Resilient and Energy-Efficient Scheduling for Mobile Devices using Multipath TCP

Christopher Pluntke

Joint work with Lars Eggert and Niko Kiukkonen

Outline

1. Problem formulation and related work
2. Architecture proposal
3. Automatic scheduler generation
4. Evaluation

Different data interfaces have different properties.

Mobile phones are equipped with multiple data interface (e.g. 3G and Wifi) that have different characteristics in terms of

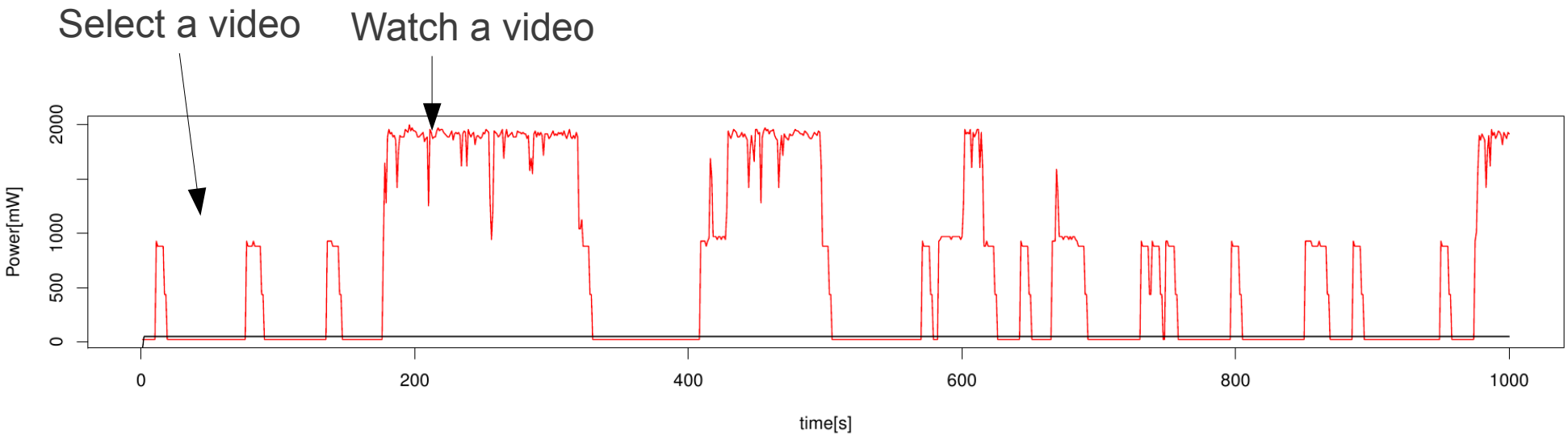
- **energy consumption**
- **connectivity**
- throughput
- delay
- cost

How can we combine the best features of each interface?

Trade-offs for 3G and Wifi energy consumption

This is a sample run showing 3G power consumption with Youtube traffic in a simulation:

