Network Research and Linux at the Hamilton Institute, NUIM.

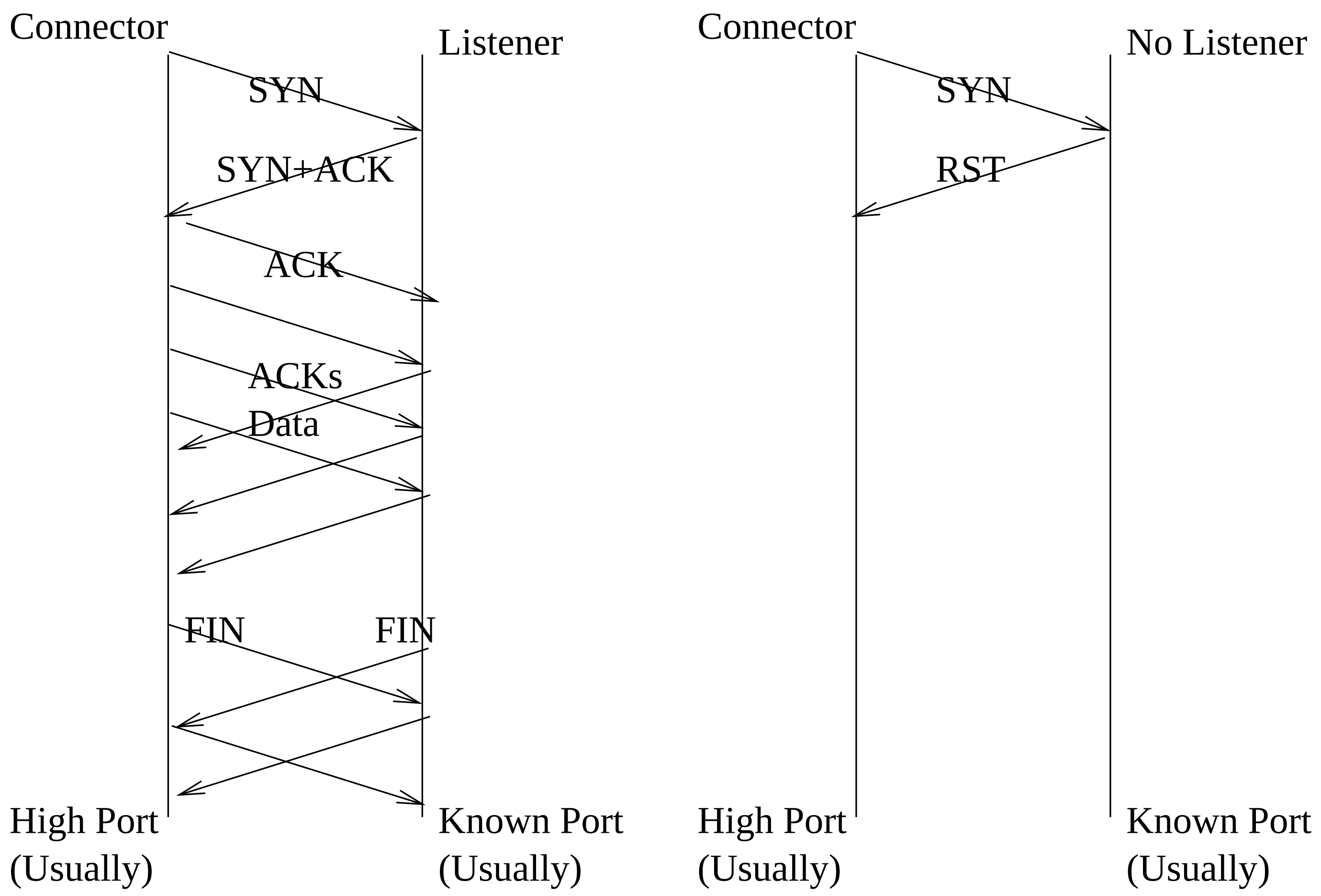
David Malone

4 November 2006

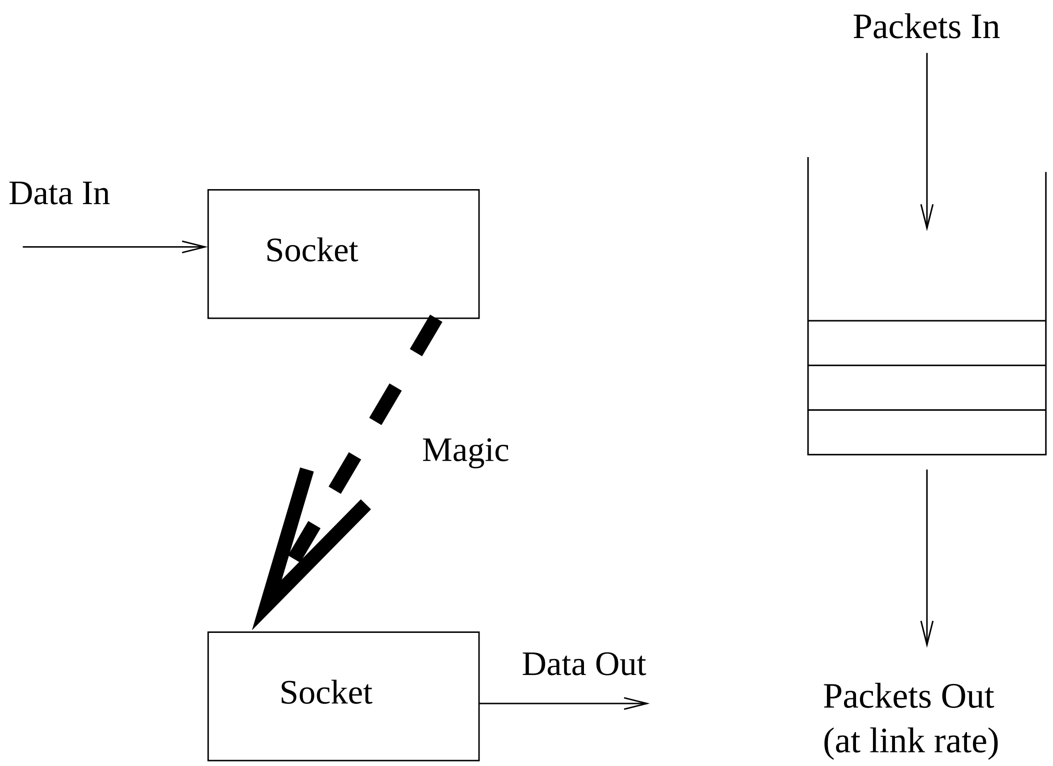
What has TCP ever done for Us?

