Rethinking Internet Traffic Management

From Multiple Decompositions to a Practical Protocol

Martin Suchara

in collaboration with:

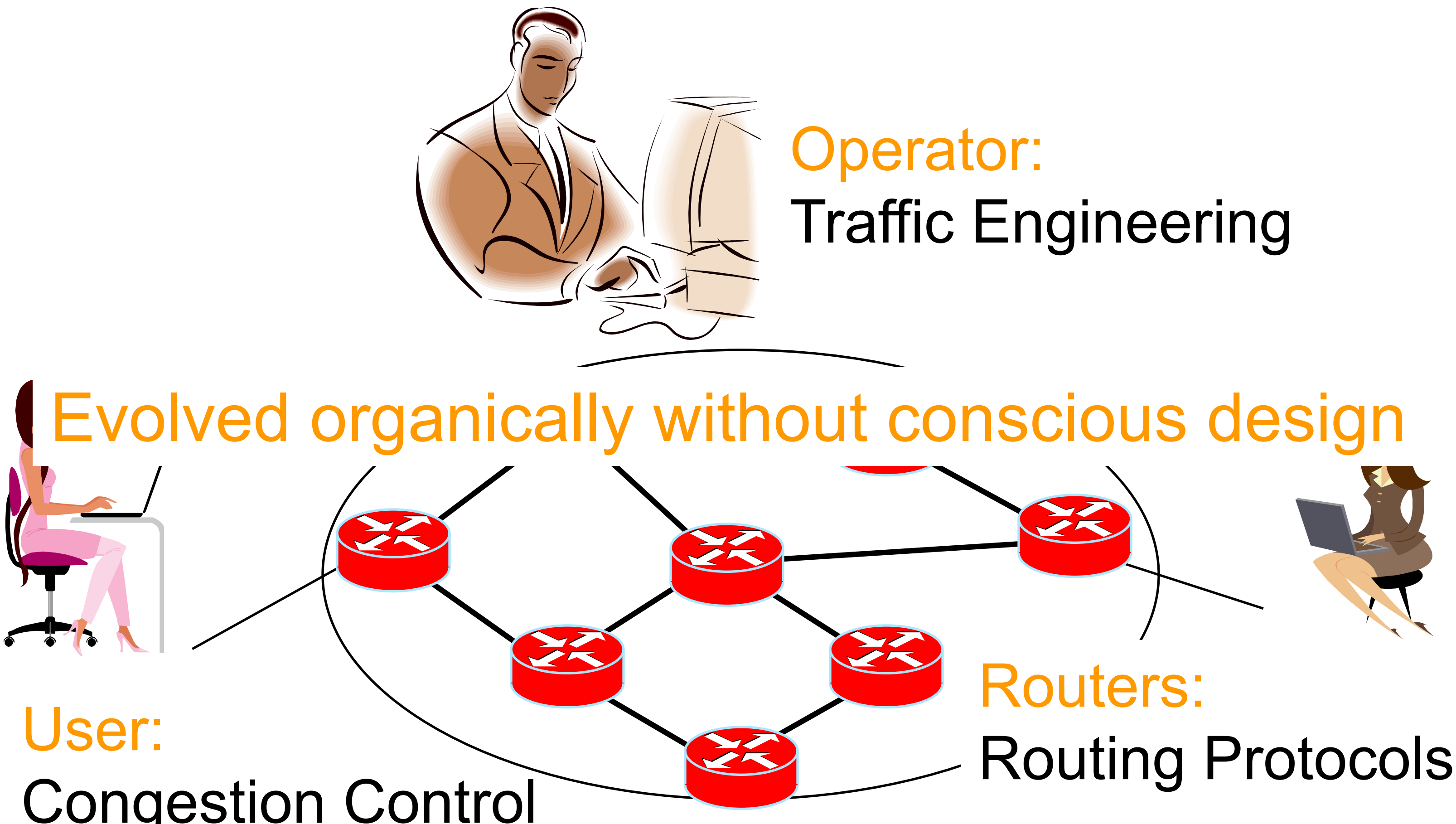
J. He, M. Bresler,

J. Rexford and M. Chiang

Why Study Traffic Management?

- Traffic management is important
  - Determines traffic rates and divides resources
  - Integrates routing, congestion control, traffic engineering, …
- The architecture has shortcomings
  - Suboptimal interactions of components
- Motivated by recent advancements in optimization theory research

Traffic Management Today



Evolved organically without conscious design

Shortcomings of Today's Architecture

- Protocol interactions ignored
  - Congestion control assumes routing is fixed
  - TE assumes the traffic is inelastic
- Inefficiency of traffic engineering
  - Link-weight tuning problem is NP-hard
  - TE at the timescale of hours or days

What would a clean-slate redesign look like?

Contributions of This Talk

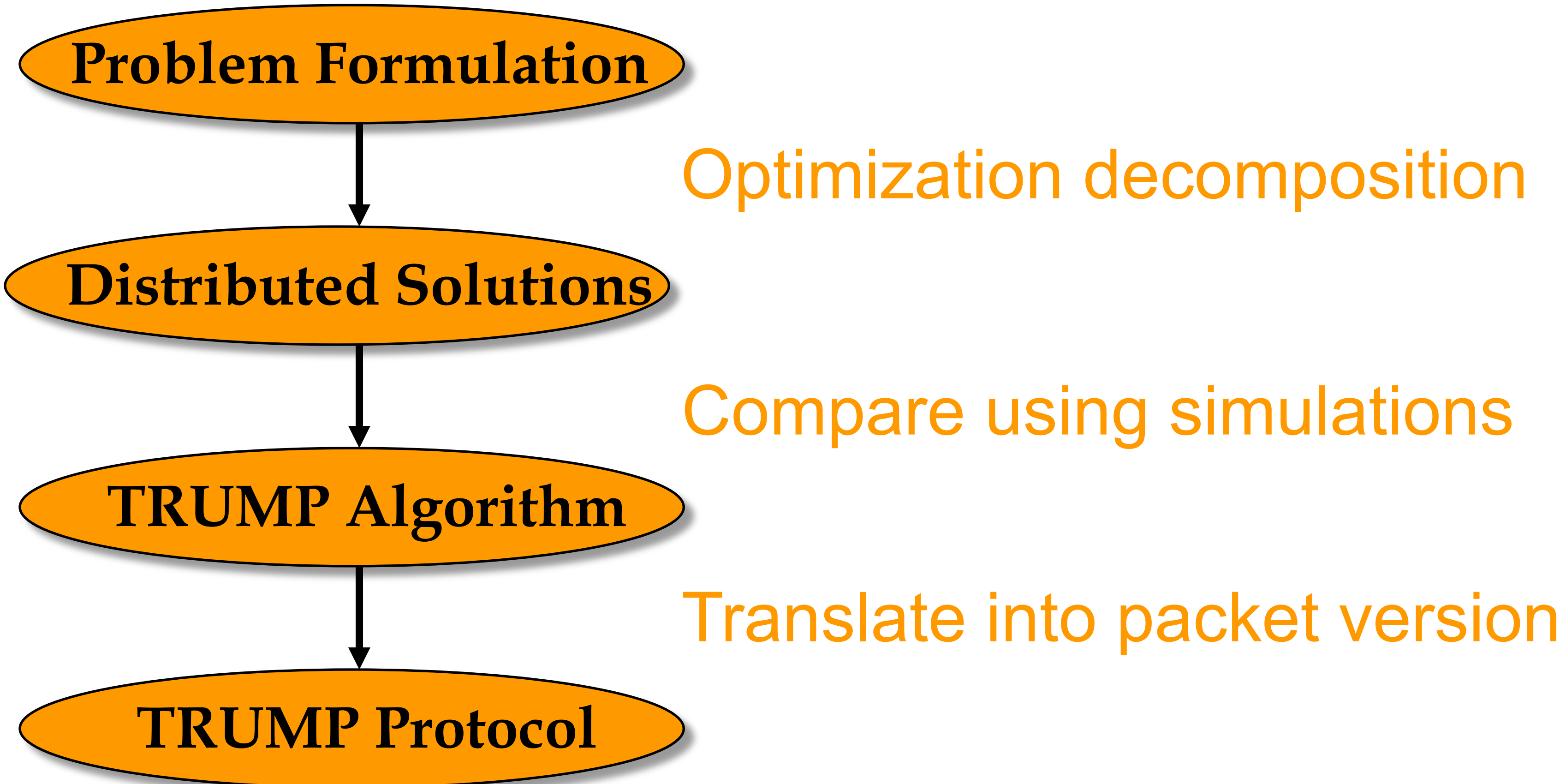
1. Case study of a design process
   - Based on optimization decompositions
   - Evaluations using simulation also needed

2. The new design
   - Of network traffic management

The next steps:

- Towards virtualized networks

Top-down Redesign



Congestion Control Implicitly Maximizes Aggregate User Utility

Source-destination pair indexed by i

User utility $U_i(x_i)$

Source rate x_i

aggregate utility

$$\max. \sum_i U_i(x_i)$$

Fair rate allocation amongst greedy users

$$\text{var.} \quad x$$

routing matrix

source rate

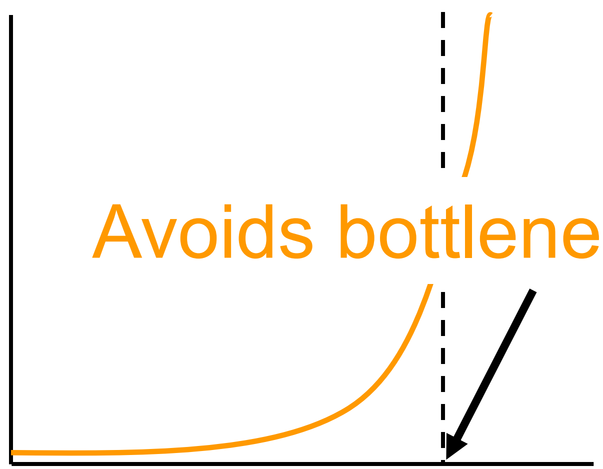
Utility represents user satisfaction and elasticity of traffic

