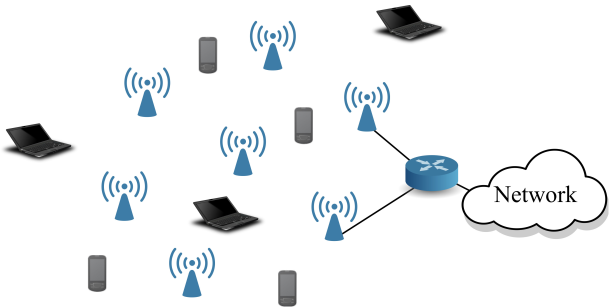
Experimental Performance Evaluation of ATP in a Wireless Mesh Network

IEEE MASS 2011

Xingang Zhang, Randy Buck, Daniel Zappala

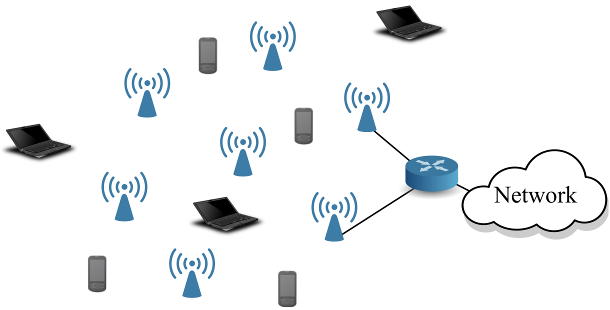
Brigham Young University
Computer Science Department

TCP in Wireless Mesh Networks



- well-known that TCP performs poorly
  - both throughput and fairness
- many attempts to design better solutions
  - ELFN, Adaptive Pacing, FeW, ATP, etc.
  - end-to-end, hop-by-hop, cross-layer

TCP in Wireless Mesh Networks



but ...

- relatively few implementations
- most work evaluated with simulations

Simulation vs Experimentation

Simulation

- easy to implement
- repeatability, quick results

but ...

- difficult to model radio wave propagation accurately
- only as good as its models

Experiment

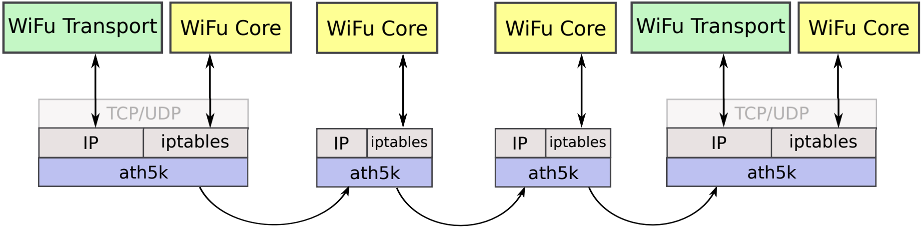
- realism – no need for modeling
- demonstrates an actual working system

but ...

- difficult to implement
- non-deterministic results
- hard to generalize
- takes time to run

$\Rightarrow$ would like to do both

WiFu



- user-level toolkit for experimental wireless transport protocols
  - *WiFu Transport*: end-to-end protocols
  - *WiFu Core*: cross-layer interactions, intercept and modify IP packets at any hop
- multi-year, open source project
- code release soon...

This Work

- implement and test ATP
  - clean-slate design of transport protocol for wireless networks
  - revised reliability, new congestion control
- first use of *WiFu Core*
- contributions
  - what works, what doesn't in ATP
  - design modifications to improve ATP performance
  - show that rate-based congestion control, using cross-layer measurements, can provide better balance of goodput and fairness in wireless mesh networks