- Demuxes applications (using port numbers).
- Makes sure lost data is retransmitted.
- Delivers data to application in order.
- Engages in congestion control.
- Allows OOB data.
- Some weird stuff with TCP options.

Standard Picture of TCP



Other Views of TCP

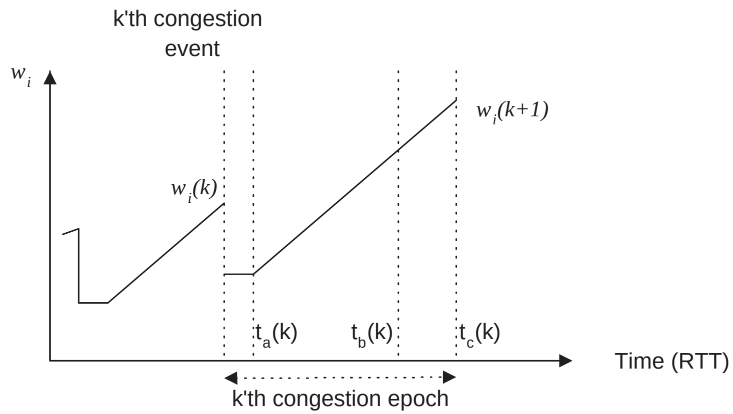


Programmer's View   Network View

Congestion Control

- TCP controls number of packets in network.
- Packets are acknowledged, so flow of ACKs.
- Receiver advertises window to avoid overflow.
- Congestion window tries to adapt to network.
- Slow start to roughly find capacity.
- Congestion avoidance gradually adapts.

The Congestion Window



- Additive increase, multiplicative decrease (AIMD).
- To fill link need to reach $BW \times \tau$.
- Backoff by 1/2, implies buffer is $BW \times \tau$.
- Fairness, responsiveness, stability, ...

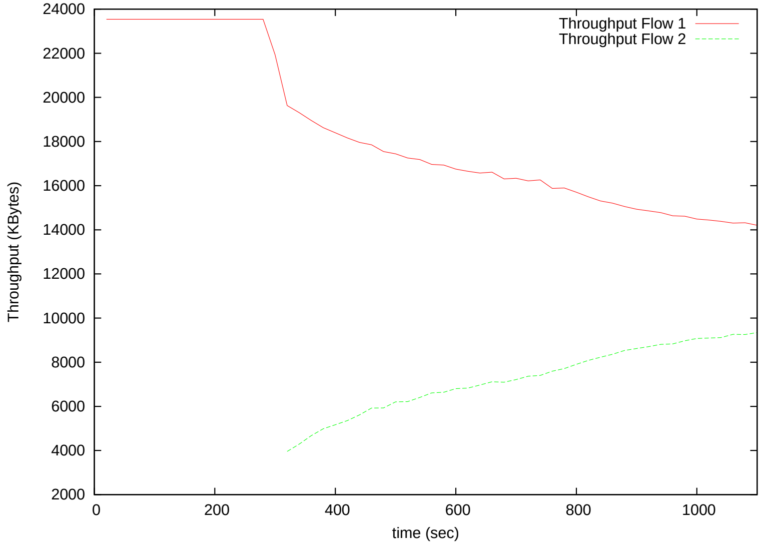
TCP On Linux

- Network stack buffers in-flight data.
- Socket buffer must be $BW \times \tau$.
- `/proc/net/core/{r,w}mem_max` $\rightarrow$ sockbuf sizes.
- `/proc/net/ipv4/tcp_{r,w,}mem` $\rightarrow$ min/def/max tcp window.
- Trade off — memory is wired, so valuable.
- Defaults have recently been increased.

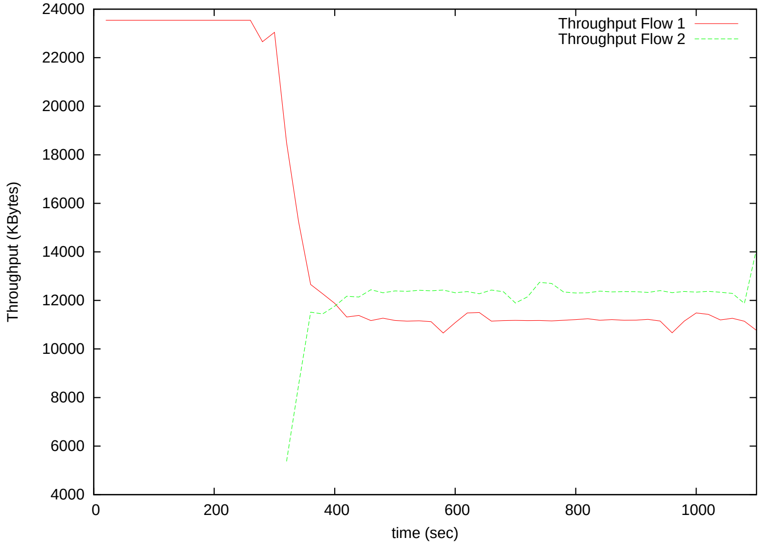
Research Work

- Packet loss not caused by congestion.
- Filling big $BW \times \tau$ product packet at a time.
- Bad for long-distance high-bandwidth links.
- Various solutions in pipeline (BIC, Scalable, High-Speed, FAST, H-TCP).
- Pluggable congestion control in Linux (behind `TCP_CONG_ADVANCED`).
- `/proc/sys/net/ipv4/tcp_congestion_control`
- Working on other congestion detection techniques.