Traffic Engineering Explicitly Minimizes Network Congestion

Links are indexed by l

aggregate congestion cost

Cost $f(u_l)$



Link Utilization u_l

$$\text{min. } \textstyle\sum_l f(u_l)$$

Avoids bottlenecks in the network

$_{li}x_i/c_l$

$$\text{var. } R$$

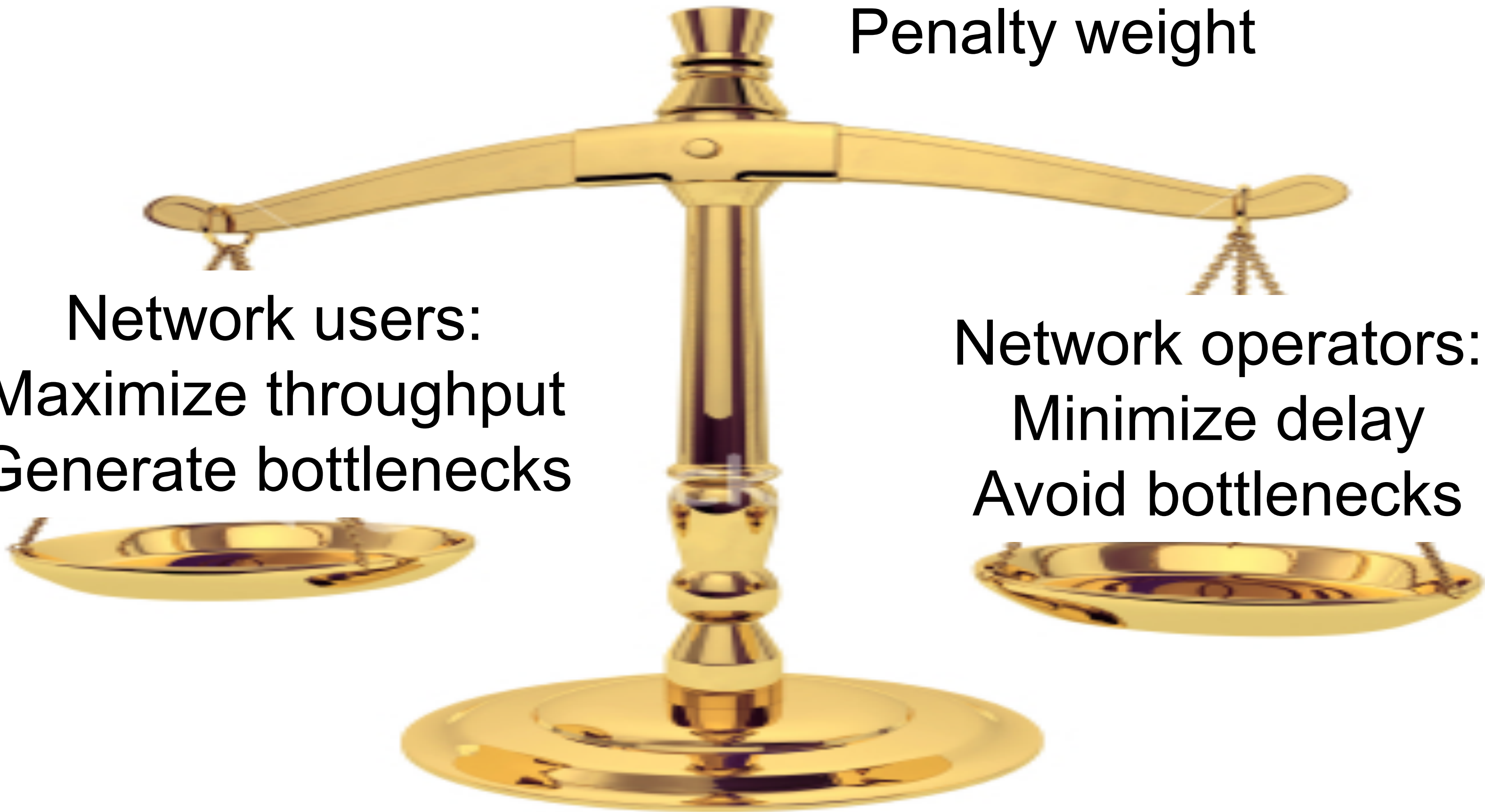
link utilization

Cost function is a penalty for approaching capacity

A Balanced Objective

$$\max. \sum_i U_i(x_i) - w\sum_l f(u_l)$$

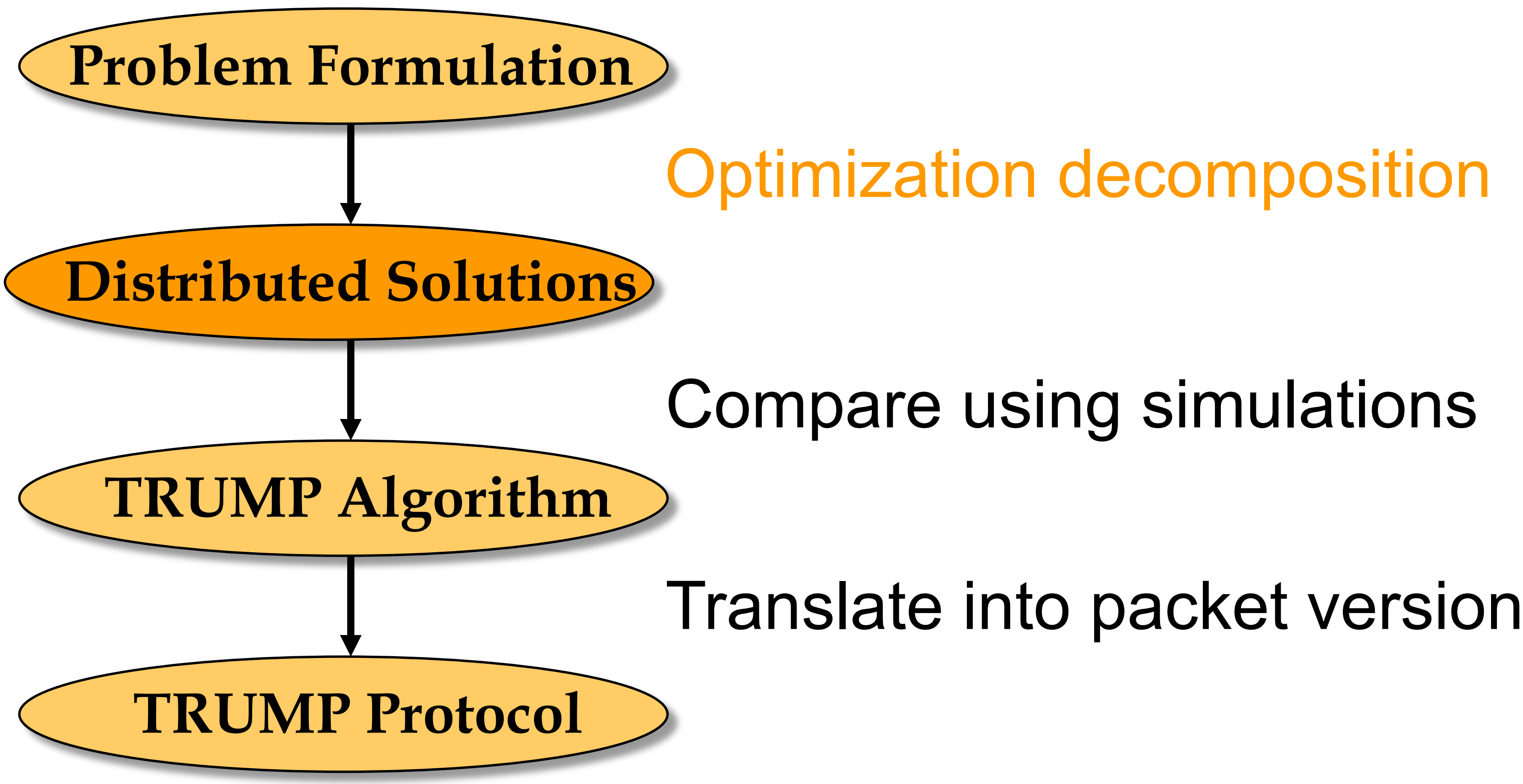
Penalty weight



Network users:
Maximize throughput
Generate bottlenecks

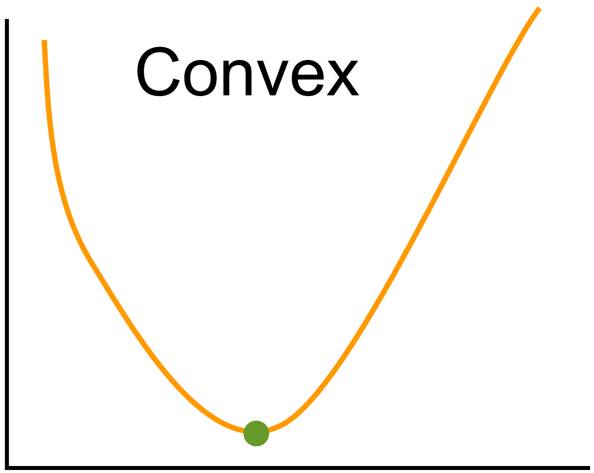
Network operators:
Minimize delay
Avoid bottlenecks

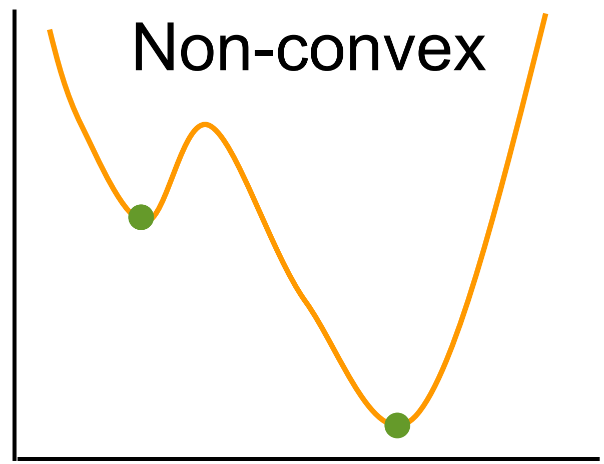
Top-down Redesign



Optimization decomposition requires convexity

Convex Problems are Easier to Solve





- Convex problems have a global minimum
- Distributed solutions that converge to global minima can be derived using decompositions

How to Make our Problem Convex?

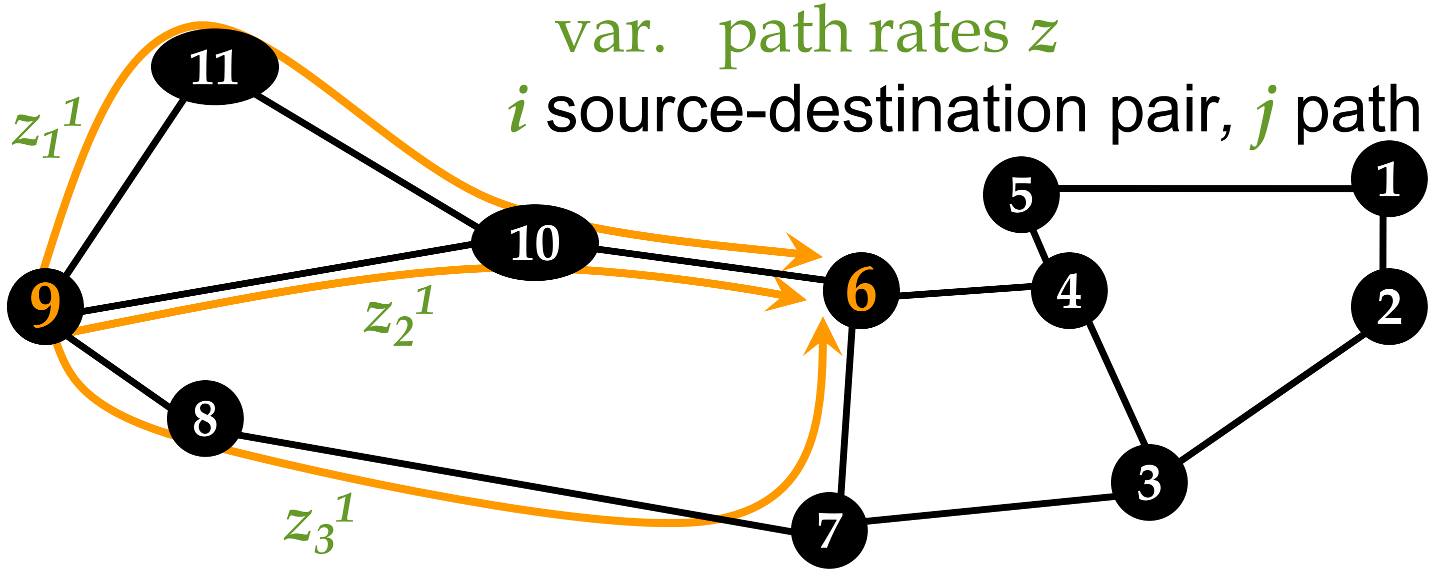
- Single path routing is non convex
- Multipath routing + flexible splitting is convex

max. $\sum_i U_i(\sum_j z_j^i) - w\sum_l f(u_l)$

s.t. link load $\leq c_l$

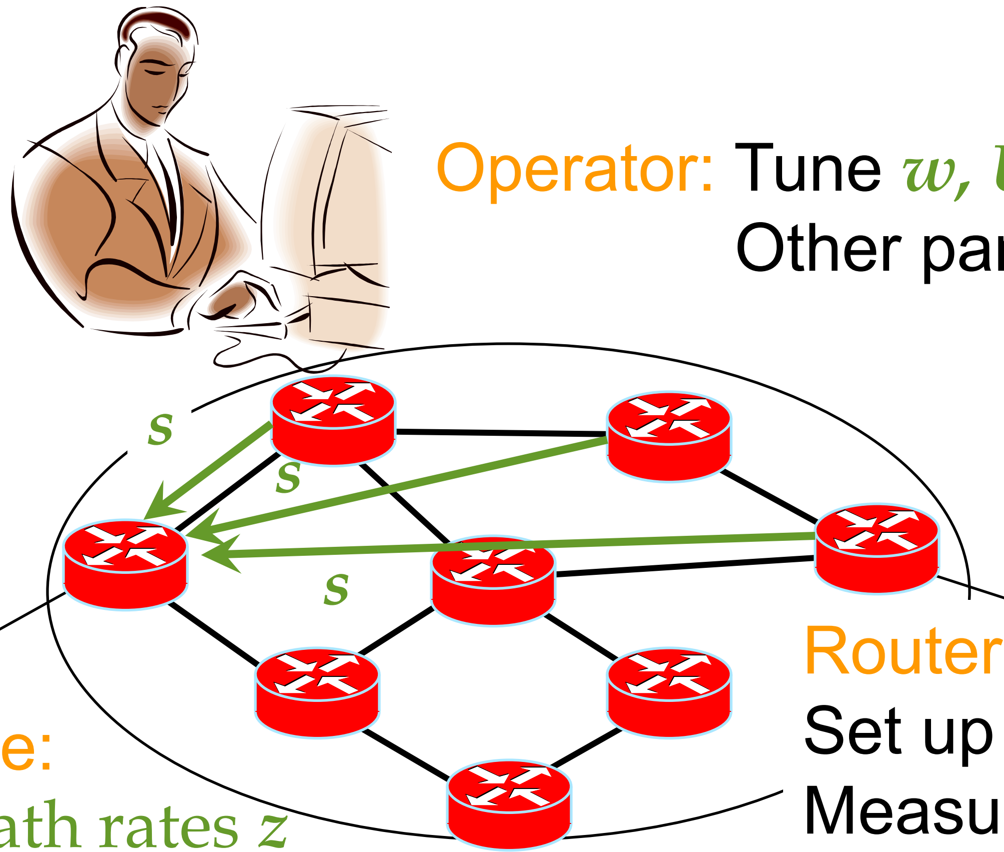
var. path rates z

i source-destination pair, j path number



Path rate captures source rates and routing

Overview of Distributed Solutions



Operator: Tune w, U, f
Other parameters

Edge node:
Update path rates z
Rate limit incoming traffic

Routers:
Set up multiple paths
Measure link load
Update link prices s

Role of Optimization Decompositions

- Derive price and path rate updates
  - Prices: penalties for violating a constraint
  - Path rates: updates driven by penalties
- Example: TCP congestion control
  - Link prices: level of packet loss or delay
  - Source rates: adjust window based on prices
- Our problem is more complicated
  - More complex objective, multiple paths

