IPADS INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

End-to-end Layer
Best-effort is not enough

IPADS, Shanghai Jiao Tong University

https://www.sjtu.edu.cn

Review: Control-plane VS. Data-plane

Control-plane

- Control the data flow by defining rules
- E.g., the routing algorithm

Data-plane

- Copies data according to the rules
- Performance critical
- E.g., the IP forwarding process

NAT

CASE: Ethernet Mapping

Mapping Internet to Ethernet

Case Study: Mapping Internet to Ethernet

Listen-before-sending rule, collision

Ethernet: CSMA/CD

- Carrier Sense Multiple Access with Collision Detection

Ethernet type

- Experimental Ethernet, 3 mpbs
- Standard Ethernet, 10 mbps
- Fast Ethernet, 100 mbps
- Gigabit Ethernet, 1000 mbps

Overview of Ethernet

A half duplex Ethernet

- The max propagation time is less than the 576 bit times, the shortest allowable packet
- So that two parties can detect a collision together
- If collision: wait random first time, exponential backoff if repeat

A full duplex & point-to-point Ethernet

- No collisions & the max length of the link is determined by the physical medium

leader	destination	source	type	data	checksum
64 bits	48 bits	48 bits	16 bits	368 to 12,000 bits	32 bits

Difference between Hub and Switch

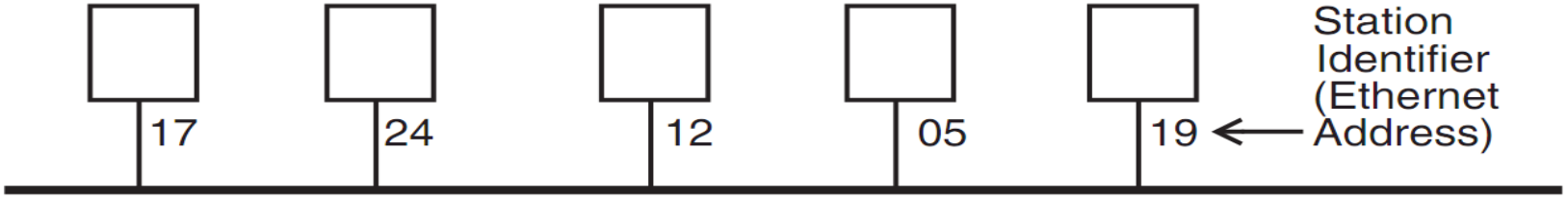
Hub

- A frame is "broadcast" to every one of its ports
- A 10/100Mbps hub must share its bandwidth with each port

Switch

- Keeps a record of the MAC addresses of all the devices
- A 10/100Mbps switch will allocate a full 10/100Mbps to each of its ports

Broadcast Aspects of Ethernet



Broadcast network

- Every frame is delivered to every station
- (Compare with forwarding network)

ETHERNET_SEND

- Pass the call along to the link layer

ETHERNET_HANDLE

- Simple, can even be implemented in hardware

Broadcast Aspects of Ethernet

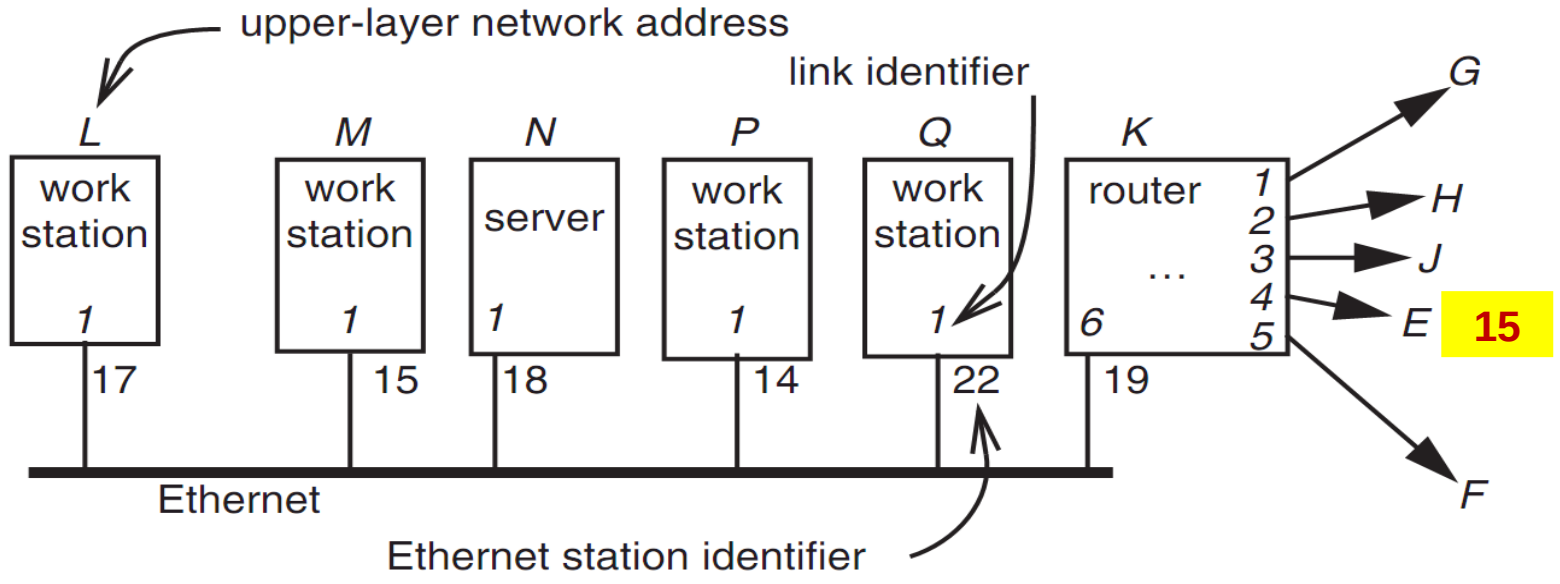
```
procedure ETHERNET_HANDLE (net_packet, length)
  destination ← net_packet.target_id
  if destination = my_station_id

    then
    GIVE_TO_END_LAYER (net_packet.data,
                       net_packet.end_protocol,
                       net_packet.source_id)
  else
    ignore packet
```

no need to do any forwarding

Broadcast Aspects of Ethernet

```
procedure ETHERNET_HANDLE (net_packet, length)
  destination ← net_packet.target_id
  if destination = my_station_id
      or destination = BROADCAST_ID
    then
    GIVE_TO_END_LAYER (net_packet.data,
                       net_packet.end_protocol,
                       net_packet.source_id)
  else
    ignore packet
```

Layer Mapping: Attach Ethernet to Forwarding Network



L sends a RPC to N by sending to station 18 of link 1

L sends a RPC to E by sending to K, E may have 15 as address, as well as M

Layer Mapping

The Internet network layer

- **NETWORK_SEND** (data, length, RPC, INTERNET, N)
- **NETWORK_SEND** (data, length, RPC, ENET, 18)

L must maintain a table

internet address	Ethernet/ station
M	enet/15
N	enet/18
P	enet/14
Q	enet/22
K	enet/19
E	enet/19

ARP (Address Resolution Protocol)

NETWORK_SEND ("where is M?", 11, ARP, ENET, BROADCAST)

NETWORK_SEND ("M is at station 15", 18, ARP, ENET, BROADCAST)

L asks E's Ethernet address, E does not hear the Ethernet broadcast, but the router at station 19 does, and it sends a suitable ARP response instead

Manage forwarding table as a cache

internet address	Ethernet/ station
M	enet/15

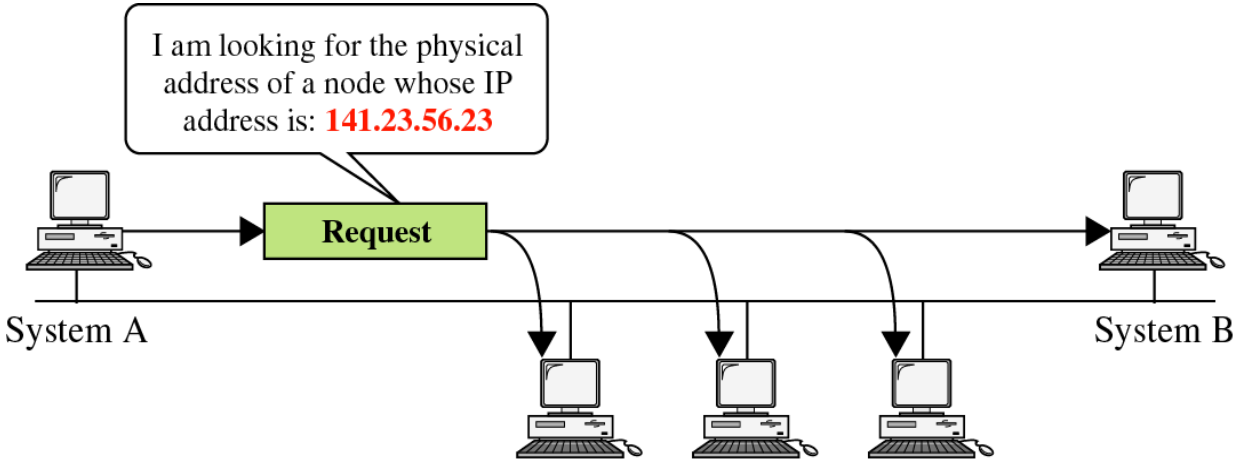
internet address	Ethernet/ station
M	enet/15
E	enet/19

ARP & RARP Protocol

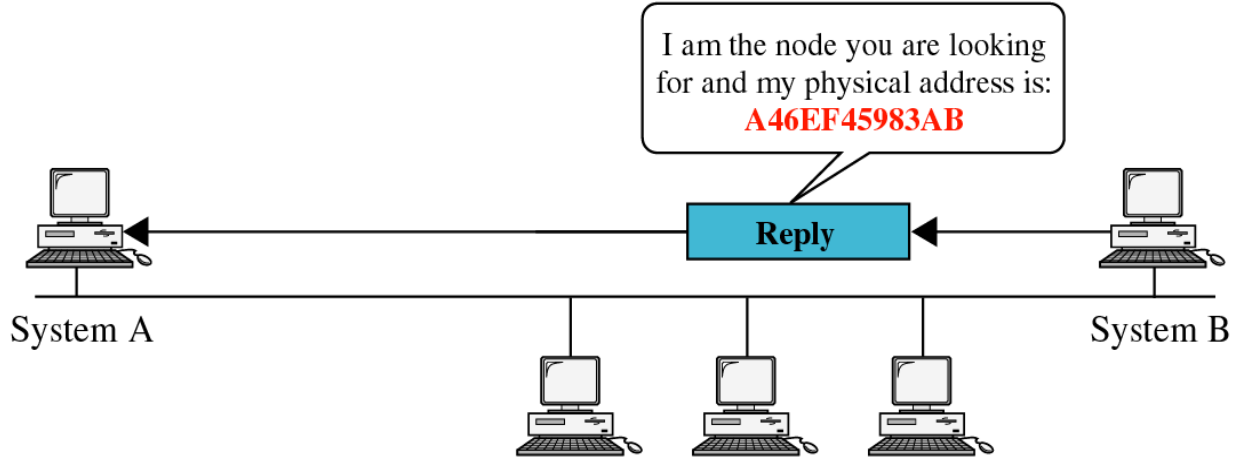
Hardware Type (16 bits)		Protocol Type (16 bits)
HA Length (8 bits)	PA Length (8 bits)	Operation (16 bits)
Sender Hardware Address (Octets 0-3)		
Sender Hardware Address (Octets 4-5)		Sender Protocol Address (Octets 0-1)
Sender Protocol Address (Octets 2-3)		Target Hardware Address (Octets 0-1)
Target Hardware Address (Octets 2-5)		
Target Protocol Address (Octets 0-3)		

Name mapping: IP address <-> MAC address

ARP & RARP



a. ARP request is broadcast



b. ARP reply is unicast

Network Topology

Take SJTU network for example

- Subnet: usually like 192.168.0.2 or 10.0.0.2
- Gateway: usually like 192.168.0.1
  - Get the global IP address: 202.120.40.82
  - A gateway usually has two (or more) IP address
- Proxy: get proxy's address
  - E.g., 106.185.46.164 (Japan)

Network Topology

How to Use socket to Access www.baidu.com ?