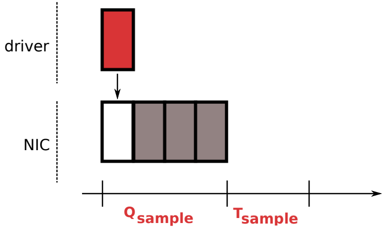
ATP

1. Intermediate Nodes
   - measure queueing and transmission delay

$$\begin{aligned} Q_t &= \alpha * Q_t + (1-\alpha) * Q_{sample}, \\ T_t &= \alpha * T_t + (1-\alpha) * T_{sample}, \\ D &= Q_t + T_t \end{aligned}$$

   where $\alpha = 0.75$

   - packet carries maximum delay D_{max} seen on the path

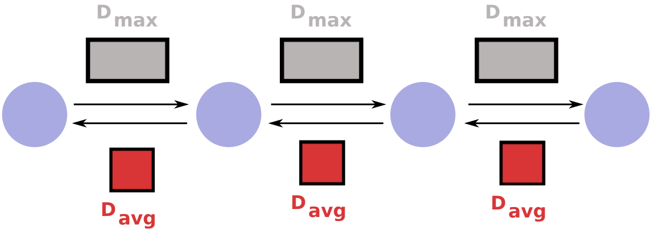


ATP

2. **Receiver**
   - receiver measures EWMA of D_{max} for each flow

$$D_{avg} = \beta * D_{avg} + (1 - \beta) * D_{max}$$

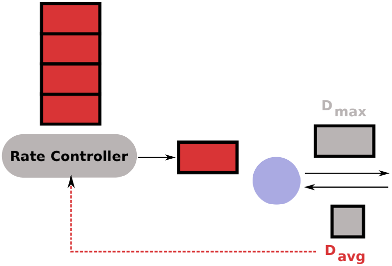
   where $\beta = 0.85$
   - sends ACK with D_{avg} every *epoch* seconds to the sender
   - includes 20 SACK blocks



ATP

3 Sender

- use D_{avg} to calculate sending rate
  - *increase* rate if D_{avg} is too small
  - *decrease* rate if D_{avg} is too big
  - *maintain* rate otherwise
- use *quick-start* to determine initial rate by probing for delay during connection establishment

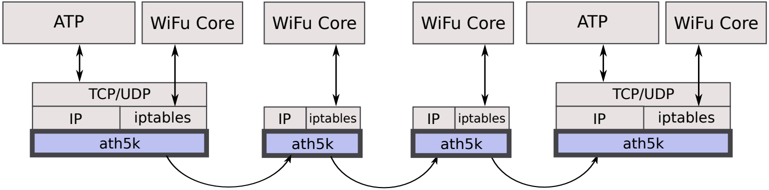


Implementation

1 Delay Averaging

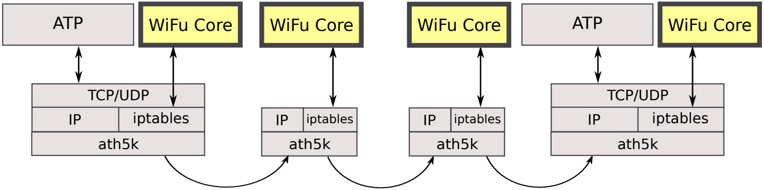
- modify `ath5k` driver
- interrupt driven, can only measure total delay

$$D = Q_t + T_t = \alpha * (Q_t + T_t) + (1 - \alpha) * (Q_{sample} + T_{sample})$$



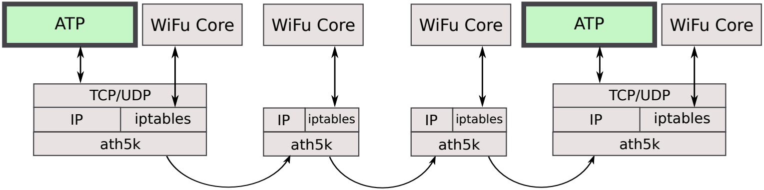
Implementation

2. Delay Collection
    - use *WiFu Core* to intercept packets
    - read *D* from /proc and overwrite ATP header field if larger than current value



