

# Robust Congestion Signaling



David Wetherall  
University of Washington  
June 2001.

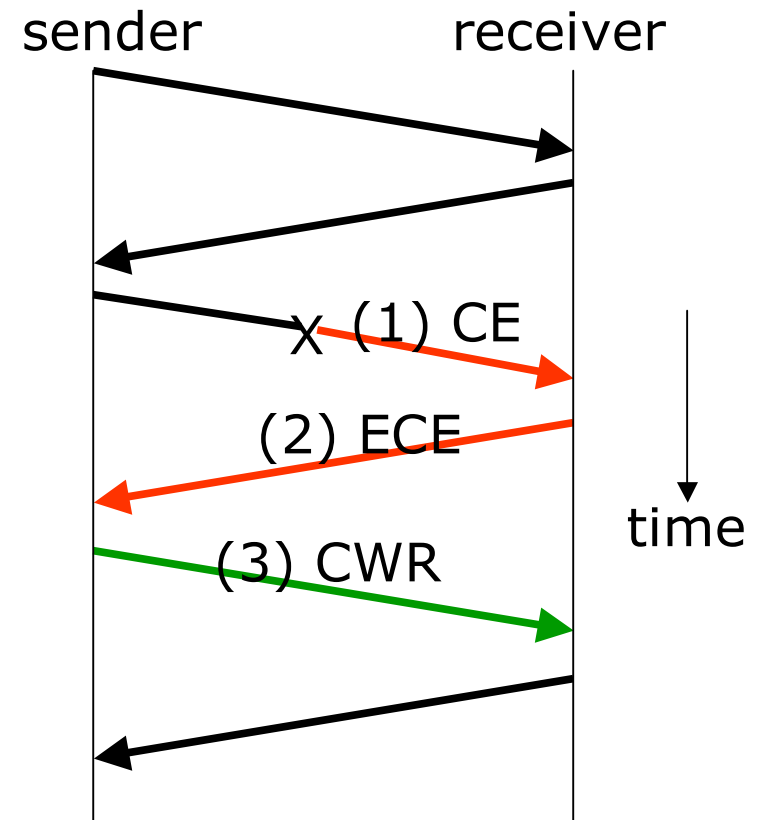
# This Talk



- A case study in congestion control
  - ECN, a problem, and a solution
- Speculation on robustness in protocols
  - An early look at faults in BGP
- Credits: David Ely, Neil Spring, Tom Anderson (UW) and Stefan Savage (UCSD)

# TCP with Explicit Congestion Notification

1. Routers mark packets (CE) instead of drop them
  2. Receivers echo marks (ECE) back to the sender
  3. Senders react (CWR) as if a drop had occurred
- Better performance, but a more complex design



# Want to surf the Web faster?

linux-2.4.0/include/net/tcp\_ecn.h

Normal:

```
51: static __inline__ void
```

```
52: TCP_ECN_send(...)
```

```
...
```