You code as if your PC connect directly with Baidu

- Call connect() with Baidu's IP address

But how does the system find next hop?

Putting All Together

App: I want to send a packet to Baidu, here is the packet with Baidu's IP in its header as target IP, and client's IP as source IP (Node-C)

OS: I don't know how to get to Baidu, I'll just send it to the router (gateway). But I cannot change the source IP of the packet, so I'll just change the MAC target address of the packet to the router's MAC address

Putting All Together

The router-1 (gateway): I get a packet with my MAC as target address. Is it my IP? No… So I'll just forward it to next hop, by changing the target MAC address to next hop's MAC address (NAT: change source IP and source port as well)

Router-2: I connect directly to Baidu, I'll just change the target MAC address to Baidu

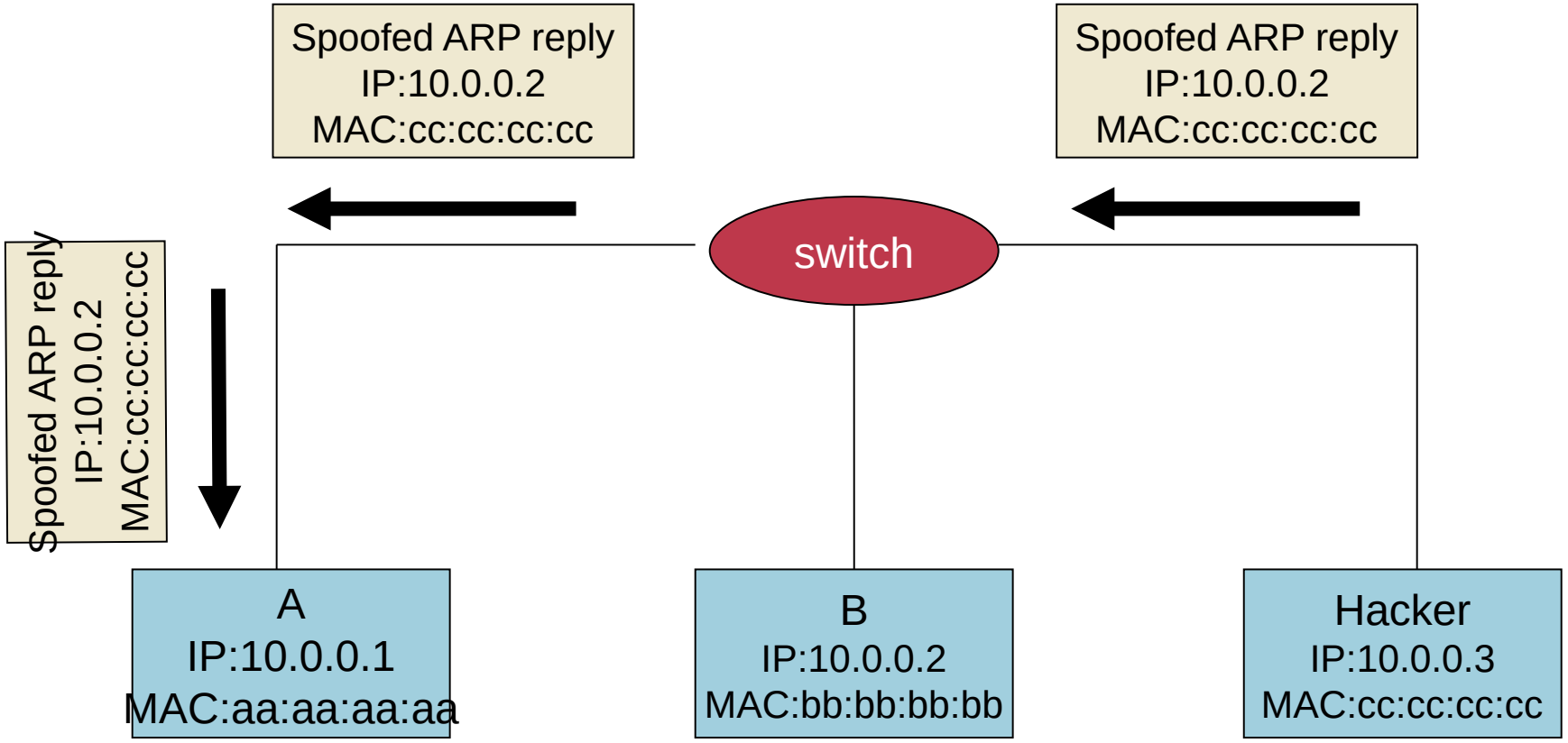
ARP Spoofing

Construct spoofed ARP replies

A target computer could be convinced to send frames destined for computer A to instead go to computer B

Computer A will have no idea that this redirection took place

This process of updating a target computer's ARP cache is referred to as "ARP poisoning"

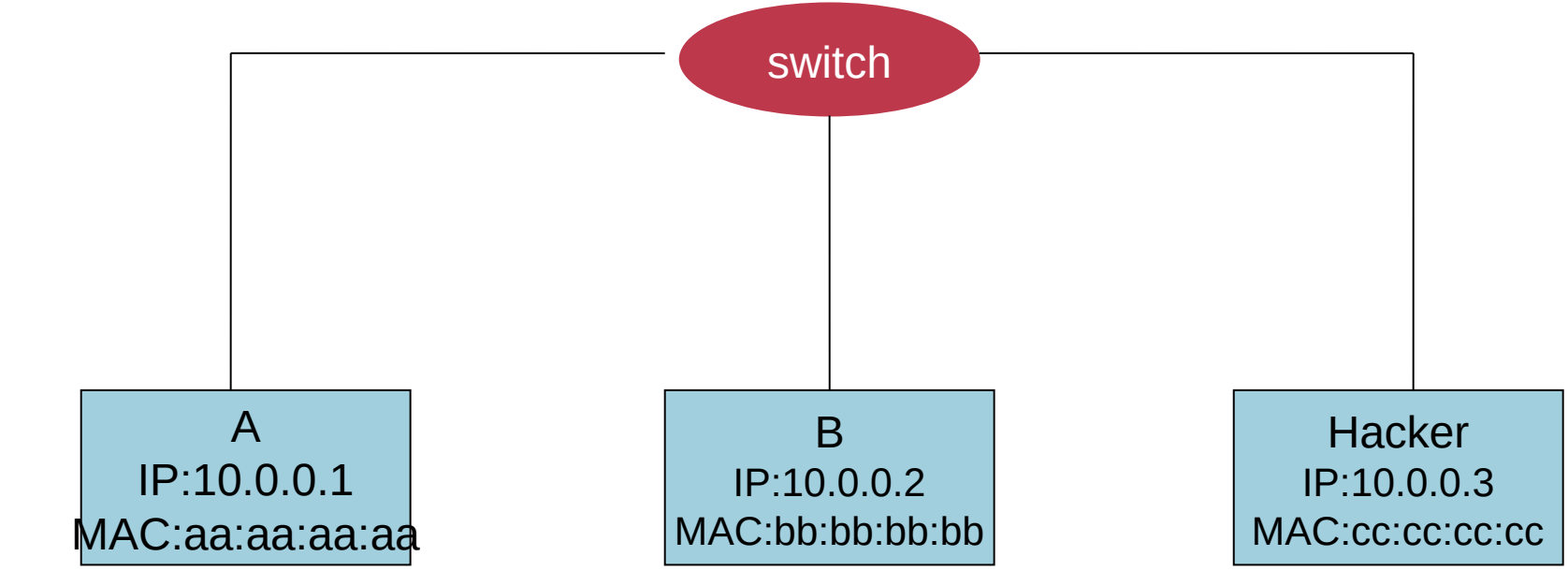


ARP cache

IP	MAC
10.0.0.2	bb:bb:bb:bb

ARP cache

IP	MAC
10.0.0.1	aa:aa:aa:aa



A's cache is poisoned

ARP cache

IP	MAC
10.0.0.2	cc:cc:cc:cc

ARP cache

IP	MAC
10.0.0.1	aa:aa:aa:aa

ARP Spoofing

Now all the packets that A intends to send to B will go to the hacker's machine

Cache entry would expire, so it needs to be updated by sending the ARP reply again

- How often?
- depends on the particular system
- Usually every 40s should be sufficient

In addition the hacker may not want his Ethernet driver talk too much

- Accomplish with ifconfig -arp

Man-in-the-Middle Attack

A hacker inserts his computer between the communications path of two target computers

The hacker will forward frames between the two target computers so communications are not interrupted

E.g., Hunt, Ettercap, etc.