Throughput

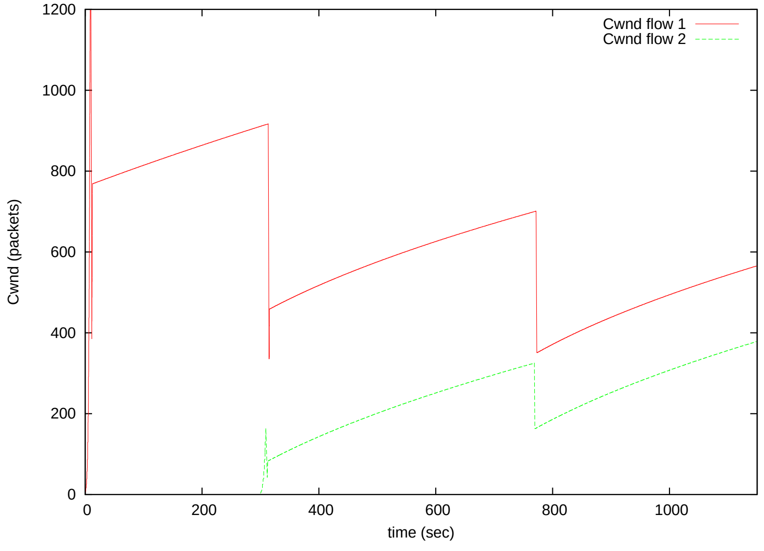


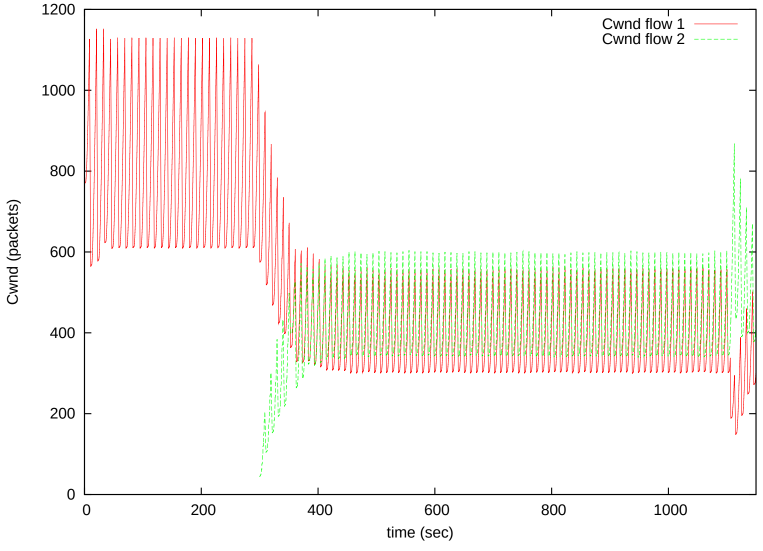
Throughput Flow 1
Throughput Flow 2
Throughput (KBytes)
time (sec)
24000
22000
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
200
400
600
800
1000



Throughput Flow 1
Throughput Flow 2
Throughput (KBytes)
time (sec)
24000
22000
20000
18000
16000
14000
12000
10000
8000
6000
4000
0
200
400
600
800
1000

Cwnd

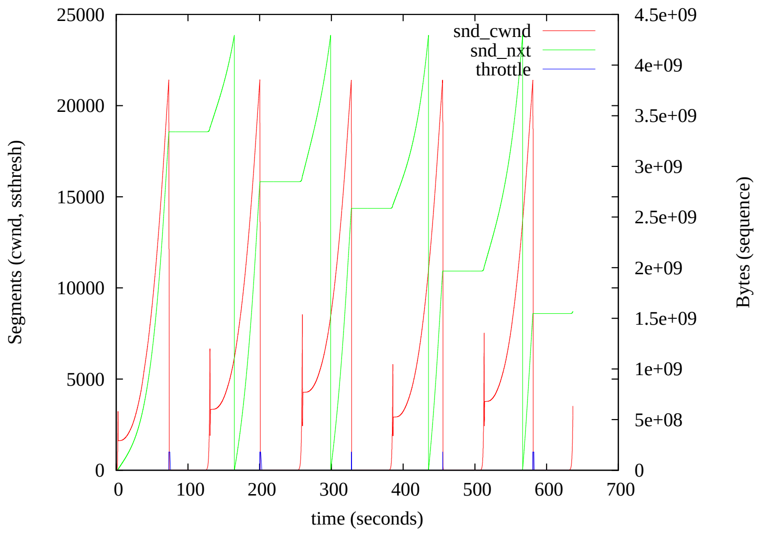

Cwnd flow 1
Cwnd flow 2
Cwnd (packets)
0
200
400
600
800
1000
1200
time (sec)


Cwnd flow 1
Cwnd flow 2
Cwnd (packets)
0
200
400
600
800
1000
1200
time (sec)

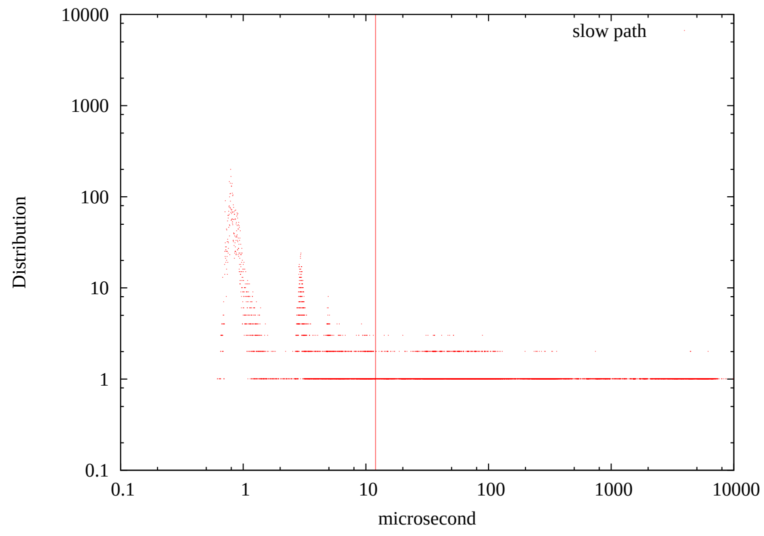
Practical Stuff

- Other issues at play, such as implementation quality.
- For example ACK processing and queueing problems.
- Testing is important: land speed records.
- Project with OSDL to build validation suite.