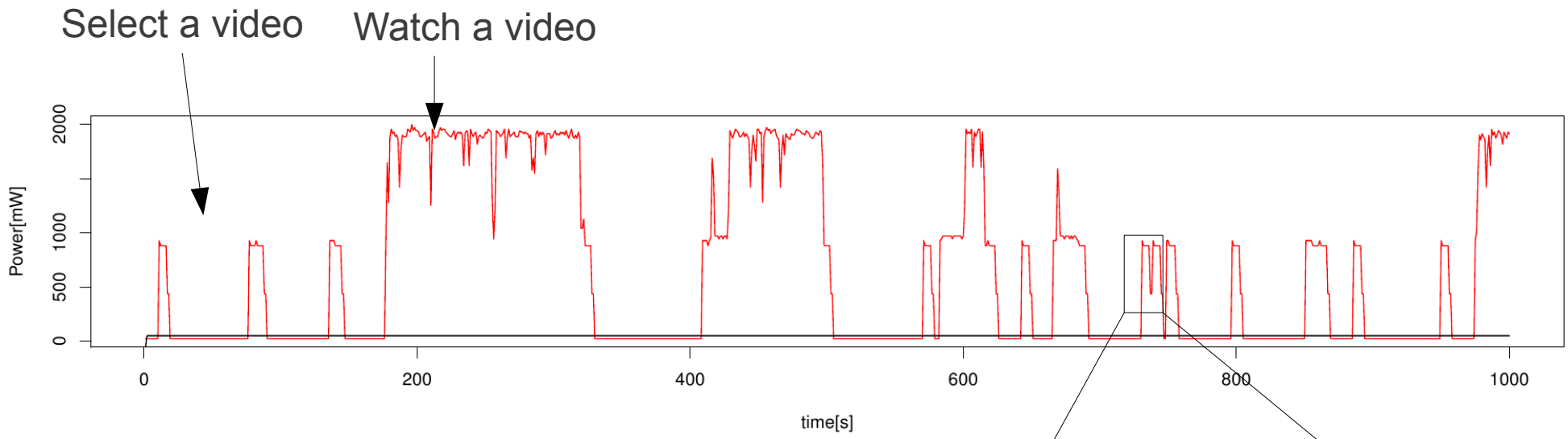
3G

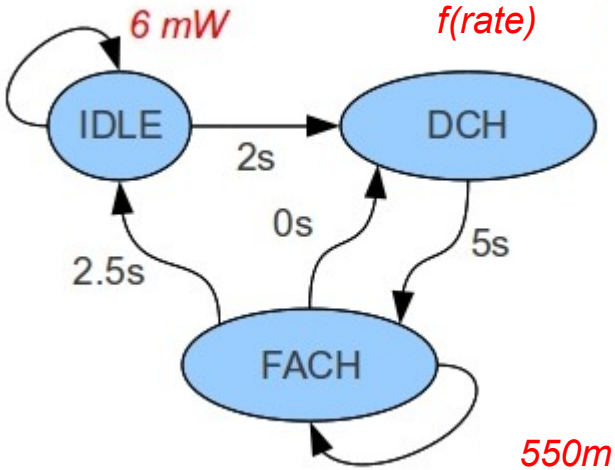


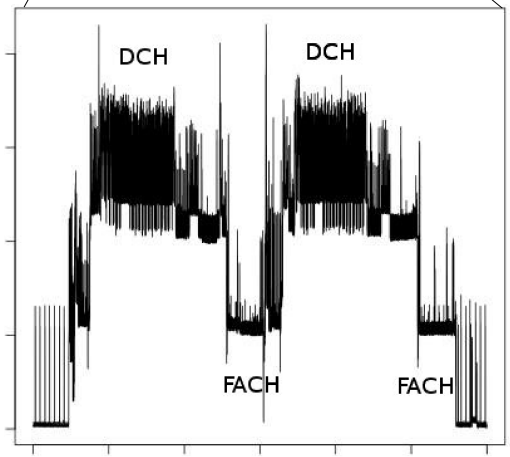
Trade-offs for 3G and Wifi energy consumption

This is a sample run showing 3G power consumption with Youtube traffic:

3G



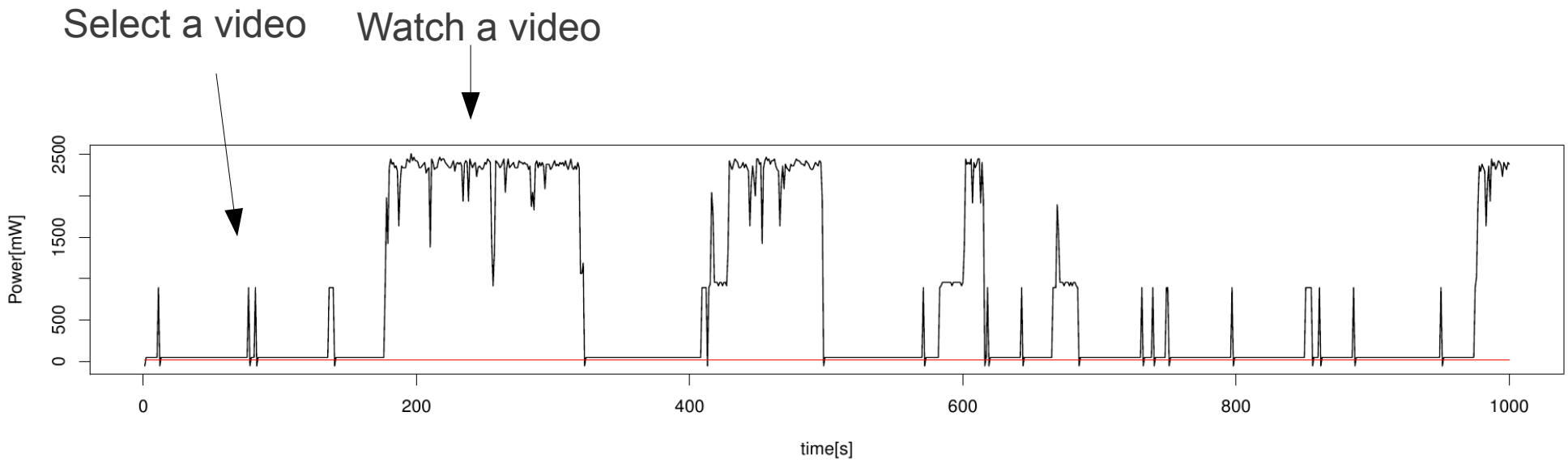


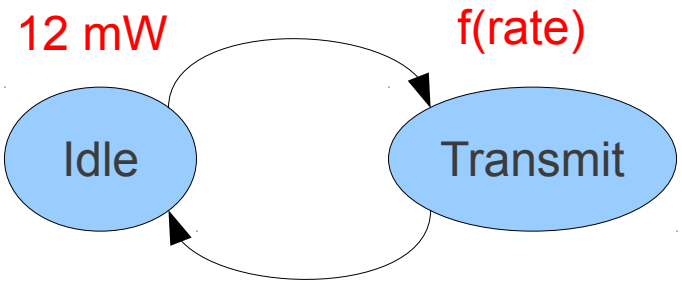


Trade-offs for 3G and Wifi energy consumption

This is a sample run showing Wifi power consumption with Youtube traffic:

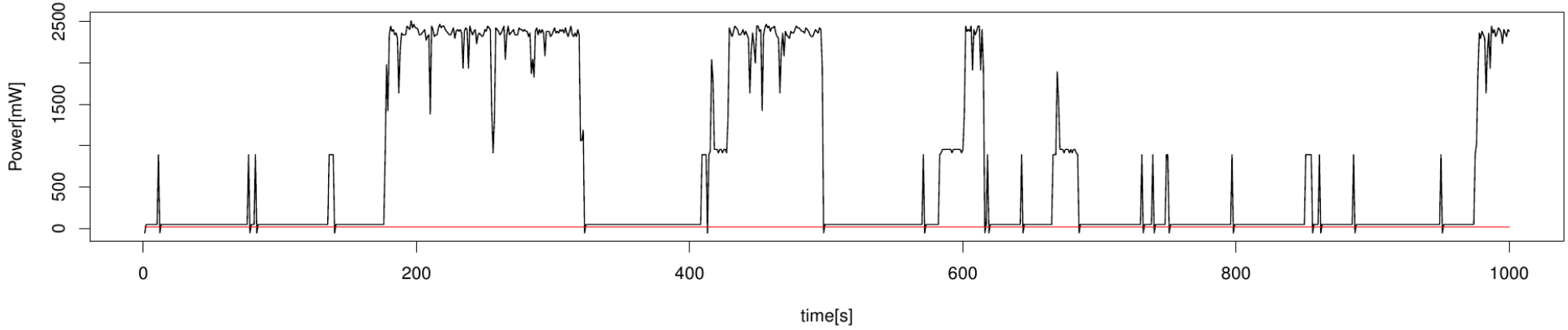
Wifi



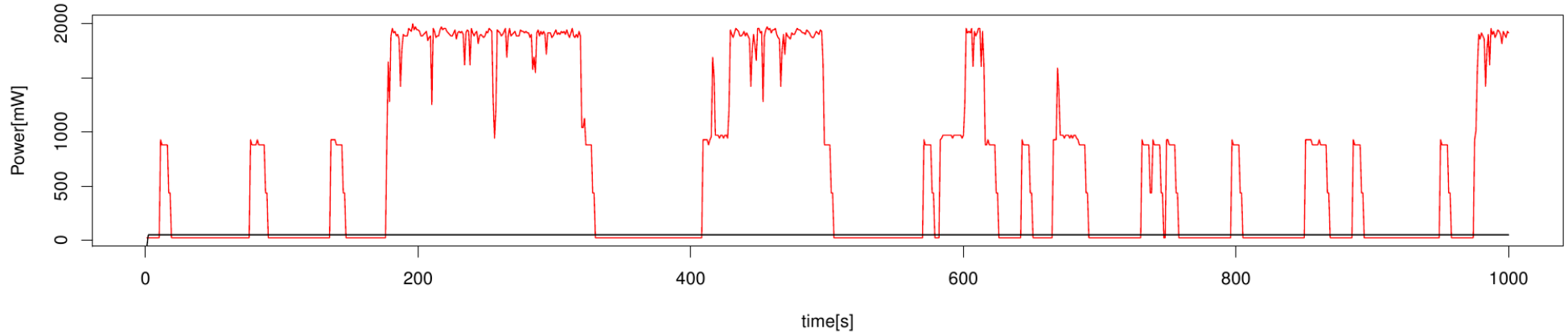


Trade-offs for 3G and Wifi energy consumption

Wifi



3G



Observations:
- Energy models are tricky to approximate in simple models.
- Power efficiency of different interface differs depending on the bitrate.

Preliminary research showed:
Opportunistic switching wastes even more energy than sticking to the worst interface!

The State of the Art for Switching Flows

You are listening to BBC iPlayer radio on Wifi at home.

As you are leaving home you lose Wifi connectivity and the iPlayer tries to reconnect.

After a while, the application gives up and even connecting 3G manually does not reestablish the stream.

Simple solution: switch manually or set up the phone in a way that it automatically switches over to 3G if it loses Wifi connectivity.

Problem: This solution also breaks the ongoing flows if the application does not try to reconnect by itself.

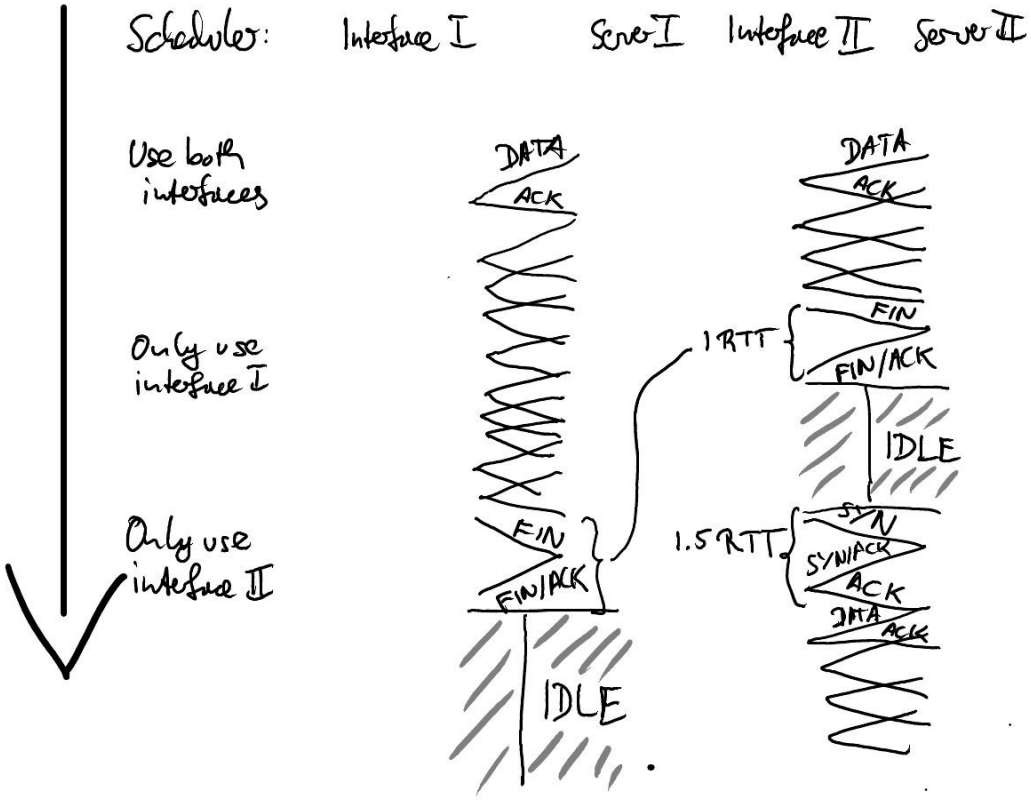
Multipath TCP – an overview

- MPTCP is a multipath extension of TCP that is currently being standardised by the IETF.

- Subflow windows can be coupled or uncoupled.

