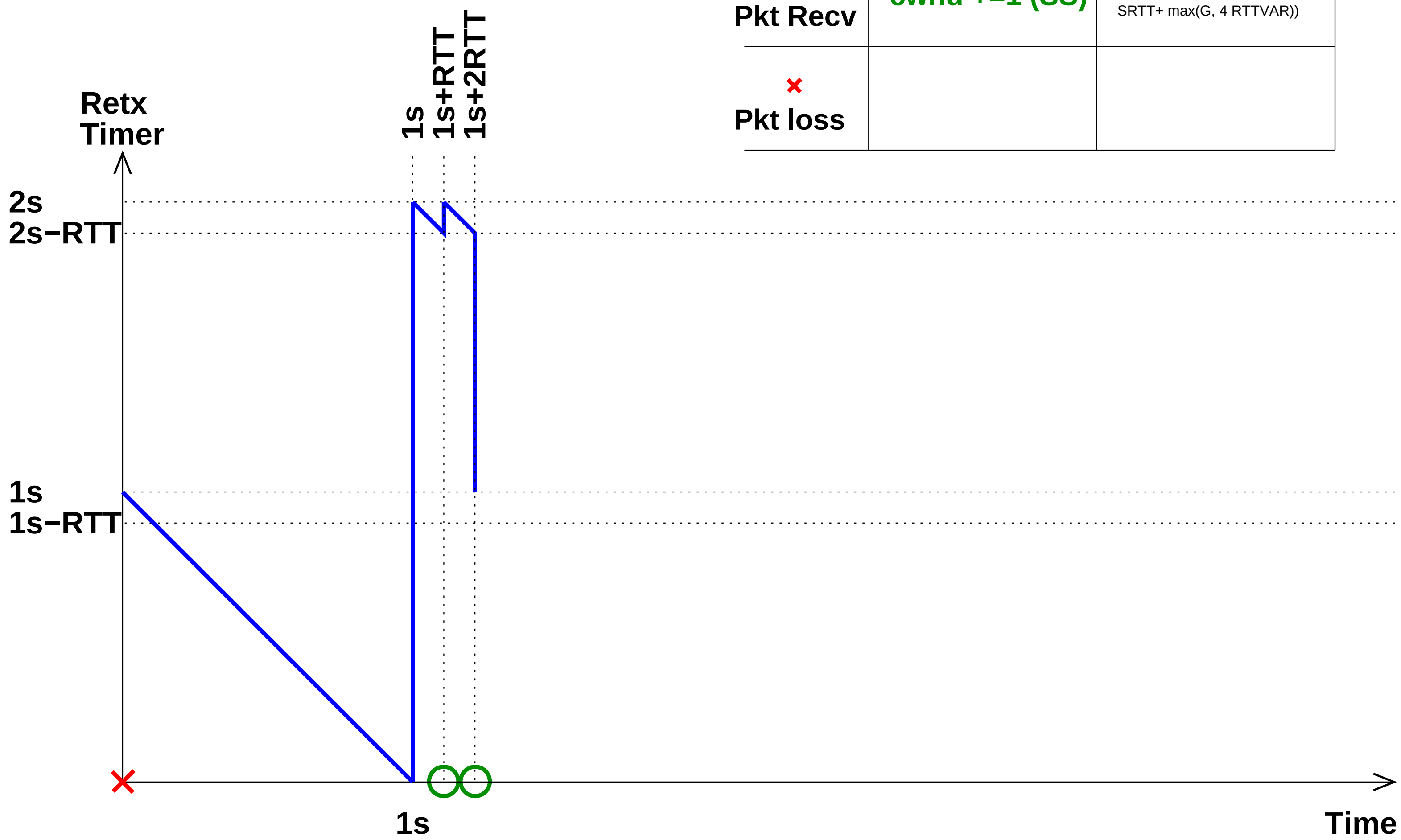
JF-drop

	Cong. Window	Timers
Pkt Recv	cwnd +=1 (SS)	RTO = max(minRTO, SRTT+ max(G, 4 RTTVAR))
Pkt loss		



