Congestion Control

Professor Yeali S. Sun
Department of Information Management
National Taiwan University
References

Outline

- What is Congestion?
- How to Deal With Congestion?
- Congestion Avoidance (Prevention) Methods
- Congestion Handling (Reactive) Methods
- Summary
What is Congestion?

- Congestion is a *situation* (state) in which *performance degrades* due to the *saturation* of network resources such as communication links, processors and memory buffers.

- Adverse (unfortunate and harmful) effects include the *long* delays, *inefficient* resource utilization or *waste* and *possible network collapse*.

- A critical and significant issue for packet-switched networks.
Congestion at a Communication Link

Finite Buffer

λ: packet arrival rate

μ: packet service rate

Link
Congestion at a Communication Link

- When excessive traffic enters a part of a network and lasts for some period of time,
  - Input traffic load exceeds link capacity
  - Excessive traffic is temporarily stored in the buffer with finite capacity
  - When buffer is full, arriving packets are being dropped
- Bursty traffic often causes congestion
Congestion at Communication Link (cont’d)

- How about *infinite* buffers?
  - Packets still experience *long delays*
  - Packets may be *timed out* in the protocol layer and duplicates are sent
  - Worsen the situation (more packets arrive)
Transmission Bottleneck

- Slow routing processor(s)
  - e.g., packet buffering, routing table lookup, etc.
- Low-bandwidth links
  - easily becoming the bottleneck
  - capacity upgrade, bottleneck shift
More about Congestion

- Congestion tends to feed upon itself and become worse
  - e.g., TCP window-based congestion control
- Congestion handling is a global issue
  - to ensure the total offered traffic won’t exceed what the network can handle
- Research on how to accurately measure or assess bottleneck link(s)
Congestion Control Schemes

- Measures for controlling network traffic in order to *prevent, avoid, or recover* from network congestion
- Two areas of control
  - congestion avoidance
  - congestion recovery
Approaches

- Window-based (buffer) flow control
- Source quench
- Slow start/congestion avoidance in TCP
Flow Control

- To control the traffic flow between a sender and a receiver
  - point-to-point (e.g., HDLC) or end-to-end (e.g., TCP)
- To ensure that a faster sender won’t overrun a slower receiver
- Usually achieved by performing direct feedback from the receiver to tell the sender its receiving capability and status
Congestion Control

- **Cause**
  - traffic load (temporarily) is *greater* than the resources can handle

- **Solutions**
  - *Increase the Resources*
    - e.g., split traffic on multiple routes vs. one best route
    - put backup router on-line in the presence of congestion
Congestion Control (cont’d)

- **Decrease the Load**
  - Blocking new traffic, denial of service to new users
  - Degrading services to some or all users
  - Having users schedule their demands in a more predictable way.

- For networks support *virtual circuits*, congestion control can be performed at the **network layer**

- For *datagram* networks, congestion control is often performed at the **transport layer**.
Congestion Control Model

- Control Theory
- Open Loop Solution
  - Act at the source
  - Act at the destination
- Closed Loop Solution
  - Explicit feedback
  - Implicit feedback
Control Theory: Closed Loop
Open Loop-based Congestion Control

- Attempts to solve the problem by *good designs* so congestion does not occur.

- Approaches
  - schemes to decide when to *accept* new traffic, e.g., *admission control*
  - schemes to decide when to *discard* packets and which ones, e.g., active *queue management* (e.g., RED) (info)
  - appropriate *scheduling* algorithms to decide which packet to transmit (e.g., weighted fair queueing (WFQ))
Closed Loop-based Congestion Control

- Based on the *feedback from the network*
- Three parts
  - *Monitor network resources* (links and routing/switching equipment) to detect *when and where* congestion occurs
  - *Pass* congestion information to *places* where action *can* be taken
  - *Take actions* to correct the problem
Congestion Control Model: Closed Loop Solution
Closed Loop-based Congestion Control (cont’d)

- **Explicit feedback**
  - *special packets* are sent back from the point of congestion to notify traffic source affected
  - e.g., the “*rate-based congestion control scheme*” for ABR traffic in ATM networks (info)

- **Implicit feedback**
  - Source deduces the existence of congestion based on its local observations
  - e.g., in TCP (info)
Closed Loop-based Congestion Control (cont’d)

- Performance metrics (measures) for congestion detection, e.g.,
  - percentage of packet drops (due to buffer overflow)
  - buffer queue length (threshold-based)
  - number of timeout packets or retransmissions
  - packet delays (and/or variations)
How to learn about the network state?