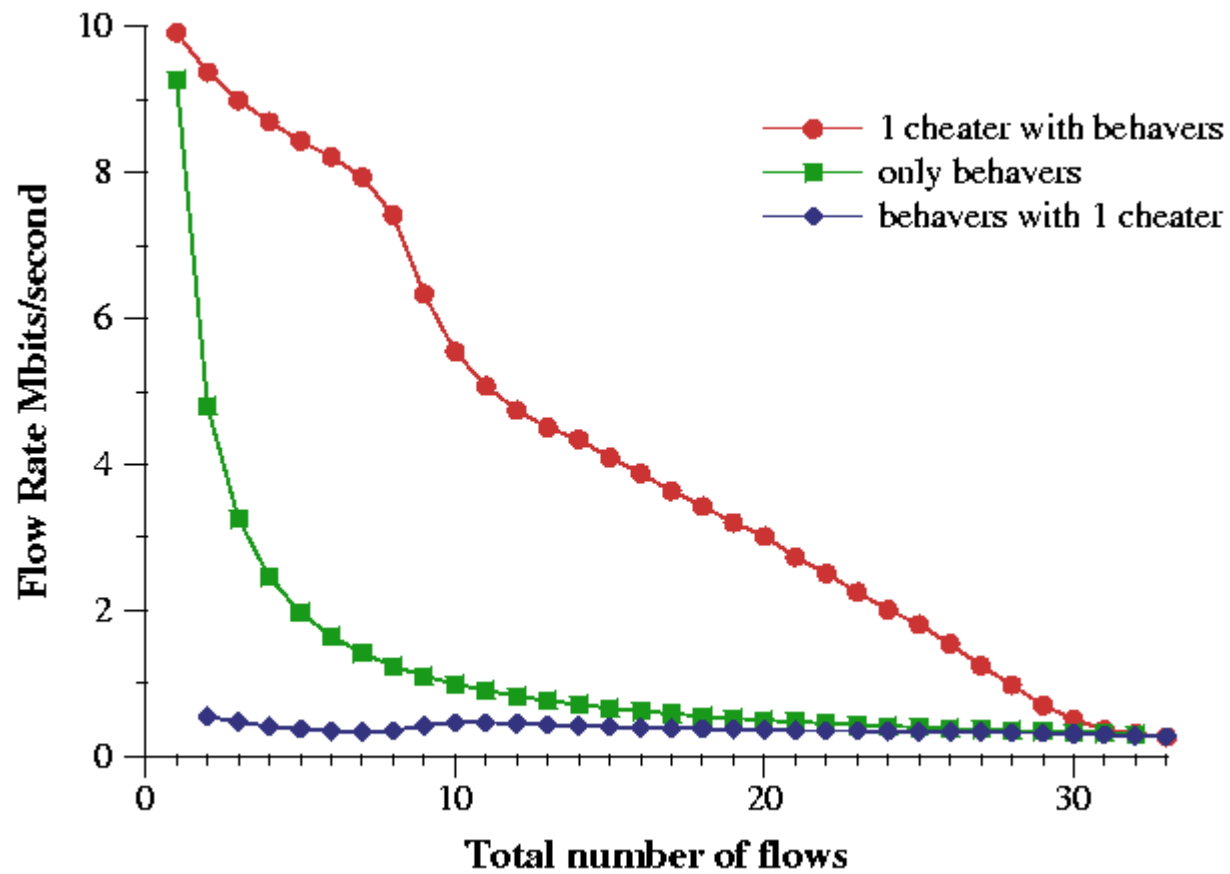
```
67:     if (tp->ecn_flags & TCP_ECN_DEMAND_CWR)
```

```
68:         skb->h.th->ece = 1;
```

Misbehaving:

```
68:         skb->h.th->ece = 0;
```

# Cheating pays



# Kinds of misbehavior



- Aggressive senders
  - Not the focus of this talk, need network support
- Cheating receivers
  - Plausible given the incentives, and simple too.
- Accidental misbehavior
  - Bugs, box incompatibilities, ...
- We focus on preventing the last two faults with only end-system support.

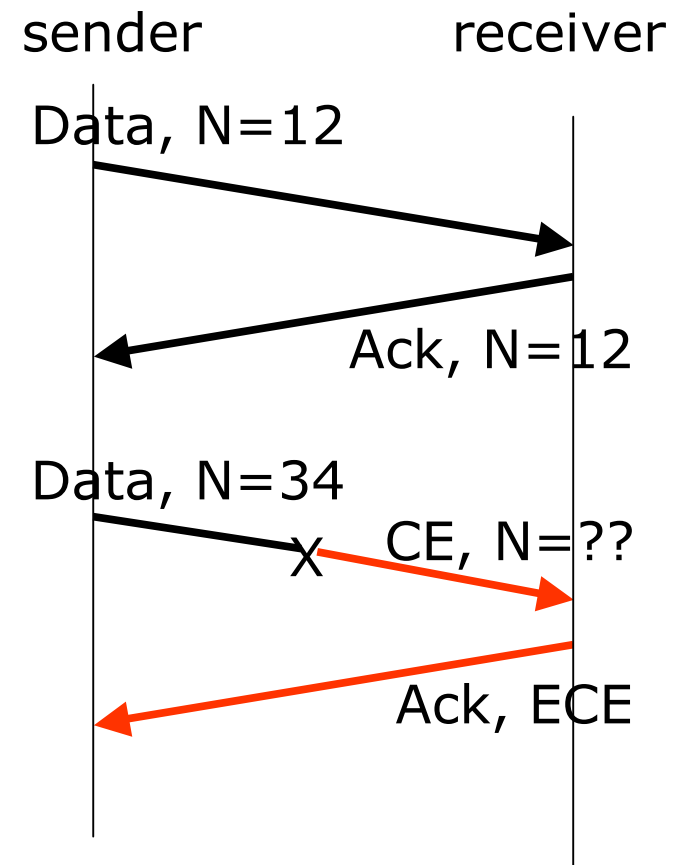
# Robust Congestion Signaling



- Goal is to provide the benefits of ECN while making the mechanism as robust as signaling with drops
  - Want to take everyone but the sender and marking router “out of the equation”
- This is valuable even when there is network support as a check that congestion control is working