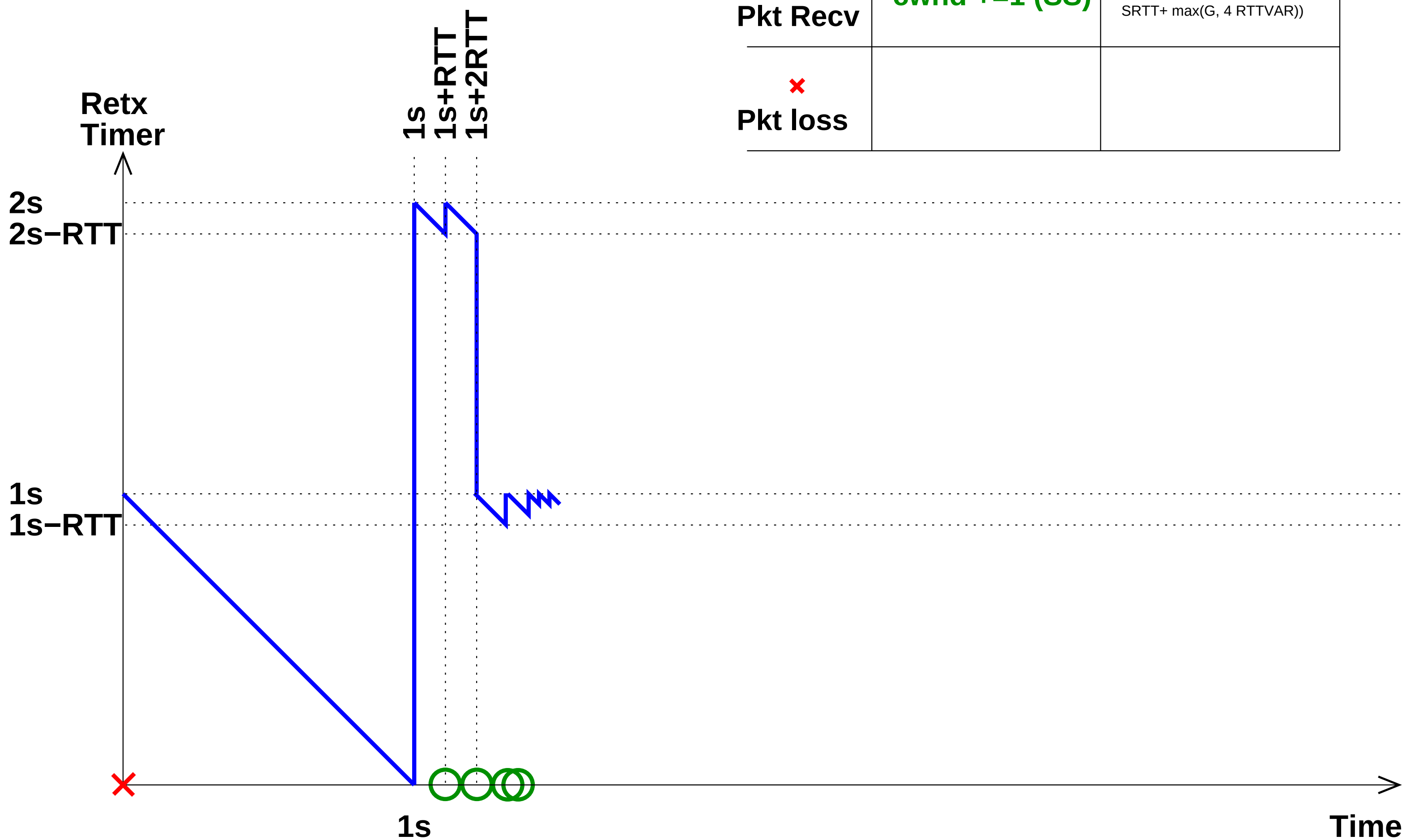
JF-drop

	Cong. Window	Timers
Pkt Recv	cwnd +=1 (SS)	RTO = max(minRTO, SRTT+ max(G, 4 RTTVAR))
Pkt loss		