Key Principles I: Effective Capacity

- Rewrite capacity constraint:

$$\text{link load} \le c_l \longrightarrow \begin{array}{l}\text{link load} = y_l \\ \text{effective capacity } y_l \le c_l\end{array}$$

- Effective capacity y_l
  - Dynamically updated
  - Advance warning of impending congestion
  - Simulates the link running at lower capacity and give feedback on that
- Effective capacity keeps system robust

Key Principles II: Consistency Price and Subgradient Updates

- Consistency price p_l
  - Relax constraint $y_l \leq c_l$ but penalize violation with price p_l
  - Allow packet loss to converge faster
- Subgradient feedback price update:

$$p_l(t+1) = [p_l(t) - \text{stepsize}*(c_l - y_l(t))]^+$$

  - Stepsize controls the granularity of reaction
  - Stepsize is a tunable parameter

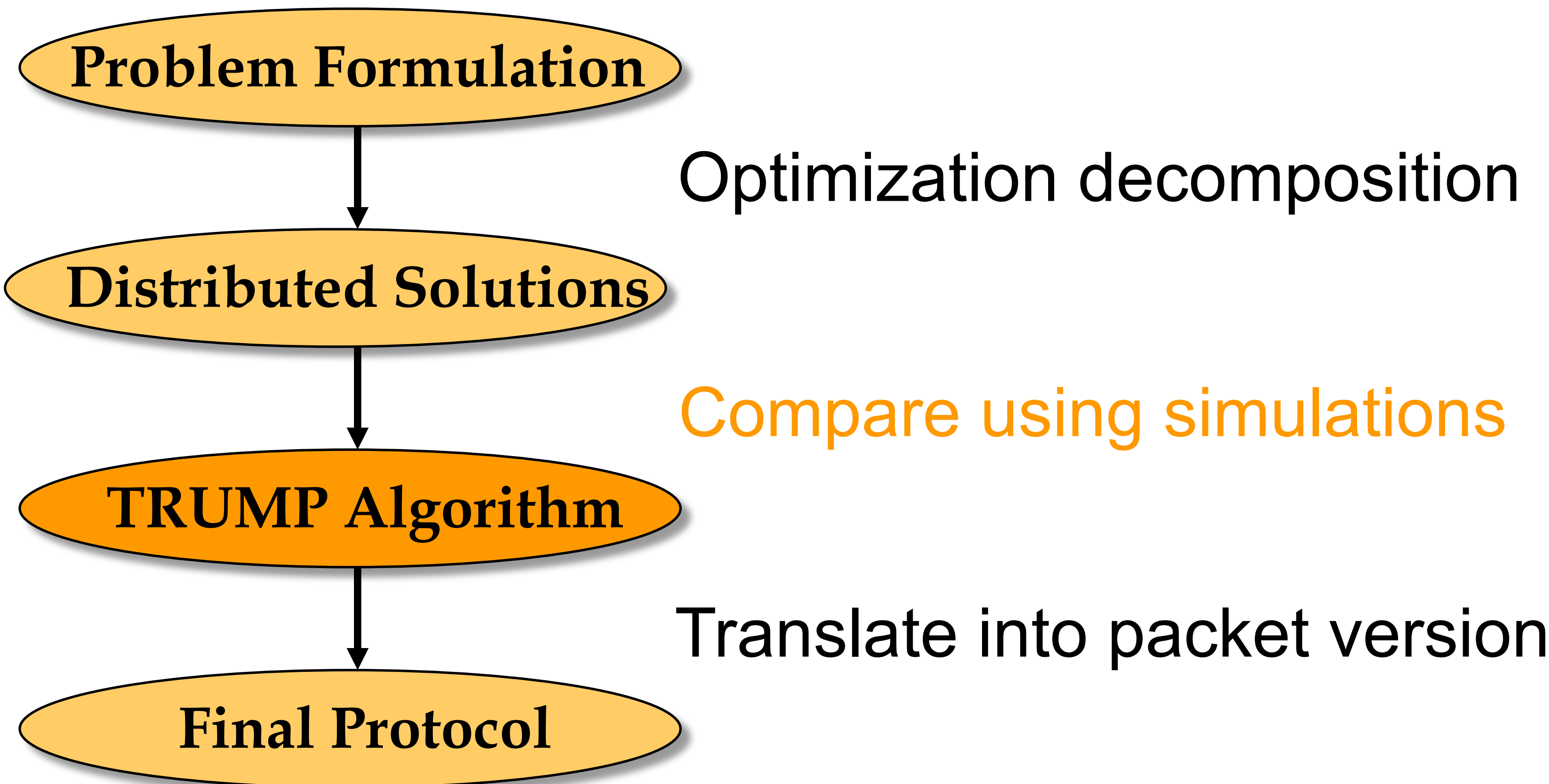
Four Decompositions – Differences

Differ in how link & source variables are updated

Algorithm	Features	Parameters
Partial-dual	Effective capacity	1
Primal-dual	Effective capacity	3
Full-dual	Effective capacity, Allows packet loss	2
Primal-driven	Direct price update	1

Iterative updates contain stepsizes:
They affect the dynamics of the distributed algorithms

Top-down Redesign

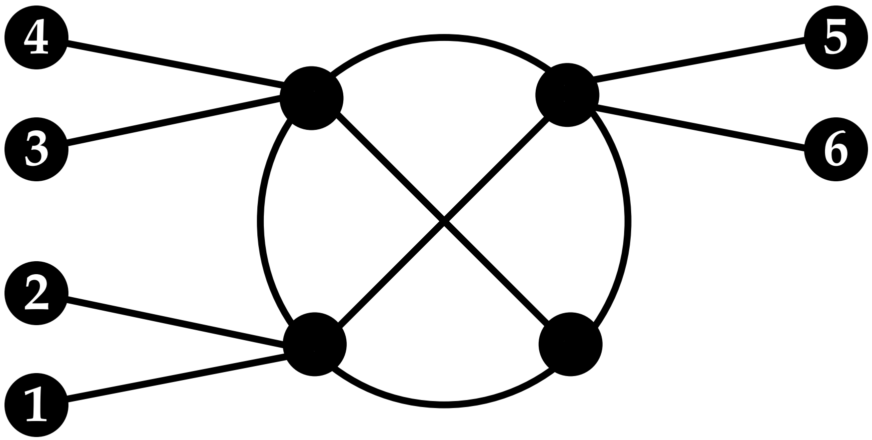


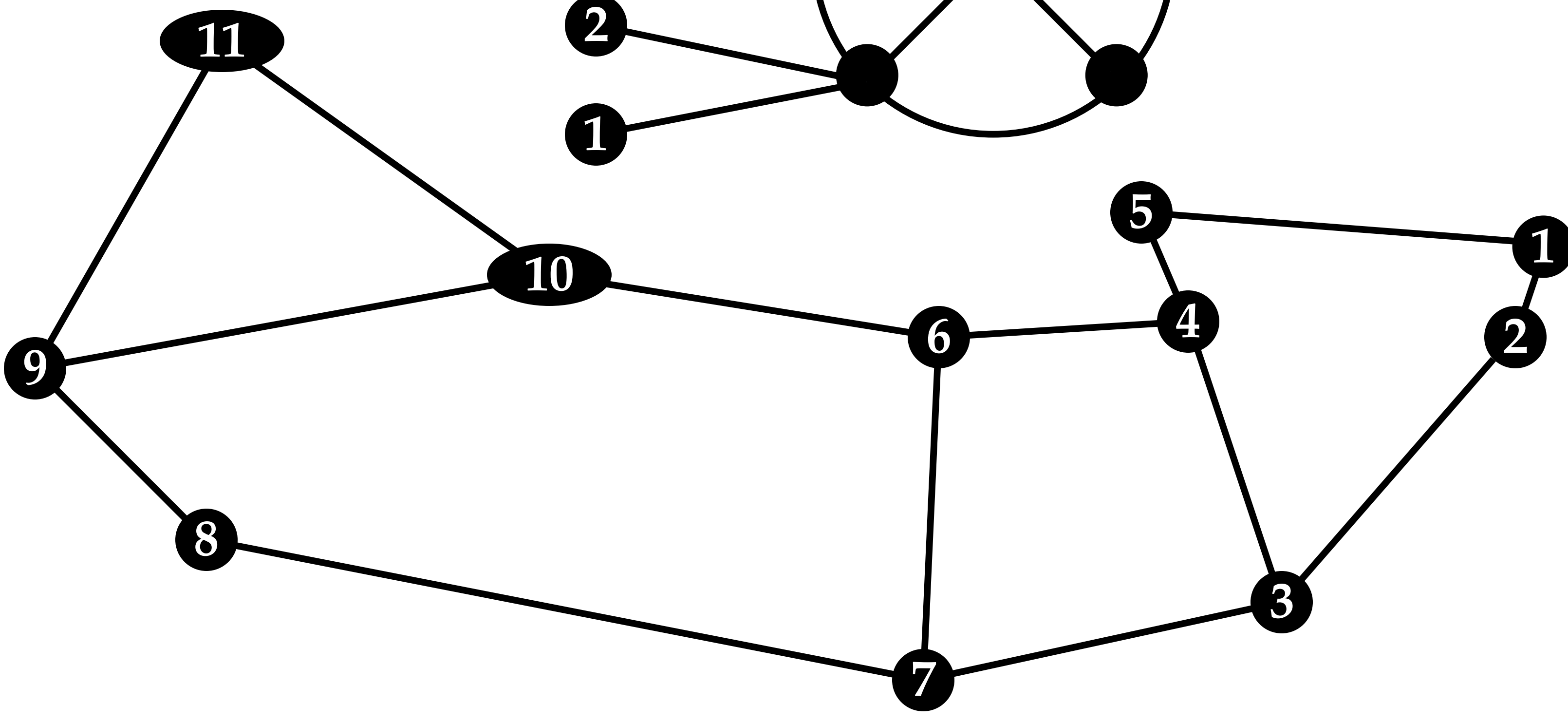
Optimization doesn't answer all the questions