# A perfect Nonce solution in theory

- Want receiver to prove receipt of unmarked packets to sender
  - random challenge
- Let the marking router erase a vital part of the “proof”



# Making Nonces work in practice



- Acknowledgements are neither sent one-for-one with data packets, nor sent reliably
  - Mightn't receive Nonce echo that reveals congestion
- Don't have a lot of space to work with
  - Leverage other TCP fields and use compact nonces

# One-bit Nonce Sums



- Reply with sum of individual nonces
  - Cumulative nonce to match cumulative TCP ack
  - Ensures delayed/lost acks can't bypass check
  - Modulo sum is no easier to predict than one nonce
- Send single bit nonces
  - Probabilistic detection of misbehavior
  - Each ack'ed congestion event is a separate trial

# Encoding One-bit Nonces

- Send using 4<sup>th</sup> ECN codepoint in IP header

Bit Value	Old Meaning	New Meaning	Symbol
00	ECN-incapable	ECN-incapable	–
01	–	ECN-capable, Nonce=1	ECT(1)
10	ECN-capable	ECN-capable, Nonce=0	ECT(0)
11	Marked	Marked	CE

- Return using a (reserved) bit in the TCP header

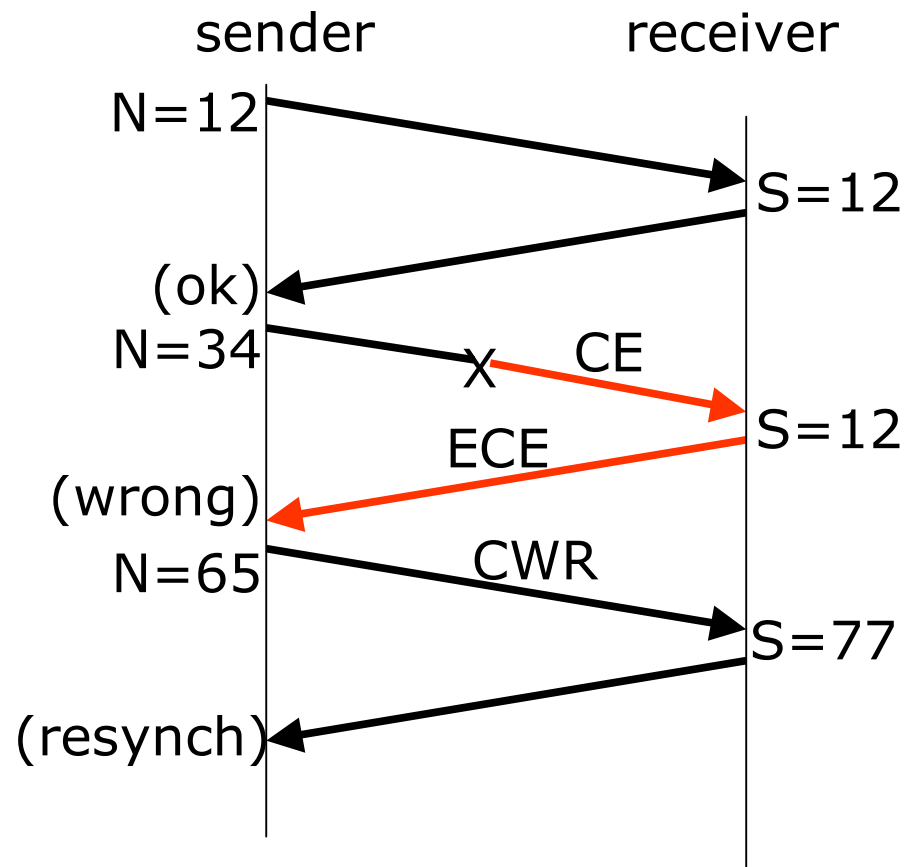
4 bit header length	reserved (3 bits)	<b>N</b>	<b>C</b>	<b>E</b>	U	A	P	R	S	F
		<b>S</b>	<b>W</b>	<b>R</b>	R	C	S	S	Y	I
			<b>R</b>	<b>E</b>	G	K	H	T	N	N