- Linux kernel implementation is underway
→ adds MPTCP capabilities in a shim layer under standard TCP
→ standard applications are unaware of the presence of MPTCP

- scheduling can be achieved by adding and deleting subflows dynamically



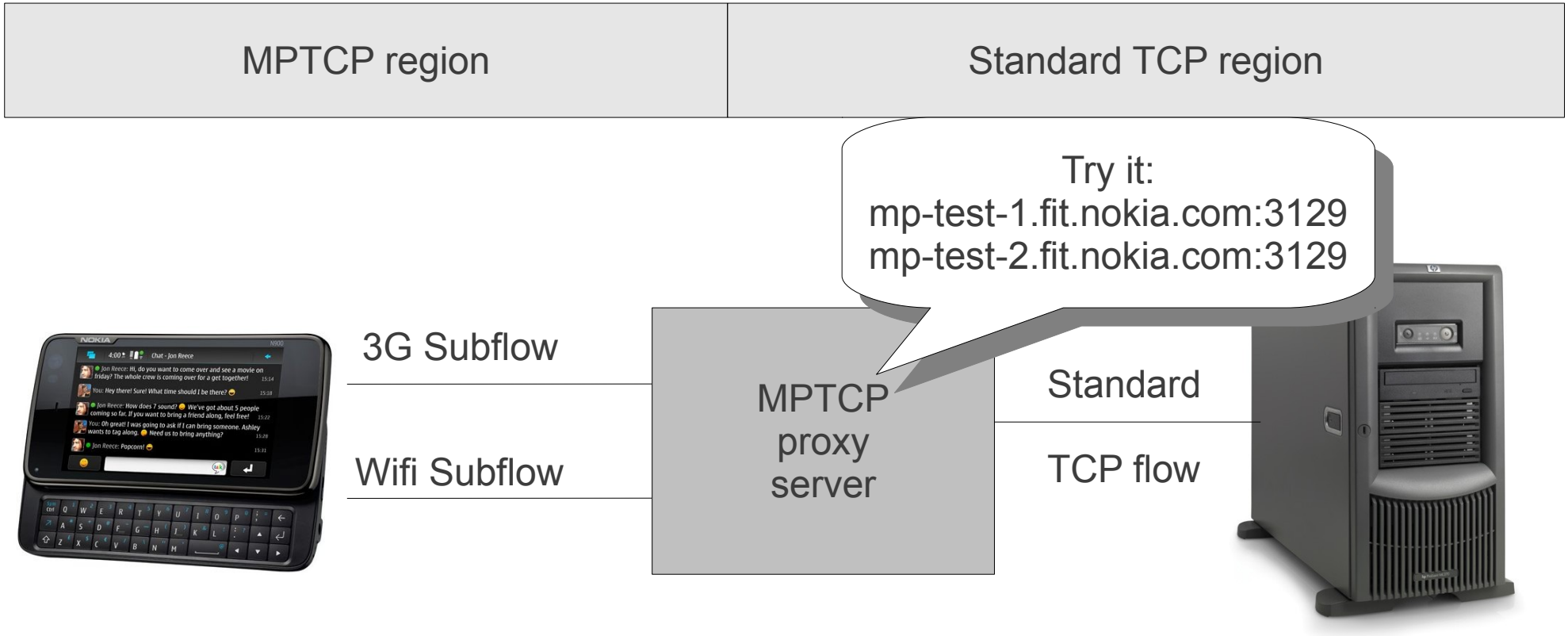
Related work

Can MPTCP be used for scheduling on mobile phones?
Ovidiu Popa, Msc thesis
→ mobile phone energy model
→ computer with mobile broadband and wifi
→ simple heuristic algorithms for load balancing

Does it make sense to use load balancing on different paths for scheduling on the uplink and on the downlink?
Many papers treat scheduling on the uplink using simple energy models and deadline constraints.
One prominent paper explores scheduling on the downlink. They use context information to decide if it is better to transfer a file with a given size on 3G or if it is better to switch to wifi
[Context-for-Wireless, Rahmati, MobySys, 2007].

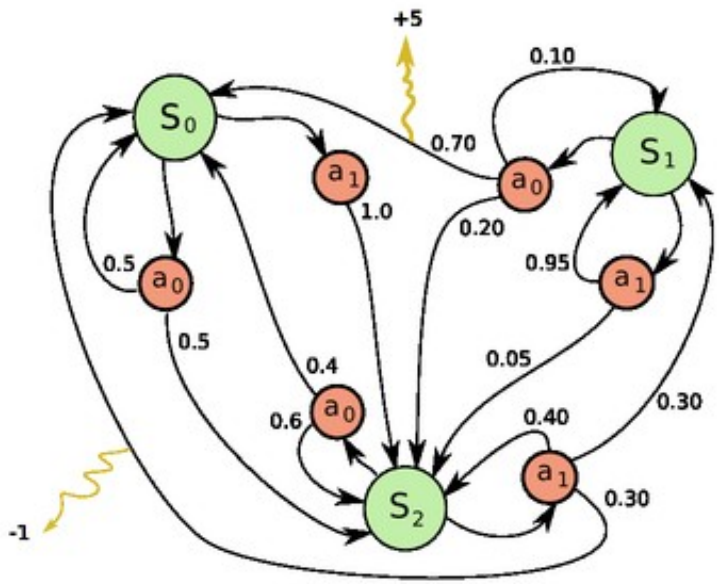
What is our contribution?
We propose an architecture based on MPTCP and show how to derive near-optimal schedulers in a completely automated way.

MPTCP scheduler infrastructure



An Overview of Markov Decision Processes (MDPs)

Markov Decision Processes are used for modelling dynamic problems involving decisions (actions) that have to be taken in certain states.



from: http://en.wikipedia.org/wiki/Markov_decision_process

An MDP consists of:

- a state space $S=\{S_0,S_1,S_2\}$
- a set of actions $A=\{a_0,a_1\}$
- transition probabilities P(y|x): probability of ending up in state y when taking action a in state x
- a reward function R(x,y,a): the reward for ending up in state y when taking action a in state x

The problem is: choose the best action in each state such that the expected cumulative long term rewards are maximised.