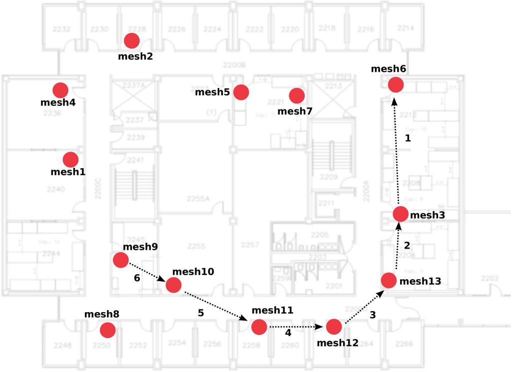
Implementation

3. Transport Protocol
   - first complete implementation of ATP, including connection management, reliability, congestion control
   - implemented in Python
   - compare to TCP Tahoe, also implemented in Python



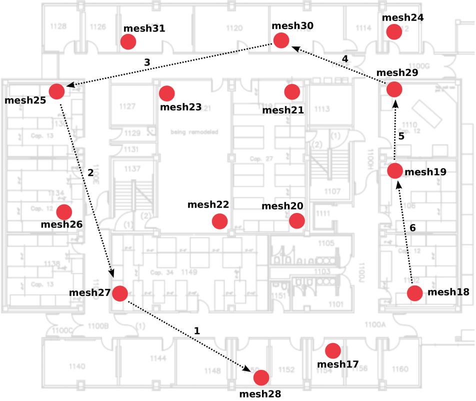
Mesh Testbeds

- Testbed A
- rate = 24 Mbps, power = 10 dBm
- high bandwidth, but lossy



Mesh Testbeds

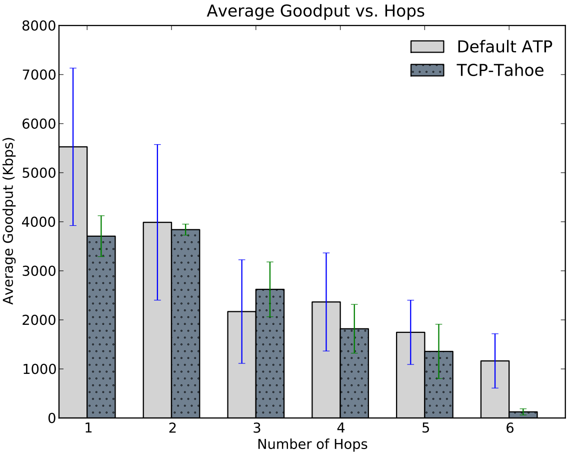
- Testbed B
- rate = 6 Mbps, power = 17 dBm
- more reliable, stable mesh with lower bandwidth



Experiment Setup

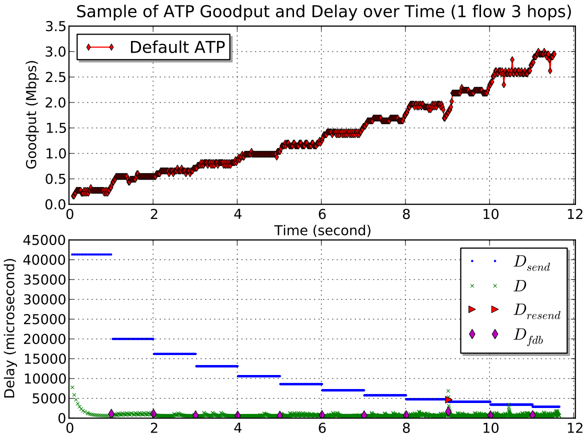
- minimize environmental variation
  - use IEEE 802.11a
  - alternate testing each protocol
- minimize confounding factors
  - MAC rate control off
  - static routing
- 10 repetitions, error bars for standard deviation

Single Flow (Testbed A)



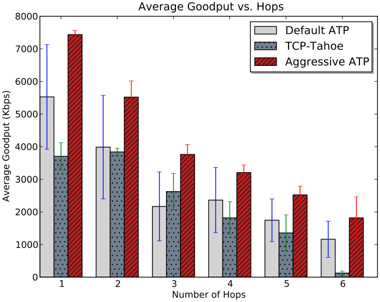
Analysis: Default ATP is Too Conservative