# Resynchronizing after a loss

- Problem: nonces are lost during congestion, so sent nonce sum will be incorrect afterwards
- Solution: sum will be off by a constant until there is a further congestion event, so reset expected value at the sender
- There's much hair here:
  - Only need one congestion indication per RTT
  - Retransmissions aren't subject to ECN

# Resynchronization example

- Returned nonce sum is (ok) before congestion and (wrong) during, as nonces have been lost.
- It is (resynched) after congestion, when CWR was received and ECE is no longer set.



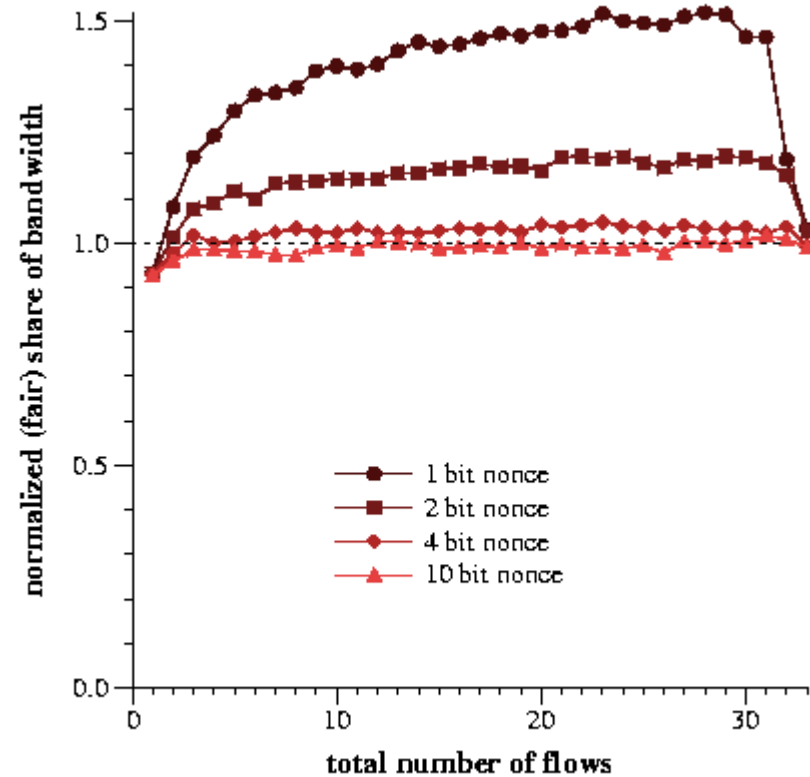
# TCP Processing and State



- Receiver:
  - Nonce bit for every out-of-order packet
  - Nonce sum of in-order packets
  - Obviously sends sum and ECE if congestion
- Sender:
  - Nonce sum expected for every unack'ed packet
  - Sum offset bit for synchronization
  - Checks sum when no congestion claimed, and re-synchronizes after congestion episodes

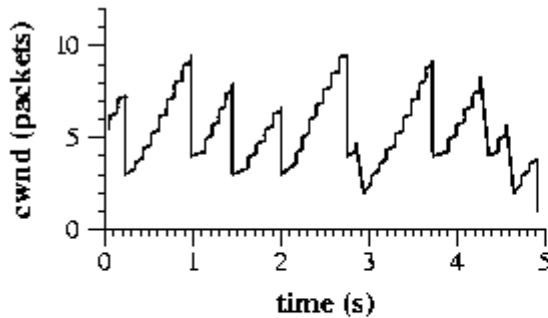
# Effectiveness of the Design

- For large nonces, sum serves as a congestion signal
- For one bit nonces, need to penalize misbehavior
  - Depends on whether bugs or cheating

