Wireless Networks
(CSC-7602)
Lecture 8
(22 Oct. 2007)

Seung-Jong Park (Jay)
http://www.csc.lsu.edu/~sjpark

Fair Queueing
Today

- Wireline Queue Drop
- Wireless Queue Drop

Types of Congestion Control Strategies

- Limit entry into the system
  - Packet level (layer 3)
    - Leaky Bucket, token bucket, WFQ
  - Flow/conversation level (layer 4)
    - Resource reservation
    - TCP backoff/reduce window
  - Application level (layer 7)
    - Limit types/kinds of applications
- Terminate existing resources
  - Drop packets
  - Drop circuits
Leaky Bucket: Analogy

- Across a single link, only allow packets across at a constant rate
- Packets may be generated in a bursty manner, but after they pass through the leaky bucket, they enter the network evenly spaced
- If all inputs enforce a leaky bucket, it's easy to reason about the total resource demand on the rest of the system

Leaky Bucket (cont’d)

- The leaky bucket is a “traffic shaper”: It changes the characteristics of a packet stream
- Traffic shaping makes the network more manageable and predictable
- Usually the network tells the leaky bucket the rate at which it may send packets when a connection is established

- Leaky Bucket: Doesn’t allow bursty transmissions
  - In some cases, we may want to allow short bursts of packets to enter the network without smoothing them out
  - For this purpose we use a token bucket, which is a modified leaky bucket
Token Bucket

- The bucket holds logical tokens instead of packets.
- Tokens are generated and placed into the token bucket at a constant rate.
- When a packet arrives at the token bucket, it is transmitted if there is a token available. Otherwise, it is buffered until a token becomes available.
- The token bucket holds a fixed number of tokens, so when it becomes full, subsequently generated tokens are discarded.
- Can still reason about total possible demand.

Token Bucket vs. Leaky Bucket

Case 1: Short burst arrivals

Arrival time at bucket:

Departure time from a leaky bucket:
- Leaky bucket rate = 1 packet / 2 time units
- Leaky bucket size = 4 packets

Departure time from a token bucket:
- Token bucket rate = 1 tokens / 2 time units
- Token bucket size = 2 tokens
Token Bucket vs. Leaky Bucket

Case 2: Large burst arrivals

Arrival time at bucket

Departure time from a leaky bucket
Leaky bucket rate = 1 packet / 2 time units
Leaky bucket size = 2 packets

Departure time from a token bucket
Token bucket rate = 1 token / 2 time units
Token bucket size = 2 tokens

Queue Drop Policy for Wire Link
Packet dropping

- Packets that cannot be served immediately are buffered
- Full buffers => packet drop strategy
- Packet losses happen almost always from best-effort connections (why?)
- Shouldn’t drop packets unless imperative
  - packet drop wastes resources (why?)

![Diagram of packet dropping](image)

Early vs. late drop

- Early drop => drop even if space is available
  - signals endpoints to reduce rate
  - cooperative sources get lower overall delays, uncooperative sources get severe packet loss
- Early random drop
  - drop arriving packet with fixed drop probability if queue length exceeds threshold
  - intuition: misbehaving sources more likely to send packets and see packet losses
  - doesn’t work!
Early vs. late drop: RED

- Random early detection (RED) makes three improvements
  - Metric is moving average of queue lengths
    - small bursts pass through unharmed
    - only affects sustained overloads
  - Packet drop probability is a function of mean queue length
    - prevents severe reaction to mild overload
  - Can mark packets instead of dropping them
    - allows sources to detect network state without losses
  - RED improves performance of a network of cooperating TCP sources
  - No bias against bursty sources
  - Controls queue length regardless of endpoint cooperation

Drop position

- Can drop a packet from head, tail, or random position in the queue
- Tail
  - easy
  - default approach
- Head
  - harder
  - lets source detect loss earlier
Drop position (contd.)

- Random
  - hardest
  - unlikely to make it to real routers
- Drop entire longest queue
  - easy
  - almost as effective as drop tail from longest queue

Randomization in Router Queue Management

- normally, packets dropped only when queue overflows
  - "drop-tail" queueing

![Diagram of Internet routing](image)
The case against drop-tail queue management

- large queues in routers are "a bad thing"
  - end-to-end latency dominated by length of queues at switches in network
- allowing queues to overflow is "a bad thing"
  - connections transmitting at high rates can starve connections transmitting at low rates
  - connections can synchronize their response to congestion
    - Global synchronization

Idea: early random packet drop

when queue length exceeds threshold, drop packets with queue length dependent probability
- probabilistic packet drop: flows see same loss rate
- problem: bursty traffic (burst arrives when queue is near threshold) can be over penalized
Random early detection (RED) packet drop

- Use exponential average of queue length to determine when to drop
  - Avoid overly penalizing short-term bursts
  - React to longer term trends
- Tie drop prob. to weighted avg. queue length
  - Avoids over-reaction to mild overload conditions
Three different types of challenges are posed to TCP in MANET

- 1. Mobility causes changing of topology and route change, cause TCP goes to exponentially back-off
- 2. The second problem deal with congestion window in use.
- 3. The third problem is significant TCP unfairness.
- This paper focuses on the third problem.
RED can improve congestion control and fairness in wired network

- Suppose the current queue size is \( q \)
- The avg queue size is computed as

\[
\text{avg} = (1 - w_q) \times \text{avg} + w_q \times q
\]

\( w_q \) is the queue weight

\[
p_b = \max_p \left( \frac{\text{avg} - \min_{th}}{(\max_{th} - \min_{th})} \right)
\]

\[
p_a = p_b / (1 - \text{count} \times p_b)
\]

\( \max_p \) is the maximum packet drop probability and \( \text{count} \) is the number of packets arrived since last packet drop.