Before

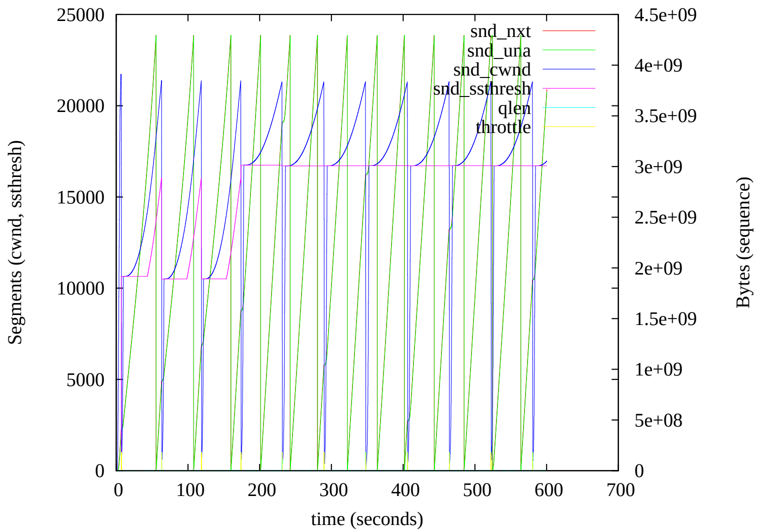


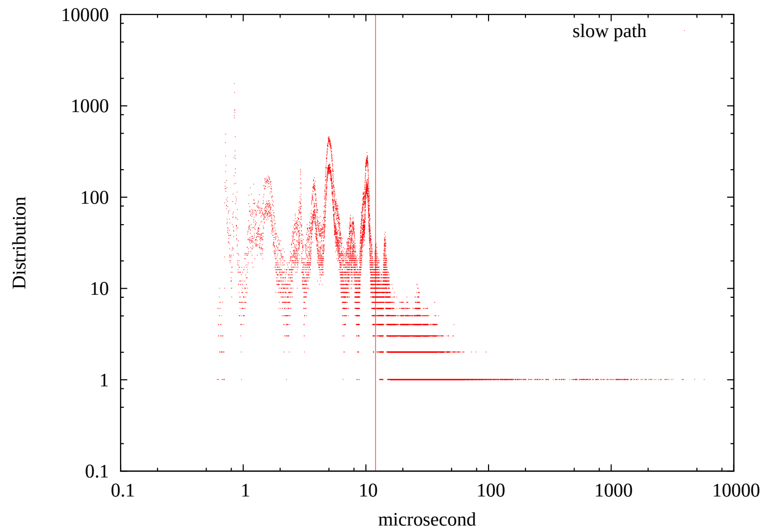
snd_cwnd
snd_nxt
throttle
Segments (cwnd, ssthresh)
Bytes (sequence)
time (seconds)
0
100
200
300
400
500
600
700
5000
10000
15000
20000
25000
5e+08
1e+09
1.5e+09
2e+09
2.5e+09
3e+09
3.5e+09
4e+09
4.5e+09



slow path
Distribution
microsecond
0.1
1
10
100
1000
10000

After


snd_nxt
snd_una
snd_cwnd
snd_ssthresh
qlen
throttle
Segments (cwnd, ssthresh)
Bytes (sequence)
time (seconds)
25000
20000
15000
10000
5000
0
4.5e+09
4e+09
3.5e+09
3e+09
2.5e+09
2e+09
1.5e+09
1e+09
5e+08
0
0
100
200
300
400
500
600
700


slow path
Distribution
microsecond
10000
1000
100
10
1
0.1
0.1
1
10
100
1000
10000

802.11(e) MAC
Summary

- After TX choose $\mathrm{rand}(0, \mathrm{CW} - 1)$.
- Wait until medium idle for DIFS($50\mu s$),
- While idle count down in slots ($20\mu s$).
- TX when counter gets to 0, ACK after SIFS ($10\mu s$).
- If ACK then $\mathrm{CW} = \mathrm{CW}_{\mathrm{min}}$ else $\mathrm{CW} * = 2$.

Ideally produces even distribution of packet TX.

In 11e have multiple queues. Each has own $\mathrm{CW}_{\mathrm{min}}$, DIFS(aka AIFS) and can have TXOP.

Testbed setup

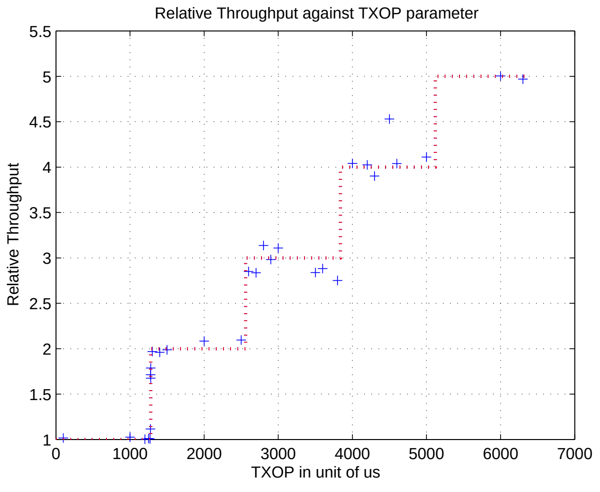
Multiple STA (Linux) connected to AP (Linux hostap).