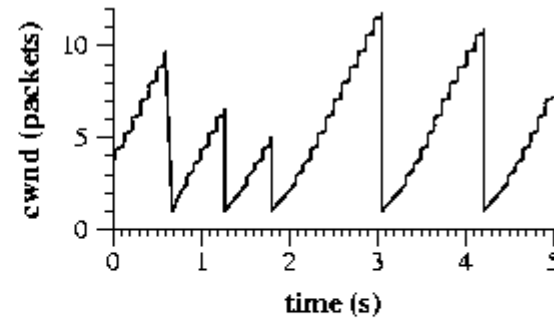


# Penalizing cheaters (sawtooths)

- One possible mechanism: set  $cwnd = ssthresh = 1$



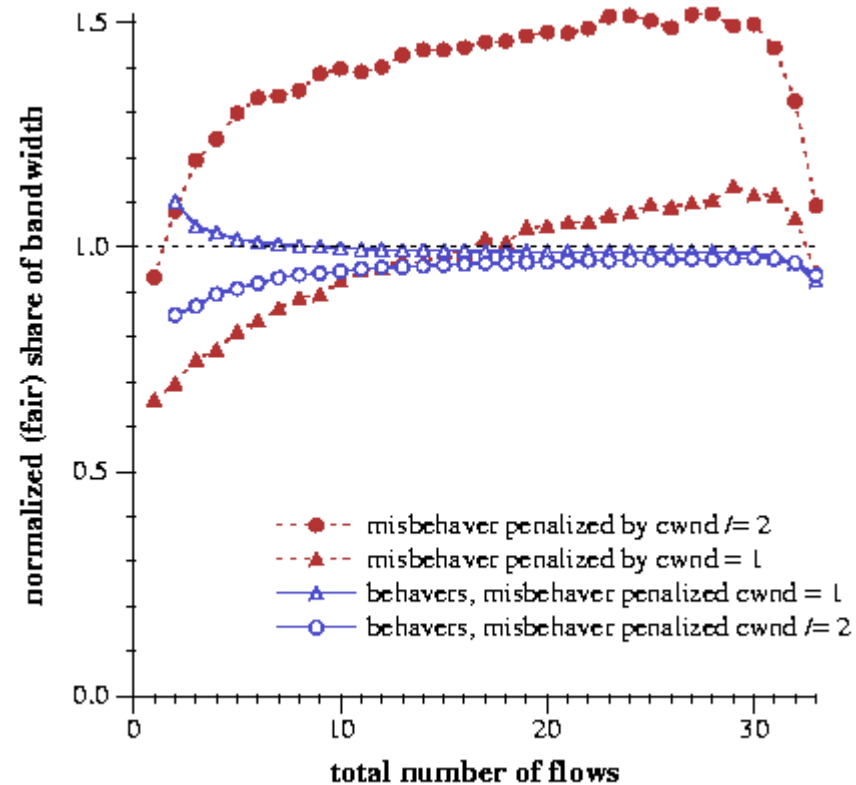
Normal cwnd sawtooth



$cwnd=1$  on misbehavior  
(but not disabling ECN)

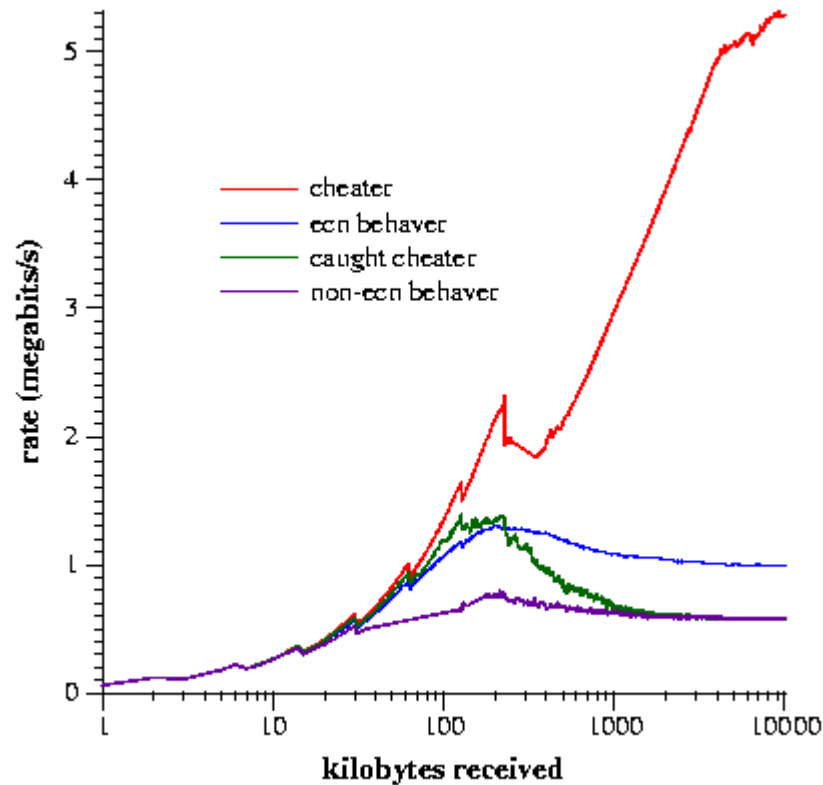
# Penalizing cheaters (throughput)