TCP UNFAIRNESS AND RED IN MANET

- FTP 2 is always starved. RED does not improve fairness.
  This is because congestion does not happen in a single node, but in an entire area.
Neighborhood and its Distributed Queue

- Neighborhood: A node’s neighborhood consists of the node itself and the nodes which can interfere with this node’s signal, which includes 1-hop neighbors and 2-hop neighbors.
- (normally interference range is much larger than data transmission range, which is simplified in this paper).

The main idea of this paper is to treat this distributed queue of a neighborhood in a MANET the same way as we would on a single link queue in a wired net and apply RED to it.

1. A neighborhood queue consists of multiple queues located at the neighboring nodes that are part of the same spatial reuse constraint set.

2. Multiple sub-queues have different relative priorities in terms of acquiring the wireless channel due to various factors including MAC unfairness, channel capture, hidden and exposed terminal etc.

3. The priority of a sub-queue may change dynamically due to topology or traffic pattern changes.
NEIGHBORHOOD RANDOM EARLY DETECTION

- Make packet drop probability and packet delay proportional to the share of bandwidth used by each TCP flow.

- 1. Neighborhood Congestion Detection (NCD)
- 2. Neighborhood congestion Notification (NCN)
- 3. Distributed Neighborhood Packet Drop (DNPD)

Neighborhood Congestion Detection (NCD)

- When a packet in any outgoing queue is transmitted, node A will detect the medium as busy.
- When a packet is received to any incoming queue, node A can also learn this through the CTS packet.
Neighborhood Congestion Detection (NCD)

- A node will monitor 5 different radio states
- 1. Transmitting
- 2. Receiving
- 3. Carrier sensing busy (RTS, CTS)
- 4. Virtual carrier sensing busy
- 5. Idle

State 1&2 is for the current node, 3&4 is for its neighbors. The authors assume state 5 means an empty queue.

Neighborhood Congestion Detection (NCD)

\[ U_{busy} = \frac{T_{interval} - T_{idle}}{T_{interval}}; \ (2) \ U_{tx} = \frac{T_{tx}}{T_{interval}}; \ (3) \ U_{rx} = \frac{T_{rx}}{T_{interval}}; \]
\[ T_{interval} = T_{tx} + T_{rx} + T_{cs} + T_{idle} \]

Assume \( W \) is channel bandwidth and the average packet size is \( C \) bits
\[ q = \frac{U_{busy} \cdot W}{C};\ \text{avg} = (1 - w_q) \cdot \text{avg} + w_q \cdot q \]

We can use the same way to calculate \( \text{avg}_{tx} \) and \( \text{avg}_{rx} \).
Distributed Neighborhood Packet Drop

When a node received a NC N with a none zero normalize dPb, the local drop prob pb is caculated as normalizedPb* (avg_tx+avg_rx).

Algorithm 5.2: RANDOM DROP()  

comment: Actions performed at the outgoing queue  

Saved Variables:  

\( \text{count}_{tx} \): outgoing pkts arrived since last drop  

\( \text{avg}_{tx} \): average outgoing queue size  

Other Parameters:  

\( p_u \): current packet dropping probability

for each packet arrived  

\( \text{count}_{tx} \) = \( \text{count}_{tx} \) + 1  

If normalizedPb < 1

\( p_u = \text{normalizedPb} \times \text{avg}_{tx} \)

else  

\( p_u = 1 \)

If \( p_u > 0 \)  

\( a\text{RandomNumber} = \text{random}([0, 1]) \)

If \( a\text{RandomNumber} \leq p_u \)  

drop the arriving pkt  

\( \text{count}_{tx} = 0 \)

else  

\( \text{count}_{tx} = -1 \)

VERIFICATION AND PARAMETER TUNING

- Verification of Queue
- Size Estimation

Figure 5: Scenario for verifying queue size estimation algorithm.

Figure 6: Estimated average queue size and the real average queue size of Node 5’s neighborhood under FTP/TCP connections.

Figure 7: Estimated average queue size and the real average queue size of Node 5’s neighborhood under HTTP/TCP connections.
VERIFICATION AND PARAMETER TUNING

- Parameter Tuning with Basic Scenarios with hidden and exposed terminal scenario.

![Diagram of network topologies](image)

**Figure 8:** The hidden terminal scenario, where Node 2 is hidden by transmission from node 4 to node 3 and Node 3 is hidden by transmission from node 1 to node 2.

![Diagram of network topologies](image)

**Figure 9:** The exposed terminal scenario, where node 2 is exposed to transmissions from node 3 to node 4.

VERIFICATION AND PARAMETER TUNING

- Fairness index
  
  \[ F(X_1, X_2) = \frac{(X_1 + X_2)^2}{2(X_1^2 + X_2^2)} \]

- MAXMin fairness is bounded between 0 and 1

![Graphs of fairness index](image)
VERIFICATION AND PARAMETER TUNING

- Aggregated Throughput (kbps) under hidden and exposed terminal situation

---

PERFORMANCE EVALUATION OF NRED

- Overall throughput of the 3 flows.
PERFORMANCE EVALUATION OF NRED

- Multiple Congested Neighborhood

![Diagram](image1.png)

Figure 15: Scenario of the multiple congested neighborhood topology.

![Graph](image2.png)

Figure 16: Overall throughput of each flow with and without NRED.

---

PERFORMANCE EVALUATION OF NRED

- More Realistic Scenario. 50 nodes randomly deployed in 1000X1000m. 5FT P/TCP are randomly selected.

![Graph](image3.png)

---

37