Evaluating Four Decompositions

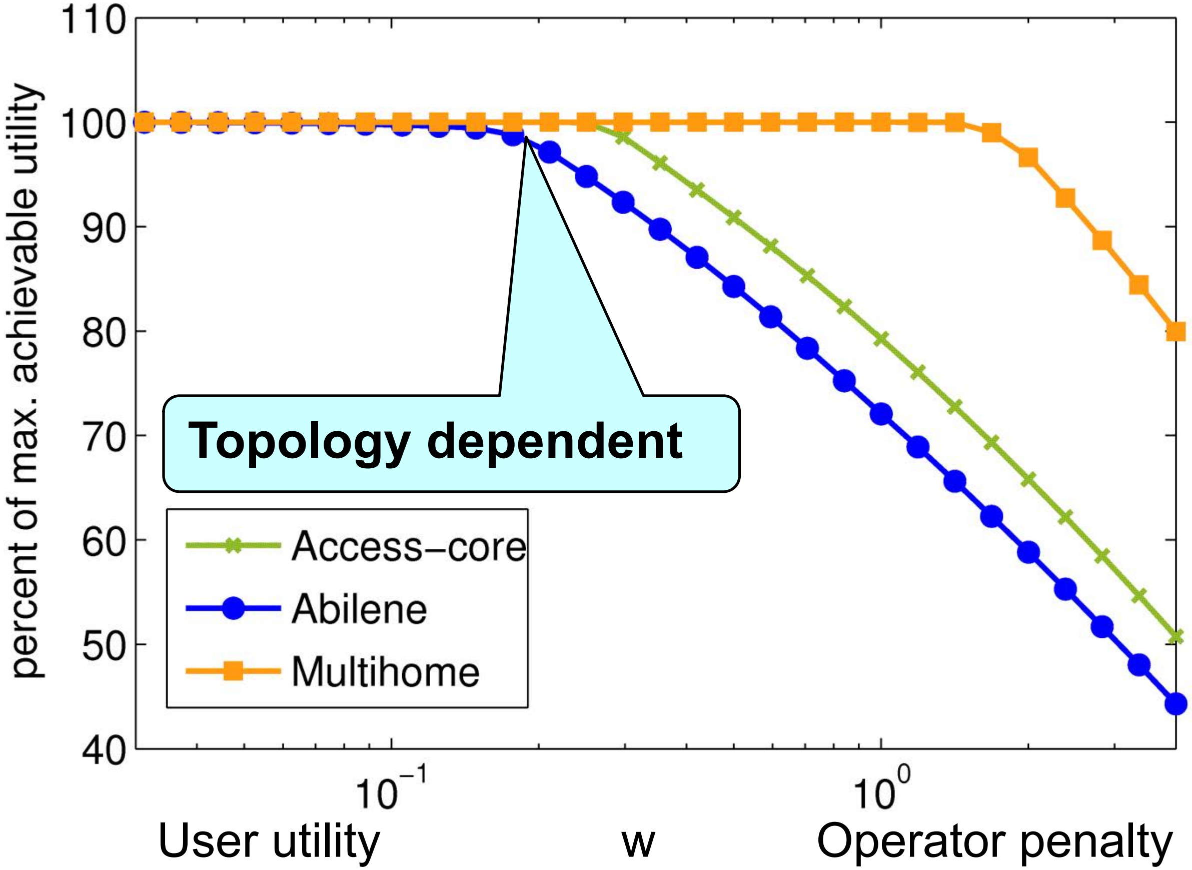
- Theoretical results and limitations:
  - All proven to converge to global optimum for well-chosen parameters
  - No guidance for choosing parameters
  - Only loose bounds for rate of convergence
- Sweep large parameter space in MATLAB
  - Effect of w on convergence
  - Compare rate of convergence
  - Compare sensitivity of parameters

Simple Topologies Used in MATLAB



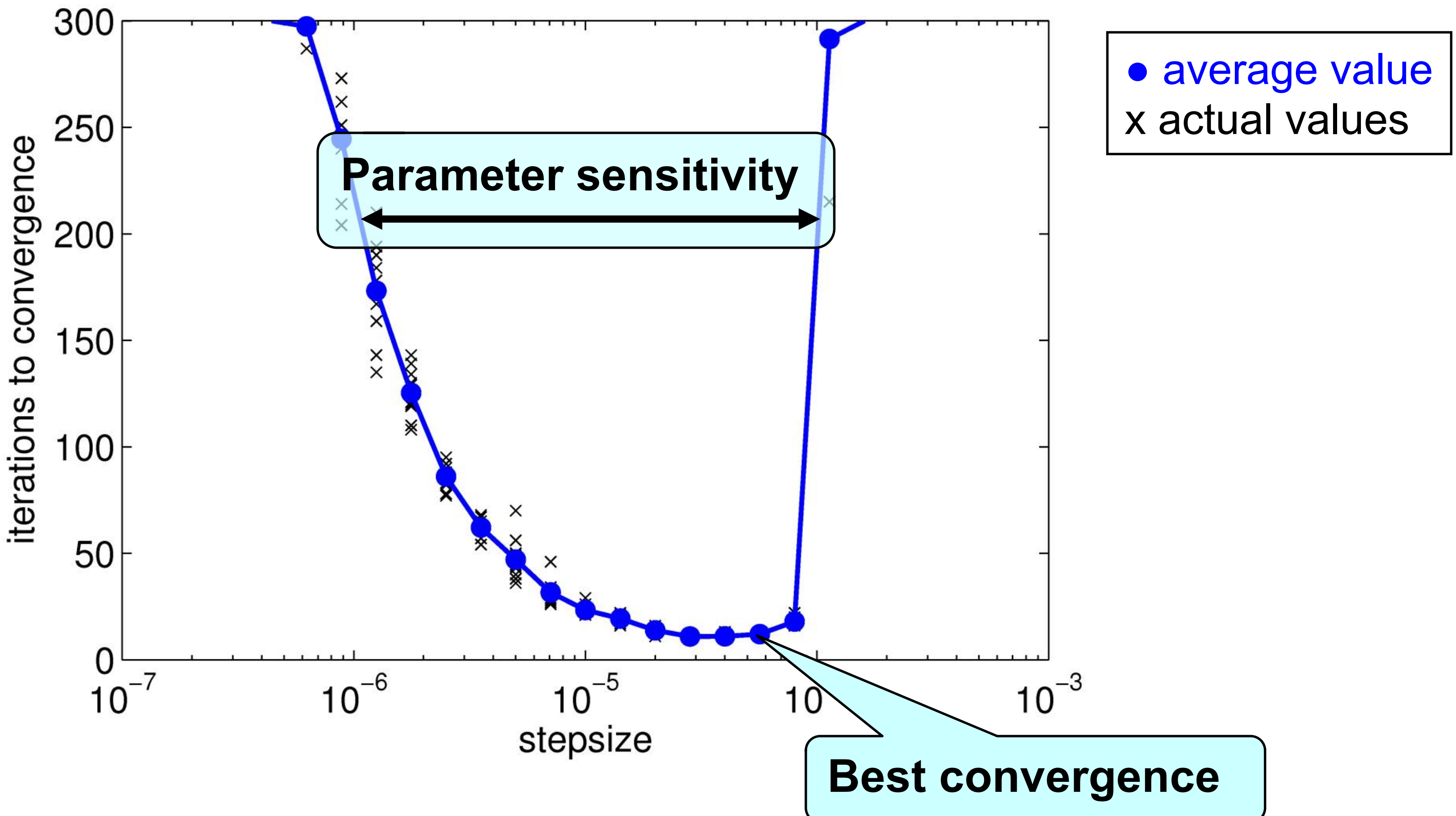


Effect of Penalty Weight w



Can achieve high aggregate utility for a range of w

Convergence Properties: Partial Dual in Access Core Topology



Tunable parameters impact convergence time

Convergence Properties (MATLAB)