- Penalty negates the benefit of cheating



# The bottom line

- An incentive for ECN, a disincentive for lying



# Additional benefits



- Prevents optimistic acks (acks sent before data is received) even with drop tail gateways
- Checks retransmission ambiguities
  - Eiffel receivers identify spurious retransmissions
- General design technique
  - New transports such as SCTP, rate-based, unreliable transports like TFRC, and some multicast too.

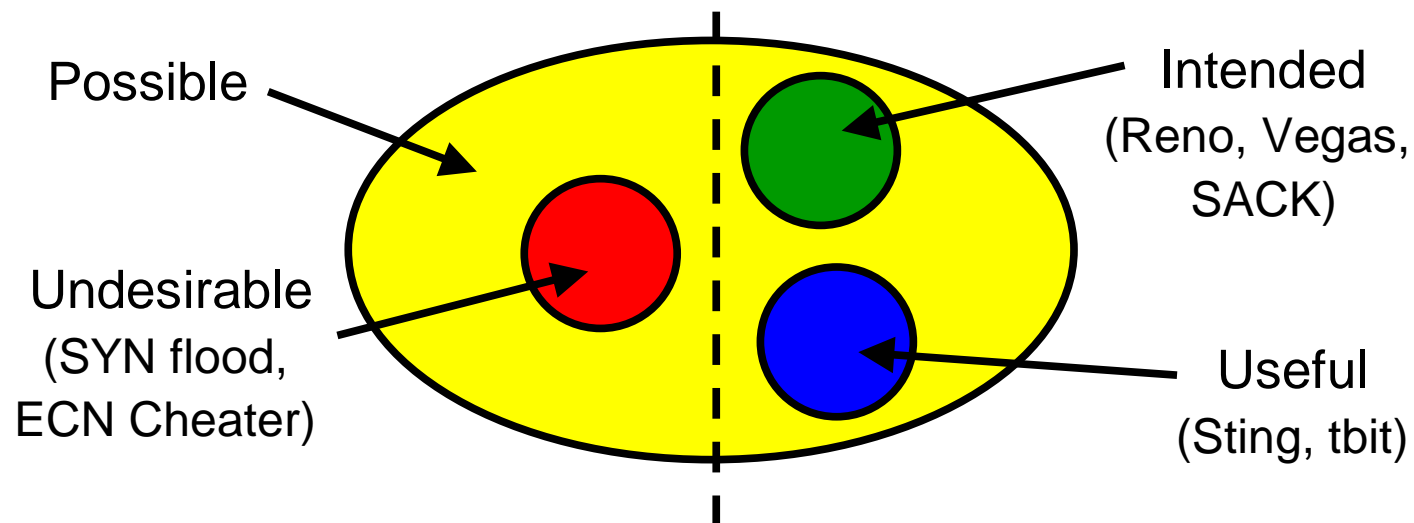
# Lessons learned



- (Again) observed there is little support for consistency checking in TCP
- Demonstrated that it is possible to modify protocols to check at least some consistency properties fairly cheaply
- “Be conservative in what you believe”

# Speculations on Robust Protocol Design

- There is a surprisingly large range of TCP behaviors ...



- What about BGP, DNS, IP, etc., ... ?
- Can we efficiently contain undesirable behaviors?