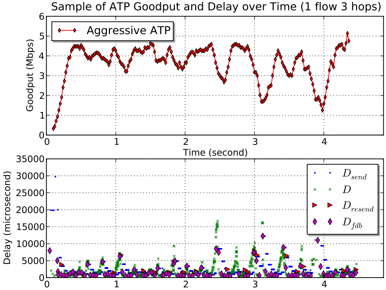
- epoch timeout of 1 second is *too large*
- rate increase factor of 0.2 is *too small*



Aggressive ATP

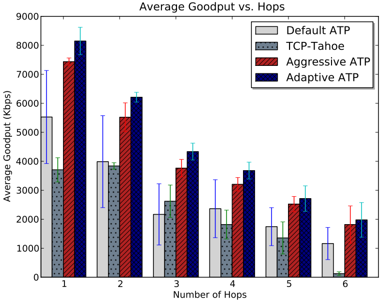
- reduce epoch timeout to 40 ms
- boost rate increase factor from 0.2 to 0.5

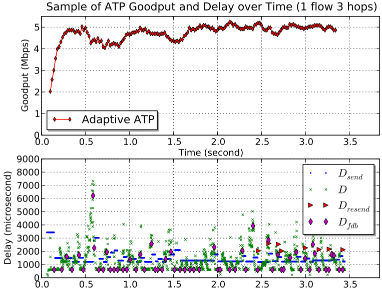




Adaptive ATP

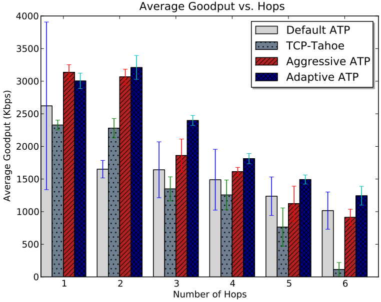
- choosing optimal parameters is difficult
- dynamically adjust epoch parameter instead of using fixed setting



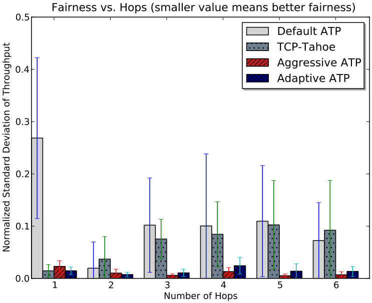


Two Flows (Testbed A)

- Adaptive ATP performs well
- Aggressive ATP's goodput advantage over default ATP diminishes over multiple hops
- both Aggressive and Adaptive ATP have better fairness



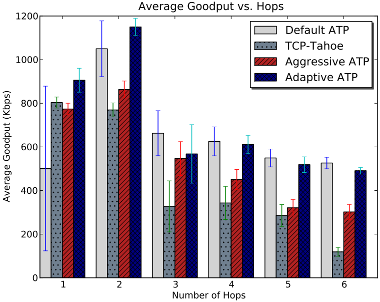
goodput



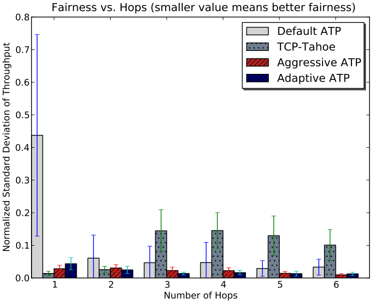
fairness

Five Flows (Testbed A)

- both Aggressive and Adaptive ATP provide good fairness
- TCP generally performs worst in both goodput and fairness
- Default ATP's conservative rate adjustment results in high but unfair goodput in saturated links



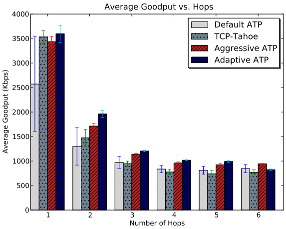
goodput



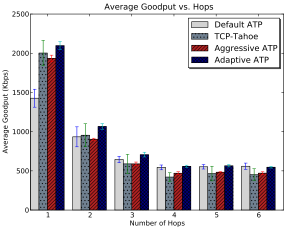
fairness

Generality: Goodput Results from Testbed B

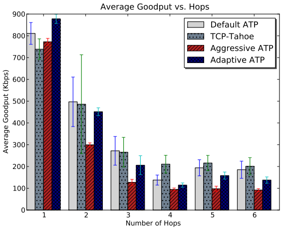
- both Default ATP and TCP compete more favorably on goodput
- Adaptive ATP adjusts well and performs on par with Default ATP