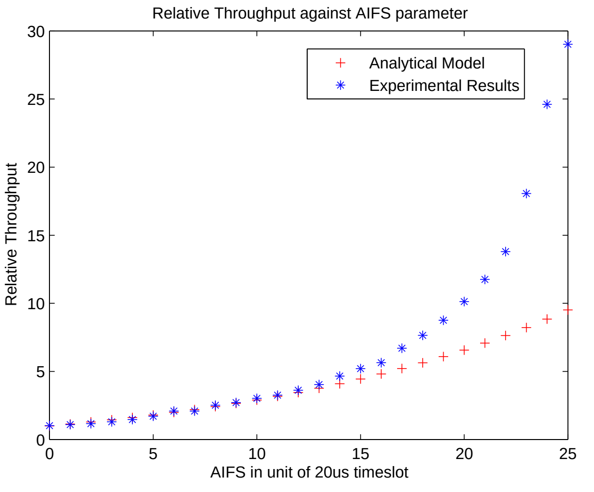
Hardware	model
1× AP	Desktop PC
18× STA	Soekris
1× STA	Desktop PC
WLAN NIC	Atheros AR5212

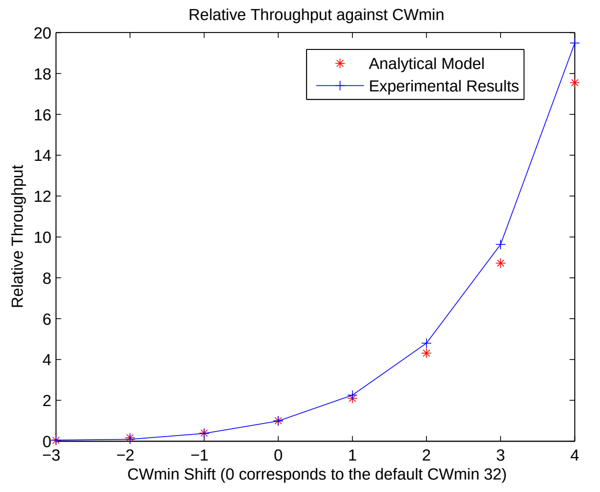
External antenna, PCI interface, Madwifi driver with local patches for 11e parameter setting.


192.168.0.180&180
192.168.0.187
187
192.168.0.185
192.168.0.7&177

Validation

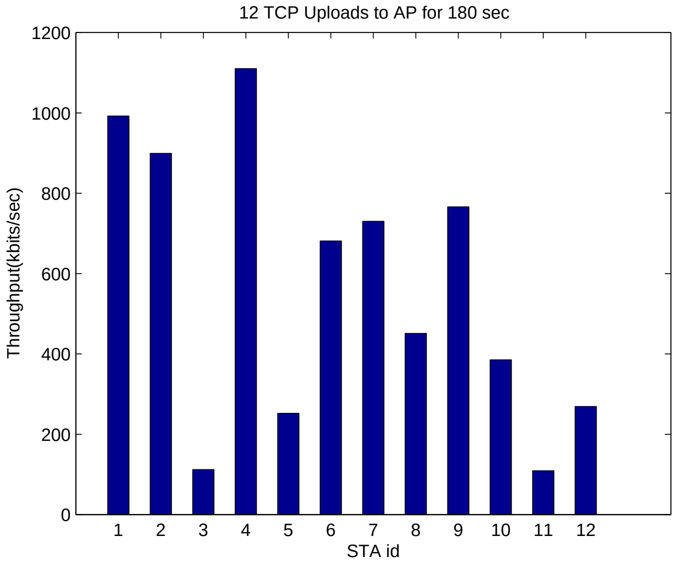


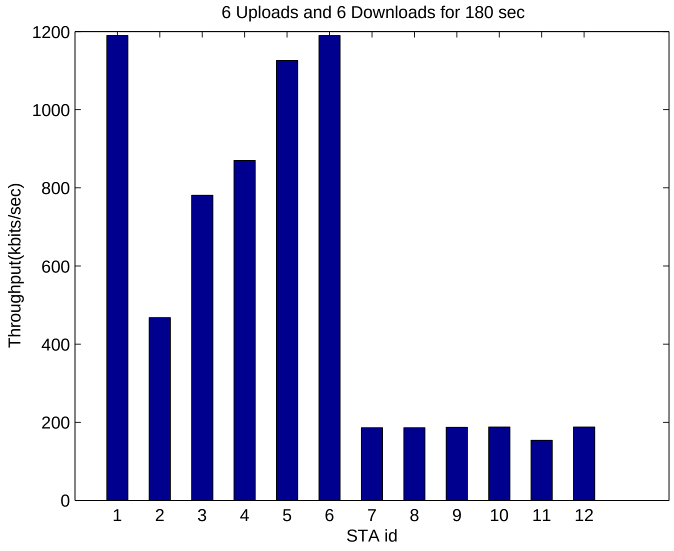




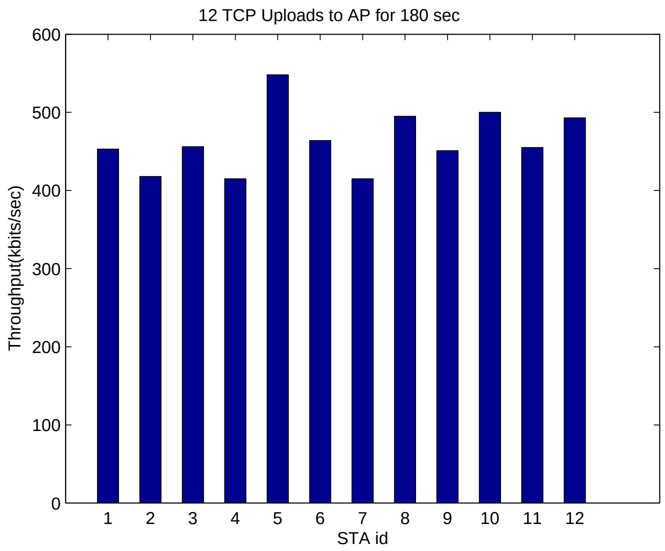
Measure relative performance of two saturated flows while varying TXOP, AIFS and CW_{min}. Compare to well-known models.