Formally, $\text{maximise } \mathrm{E}\left[\sum_{t=0}^{\infty}\gamma^t \mathrm{R}(s_t)|\pi\right] \text{ over } \pi \in \{0,1\}^k$ where gamma is a discount factor.

→ There are efficient algorithms to solve for the optimal policy.

Our scheduling problem can be cast as an MDP.

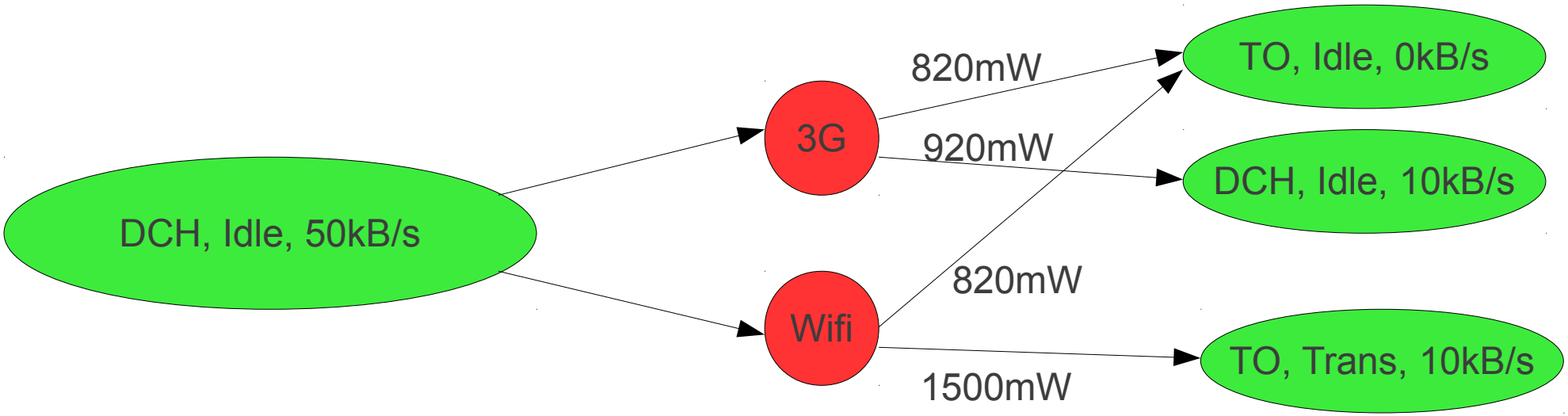
The scheduling problem can be described as an MDP with the following properties:

- state space (3G state, wifi state, throughput) where 3g state and wifi state are all interface states in the FSM description of the interface energy model.
  Each state will be occupied for Δt time units before taking the next decision and moving on.

- transition probabilities: From the finite state machine for 3G and Wifi states and the throughput transition probabilities from the application model it is easy to work out the joint probability distribution

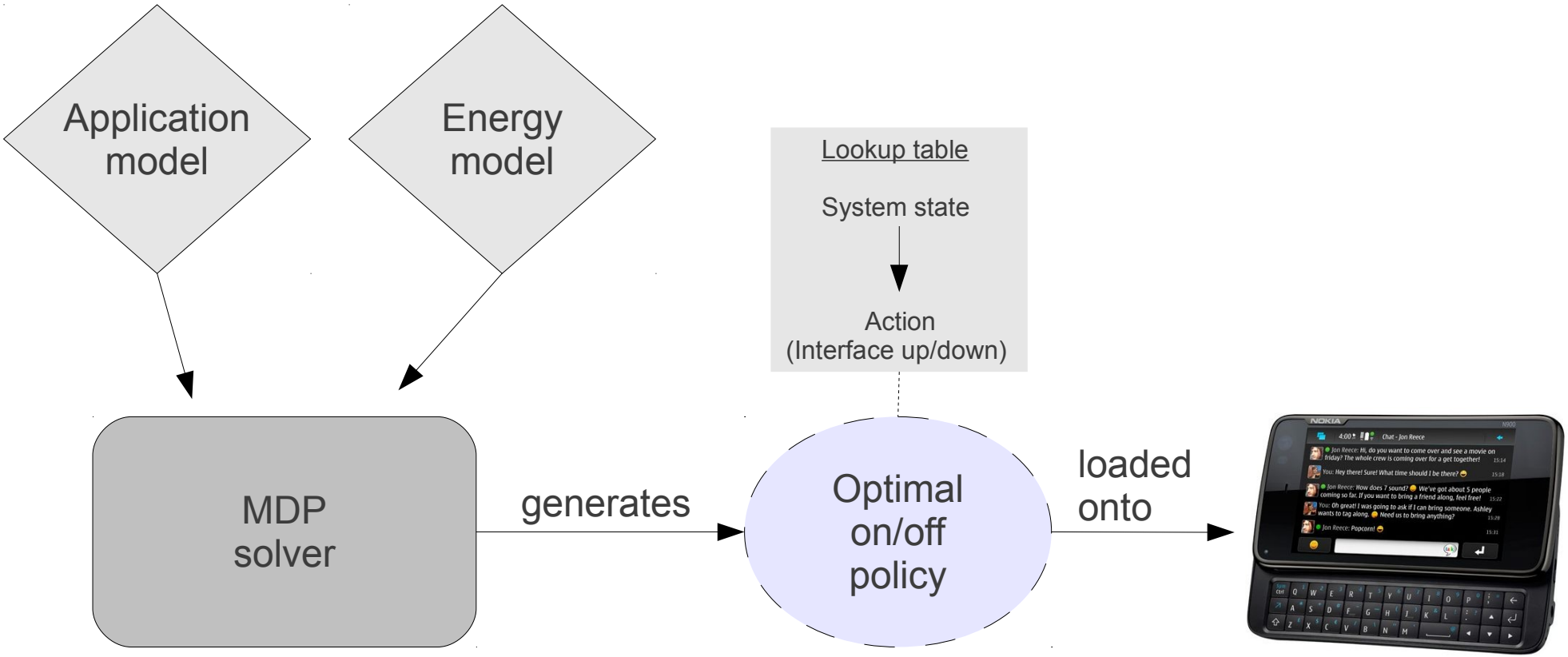
- actions: send on 3G or send on Wifi.