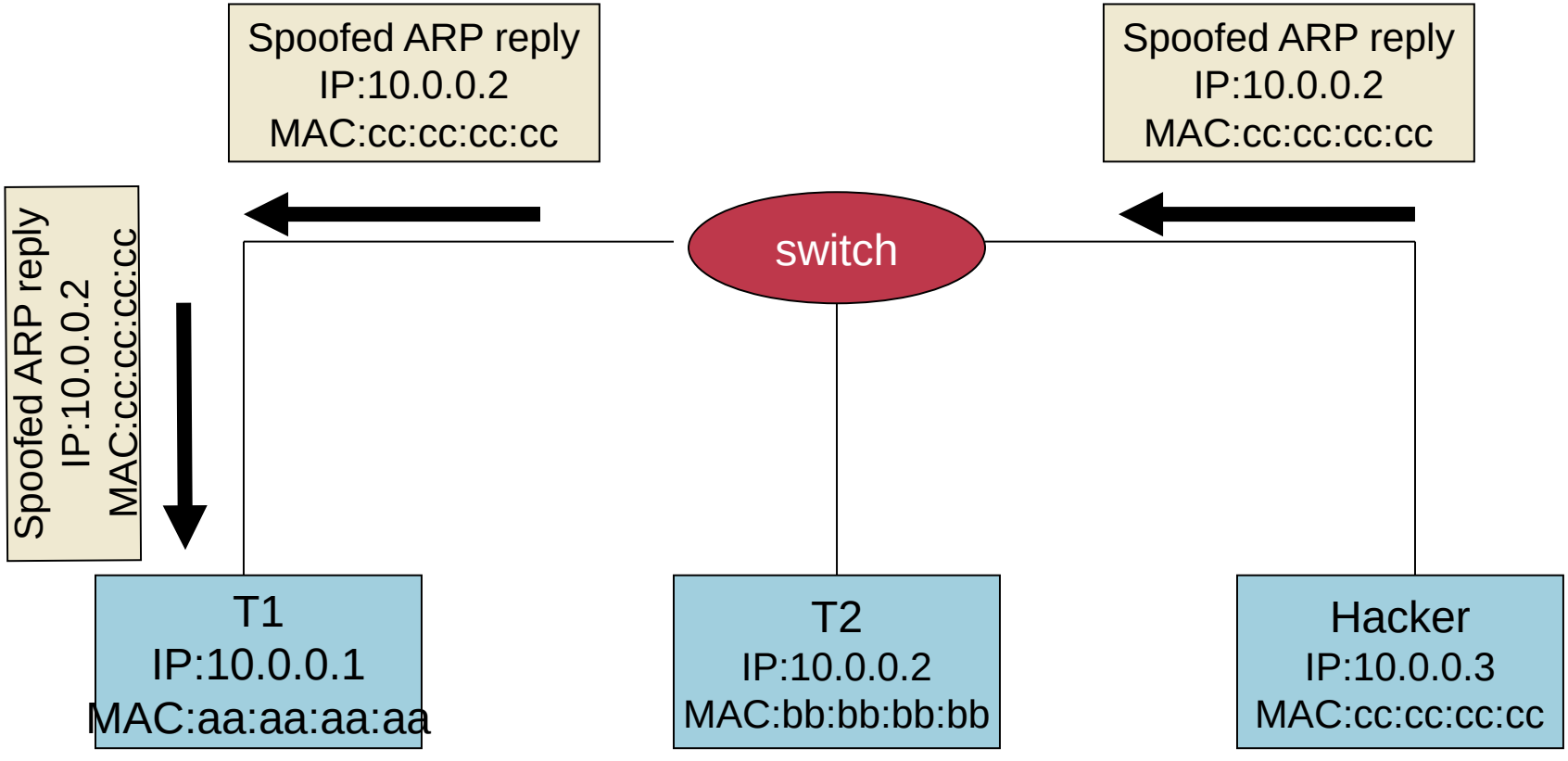
- Can be obtained easily in many web archives

Man-in-the-Middle Attack

The attack is performed as follows:

- Suppose X is the hacker's computer
- T1 and T2 are the targets

1. X poisons the ARP cache of T1 and T2
2. T1 associates T2's IP with X's MAC
3. T2 associates T1's IP with X's MAC
4. All of T1 and T2's traffic will then go to X first, instead of directly to each other