Parameter sensitivity correlated to rate of convergence

Algorithms	Convergence Properties
All	Converges slower for small *w*
Partial-dual vs. Primal-dual	Extra parameters do not improve convergence
Partial-dual vs. Full-dual	Allowing some packet loss may improve convergence
Partial-dual vs. Primal-driven	Direct updates converge faster than iterative updates

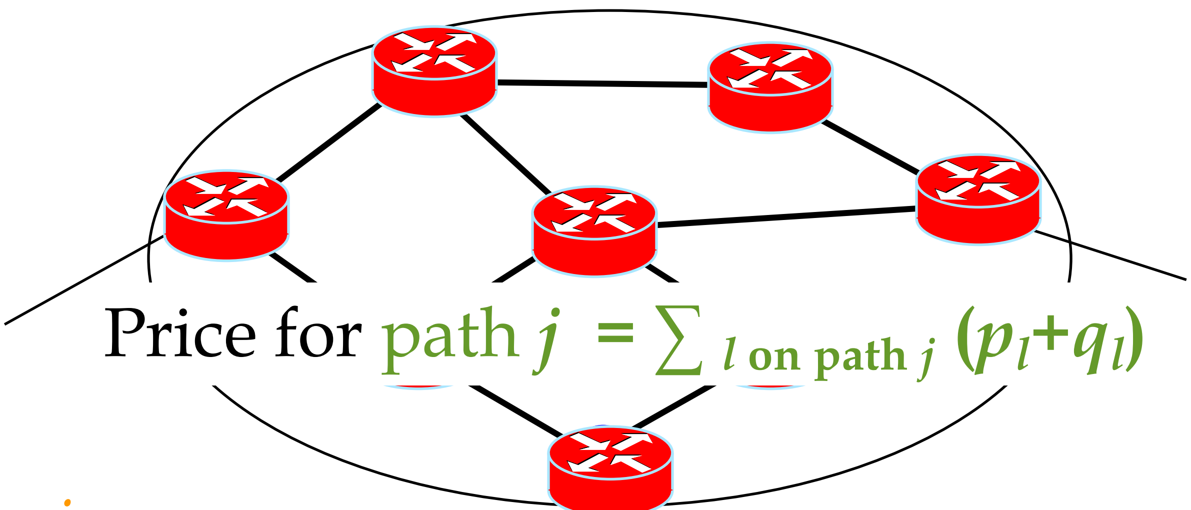
TRUMP: TRaffic-management Using Multipath Protocol

- Insights from simulations:
  - Have as few tunable parameters as possible
  - Use direct update when possible
  - Allow some packet loss
- Cherry-pick different parts of previous algorithms to construct TRUMP
- One tunable parameter

TRUMP Algorithm

Link l: loss $p_l(t+1) = [p_l(t) + \text{stepsize}^*(\text{link load} - c_l)]^+$

queuing delay $q_l(t+1) = wf'(u_l)$



Price for path j $= \sum_{l \text{ on path } j} (p_l+q_l)$

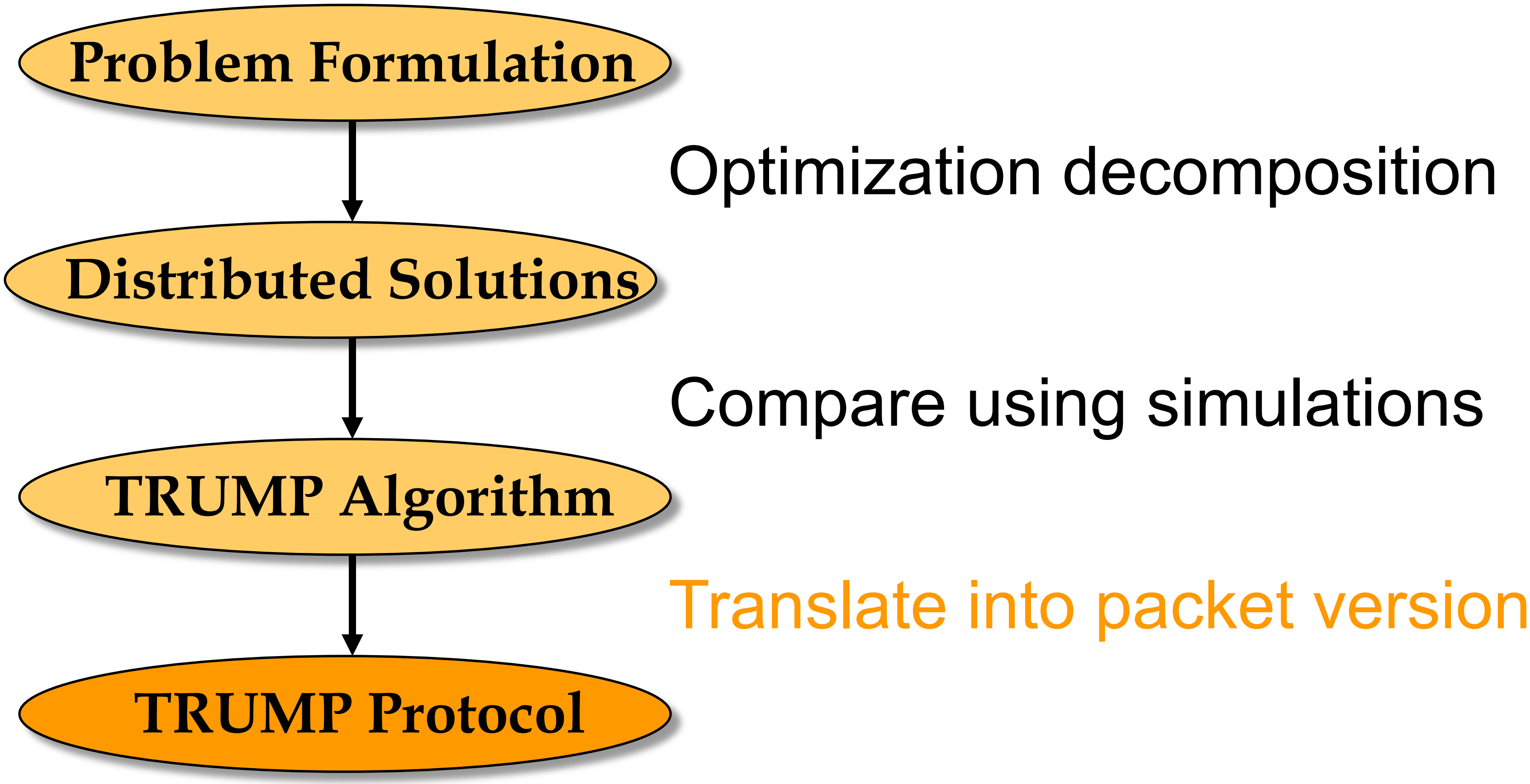
Source i:

Path rate $z_j^i(t+1) = \max.\ (U_i(\sum_k z_k^i) - z_j^i \,{}^*\text{path price})$

TRUMP Versus Other Algorithms

- TRUMP is not another decomposition
  - We can prove convergence, but only under more restrictive conditions
- From MATLAB:
  - Faster rate of convergence
  - Easy to tune parameter

Top-down Redesign

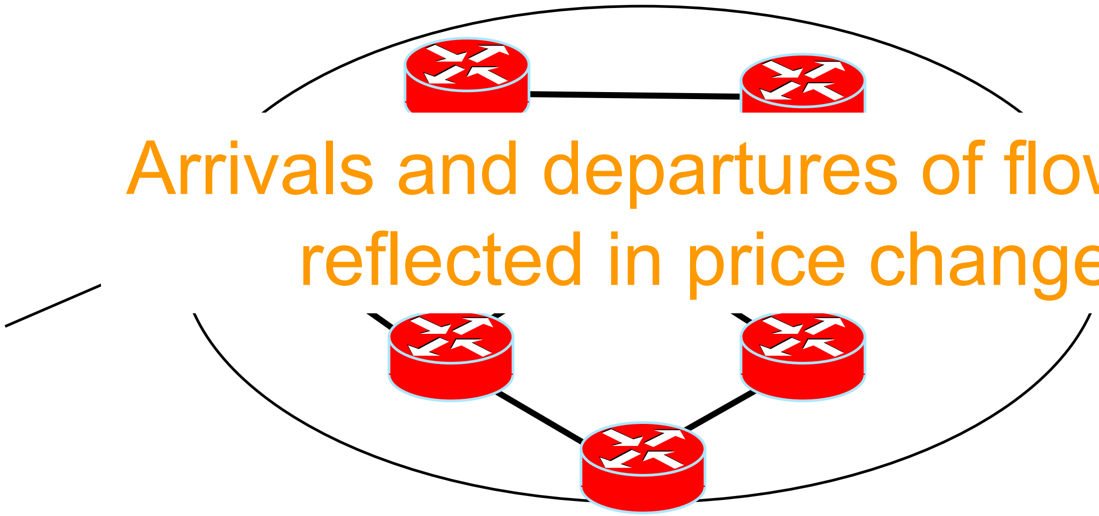


So far, assumed fluid model, constant feedback delay, greedy traffic sources

TRUMP: Packet-based Version

Link l: link load = (bytes in period T) / ($c_l T$)