- rewards: negative energy consumption for the last Δt time units



Scheduler Generation

A completely proof-free solution!



The scheduler can adapt to the user.

The basic application model is generated for an average user based on traffic measurements on large scale which are currently under way at Nokia.

The user has two ways to influence the scheduler:

- The user can set weights for preferred interfaces, for throughput, delay, and resilience requirements. There are different precalculated schedulers associated with each application which are all installed in one go.
- The user can upload traffic measurements to personalise the scheduler. The MDP is solved offline in the cloud and offered for download when ready.

Some facts about the solution process:

- Bellman's value iteration algorithm (1957) solves the dynamic programming equation for the scheduler.
- It needs about 50 iterations for a reasonably small error using a discount factor of 0.9
- Generating the scheduler takes about four minutes on my laptop (1.3 GHz processor).
- The uncompressed scheduler table is 100KB, but can be compressed.

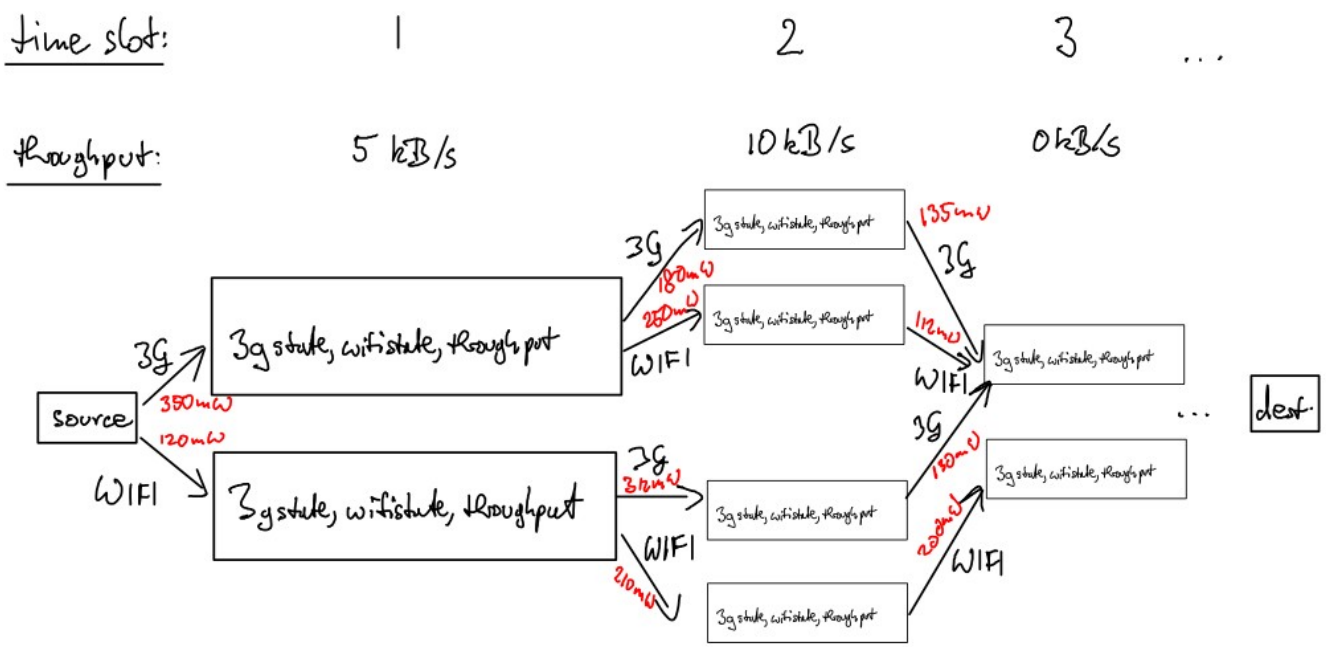
Reference algorithms

We compare the performance of the MDP scheduler to three reference algorithms:

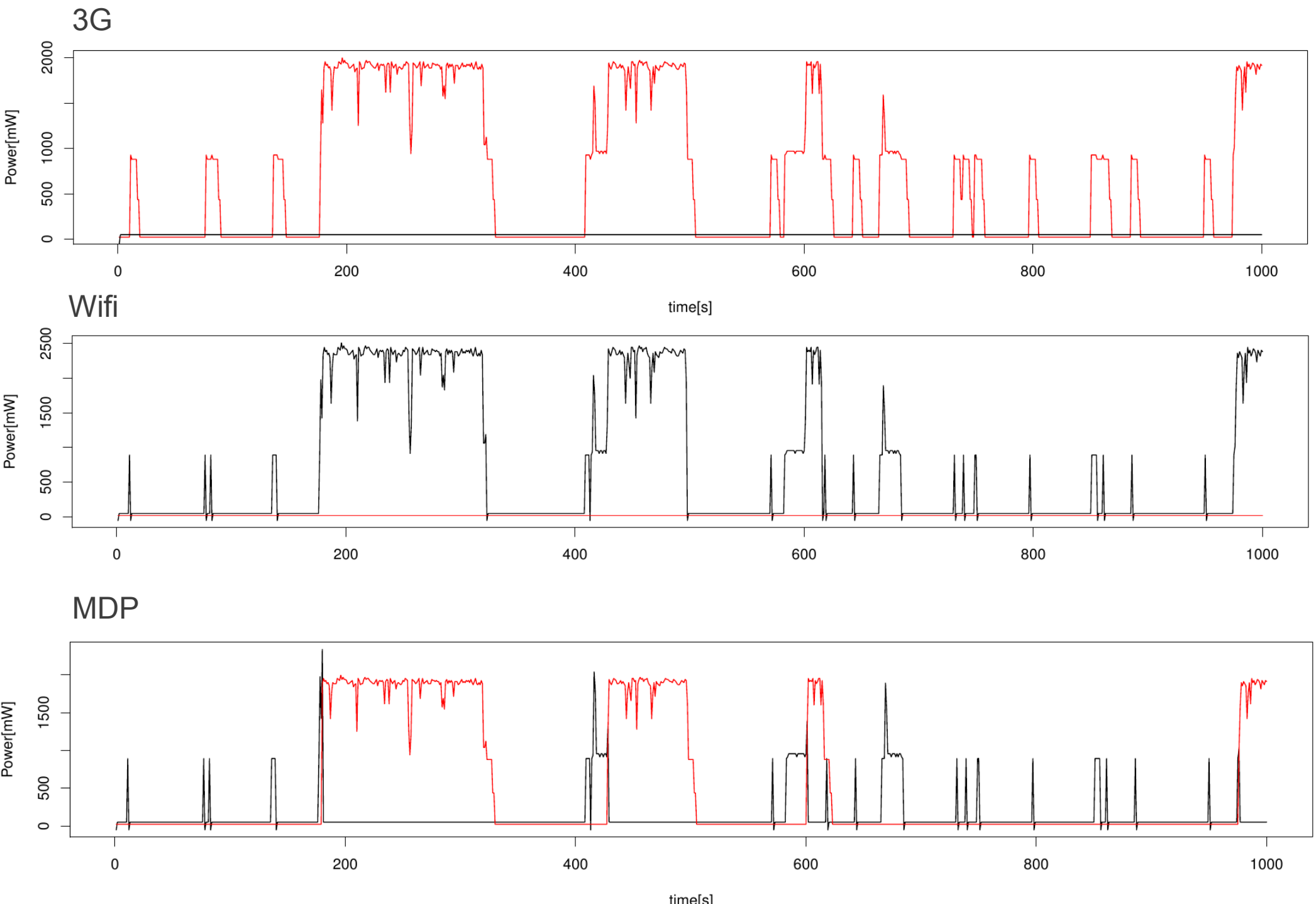
- single interface policies:
  - → 3G
  - → Wifi
- omniscient oracle scheduler
  - → knows entire future of the process
  - → solves a shortest path problem in the policy space

This is how it is done in practice

This is as good as it can ever get

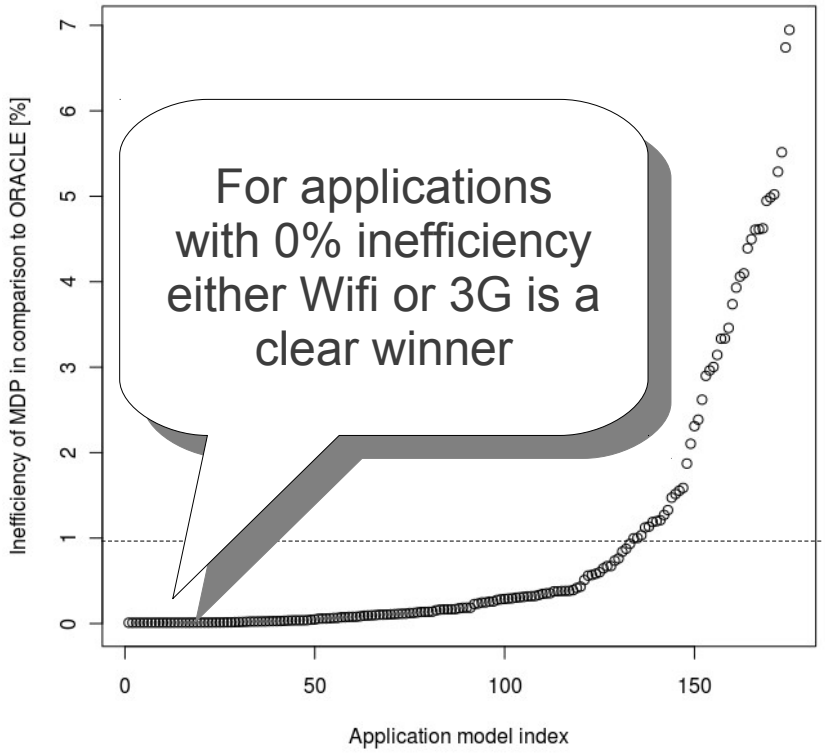


A sample run of the MDP scheduler with Youtube traffic



The scheduler is robust against a wide variety of application models

We used the Nokia N900 energy model for Wifi and 3G and generated random application throughput models. The graph shows the inefficiency of the MDP policy against ORACLE.