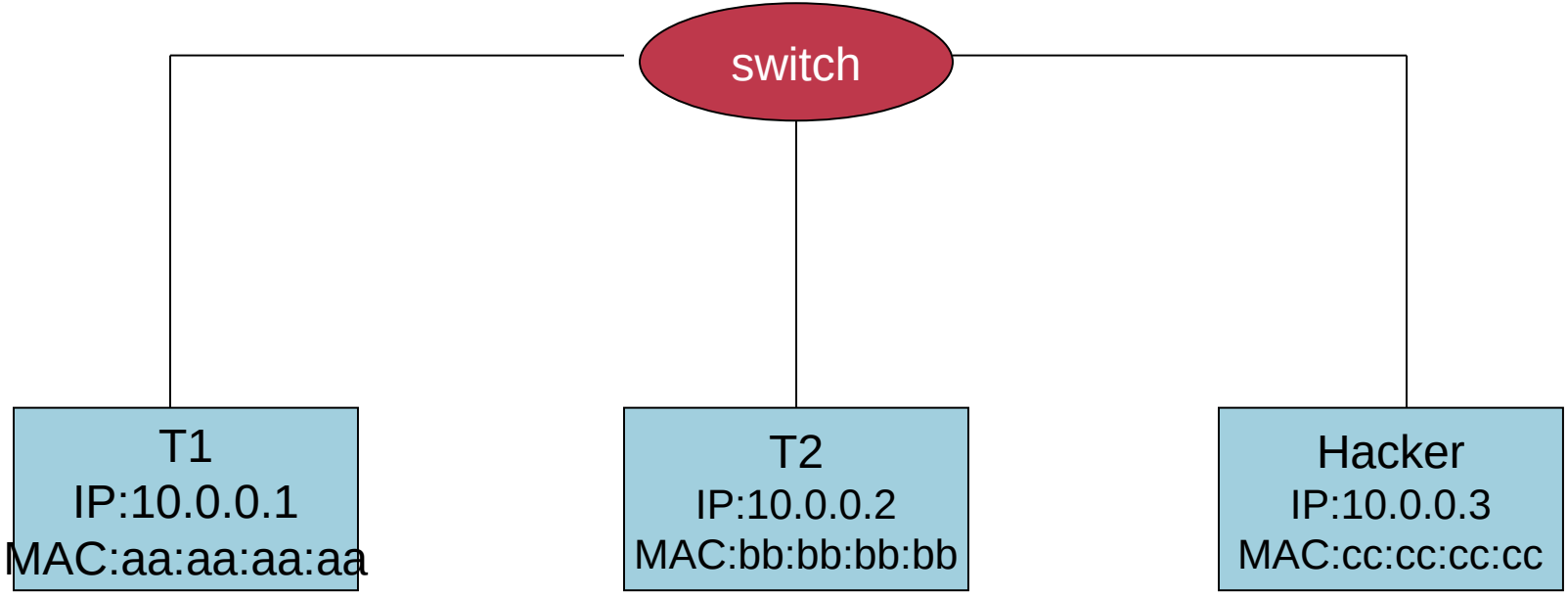


ARP cache

IP	MAC
10.0.0.2	bb:bb:bb:bb

ARP cache

IP	MAC
10.0.0.1	aa:aa:aa:aa



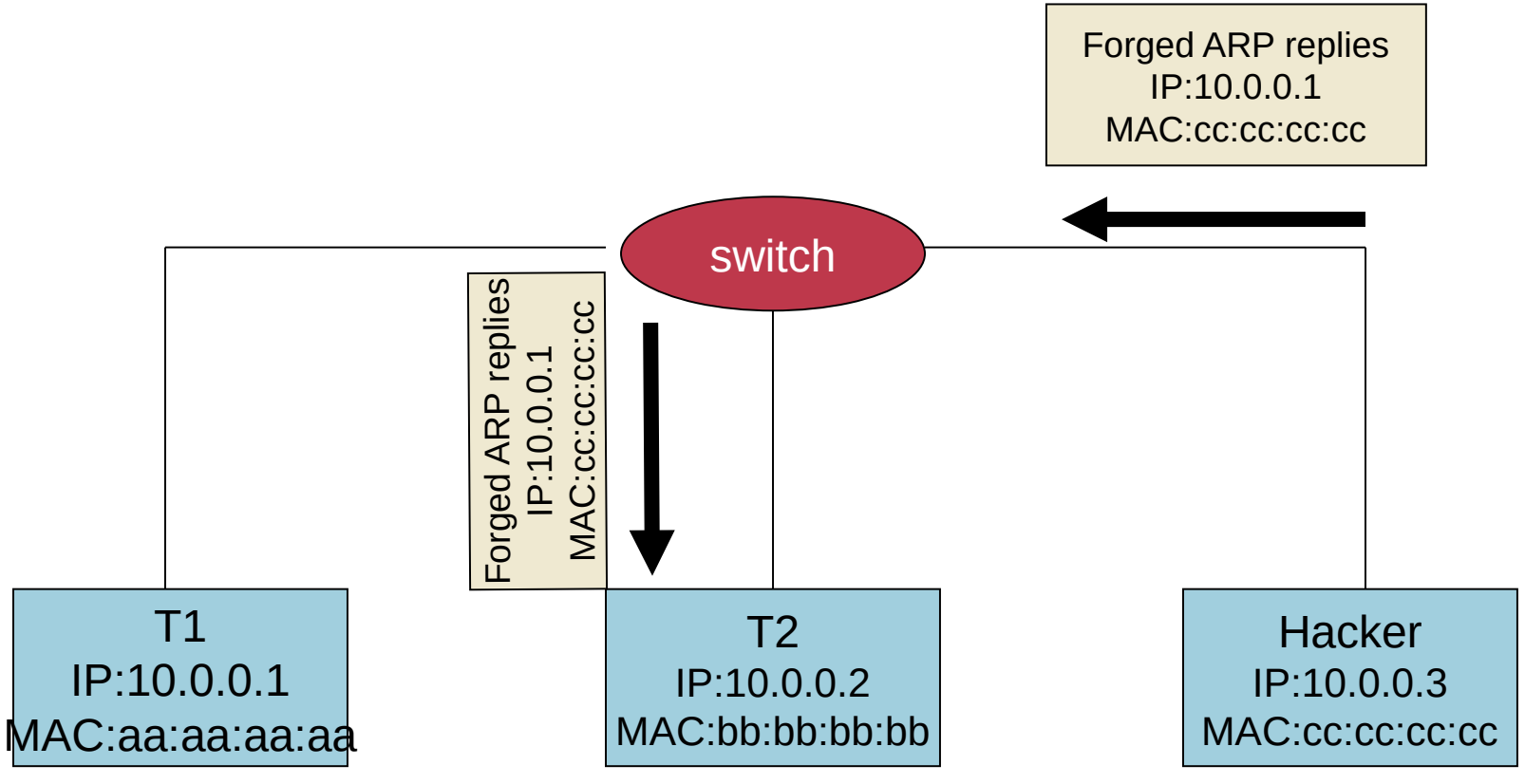
ARP cache

IP	MAC
10.0.0.2	cc:cc:cc:cc

ARP cache

IP	MAC
10.0.0.1	aa:aa:aa:aa

T1's cache is poisoned

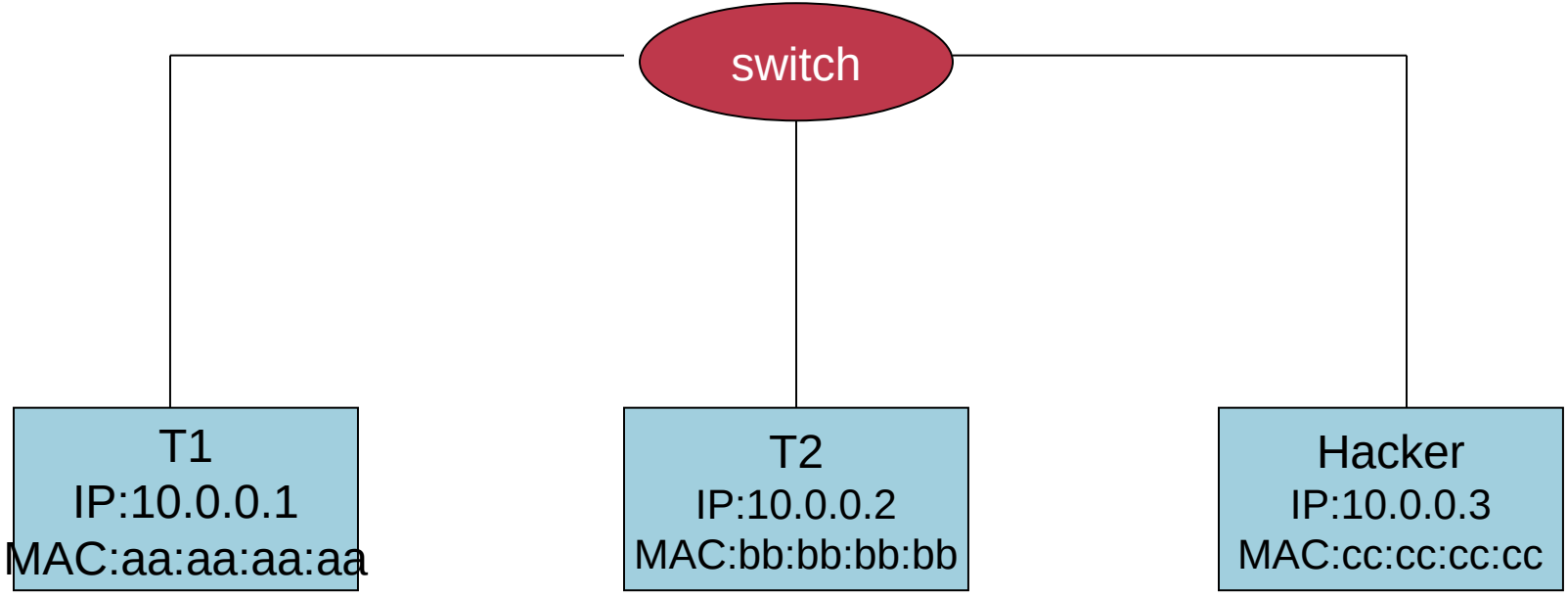


ARP cache

IP	MAC
10.0.0.2	cc:cc:cc:cc

ARP cache

IP	MAC
10.0.0.1	aa:aa:aa:aa



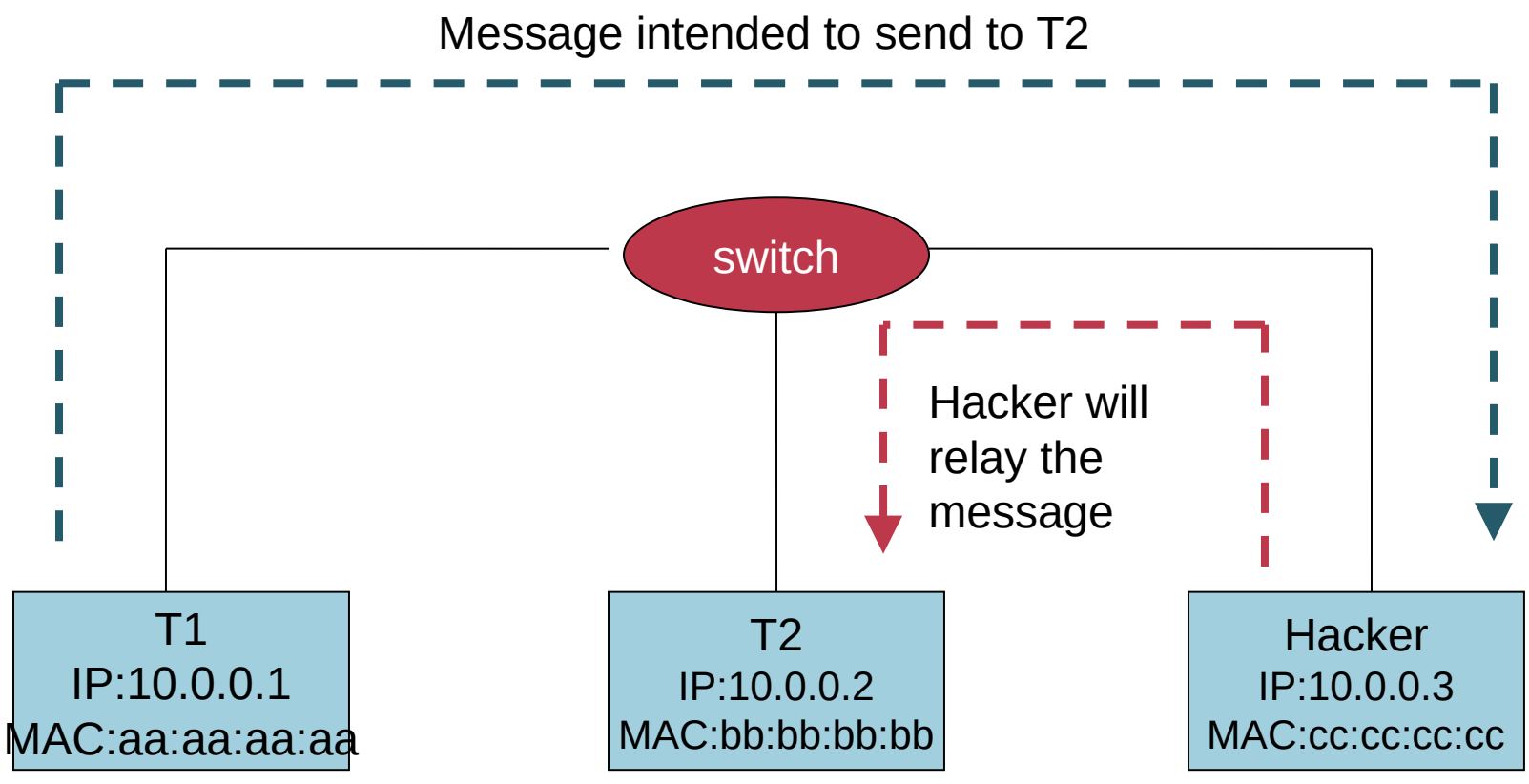
ARP cache

IP	MAC
10.0.0.2	cc:cc:cc:cc

ARP cache

IP	MAC
10.0.0.1	cc:cc:cc:cc

T2's cache is poisoned

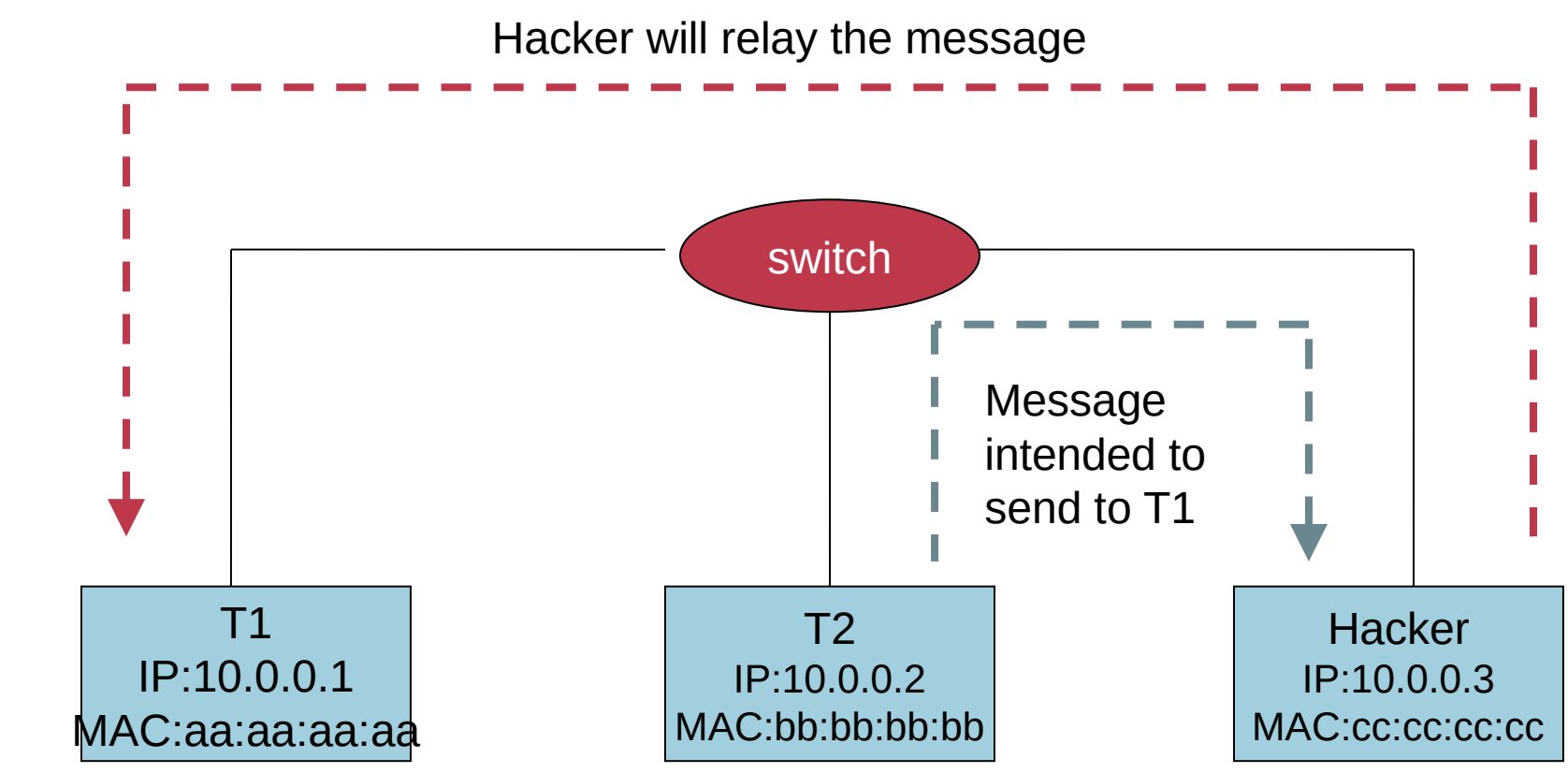


ARP cache

IP	MAC
10.0.0.2	cc:cc:cc:cc

ARP cache

IP	MAC
10.0.0.1	cc:cc:cc:cc



ARP cache

IP	MAC
10.0.0.2	cc:cc:cc:cc

ARP cache

IP	MAC
10.0.0.1	cc:cc:cc:cc

Defenses against ARP Spoofing

No Universal defense (!)