Before

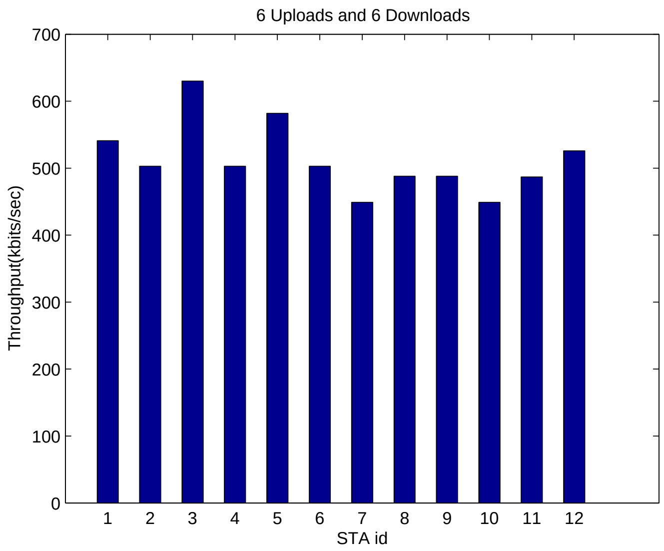

12 TCP Uploads to AP for 180 sec
Throughput(kbits/sec)
0
200
400
600
800
1000
1200
1 2 3 4 5 6 7 8 9 10 11 12
STA id


6 Uploads and 6 Downloads for 180 sec
Throughput(kbits/sec)
0
200
400
600
800
1000
1200
1 2 3 4 5 6 7 8 9 10 11 12
STA id

After



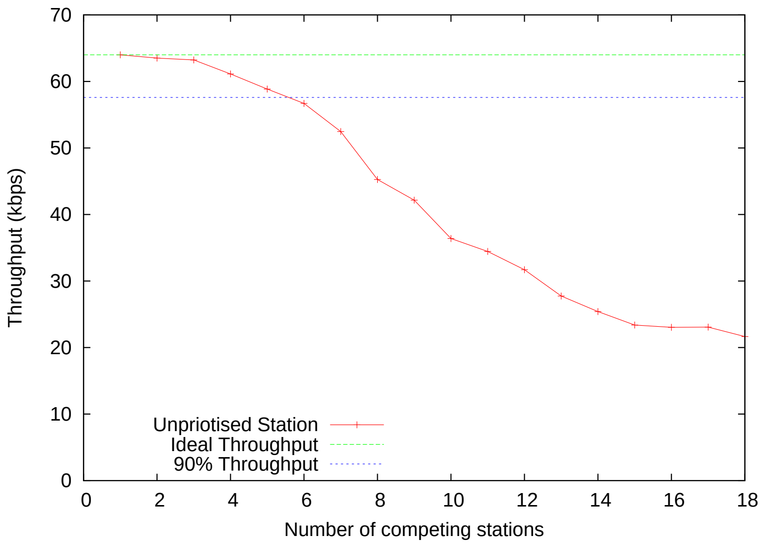
12 TCP Uploads to AP for 180 sec
Throughput(kbits/sec)
0
100
200
300
400
500
600
1 2 3 4 5 6 7 8 9 10 11 12
STA id



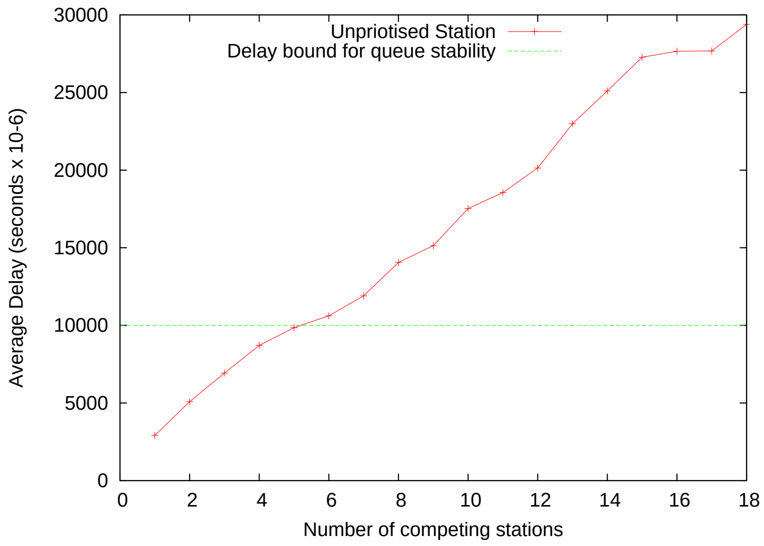
6 Uploads and 6 Downloads
Throughput(kbits/sec)
0
100
200
300
400
500
600
700
1 2 3 4 5 6 7 8 9 10 11 12
STA id