1 flow



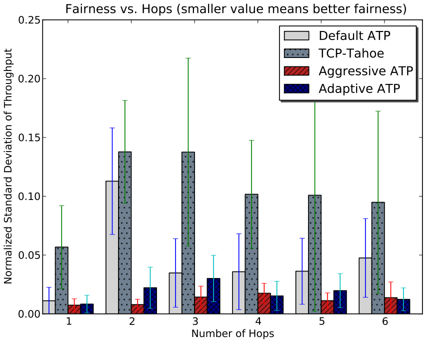
2 flows



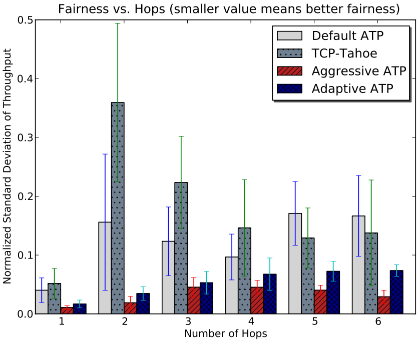
5 flows

Generality: Fairness Results from Testbed B

- both Aggressive and Adaptive ATP still have better fairness



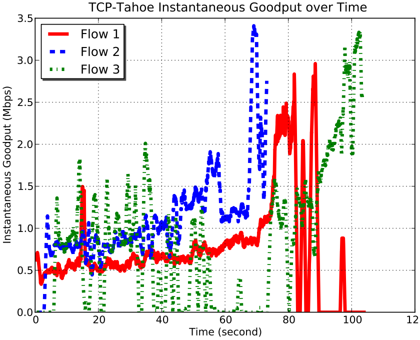
2 flows



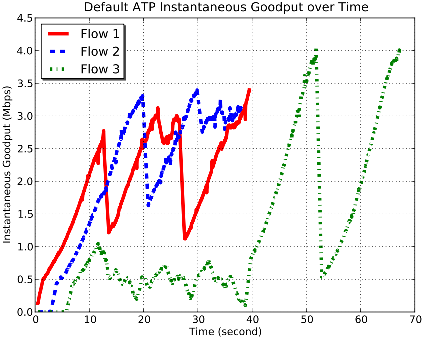
5 flows

Fairness with Stack Topology

- green flow "in the middle" of the red and blue flows
- classic case of starvation in mesh networks



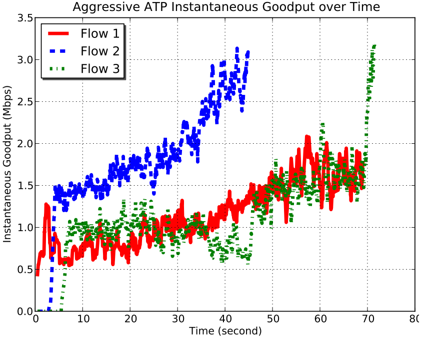
TCP Tahoe, Log utility: 8.60



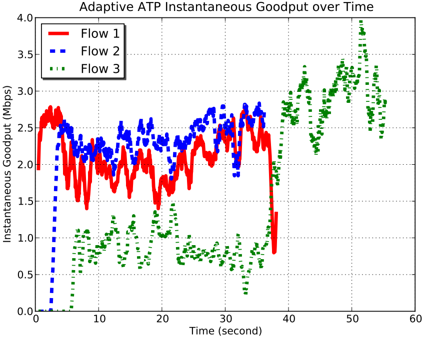
Default ATP, Log utility: 9.48

Fairness with Stack Topology

- green flow "in the middle" of the red and blue flows
- classic case of starvation in mesh networks