Use static ARP entries

- Cannot be updated; Spoofed ARP replies are ignored
- ARP table needs a static entry for each machine on the network
- Large overhead
  - Deploying these tables; Keep the table up-to-date

Arpwatch

- A free UNIX program which listens for ARP replies on a network
- Build a table of IP/MAC associations and store it in a file
- When a MAC/IP pair changes (flip-flop), an email is sent to an administrator

End-to-end Layer

The End-to-end Layer

Network layer has no guarantees on:

- Delay - Order of arrival
- Certainty of arrival - Accuracy of content
- Right place to deliver

End-to-end layer

- No single design is likely to suffice
- Transport protocol for each class of application

Famous Transport Protocols

UDP (User Datagram Protocol)

- Be used directly for some simple applications
- Also be used as a component for other protocols

TCP (Transmission Control Protocol)

- Keep order, no missing, no duplication
- Provision for flow control

RTP (Real-time Transport Protocol)

- Built on UDP
- Be used for streaming video or voice, etc.

No "one size fits all"

Assurance of End-to-end Protocol

1. Assurance of at-least-once delivery
2. Assurance of at-most-once delivery
3. Assurance of data integrity
4. Assurance of stream order & closing of connections
5. Assurance of jitter control
6. Assurance of authenticity and privacy
7. Assurance of end-to-end performance

1. Assurance of At-least-once Delivery

RTT (Round-trip time)

- **to_time + process_time + back_time (ack)**

At least once on best effort network

- Send packet with **nonce**
- Sender keeps a copy of the packet
- Resend if timeout before receiving acknowledge
- Receiver acknowledges a packet with its **nonce**

Try *limit* times before returning error to app

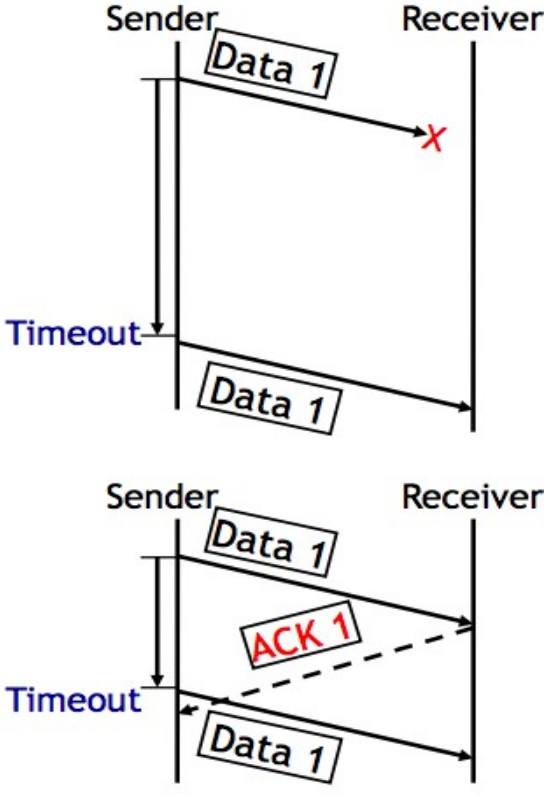
1. Assurance of At-least-once Delivery

Dilemma

- 1. The data was not delivered
- 2. The data was delivered, but no ACK received
- No way to know which situation

At-least-once delivery

- No absolute assurance for at-least-once
- Ensure if it is possible to get through, the message will get through eventually
- Ensure if impossible to confirm delivery, app will know
- No assurance for no-duplication



How to Decide Timeout?

Fixed timer: dilemma of fixed timer

- Too short? unnecessary resend
- Too long? take long time to discover lost packets

Adaptive timer

- E.g., adjust by currently observed RTT, set timer to 150%
- Exponential back-off: doubling from a small timer

NAK (Negative AcKnowledgment)

- Receiver sends a message that lists missing items
- Receiver can count arriving segments rather than timer
- Sender can have no timer (only once per stream)

Fixed Timer is Evil

Fixed timers lead to congestion collapse in NFS

Emergent phase synchronization of periodic protocols

Wisconsin time server meltdown

Emergent Phase Synchronization of Periodic Protocols

Periodic polling

- E.g. picking up mail, sending "are-you-there?"
- A workstation sends a broadcast packet every 5 minutes
- All workstations try to broadcast at the same time

Each workstation

- Send a broadcast
- Set a fixed timer

Lesson: Fixed timers have many evils. Do not assume that unsynchronized periodic activities will stay that way

Wisconsin Time Server Meltdown

NETGEAR added a feature to wireless router

- Logging packets -> timestamp -> time server (SNTP) -> name discovery -> 128.105.39.11
- Once per second (!) until receive a response
- Once per minute or per day after that

Wisconsin Univ.

- On May 14, 2003, at about 8:00 am
- From 20,000 to 60,000 requests per second, filtering 23457
- After one week, 270,000 requests per second, 150Mbps

Wisconsin Time Server Meltdown

Lesson(s)

- Fixed timers, again
- Fixed Internet address
- The client implements only part of a protocol
  - There is a reason for features likes the "*go away*" response in SNTP

How to Decide Timeout?

Fixed timer: dilemma of fixed timer

- Too short: unnecessary resend
- Too long: take long time to discover lost packets

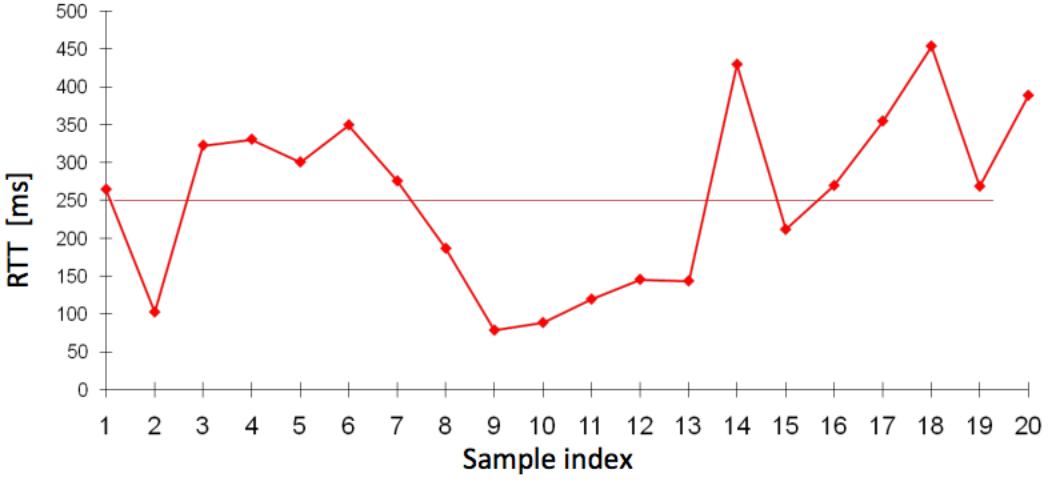
Adaptive timer

- E.g., adjust by currently observed RTT, set timer to 150%
- Exponential back-off: doubling from a small timer

NAK (Negative AcKnowledgment)

- Receiver sends a message that lists missing items
- Receiver can count arriving segments rather than timer
- Sender can have no timer (only once per stream)

RTT Could be Highly Variable



Example from a TCP connection over a wide-area wireless link
Mean RTT = 0.25 seconds; Std deviation = 0.11 seconds!

Can't set timeout to an RTT sample; need to consider variations

Calculating RTT and Timeout (in TCP)

Exponentially Weighted Moving Average

- Estimate both the average *rtt_avg* and the deviation *rtt_dev*
- Procedure *calc_rtt*(rtt_sample)
  - `rtt_avg = a*rtt_sample + (1-a)*rtt_avg; /* a = 1/8 */`
  - `dev = absolute(rtt_sample – rtt_avg);`
  - `rtt_dev = b*dev + (1-b)*rtt_dev;  /* b = 1/4 */`
- Procedure *calc_timeout(rtt_avg, rtt_dev)*
  - `Timeout = rtt_avg + 4*rtt_dev`

How to Decide Timeout?

Fixed timer: dilemma of fixed timer

- Too short: unnecessary resend
- Too long: take long time to discover lost packets

Adaptive timer

- E.g., adjust by currently observed RTT, set timer to 150%
- Exponential back-off: doubling from a small timer

NAK (Negative AcKnowledgment)

- Receiver sends a message that lists missing items
- Receiver can count arriving segments rather than timer
- Sender can have no timer (only once per stream)

2. Assurance of At-most-once Delivery

At-least-once delivery

- Remember state at the sending side
- Tends to generate duplicated requests

At-most-once delivery

- Maintains a table of nonce at the receiving side
- The table may grow indefinitely
  - Space and search time
  - Tombstones (something that cannot be deleted *forever*)
- Another way: Make the application tolerate duplicated requests
  - Recall the NFS collapse case: server wastes time to execute duplicated requests

Duplicate Suppression

Monotonically increasing sequence number

- Receiver discards smaller nonce, only holds the last nonce, one per sender
- Now, the old nonce is a *tombstone*!

Use a different port for each new request

- Should never reuse the old port number
- But, the old port is now *tombstone*!

Duplicate Suppression

Accept the possibility of making a mistake

- E.g., if sender always gives up after five RTT (cannot ensure at-least-once), then receiver can safely discard *nonces* that are older than five RTT
- It is possible that a packet finally shows up after long delay (solution: wait long time)

Receiver crashes and restarts: lose the table

- One solution is to use a new port number each time the system restarts
- Another is to ignore all packets until the number of RTT has passed since restarting, if sender tries limit times

Anyway, duplicate suppression makes the system complex

3. Assurance of Data Integrity

Data integrity

- Receiver gets the same contents as sender

Reliable delivery protocol

- Sender: adds checksum to the end-to-end layer
- Receiver: recalculates the checksum, discards if not match

Q: Is it redundant since link layer provides checksum?

- No. Besides network, there could be other errors, e.g., in memory copying

The assurance is not absolute

- What if a packet is mis-delivered and the receiver ACK?

4. Segments and Reassembly of Long Messages

Bridge the difference between message and MTU

- Message length: determined by application
- MTU: determined by network

Segment contains ID for where it fits

- E.g., "message 914, segment 3 of 7"
- Can be used for *at-least-once* and *at-most-once* delivery

Reassembly

- Out-of-order, mingled with other message's segments
- Allocating a buffer large enough to hold the message

When Out of Order…

Solution-1: Receiver only ACK in order packets, discards others

- Waste of bandwidth

Solution-2: ACK every packet and hold early packets in buffer, release the buffer when all in order

- Need using large buffer when waiting for a bad packet

Solution-3: Combine the two above

- Discard if buffer is full
- New problem: how much buffer?