■ **Approach #1 - Probe packets**
  ■ A *traffic source* periodically sends *probe packets* to the network.
  ■ A network node(s) (routers, switches, etc.) will *mark* probe packets when congestion is detected.

■ **Approach #2 - Congestion Indication**
  ■ Packet header carries a “*congestion indication*” field.
How to learn about the network state? (cont’d)

- It is marked when an intermediate node is under congestion.
- Receiver sends a *congestion notification message* back to sender if any packets received have this field marked.

**Approach #3 – Network Notification**

- *Network nodes automatically* issue congestion notification message to traffic sources.
- Traffic sources *reduce* sending rates.
How to Avoid or Reduce Congestion?

- **Bursty traffic** often causes network congestion
- Let a traffic source perform *traffic spacing* to reduce the degree of burstiness when injecting traffic into the network
  - to lessen *spiky* bursts
  - cannot avoid congestion but it *helps* to reduce the probability of congestion
- Let a traffic source perform *traffic shaping* so that traffic enters network in a more *predictable* rate
Traffic Shaping

- To regulate the average rate and peak rate (burstiness) of data transmission
  - sliding window limits the amount of data in transit, *not* the sending rate
- Used in ATM networks to enforce a traffic source to send data complying with the traffic description in the service contract.
- It helps the network to keep up its promised delivery QoS (more predictable traffic load).
Traffic Shaping and Policing

- A source can use a leaky bucket (or token bucket) to shape its traffic flow for traffic description conformance.
- A network can use a leaky bucket (or token bucket) for the conformance check of a traffic flow (or a connection or virtual circuit).
Admission Control

- Blocking any new traffic entering the network in the presence of congestion
- Direct traffic flow away from congested spots
Choke Packets

- Similar concepts with rate-based congestion control scheme in ATM networks
- Criterion for congestion condition
  - metrics, e.g., link utilization, queue length, etc.
  - threshold-based
- Scheme
  - A congested node (router) sends a **choke packet** to the source host
  - the original packets are tagged to avoid generating any more choke packets further along the path
Choke Packets (cont'd)

- When receiving a choke packet, a source host reduces its data sending rate to the specified destination.
- When a source does not receive any more choke packets within a period of time, it starts increasing the sending rate.
Summary

- Congestion control has been one of the major issues since the inception of networks.
- It has become a nightmare of both users and network operators.
- This problem will continue to exist.
- Congestion must be controlled in multiple ways – avoidance (good designs, admission control, traffic control, etc.), handling and recovery.
**Active Queue Management:**
Random early detection gateway for congestion avoidance

Motivation

- High-speed networks with large delay-bandwidth products
- Gateways should have large maximum queues to accommodate transient congestion.
- But it is undesirable to have large queues which were full much of the time (large average queueing delay, possibly packet drops which may cause TCP congestion avoidance phase to slow start phase)
- What desired are mechanisms to keep throughput high but average queue size low
Congestion control: various approaches

- Explicit feedback from the network
- If no explicit feedback, it is left to the transport layer protocols to infer congestion from various observations and estimates, e.g., changes in throughput, changes in round-trip time, packet drops, etc.
Random early detection: an active queue management method

- The focus is on the design of gateway-queue management.
- The idea is to monitor and control the average queue size at the gateway and notify the connections causing congestion.
- Assume FIFO queueing discipline.
  - Scales well and is easy to implement
- Other related method such as
  - DECbit congestion avoidance scheme
    - Give explicit feedback when the average queue size exceeds a certain threshold.
Global synchronization problem

- TCP – one single packet drop (BSD 4.3 TCP Tahoe) makes the connection enter the Slow-Start phase, reducing the congestion window size to one.
- It is resulted from many TCP connections reducing their windows at the same time.
- In the case of a shared queue and tail drop discipline, global synchronization may result in low utilization and throughput.
- TCP recovering from a burst of packets drops is more difficulty.
Active queue management: Fairness

- Queue management does *not* provide general fairness among flows.

- For example:
  - Two TCP connections may receive different bandwidths because they have different round-trip times.
  - A flow without using congestion control may receive more bandwidth than a flow that does.

