Cheating [+detection] in Wireless Networks

Imad Aad
14.4.2014

Collaborative (old) work/papers with:
Maxim Raya, Mario Cagalj, Alaeddine El Fawal, Saurabh Ganeriwal and Jean-Pierre Hubaux
Motivation

It's obvious: you cheat, you can get more bandwidth

How to detect?
System model

- Infrastructure mode
- DCF
- Single trusted AP
- Misbehavior is greedy as opposed to malicious
- Detection is implemented only at the AP
Example scenario

Well-behaved node

Cheater

Well-behaved node
IEEE 802.11 MAC – Brief reminder

CW: Contention Window
SIFS: Short Inter-Frame Spacing
DIFS: Distributed Inter-Frame Spacing

RTS / CTS: Request To Send / Clear To Send
ACK: ACKnowledgement
NAV: Network Allocation Vector
Question

How can one cheat?
Misbehavior techniques – Frame scrambling

CW: Contention Window
SIFS: Short Inter-Frame Spacing
DIFS: Distributed Inter-Frame Spacing

RTS / CTS: Request To Send / Clear To Send
ACK: ACKnowledgement
NAV: Network Allocation Vector
Misbehavior techniques – NAV

CW: Contention Window
SIFS: Short Inter-Frame Spacing
DIFS: Distributed Inter-Frame Spacing
RTS / CTS: Request To Send / Clear To Send
ACK: ACKnowledgement
NAV: Network Allocation Vector
Misbehavior techniques – DIFS

CW: Contention Window
SIFS: Short Inter-Frame Spacing
DIFS: Distributed Inter-Frame Spacing
RTS / CTS: Request To Send / Clear To Send
ACK: ACKnowledgement
NAV: Network Allocation Vector
Misbehavior techniques – Backoff

**CW**: Contention Window  
**RTS/CTS**: Request To Send/Clear To Send  
**ACK**: ACKnowledgement  
**NAV**: Network Allocation Vector  
**RTS/CTS**: Request To Send/Clear To Send  
**ACK**: ACKnowledgement  
**NAV**: Network Allocation Vector

**CW**: Contention Window  
**RTS/CTS**: Request To Send/Clear To Send  
**ACK**: ACKnowledgement  
**NAV**: Network Allocation Vector  
**RTS/CTS**: Request To Send/Clear To Send  
**ACK**: ACKnowledgement  
**NAV**: Network Allocation Vector
Implementation – Throughput

![Graph showing throughput vs. misbehavior coefficient]

- **Cheater** line: Red, increasing with misbehavior coefficient.
- **Well-behaved** line: Blue, decreasing with misbehavior coefficient.

Throughput (b/s) on the y-axis, Misbehavior coefficient on the x-axis.
## Components of DOMINO

<table>
<thead>
<tr>
<th>Cheating method</th>
<th>Detection test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame scrambling</td>
<td>Number of retransmissions</td>
</tr>
<tr>
<td>Oversized NAV</td>
<td>Comparison of the declared and actual NAV values</td>
</tr>
<tr>
<td>Transmission before DIFS</td>
<td>Comparison of the idle time after the last ACK with DIFS</td>
</tr>
<tr>
<td>Backoff manipulation</td>
<td>Maximum backoff: the maximum should be close to $CW_{min} - 1$</td>
</tr>
<tr>
<td></td>
<td>Actual backoff</td>
</tr>
<tr>
<td></td>
<td>Consecutive backoff</td>
</tr>
</tbody>
</table>
Actual backoff test

- Compares the average actual backoff of each station to the average actual backoff of the AP
- Collisions are not taken into account
- Unsuitable for sources with interframe delays (e.g., due to TCP congestion control)
Consecutive backoff test

- Useful when cheaters have interframe delays (mainly TCP sources)
- Does not work if the traffic is very high due to the lack of samples
- Complementary to the actual backoff test
Simulation – Topology

- ns-2
- Backoff manipulation
- CBR / UDP traffic
- FTP / TCP traffic
- misbehavior coefficient \((m)\):
  cheater chooses its backoff from the fixed contention window \((1-m) \times CWmin\)
Simulation – DOMINO performance – UDP case
Simulation – DOMINO performance – TCP case

![Graph showing the accuracy of detection and misdetection against misbehavior coefficient. The graph plots Accuracy (%) on the y-axis and Misbehavior coefficient on the x-axis. The detection line shows a steep increase in accuracy as the misbehavior coefficient increases, while the misdetection line remains relatively flat.](image-url)
Implementation

- **Equipment**
  - Adapters based on the Atheros AR5212 chipset
  - MADWIFI driver