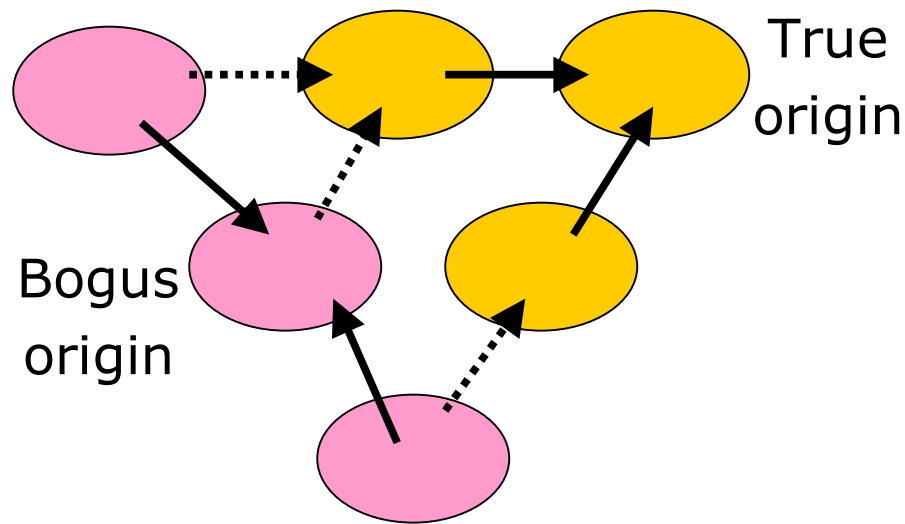
# Possible BGP routing faults



- Can a third party interfere with connectivity?
- Well, yes.
  - Blackholes (bogus origins)
  - Customer provides transit (impaired forwarding)
  - Flapping (damping)
  - Policy disputes (in practice?)

# Blackholes

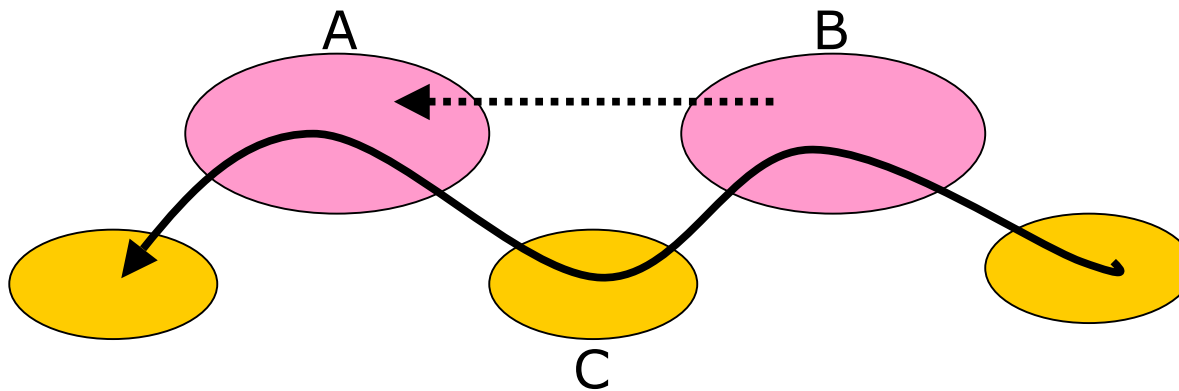
- Bogus origin advertises prefix and attracts traffic; more-specific prefix takes precedence



- True origin might not even be aware of this!

# Customer provides Transit

- Multi-homed customer C advertises transit from providers B to A and B believes it



- B reaches A through C, instead of directly

# Defense in the Internet today



- Routing Registries detail who owns what and could in theory be used to check validity
  - But incomplete and not heavily used in practice
- Route filtering at ISP edges rejects non-customer routes heard from customers
  - But fairly static lists and not strict in practice

# Looking for misconfigurations



- Obviously broken announcements easy to find
  - e.g., routes to 10/8, AS paths with “loops”
  - Study “Routeviews” BGP database
- Plausible but incorrect announcements require more work ...
  - No well-defined model to check against

# Can we prevent misconfigurations?



- Crypto and certification authority solves origin problem ... can we do so more efficiently?
  - Detect conflicting origins before validating
- Recent work dynamically checks that paths are “policy safe” using BGP community signaling
  - Can we extend it to raise the bar on unintended transit?

# Questions?