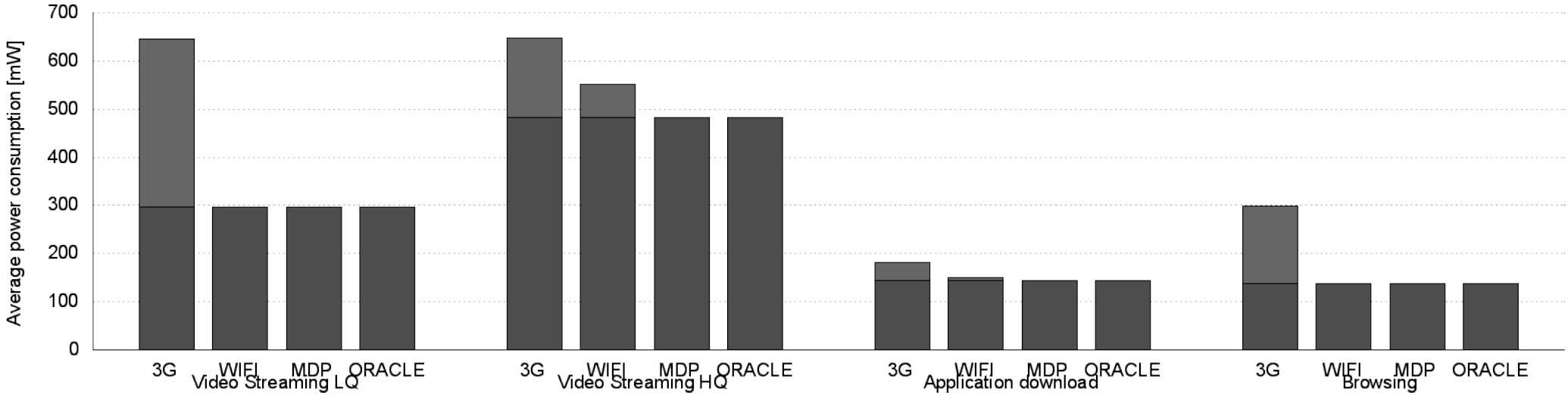


MDP in comparison to 3G:
Maximum 78% less energy
Average 11% less energy

MDP in comparison to Wifi:
Maximum 18% less energy
Average 4% less energy

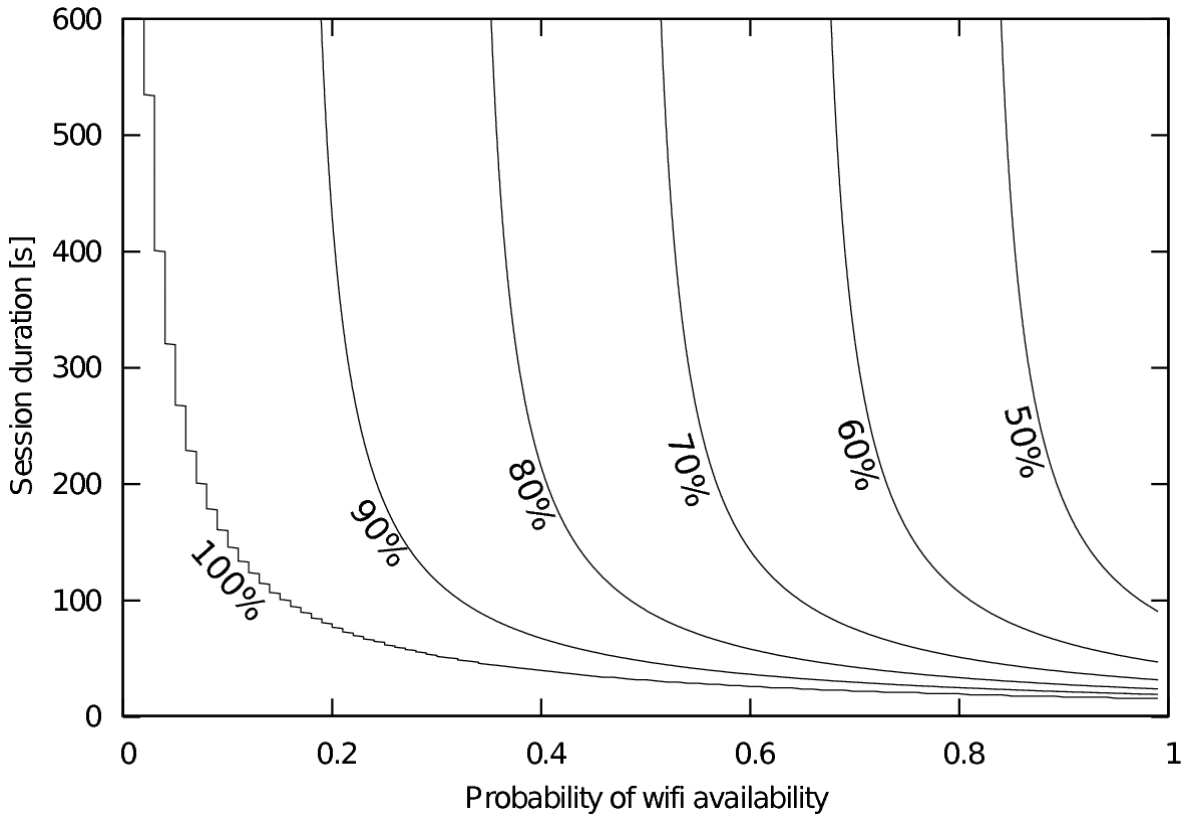
For 134 out of 175 (77%) applications the power consumption of MDP is within 1% of the power consumption of ORACLE.

For realistic traffic models the MDP scheduler matches the oracle policy



For high bandwidth applications, it makes sense to switch flows seamlessly during operation.

Wifi offloading with MDP



Wifi offloading adds an environment variable to the model.
While 3G is assumed to be always available, Wifi is available with a given probability.
Wifi offloading with MDP means, if the scheduler decides to probe for Wifi availability, it will use 3G and Wifi simultaneously when available to even reduce energy consumption.

Summary and Outlook

– We proposed an **architecture** for seamless flow switching.

– We showed how to describe the problem of deriving near-optimal scheduling rules for the architecture using **MDP modelling.**

– Evaluations show that the MDP policy **matches an unachievable oracle scheduler** in most cases

What is **still missing**?

– Solving MDPs with many states takes time. We have to explore **heuristic solutions**. **Bandit algorithms** can be used to reduce the state space if there are many interfaces. **Approximation algorithms** to MDP programming are widely available.

– We are planning to present a **working implementation** within the next 6 months.

– What do you do with UDP flows? **Multipath UDP**.