Unprioritised Voice

Throughput

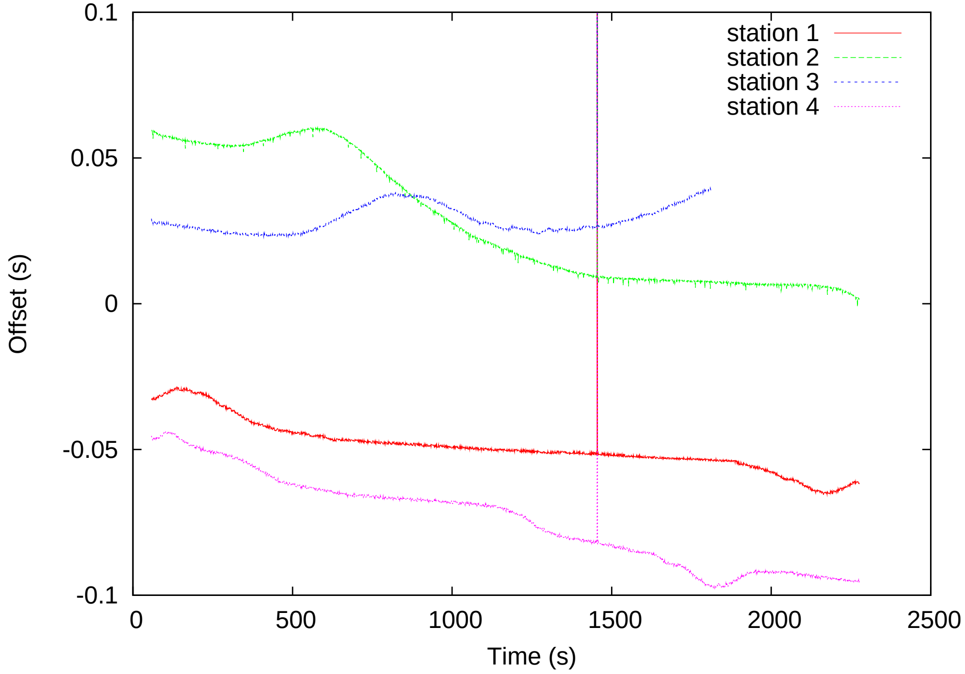


Delay



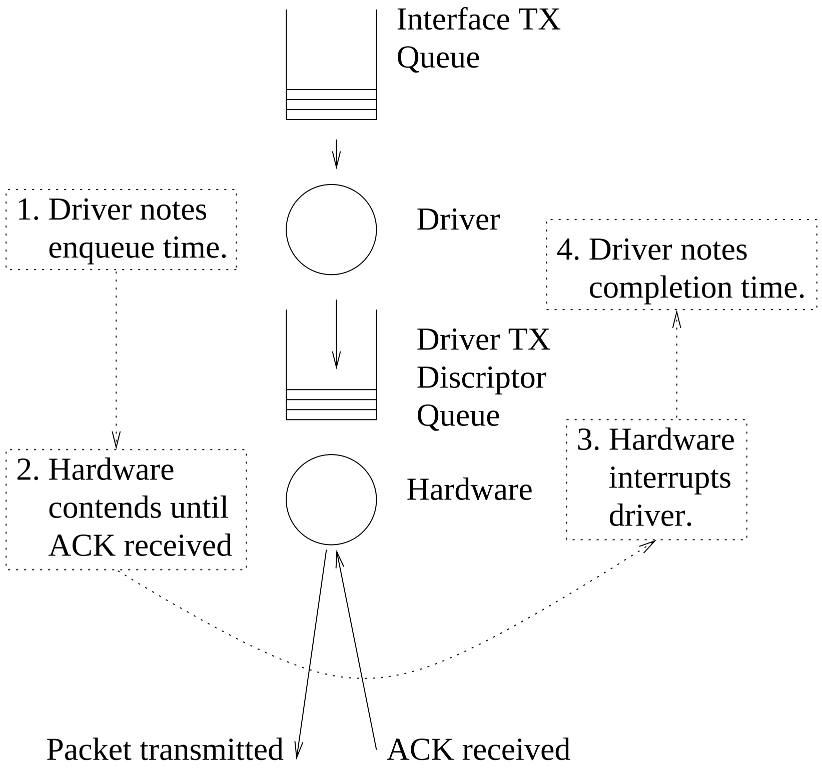
Measuring Delay

- Want to measure one-way MAC delay.
- NTP slow and insufficiently accurate.
- Simultaneously observable TX better, largish noise.