- General fairness can be achieved by **adding per-flow scheduling** such as Weighted Fair Queueing.
Design ideas

- Randomization – choose connections to notify of congestion
- For transient congestion – a temporary increase in the queue
- Long-lived congestion – an increase in the average queue size; send congestion to randomly selected connections to decrease their windows.
- Wish to make the probability that a connection is notified is proportional to that connection’s share of the throughput through the gateway.
RED - an active queue management algorithm

- Drop arriving packets probabilistically when based on the average queue size
- **Calculate average queue size** (either in units of packets or of bytes)
  - Use a low-pass filter – exponential weighted moving average
- **Packet marking (or drop) decision**
  - Two parameters, \( \text{min}_{th} \) and \( \text{max}_{th} \) are used to determine if to mark an incoming packet.
RED - packet marking decision

- Goals
  - to ensure average queue size does not significantly exceed the maximum threshold.

- Method - compare average queue size with maximum threshold and minimum threshold

For each packet arrival
- calculate the average queue size $q_{avg}$
  - If $min_{th} \leq q_{avg} \leq max_{th}$
    - calculate probability $p_a$;
    - with probability $p_a$ mark the arriving packet;
  - else if $max_{th} \leq q_{avg}$
    - mark the arriving packet;
The RED Algorithm

Notations

$q_{avg}$: average queue size
$q_{time}$: start of the queue idle time
$count$: packets since last marked packet

$wq$: queue weight
$min_{th}$: minimum threshold for queue
$max_{th}$: maximum threshold for queue
$max_{p}$: maximum value for $pb$

$p_{a}$: current packet-marking probability
$q$: current queue size
$time$: current time
$f(t)$: a linear function of the time $t$
The RED Algorithm

Initialization:

\[ avg \leftarrow 0; \]
\[ count \leftarrow 1; \]

for each packet arrival:
   calculate the new average queue size \( avg \);
   if the queue is nonempty
      \[ q_{avg} \leftarrow (1-w_q)q_{avg} + w_q q \]
   else
      \[ m \leftarrow f(time - q_time) \]
      \[ q_{avg} \leftarrow (1-w_q)mq_{avg} \]
   if \( min_{th} \leq q_{avg} < max_{th} \)
      increment \( count \);
   calculate probability \( p_a \):
      \[ p_b \rightarrow \]
      \[ max_p(q_{avg} - min_{th})/(max_{th} - min_{th}) \]
      \[ p_a \rightarrow p_b / (1 - count \times p_b) \];
      with probability \( p_a \), mark the arriving packet;
      \[ count \leftarrow 0; \]
      else if \( max_{th} < q_{avg} \)
      mark the arriving packet;
      \[ count \leftarrow 0; \]
      else \( count \leftarrow -1; \)
when queue becomes empty
\[ q_time \leftarrow time; \]
Design guidelines

- *Average queue size* determines the degree of burstiness allowed in the gateway queue.

- Wish to mark packets at *fairly evenly-spaced intervals*.
  - to avoid biases and global synchronization, and mark packets sufficiently frequently to control the average queue size.

- The design of $p_b$
  - It varies linearly from 0 to $\max_p$
  - $\max_p$: the maximum value of $p_b$
  - Wish to assure $p_b$ increases slowly as the count increases since the last marked packet.

- $\max_p$,
  - e.g. $\max_p = 1/50$ (on average, roughly one out of 50 arriving packets is dropped.)
  - Packet marking prob. changes slowly as the average queue size changes to avoid oscillations.
  - Experiments show that never set $\max_p < 0.1$
Measure the queue in bytes instead of packets

- A packet is marked proportional to the packet size in bytes
  \[ p_b = \max_p (q_{avg} - \max_{th})/(\max_{th} - \min_{th}) \]
  \[ p_b = p_b (\text{pkt}_\text{size}/\max_{\text{pkt}_\text{size}}) \]
  \[ p_a = p_b / (1 - \text{count} \times p_b) \]

- A large FTP packet is more likely to be marked than a small Telnet packet.
Calculating $p_b$, $p_a$

- Let $X$ be the number of packet arrivals during intermarking time.
- We know:
  \[ \text{Prob}[X=n] = (1-p_b)^{n-1}p_b \]
- $X$ is a geometric random variable and $\text{E}[X] = 1/p_b$

- But … if to maintain a constant average queue size, one would need to mark packets at fairly regular intervals.
  - Namely, we do not want to have too many marked packets close together, nor too long to mark a packet. (they may cause global synchronization)
Calculating $p_b, p_a$

- So … we wish to $X$ be a uniform random variable from \{1, 2, \ldots, 1/p_b\}
- To achieve this, we need the marking probability be $p_b/(1 - \text{count} \times p_b)$
- Count – the number of unmarked packets that have arrived since last marked packet.
- We have

$$
\Pr\{X = n\} = \frac{p_b}{1 - (n - 1)p_b} \prod_{i=0}^{n-2} \left(1 - \frac{p_b}{1 - ip_b}\right)
$$

$$
= p_b \quad \text{for} \ 1 \leq n \leq 1/p_b \\
= 0 \quad \text{for} \ n > 1/p_b
$$

$$
E[X] = 1/2 p_b + 1/2
$$
Comparing packet marking under geometric and uniform probability distributions

- \( p_a = 0.02 \) for geometric
- \( p_b = 0.01, \ p_a = p_b / (1 + ip_b) \)
- They all mark roughly 100 out of 5000
Fig. 4. Simulation network.