Speedup for common case

- NAK to avoid timeout
- If NAKs are causing duplicates, stop NAKs

TCP is based on ACK, not NAK

5. Assurance of Jitter Control

Real-time

- When reliability is less important than timely delivery
- A few error in a movie may not be noticed
- Jitter: variability in delivery time

Strategy

- Basic: delay all arriving segments

5. Assurance of Jitter Control

Measure the distribution of delays in a chart showing delay time vs. frequency of that delay

Choose an acceptable frequency of delivery failure

Determine D_{long} that longer than 99% delay

Determine the shortest delay, D_{short}

Calculate number of segment buffer:

$$\text{Number of segment buffers} = \frac{D_{\text{long}} - D_{\text{short}}}{D_{\text{headway}}}$$

– $D_{headway}$ is average delay between arriving segments

6. Assurance of Authenticity and Privacy

Internet is dangerous

- Hostile intercepts and maliciously modifies packets
- Violate a protocol with malicious intent

Key-based mathematical transformations to data

- Sign and verify: establish the authenticity of the source and integrity of contents
- Encrypt and decrypt: maintain privacy of contents

Consideration

- False sense of security, worse than no assurance

6. Security: Asymmetric Encryption

Public Key VS. Private Key

- Public key: encrypt to identify *reader* (only me can read this)
- Private key: encrypt to identify *writer* (yes, it's me who wrote this)
- Poor performance, so just used to exchange symmetric key

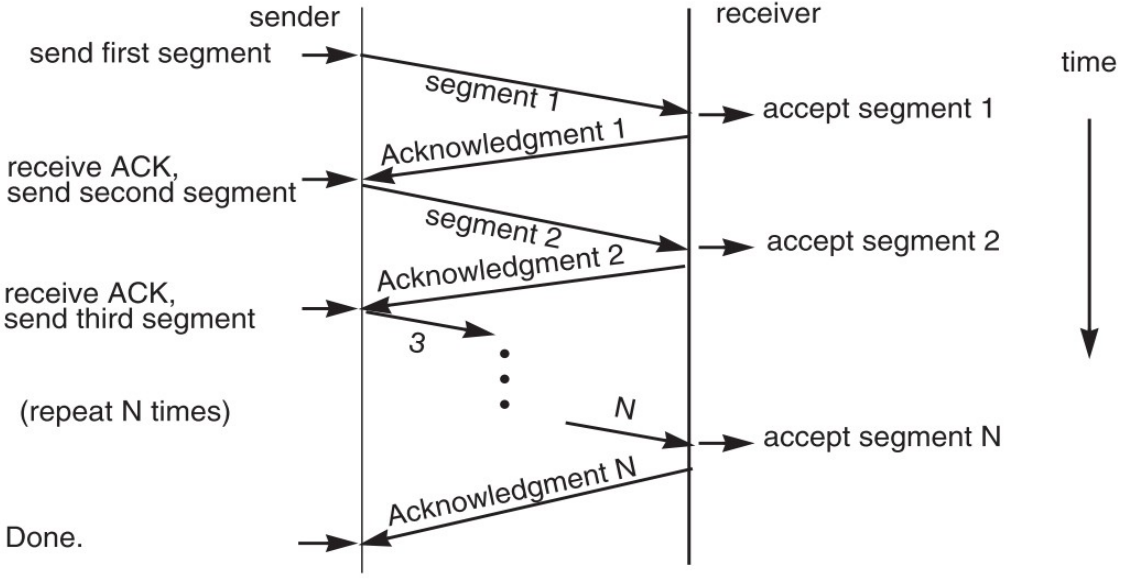
Questions

- What is a certificate? Why using a CA (Certificate Authority)?
- How to exchange a symmetric key in HTTPS or SSH?
- What is the root of trust?
- Will be addressed later, in Security part

7. End-to-end Performance

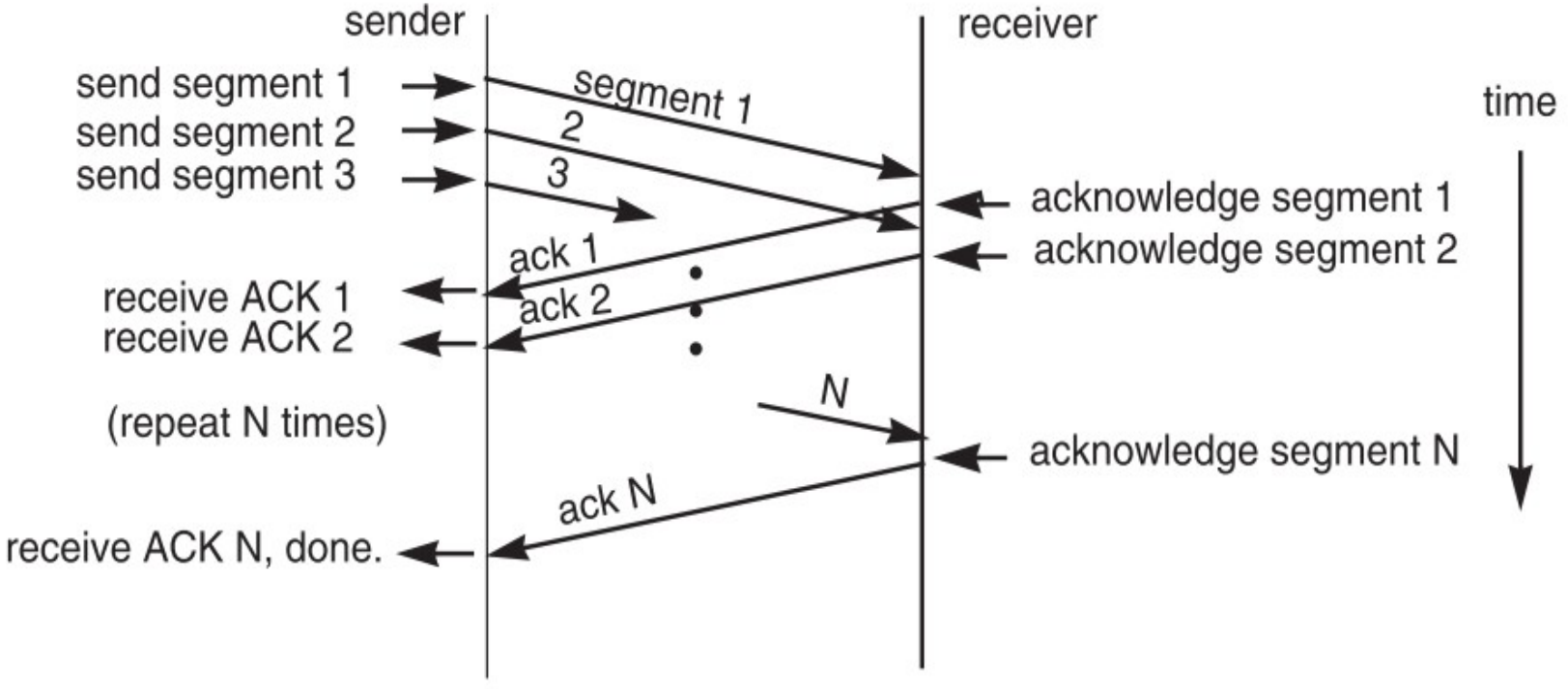
Multi-segment message questions

- Trade-off between complexity and performance
- Lock-step protocol



Overlapping Transmissions

Pipelining technique



Overlapping Transmissions

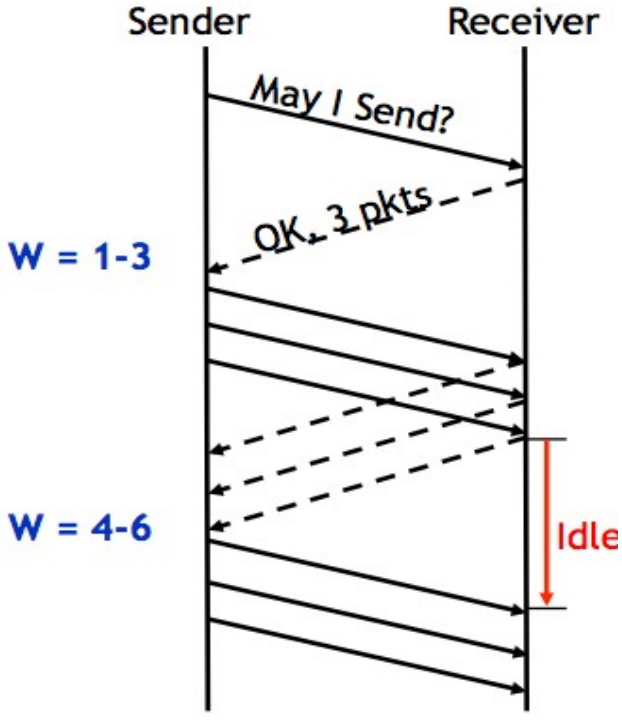
Packets or ACK may be lost

- Sender holds a list of segments sent, check it off when receives ACK
- Set a timer (according to RTT) for last segment

If list of missing ACK is empty, OK

If timer expires, resend packets and another timer

Fixed Window



Receiver tells the sender a window size

Sender sends window

Receiver acks each packet as before

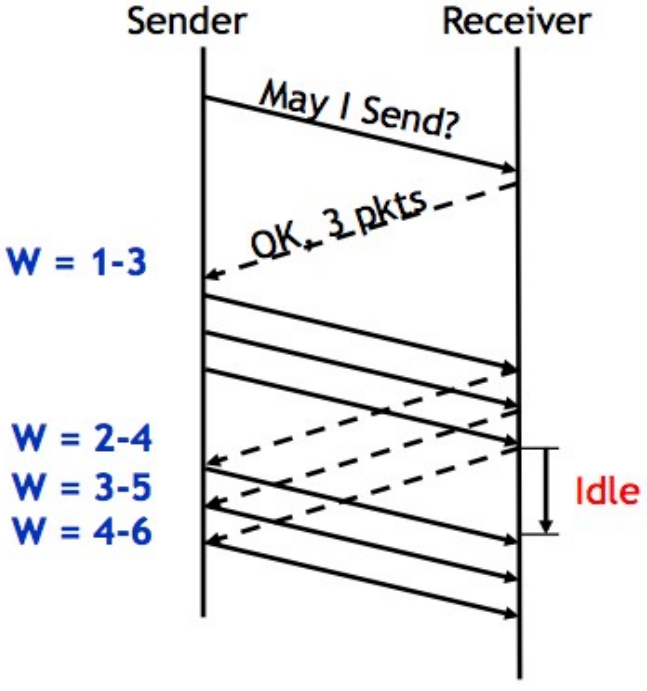
Window advances when all packets in previous window are acked