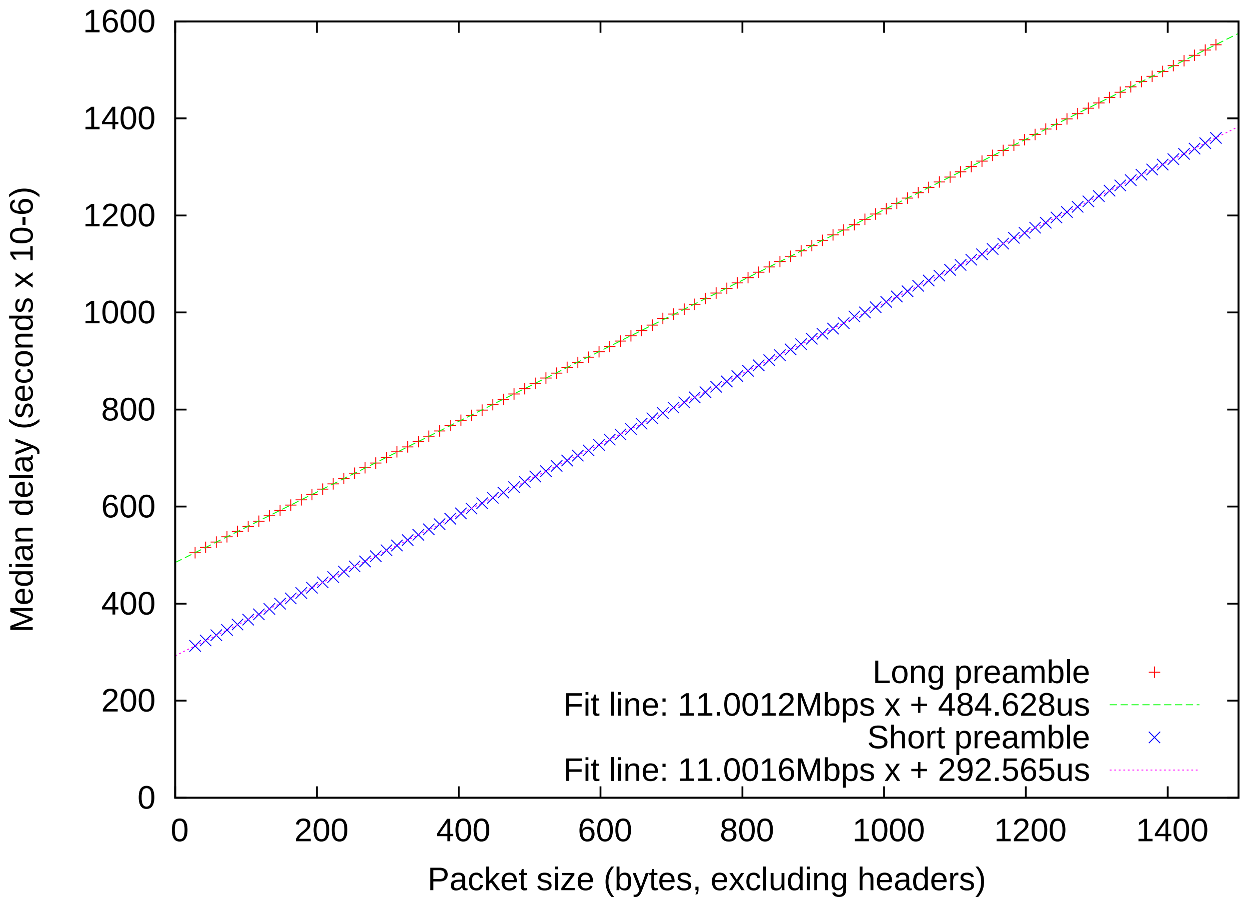


Delay Technique

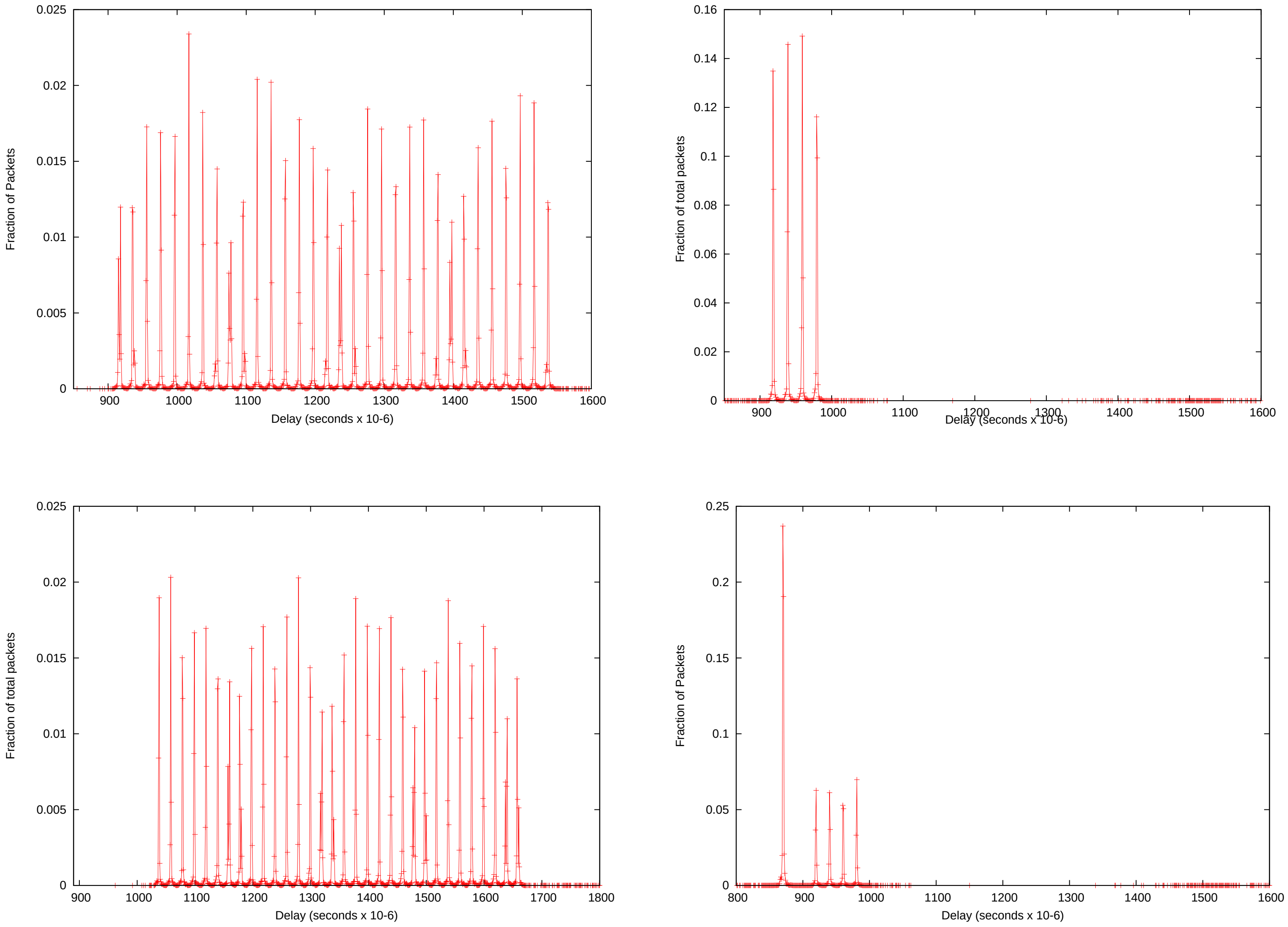
- Transmission not complete until MAC ACK.
- Hardware supports interrupt after ACK.



Validation



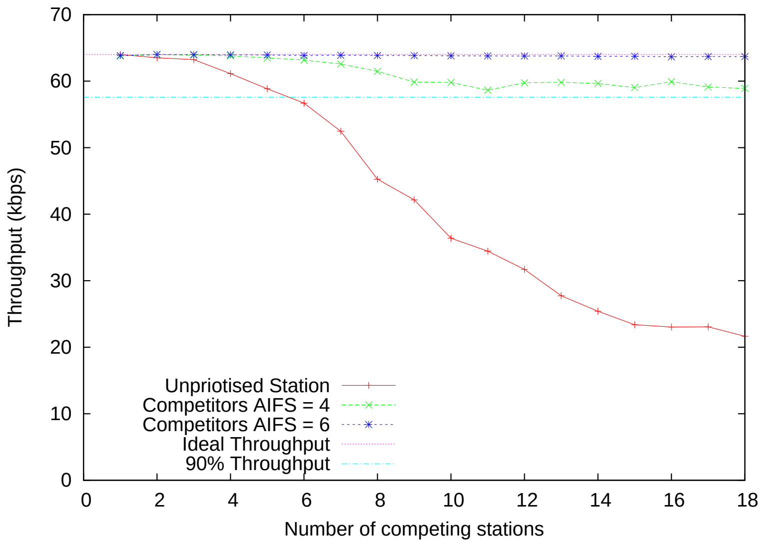
Median delay (seconds x 10-6)
1600
1400
1200
1000
800
600
400
200
0
0
200
400
600
800
1000
1200
1400
Packet size (bytes, excluding headers)
Long preamble
Fit line: 11.0012Mbps x + 484.628us
Short preamble
Fit line: 11.0016Mbps x + 292.565us



Fraction of Packets
Delay (seconds x 10-6)
Fraction of total packets
Delay (seconds x 10-6)
Fraction of total packets
Delay (seconds x 10-6)
Fraction of Packets
Delay (seconds x 10-6)

AIFS Impact

Throughput



Delay