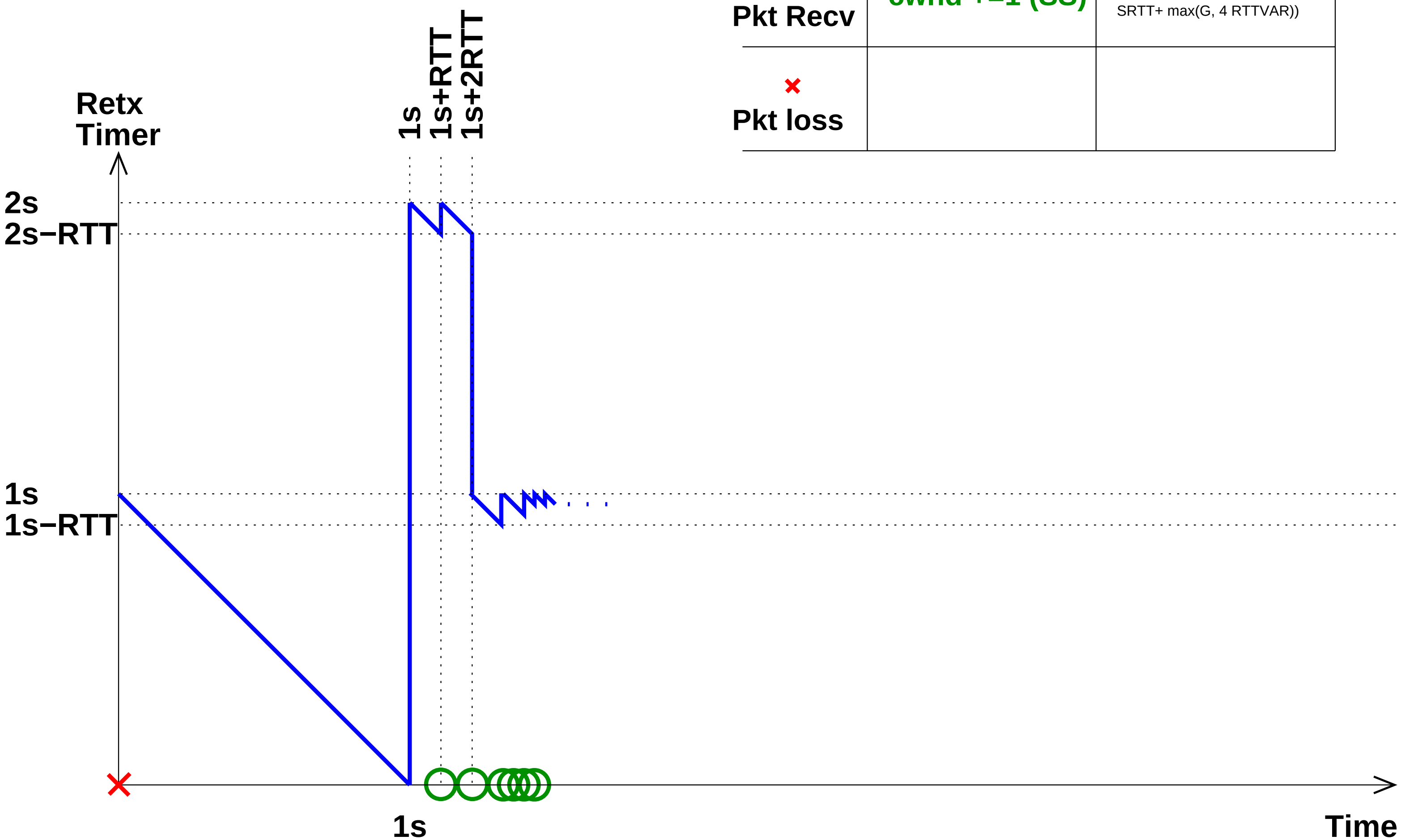
JF-drop

	Cong. Window	Timers
○ Pkt Recv	cwnd +=1 (SS)	RTO = max(minRTO, SRTT+ max(G, 4 RTTVAR))
× Pkt loss		