Update link prices every T



Arrivals and departures of flows are reflected in price changes

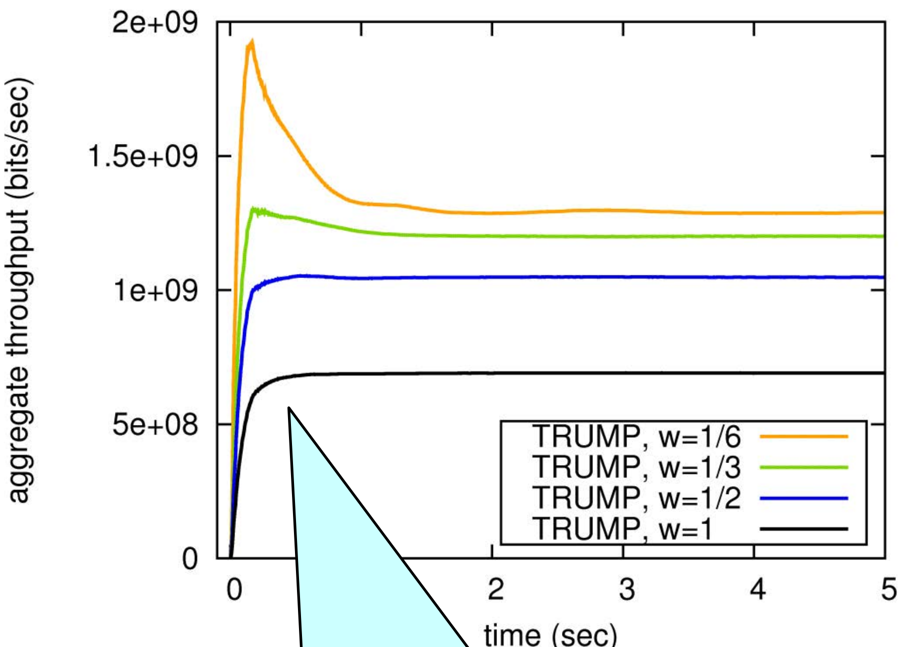
Source i: Update path rates at $\max_j \{RTT_j^i\}$

Packet-level Experiments in NS-2

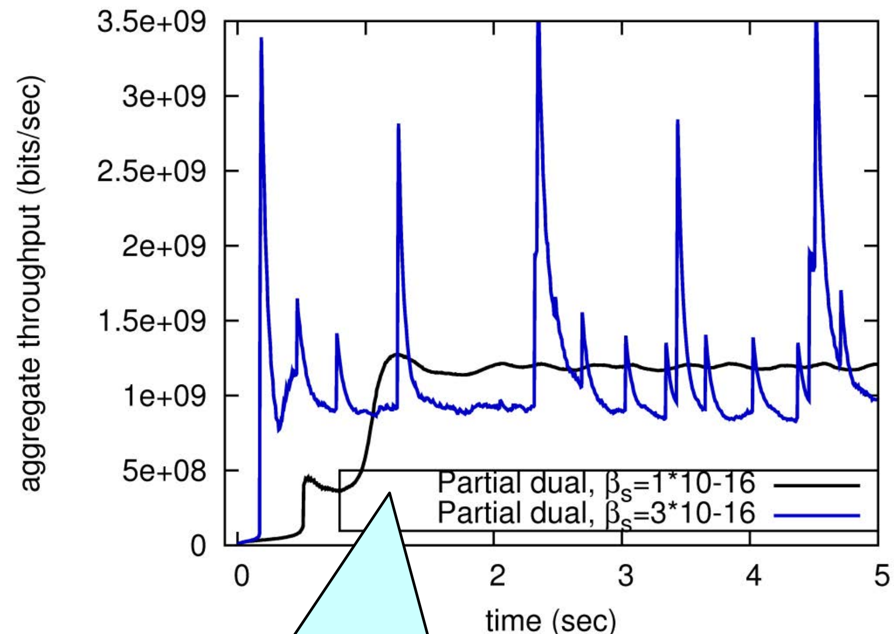
- Set-up:
  - Synthetic topologies + realistic topologies and delays of large ISPs
  - Multiple paths with 1ms to 400ms of delay
  - Realistic ON-OFF traffic model
- Questions:
  - Do MATLAB results still hold?
  - Does TRUMP react quickly to link dynamics? Can it handle ON-OFF flows?
  - Number of paths needed?