- E.g., packets 4-6 sent, after 1-3 ack'd

If a packet times out -> resend packets

Still much idle time

Sliding Window



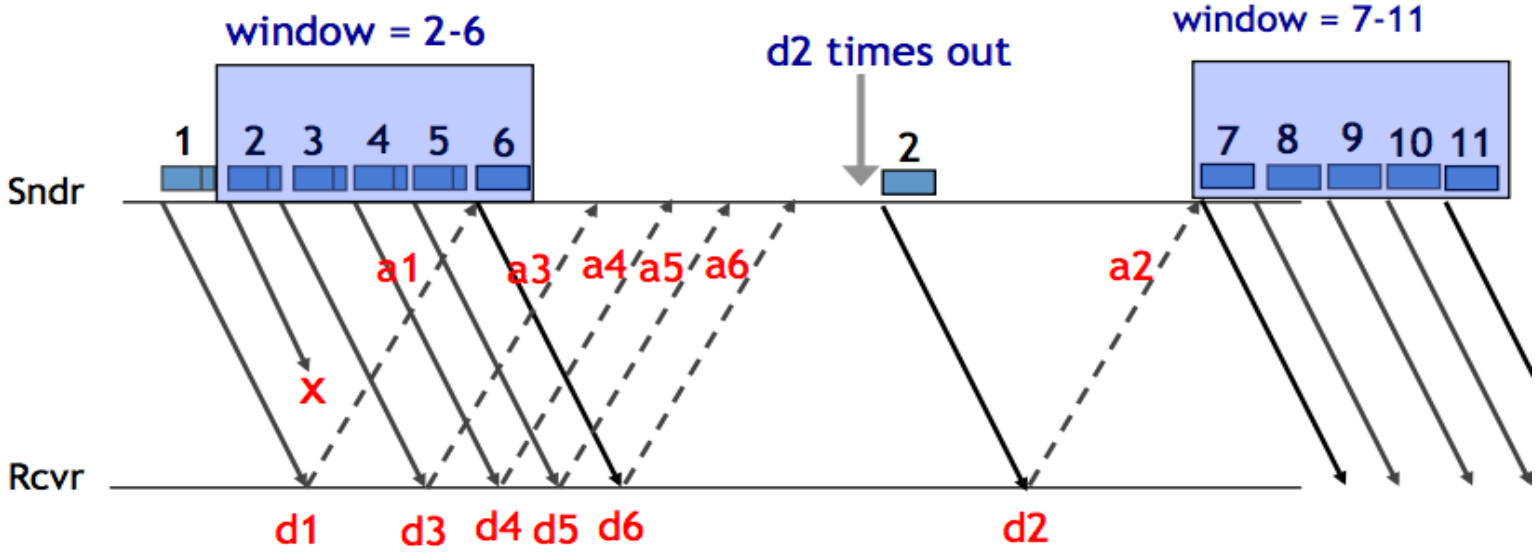
Sender advances the window by 1 for each in-sequence ACK it receives

- Reduces idle periods
- Pipelining idea

But what's the correct value for the window?

- We'll revisit this question
- First, we need to understand windows

Handling Packet Loss



Sender advances the window on arrivals of in-sequence acks

→ Can't advance on a3's arrival

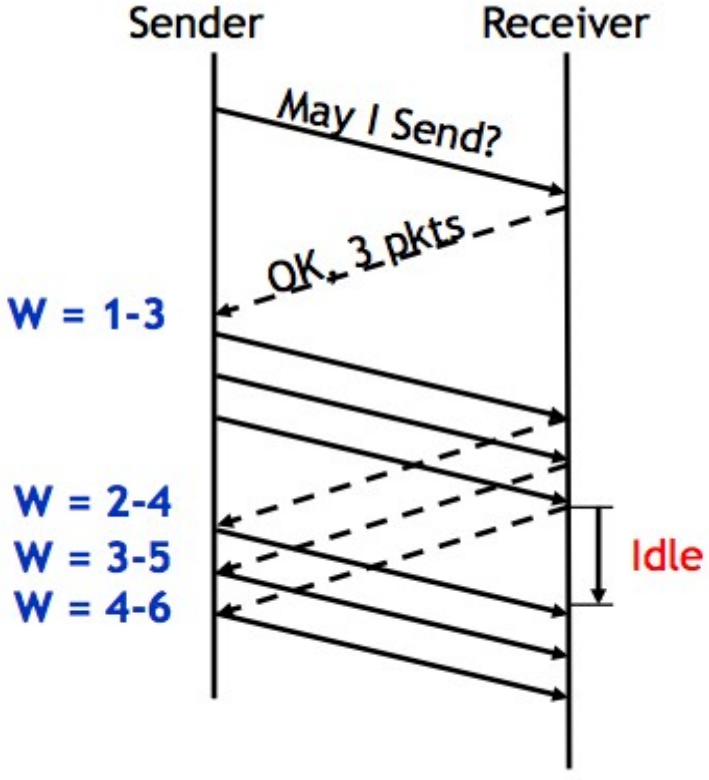
Chose the Right Window Size

If window is too small

- Long idle time
- Underutilized network

If window too large

- Congestion



Sliding Window Size

window size ≥ round-trip time × bottleneck data rate

Sliding window with one segment in size

– Data rate is window size / RTT

Enlarge window size to bottleneck data rate

– Data rate is window size / RTT

Enlarge window size further

– Data rate is still bottleneck

– Larger window makes no sense

- Receive 500 KBps
- Sender 1 MBps
- RTT 70ms
- A segment carries 0.5 KB

- Sliding window size = 35KB (70 segment)

Self-pacing: Sliding Window Size

Although the sender doesn't know the bottleneck, it is sending at exactly that rate

Once sender fills a sliding window, cannot send next data until receive ACK of the oldest data in the window

The receiver cannot generate ACK faster than the network can deliver data elements

RTT estimation still needed

TCP Congestion Control

Congestion

Definition: Too many packets present in (a part of) the network causes packet delay and loss that degrades performance.

Network & End-to-end layers *share the responsibility* for handling congestion

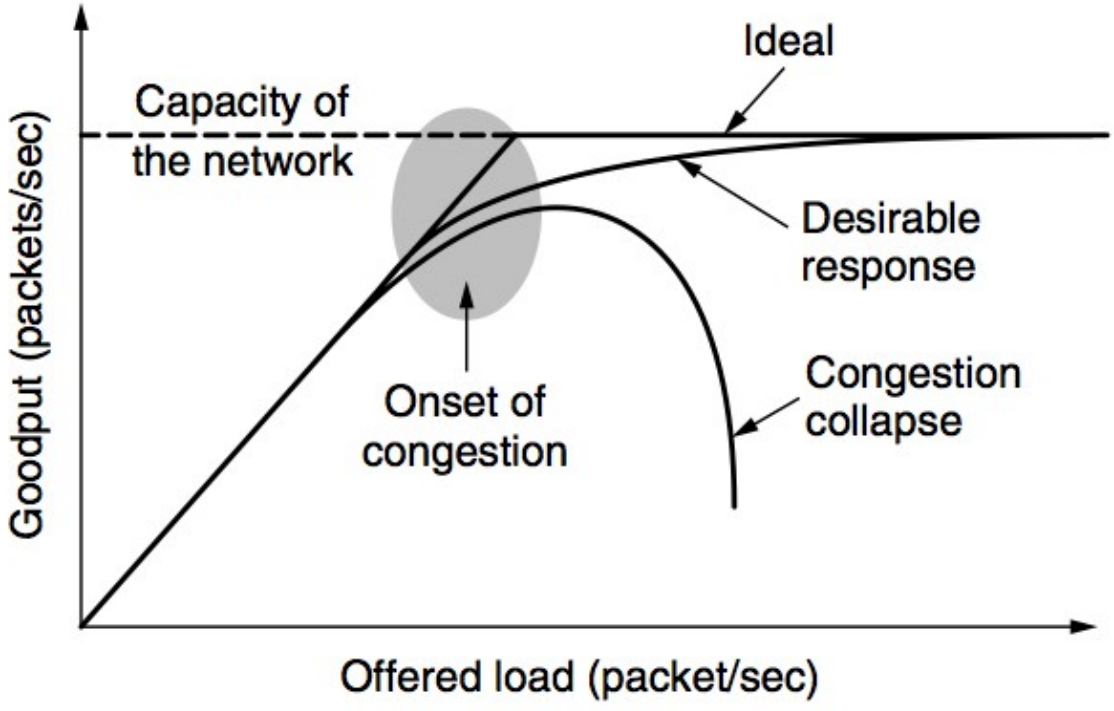
1. Network layer

- Directly experiences the congestion
- Ultimately determine what to do with the excess packets

2. End-to-end layer

- Control to reduce the sending rate, is the most effective way

Network Congestion



Why Congest?

If all of a sudden, streams of packets begin arriving on three or four input lines and all need the same output line, a queue will build up

If there is insufficient memory to hold all of them, packets will be lost

Adding more memory may help up to a point, but

- Nagle (1987) realized that if routers have an infinite amount of memory, congestion gets worse, not better
- This is because by the time packets get to the front of the queue, they have **already timed out** (repeatedly) and duplicates have been sent

Load Shedding: Setting Window Size

For performance:

- window size ≥ round-trip time × bottleneck data rate

For congestion control:

- window size ≤ min(RTT x bottleneck data rate, Receiver buffer)
- Congestion window

2 windows become 1

- to achieve best performance and avoid congestion

Congestion Control

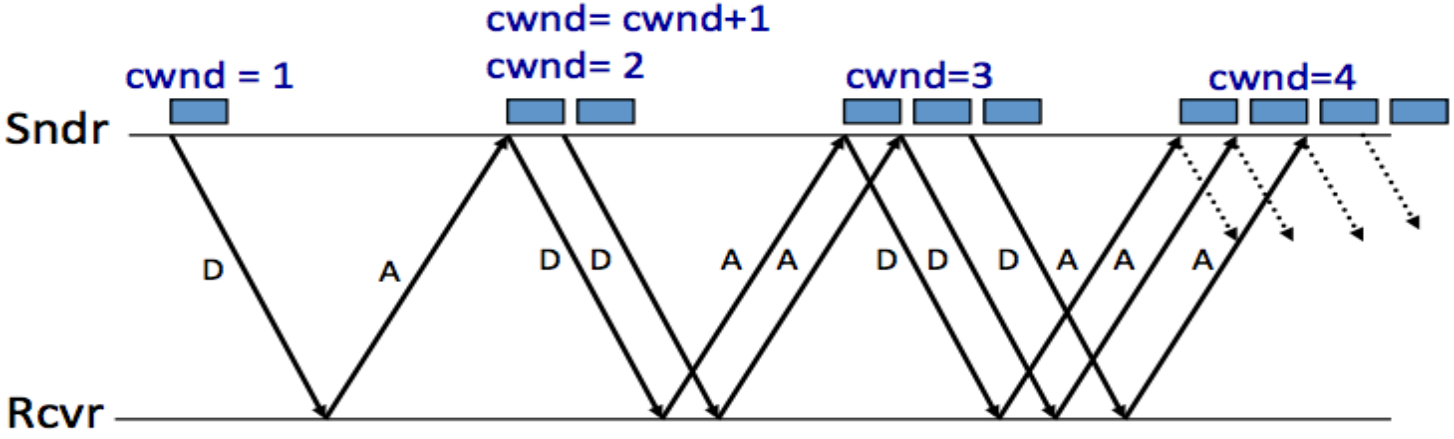
Basic idea:

- Increase congestion window slowly
- If no drops -> no congestion yet
- If a drop occurs -> decrease congestion window quickly

Use the idea in a distributed protocol that achieves:

- Efficiency: i.e., uses the bottleneck capacity efficiently
- Fairness, i.e., senders sharing a bottleneck get equal throughput (if they have demands)

AIMD (Additive Increase, Multiplicative Decrease)



Every RTT:

- No drop: cwnd = cwnd + 1
- A drop: cwnd = cwnd / 2

Problems with AIMD

Increases very slowly at the beginning

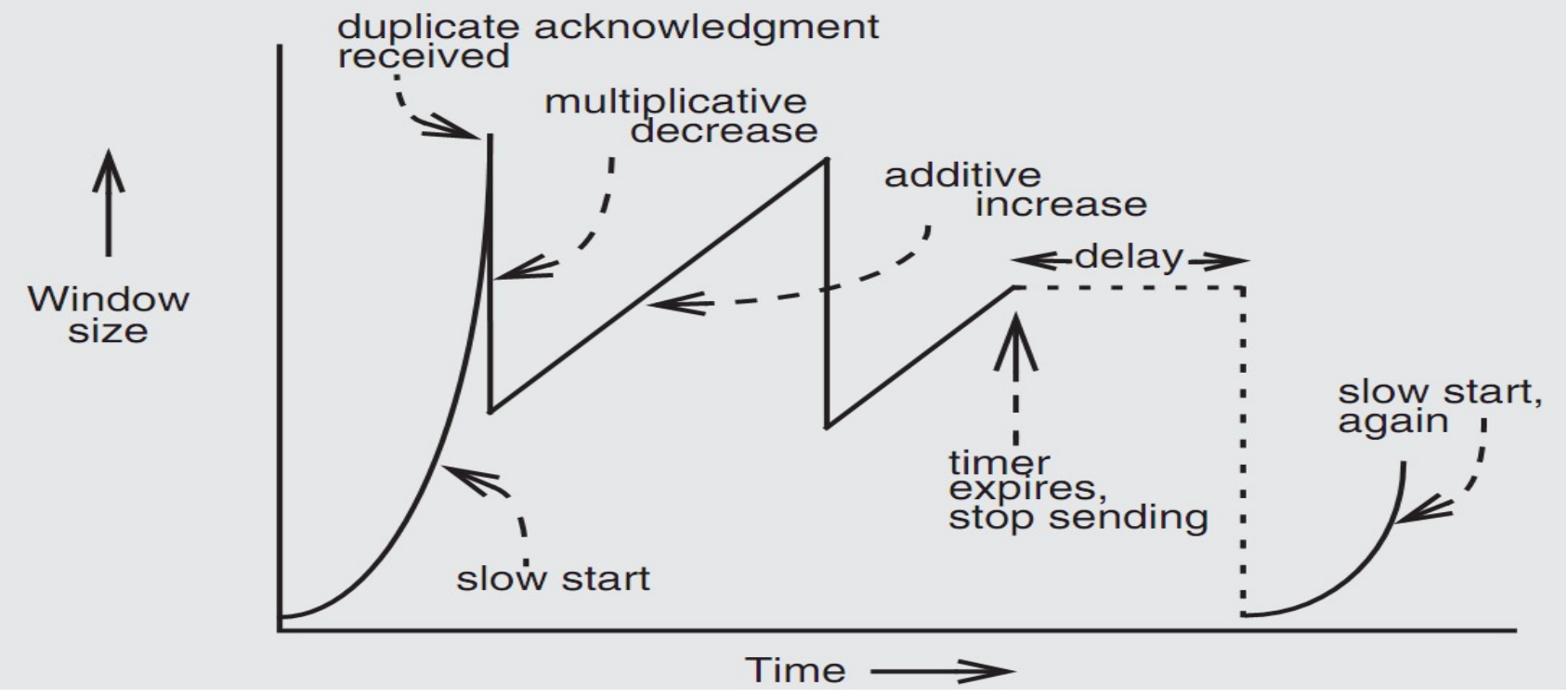
Initial window size is 1

- Probably too small in practice

Solution: do multiplicative increase at the beginning

- *Congestion_window* $_{init}$ = 1
- Initially, do *Congestion_window* ← 2 * *Congestion_window* each RTT until we hit congestion
- Named "slow start" (even though it's exponentially fast!)

Retrofitting TCP



Retrofitting TCP

1. Slow start: one packet at first, then double until

- Sender reaches the window size suggested by the receiver
- All the available data has been dispatched
- Sender detects that a packet it sent has been discarded

2. Duplicate ACK

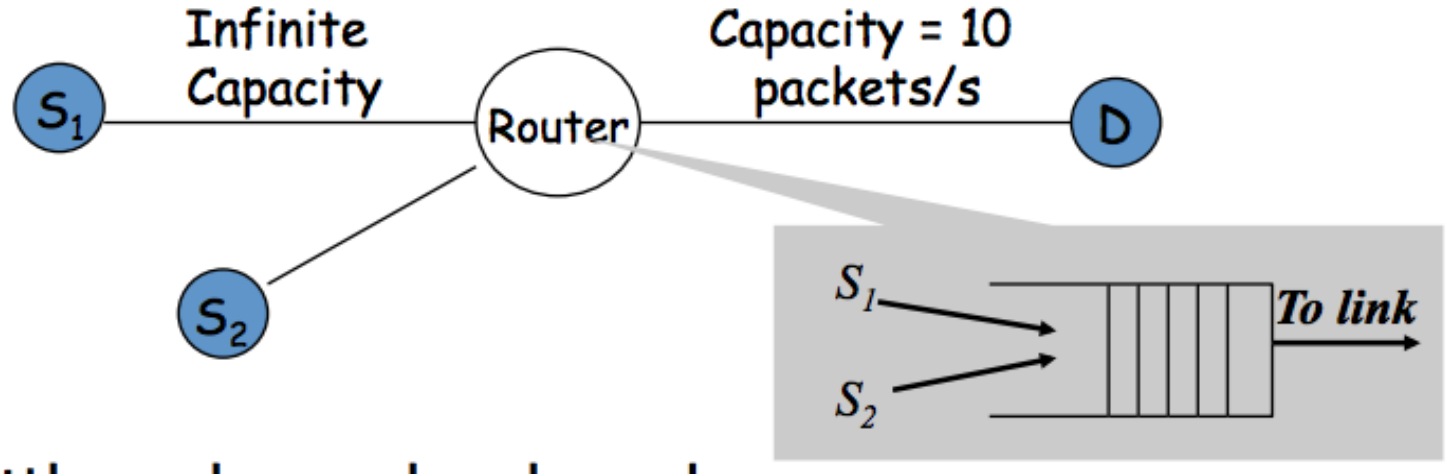
- When receiver gets an out-of-order packet, it sends back a duplicate of latest ACK

3. Equilibrium

- Additive increase & multiplicative decrease

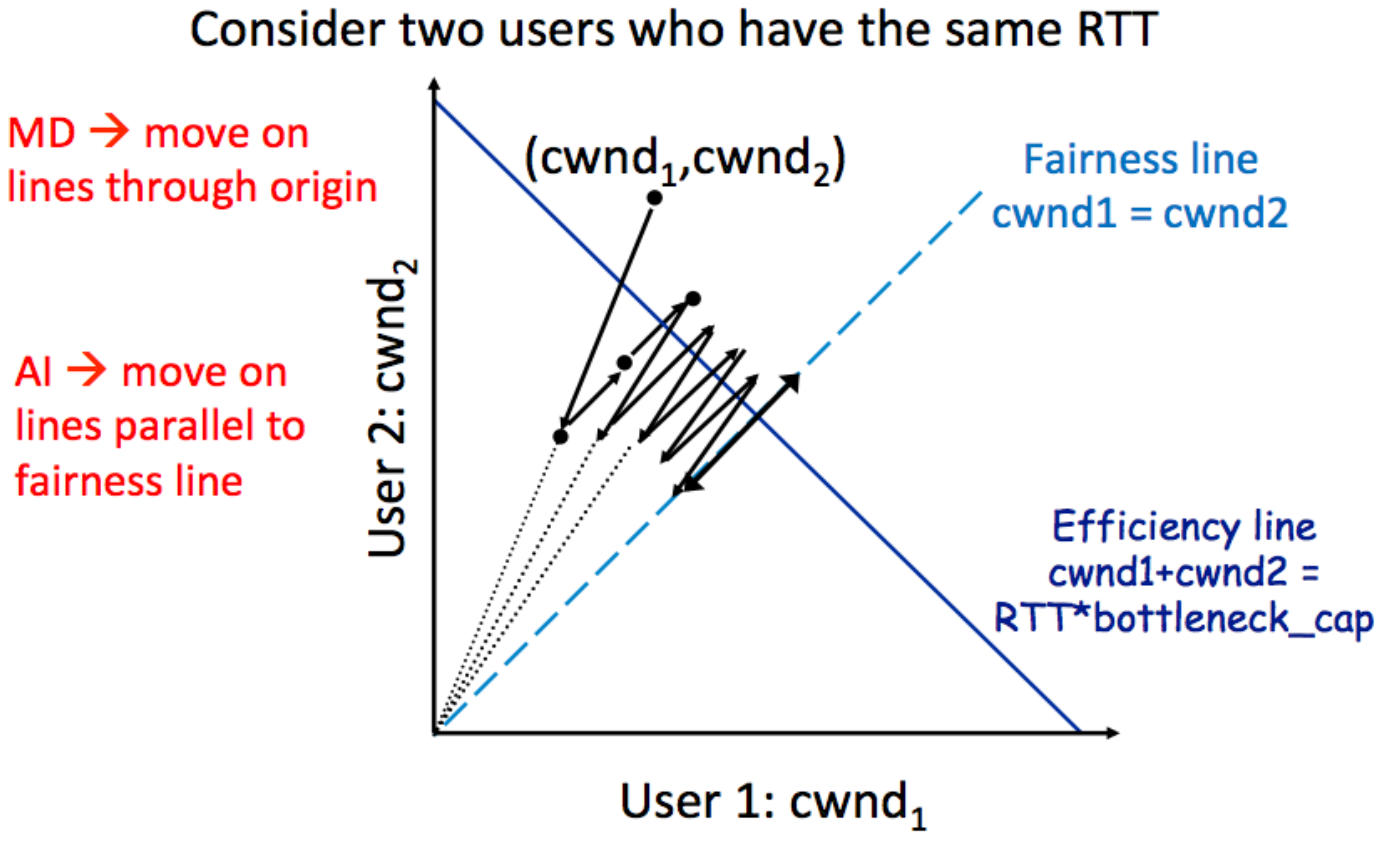
4. Restart, after waiting a short time

Fairness between Links



Bottleneck may be shared

AIMD Leads to Efficiency and Fairness



Q: Why not Additive Decrease

It does not converge to fairness

- from a congested point, (x',y'), reducing each by 1 worsens fairness and takes us away from the "ideal" outcome

Weakness of TCP

If routers have too much buffering, causes long delays

Packet loss is not always caused by congestion

- Consider wireless network: if losing packet, sender may send faster instead

TCP does not perform well in datacenters

- High bandwidth, low delay situations

TCP has a bias against long RTTs

- Throughput inversely proportionally to RTT
- Consider when sending packets really far away vs really close

Assumes cooperating sources, which is not always a good assumption

Summary of Congestion Window

Reliability Using Sliding Window

- Tx Rate = W / RTT

Congestion Control

- W = min(Receiver_buffer, cwnd)
- Congestion window is adapted by the congestion control protocol to ensure efficiency and fairness
- TCP congestion control uses AIMD which provides fairness and efficiency in a distributed way