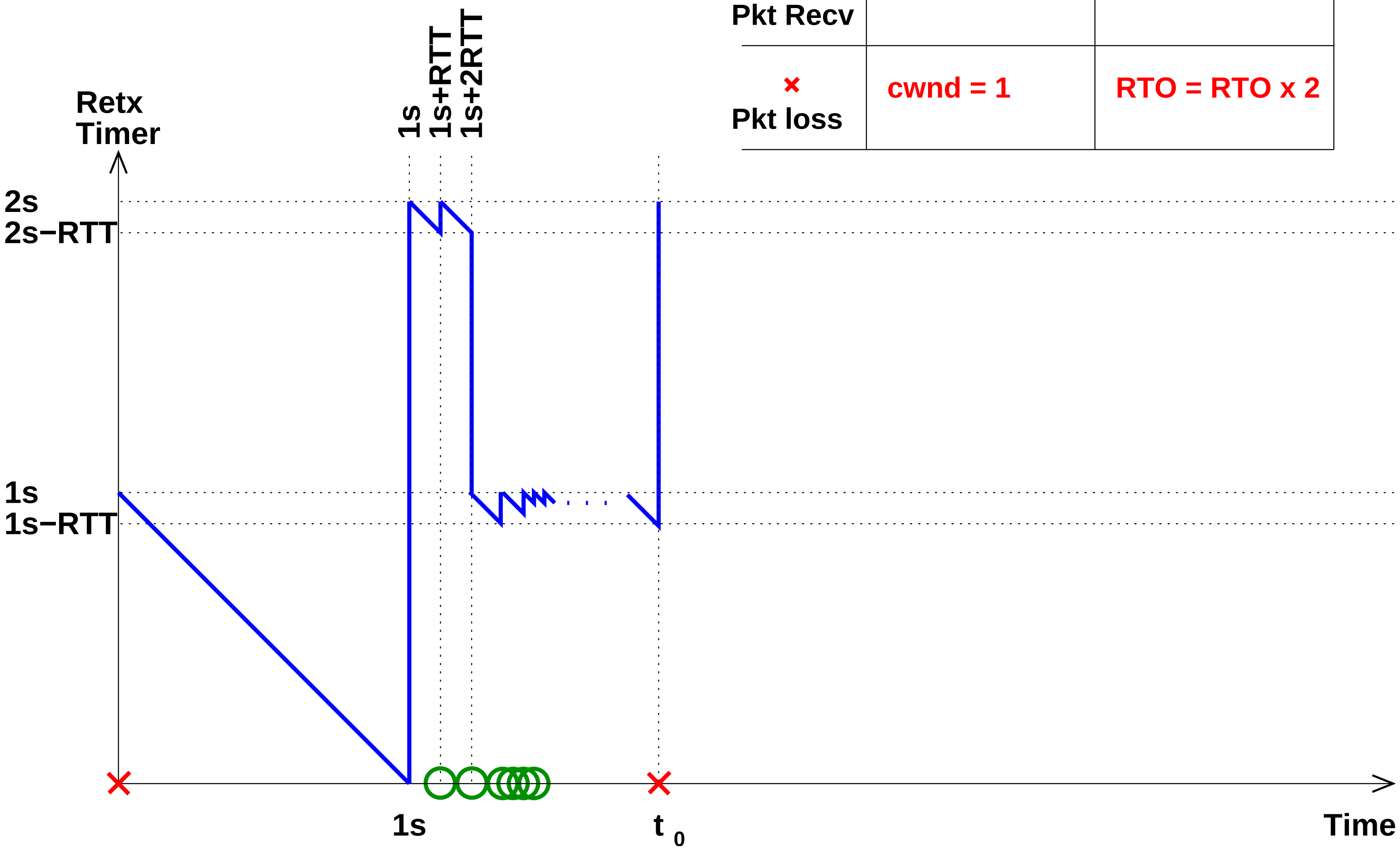
JF-drop

	Cong. Window	Timers
Pkt Recv	cwnd +=1 (SS)	RTO = max(minRTO, SRTT+ max(G, 4 RTTVAR))
Pkt loss		