TRUMP Versus Partial Dual in Sprint

TRUMP



Partial dual

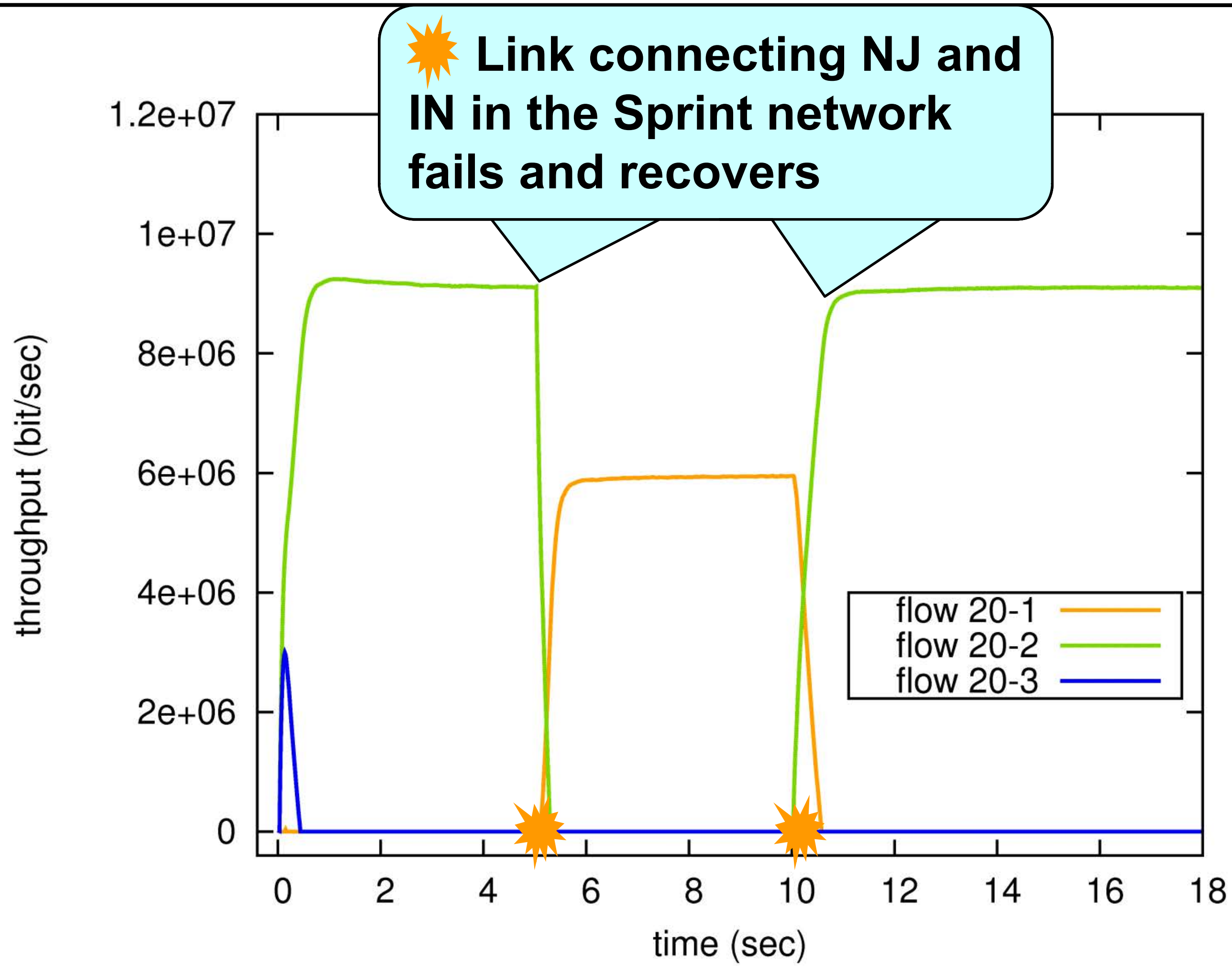


Aggregate throughput stabilizes for a range of w's

No stepsize allows satisfactory convergence

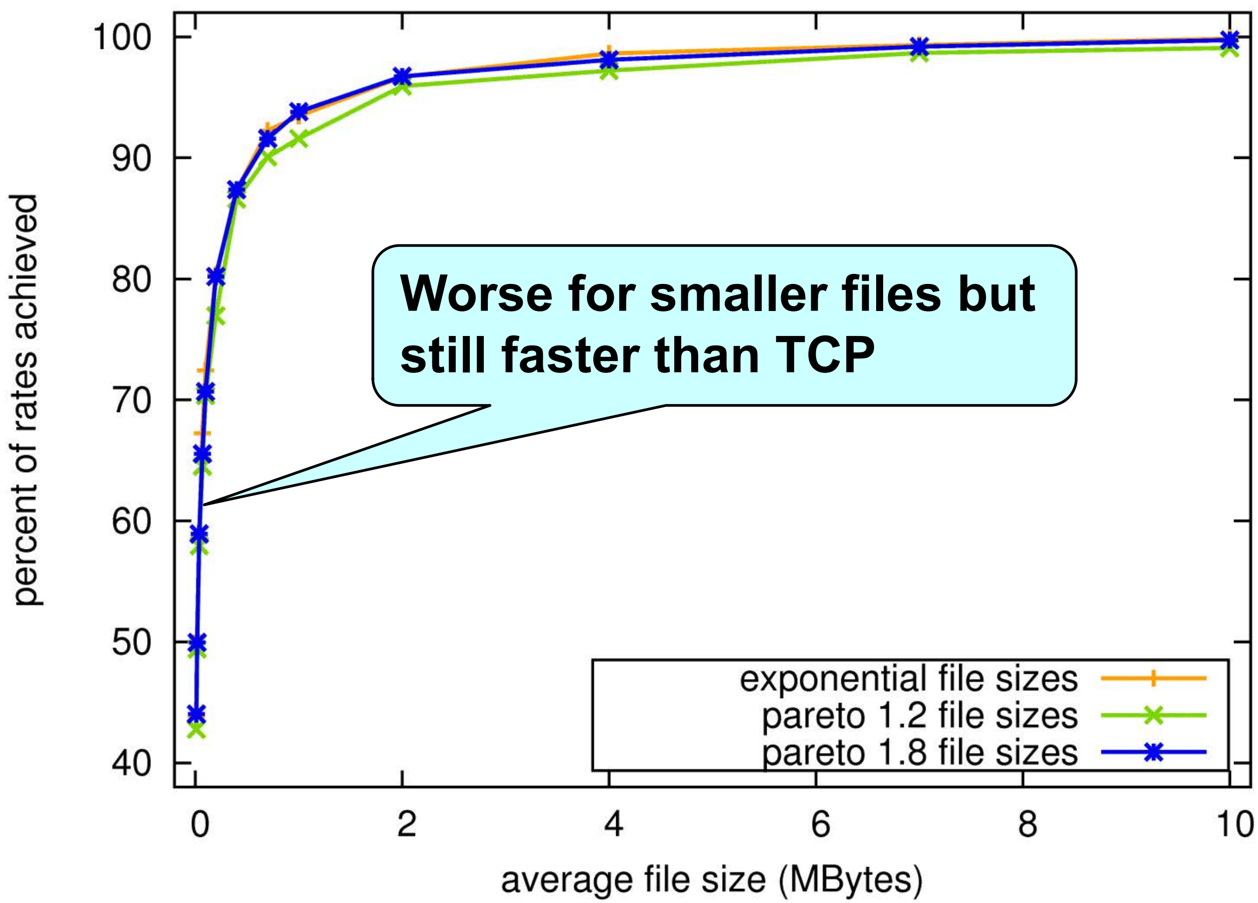
TRUMP trumps partial dual for $w \leq 1/3$

TRUMP Link Dynamics



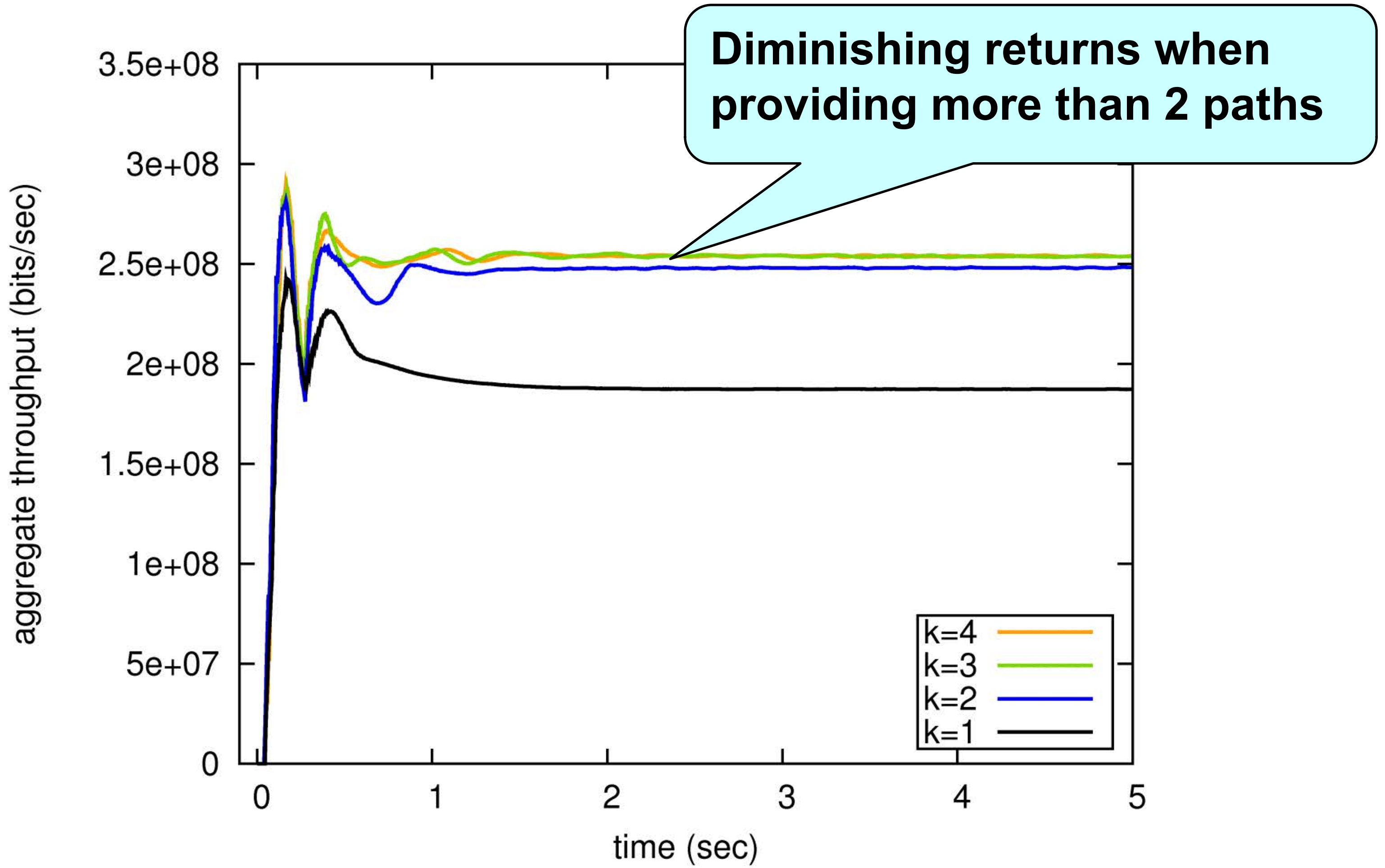
TRUMP reacts quickly to link dynamics

TRUMP Versus File Size



TRUMP's performance is independent of variance

TRUMP: A Few Paths Suffice



Sources benefit the most when they learn a few paths

Summary of TRUMP Properties

Property	TRUMP
Tuning Parameters	One easy to tune parameter Only needs to be tuned for small *w*
Robustness to link dynamics	Reacts quickly to link failures and recoveries
Robustness to flow dynamics	Independent of variance of file sizes, more efficient for larger files
General	Trumps other algorithms

Division of Functionality

	Today	TRUMP
Operators	Tune link weights Set penalty function	Set-up multipath Tune *w* & stepsize
Sources	Adapt source rates	Adapt path rates
Routers	Shortest path routing	Compute prices

- Sources: end hosts or edge routers?
- Feedback: implicit or explicit?

Mathematics leaves open architecture questions

The Next Steps

- So far the utility function maximizes utility of throughput sensitive traffic
- However, not all traffic throughput sensitive:

The Next Steps: Virtualization

- Need to support multiple types of applications
  - Throughput-sensitive: file transfers
  - Delay-sensitive: VoIP and gaming
- Questions
  - What should the utility for each application look like?
  - How to share network resources dynamically?

Conclusions

- Traffic engineering
  - Started with multiple decompositions
  - Designed TRUMP: new traffic-management protocol
- What to do next?
  - Support of different traffic classes

Thank You!

Related Work

- Advancements in optimization theory
  - Protocol reverse engineering ***(Kelly98, Low03)***
  - Design of new protocols ***(Low06)***
  - Multiple decompositions ***(Chiang06)***
- Traffic management protocols consider congestion control or traffic engineering
  - Congestion control alone (FAST TCP, RCP, XCP, etc.)
  - Use of multiple paths without adjusting source rates (MATE, REPLEX, etc.)