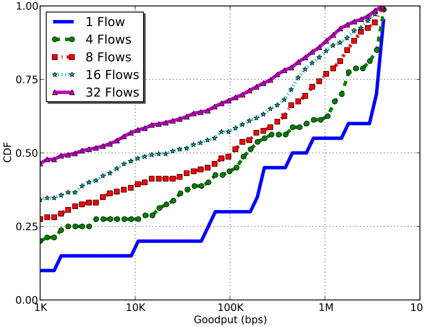
Aggressive ATP, Log utility: 9.26



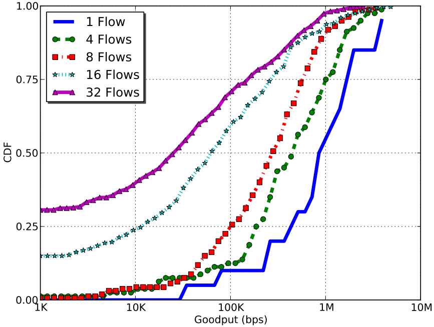
Adaptive ATP, Log utility: 9.64

Random Flows: 1 MB File Transfer

- uses entire mesh, OLSR for routing
- TCP starves half of 32 flows, Default ATP starves 30%



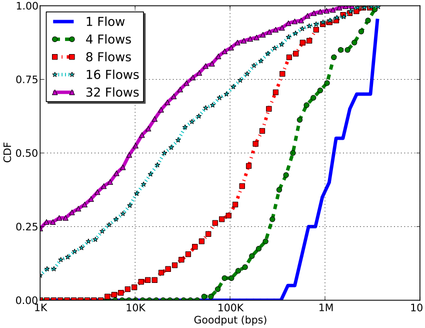
TCP Tahoe



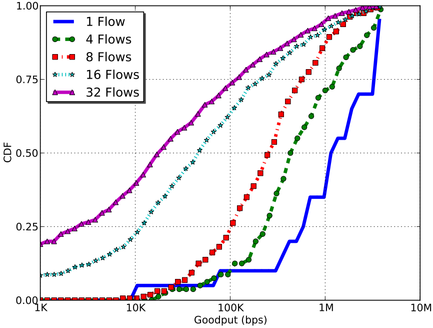
Default ATP

Random Flows: 1 MB File Transfer

- uses entire mesh, OLSR for routing
- Adaptive ATP starves the fewest, good balance of fairness and goodput



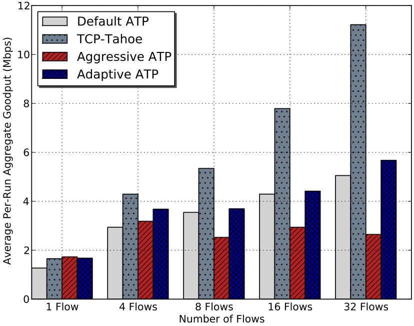
Aggressive ATP



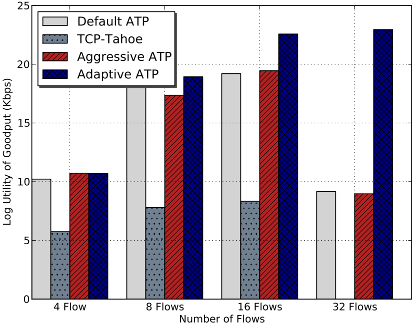
Adaptive ATP

Random Flows: 1 MB File Transfer

- tradeoff between aggressiveness and fairness
- TCP has highest goodput, Adaptive ATP is more fair



Average aggregate goodput



Average log utility

Conclusions

- original ATP design has several shortcomings
  - epoch-based feedback too inefficient
  - rate increase too conservative
- modifications to ATP provide better goodput and fairness
  - shorter and more dyanmic epoch settings
  - more aggressive rate increase

$\Rightarrow$ with good design, rate-based congestion control, using cross-layer measurements, is a promising direction

Future Work

- split ATP-style congestion control from epoch-based feedback completely and implement in *WiFu Transport* (C++)
- implement and compare a wider variety of congestion control designs
- incorporate mobility into tests