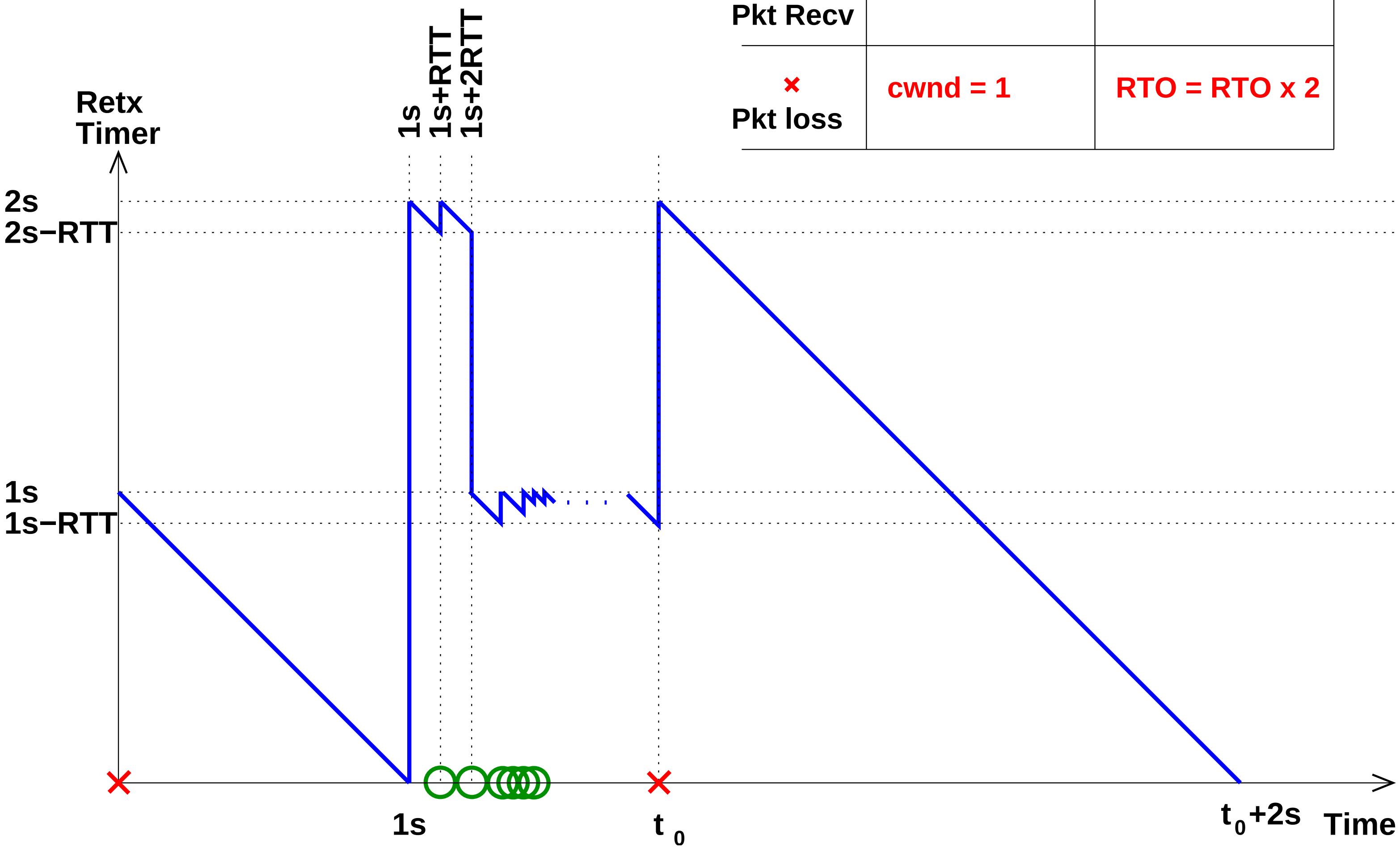
JF-drop

	Cong. Window	Timers
Pkt Recv		
Pkt loss	cwnd = 1	RTO = RTO x 2



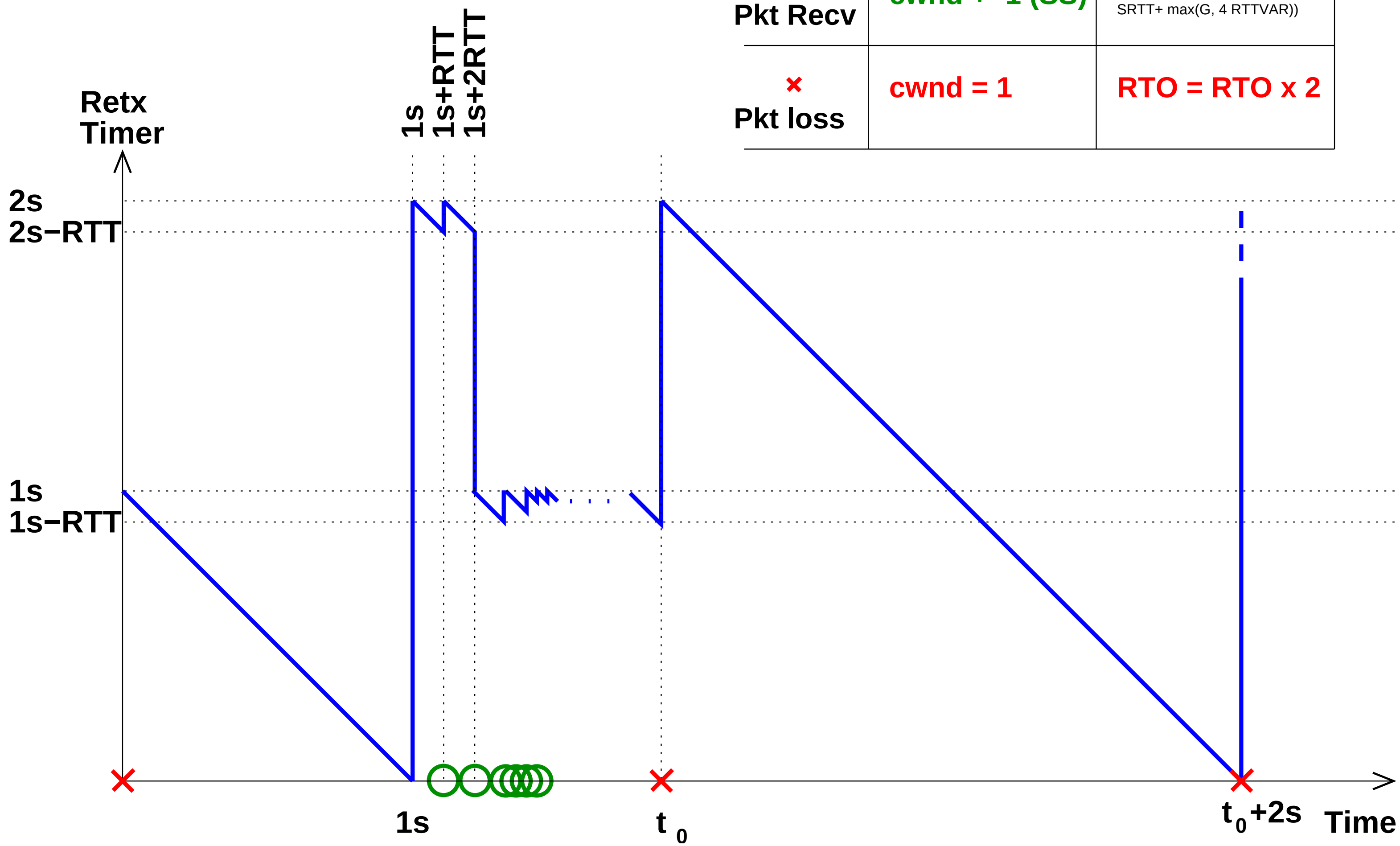
JF-drop

	Cong. Window	Timers