- **Misbehavior (backoff)**
  - Write to the register containing CWmin and CWmax (in driver)

- **Monitoring**
  - The driver in MONITOR mode
  - prism2 frame header
Implementation – Throughput

![Graph showing throughput vs. misbehavior coefficient]

Throughput (b/s)

Misbehavior coefficient
Implementation – Backoff and DOMINO

![Graph showing observed average backoff and detection percentage vs. misbehavior coefficient.](image)
Discussion

- **Hidden terminals**
  - *Well-chosen detection thresholds can reduce false positives*

- **Security**
  - *Hybrid attacks: limited efficiency*

- **Adaptive cheating**
  - *Hard to implement*

- **Overhead**
  - *Negligible*
Question

Do you see any “relevance” issue here?
And the answer is...

Traffic is mainly on the downlink, not on the uplink!

→ the previous cheating/detection methods apply to a small fraction of traffic only
A new downlink scenario
Attack = jamming TCP packets on the downlink
Another question...

What happens if everybody does this (backoff cheat)?

Simulation study (N=20 nodes, C=1 cheater, 2Mbps)
Pareto-optimal point...

- $N = 20$, $C = 10$, $2$ Mbps
- There is an optimal point ($W^* = 27$) at which throughput of each cheater is maximized
- $W^*$ is Pareto-optimal, since the channel is saturated
- However, $W^*$ is not an equilibrium point!
Then the questions become...

- How do players know what is the optimal point?
- How can one enforce an equilibrium at this Pareto-optimal point?
Key building blocks:

- **Detection mechanism**
- **Penalizing mechanism**
- **Adaptive strategy**
  - *Action taken by the "non-cooperative" cheater in response to being punished*
- **Gradient-search mechanism**
  - *To pinpoint the Pareto-optimal point (W*) of operation in a distributed way*
Detection mechanism

Throughput-based

- Cheaters should have similar throughputs ("communist fair")
- Cheater $i$ measures the throughput of cheater $j$ for $T_{obs}$ sec. If $\frac{\text{thr}_j}{\text{thr}_i} > 1 + \epsilon$, $i$ detects that $j$ is deviating ($\epsilon$ is a tolerance margin; important due to the inherent short term unfairness of IEEE 802.11).
Penalizing mechanism

Based on selective jamming (with IEEE 802.11, only one station can transmit at a time)

Enforcing "communist fairness", i.e., the duration of a punishment $T_{jam}$ is selected such that $thr_i$ and $thr_j$ are the same over the time $(T_{obs} + T_{jam})$: 
Adaptive strategy

- Is an action taken by a "non-cooperative" cheater in response to being punished

- Each time a "non-cooperative" cheater is punished ( jammed) consecutively for $D$, it increases its contention window by $b$

- Simulations: $N=20$, $C=10$ out of which 3 "non-cooperative" cheaters ($X,Y,Z$)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Non-adaptive</th>
<th>Adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheater $X$</td>
<td>2843</td>
<td>10356</td>
</tr>
<tr>
<td>Cheater $Y$</td>
<td>2686</td>
<td>10185</td>
</tr>
<tr>
<td>Cheater $Z$</td>
<td>2565</td>
<td>10239</td>
</tr>
<tr>
<td>Other cheaters</td>
<td>2544</td>
<td>10172</td>
</tr>
<tr>
<td>Well-behaved nodes</td>
<td>270</td>
<td>1981</td>
</tr>
</tbody>
</table>
Gradient-search mechanism...

- In a distributed setting, cheaters do not know $W^*$
- **IDEA:** the first derivative changes sign at $W^*$
  - Set $W$ to some small ($<W^*$) initial value $W_{in}$
  - Each cheater increases its contention window by $b$ after expiration of its random timer
  - Cheater $X$ with the largest contention window receives lower throughput than others (i.e., $X$ detects an act of "non-cooperation")
  - Cheater $X$ penalizes others, who in turn increases their contention window sizes by $b$ (they also stop their timers)
  - System stabilizes when all cheaters have similar throughputs and window sizes
    $W = W_{in} + b$
  - At this stage, cheater $i$ checks if $\text{thr}(W_{in}) - \text{thr}(W_{in} + b) < 0$ in which case it terminates the search for $W^*$
  - Otherwise, cheater $i$ again sets up random timer to increase $W$
Equilibrium study

Simulation: N=20, C=7, W*=20
Conclusions

- Identified simple and practical mechanisms for cheating
- Evaluated the impact
- Identified methods to detect cheating
- On the uplink and downlink
- Evaluated the cases of multiple cheaters
- Devised methods for cooperative (optimal) cheating

• Refer to the papers below for further details:
  