Fig. 5. Comparing Drop Tail and RED gateways.
ATM ABR Traffic Management Framework

Figure 1. ABR traffic management model: source, switch, destination, and resource management cells.
**ABR Congestion Control**

- **A Rate-based End-to-end Closed-Loop approach**
  - *Continuous* feedbacks between network and sources.
  - Resource Management (RM) cells travel from the source to the destination and back to the source.

- **Rate-based**
  - Sources send data at a specified rate.
  - Different from TCP window-based congestion control.
    - Sources limit their transmission to a particular number of packets.
Ways of Sending Congestion Feedback to Sources

- Explicit Forward Congestion Indication (EFCI)
- Congestion Indication (CI)
- Explicit Rate (ER)
Explicit Forward Congestion Indication (EFCI)

- One bit in data cell
- Set by a congested switch
- Destination saves the EFCI state of last data cell.
- If EFCI is set when it turns around an RM cell, it uses the CI bit to give feedback to the source.
- Sender may adjust its sending rate accordingly
- Binary or EFCI switches.
Congestion Indication (CI)

- RM cells have two bits in the payload:
  - CI bit
  - no increase (NI) bit
- CI bit is set by a congested switch if severe congestion occurs.
- Relative rate marking switches.
Explicit Rate (ER)

- A field in the payload of an RM cell
- Can be reduced by congested switch to any desired value.
- Explicit rate switches.
- Under certain circumstances, congested switches can generate RM cells and send them immediately to the sources.
TCP: Congestion Control
TCP Congestion Control

- Slow start
- Congestion avoidance
Slow-Start

- To get data flowing there must be **acks** to clock out packets; but to get **acks** there must be data flowing.

- Maintain a per connection state variable in the sender – "**congestion window**" \( cwnd \)

- "**When to enter Slow-Start Phase?**"
  - When a connection begins
  - After a timeout
Slow Start (cont'd)

Algorithm –

- When starting or restarting after a loss, set $cwnd=1$ packet.

- One ack for each new data, i.e. $cwnd = cwnd + 1$.

- When sending, send the $\min(\text{receiver's\_adviseWin}, cwnd)$
Details of Slow Start

- Each time an ACK is received, $cwnd$ is incremented by one segment size.
  - $cwnd$ is maintained in bytes.
  - The *segment size* is announced by the *receiver*.
TCP Slow Start (continued)

- When connection begins, increase rate **exponentially** until first loss event:
  - double $cwnd$ every RTT
  - done by incrementing $cwnd$ for every ACK received

- **Summary:** initial rate is slow but ramps up exponentially fast

$\rightarrow$ PROBE network maximum "throughput"!
Congestion Avoidance

- Congestion is indicated by a *timeout* or the reception of *three* duplicate ACKs.

- The goal is to avoid increasing the window size too quickly and causing additional congestion.
**Congestion Avoidance Algorithm**

- **Slow start phase**
  - When a connection begins: \( \text{cwnd} \) is one segment and \( \text{ssthresh} \) (slow start threshold) is 65,535 bytes.
  - When congestion occurs, \( \text{ssthresh} = \text{cwnd}/2 \), \( \text{cwnd} = 1 \)

- Once \( \text{cwnd} = \text{ssthresh} \), the connection enters the congestion avoidance phase.
  - On each ack for new data, \( \text{cwnd} = \text{cwnd} + 1/\text{cwnd} \) (additive increase)
  - When sending, send the \( \text{min(receiver’s AdvertiseWindow, cwnd)} \)
**AIMD**: additive increase, multiplicative decrease

- **cwnd**: segments
- **Slow Start**
- **Congestion Avoidance**
- **Additive Increase**
- **Multiplicative Increase**

※ Visualization of slow start and congestion avoidance.
Duplicate ACKs

- If there are less than 3 duplicate ACKs, it is assumed that there is just a *reordering* of the segments.
- If 3 or more duplicate ACKs are received in a row, it is a *strong indication* that a segment has been lost.
- **Fast Retransmit** and **Fast Recovery**
  - -> TCP-tahoe and TCP-reno
Fast Retransmit

- When 3 duplicate ACKs are received, a retransmission is performed *without* waiting for a retransmission timer to expire.