Pkt Recv		
Pkt loss	cwnd = 1	RTO = RTO x 2



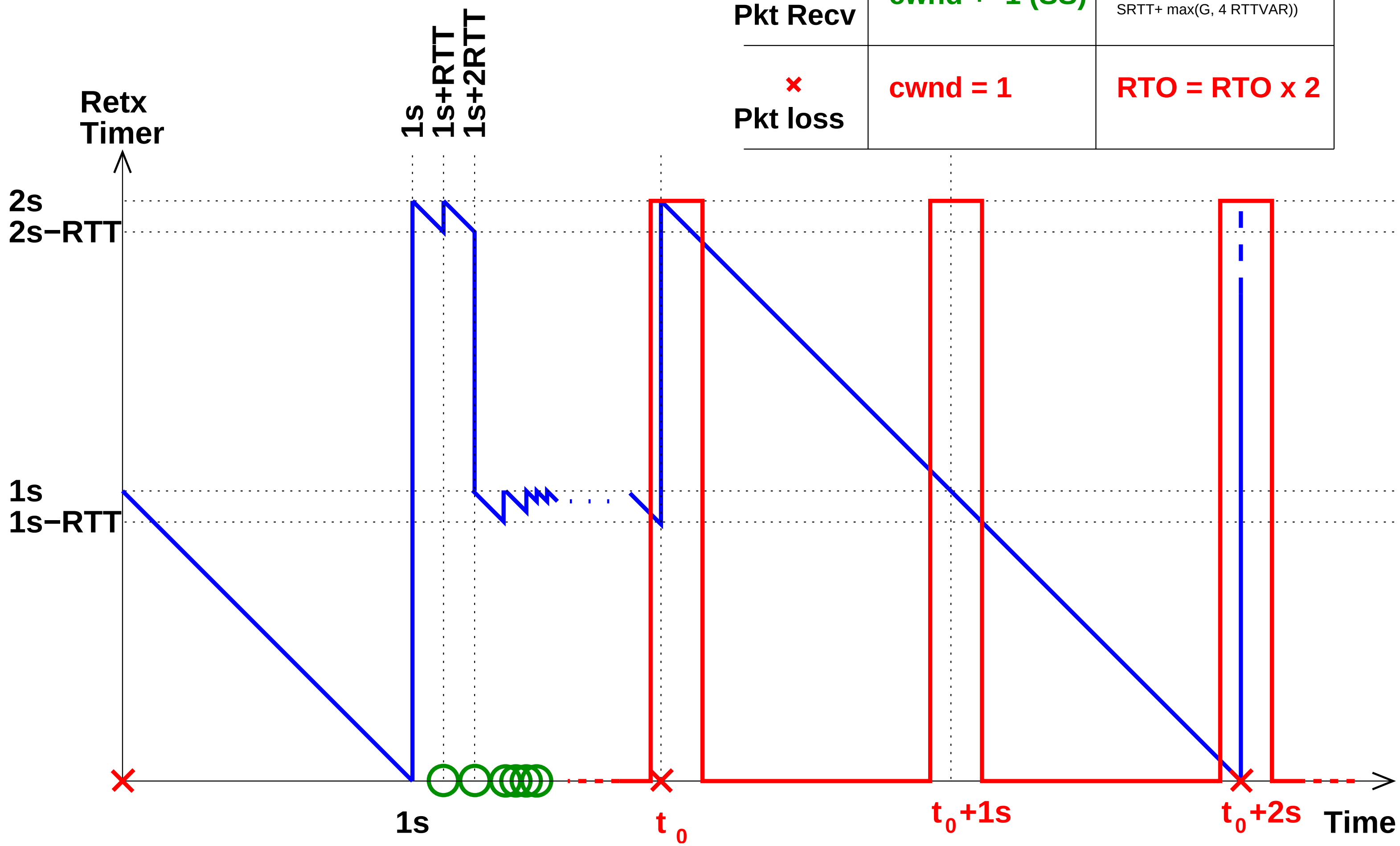
JF-drop

	Cong. Window	Timers
O Pkt Recv	cwnd +=1 (SS)	RTO = max(minRTO, SRTT+ max(G, 4 RTTVAR))
× Pkt loss	cwnd = 1	RTO = RTO x 2



JF-drop

	Cong. Window	Timers
Pkt Recv	cwnd +=1 (SS)	RTO = max(minRTO, SRTT+ max(G, 4 RTTVAR))
Pkt loss	cwnd = 1	RTO = RTO x 2



Simulation results: Number of hops