  - `ssthresh = cwnd/2` and `cwnd = 1`;
    (entering Slow Start phase)

- Retransmit the missing segment.
TCP Tahoe

Slow Start

- \( \text{cwnd} = \text{ssthresh} / 2 \)
- \( \text{cwnd} = 1 \)
- Three duplicate ACKs
- \( \text{ssthresh} = \text{cwnd} / 2 \)
- \( \text{cwnd} = 1 \)
- \( \text{cwnd} += 1 / \text{cwnd} \) per ACK

Retransmit the lost packet

Fast Retransmit

- \( \text{cwnd} = \text{cwnd} + 1 \) per ACK
- \( \text{ssthresh} = \text{cwnd} / 2 \)
- \( \text{cwnd} = 1 \)

Timeout

- \( \text{ssthresh} = \text{cwnd} / 2 \)
- \( \text{cwnd} = 1 \)

Congestion Avoidance
TCP Tahoe

- After fast retransmit, goes to “slow-start” phase to probe the network again.

- To avoid congest the network.
Fast Recovery

- Immediately after fast retransmit, instead of entering slow start, *congestion avoidance* is performed.
- To boot up throughput
  - \( ssthresh = \frac{cwnd}{2}; \quad cwnd = ssthresh + 3 \) segments
  - Each time an ACK or a duplicate ACK arrives, increment \( cwnd \) by the segment size \( cwnd++; \)
- Allow to transmit new packet
Fast Recovery (cont’d)

- When the next ACK arrives that acknowledges the lost data,
  - set $cwnd$ to $ssthresh$
  - enter congestion avoidance phase
TCP Reno

Slow Start
- \( cwnd = cwnd + 1 \)
- \( \text{timeout} \)
  - \( ssthresh = cwnd / 2 \)
  - \( cwnd = 1 \)
- \( \text{per ACK} \)

Fast Retransmit
- \( cwnd = 1 \)
- \( \text{timeout} \)
  - \( ssthresh = cwnd / 2 \)
  - \( cwnd = ssthresh + 3 \)
- \( \text{Three duplicate ACKs} \)
- \( \text{retransmit lost packet} \)

Congestion Avoidance
- \( cwnd = cwnd + \frac{1}{cwnd} \)
- \( \text{per ACK} \)
- \( \text{timeout} \)
  - \( ssthresh = cwnd / 2 \)
  - \( cwnd = 1 \)

Fast Recovery
- \( cwnd = ssthresh \)
- \( \text{retransmit lost packet} \)
- \( \text{three duplicate ACKs} \)
  - \( ssthresh = cwnd / 2 \)
  - \( cwnd = ssthresh + 3 \)
- \( \text{recovered lost packet} \)
  - \( cwnd = ssthresh + 3 \)
- \( cwnd = cwnd + 1 \)
  - \( \text{per ACK} \)
TCP-Reno: Congestion Window Size

(1) After every new acknowledgment
   \[ \text{if } (\text{CWND} < \text{SSTHRESH}) \]
   \[ \text{CWND} \leftarrow \text{CWND} + 1 \]
   \[ \text{else} \]
   \[ \text{CWND} \leftarrow \text{CWND} + 1/\text{CWND} \]

(2) Upon RTO (retransmission timeout)
   \[ \text{SSTHRESH} \leftarrow \text{CWND}/2 \]
   \[ \text{CWND} \leftarrow 1 \]

(3) When NDUP (# of duplicate ACKs) exceeds 3
   \[ \text{SSTHRESH} \leftarrow \text{CWND}/2 \]
   \[ \text{CWND} \leftarrow \text{CWND}/2 + 3 \]
Competition of TCP connection with UDP flow

- Sender 1 (140.112.8.162) 先以 8 Mbps 的速度送出 TCP traffic
- 20 秒後 Sender 2 (140.112.10.177) 再以 8 Mbps 的速度送出 UDP traffic
- The buffer space is 100KB for both queues. There is no packet drop.
- After UDP traffic starts, TCP throughput drops to less than 2Mb，UDP has the rest。
- Possible cause: Receiver (140.112.9.165) fails to send ACKs to Sender 1, causing Sender 1以为發生packet loss，因此把window size调降，而使得傳送的速率下降。
Competition of TCP connection with UDP flow (cont’d)