Capturing Complexity in Networked Systems Design: The Case for Improved Metrics

Sylvia Ratnasamy
U.C. Berkeley

Motivation

Our community values “simple”, “clean” system designs

*““The most important lesson we learned is the value of **simple** designs…”*
– Google Bigtable, OSDI ’06

*“The **complexity** of the architectural design…”* – Sprint, IEEE Network 2000

*“A design for multicast that is **simple**…”* – Perlman. Simple Multicast, IETF Draft.

*“The advantage of Chord is that it is **less complicated**…”* – Chord, Sigcomm’01

“***Complex routing algorithms** may have difficulty scaling… simple floods are sufficient”* -- TinyOS, NSDI’04

Motivation

Our community values “simple”, “clean” system designs

- But lacks metrics that rigorously capture this aesthetic
- Best practice: metrics borrowed from the theory community
  - *e.g.*, total_messages, total_state
- Problem: at times incongruent with our notion of “simplicity”
  - *e.g.*, flooding: inefficient but simple
  - *e.g.*, dijkstra’s: global state, but simple
  - *e.g.*, leaderID *vs.* neighborID

The NetComplex Work

- Can we quantify design “simplicity”?
  - more rigorously compare-and-contrast design options
  - align design goals of algorithms, systems communities

- First stab: complexity metrics for the algorithmic component of a networked system

- Note
  - just one aspect to system complexity
  - intent: complement, not replace, existing metrics

Outline

- Sketch of a solution
- Sample results
- Limitations and future directions

Approach

Builds on two observations:

- much of system design centers around <u>state</u>
  - what state is required?
  - how is it constructed/maintained?
  - how is it used?
- what distinguishes wide-area state:
  - derived from remote nodes → dependencies are distributed
  - relayed via intermediate nodes → more dependencies

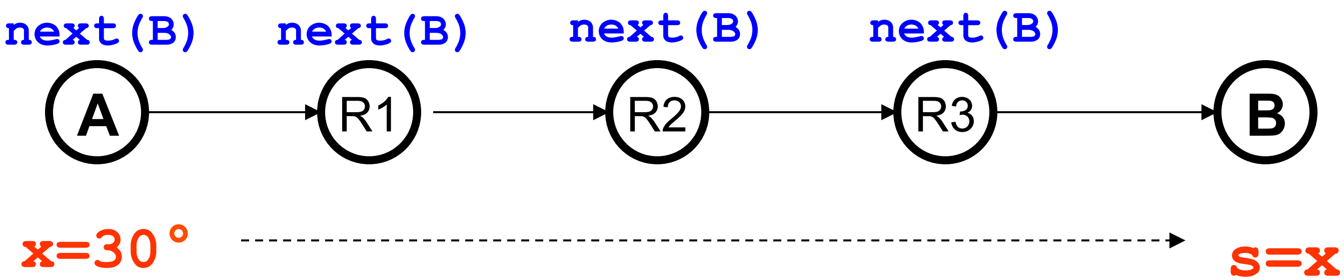
Current metrics mostly treat all state as equal

Proposal

For a given piece of state *s*, measure the ensemble of distributed dependencies that yield *s*

- First cut: measure → counting
  - akin to existing metrics: #msgs, #state
  - amenable to evaluation by simple examination, simulation

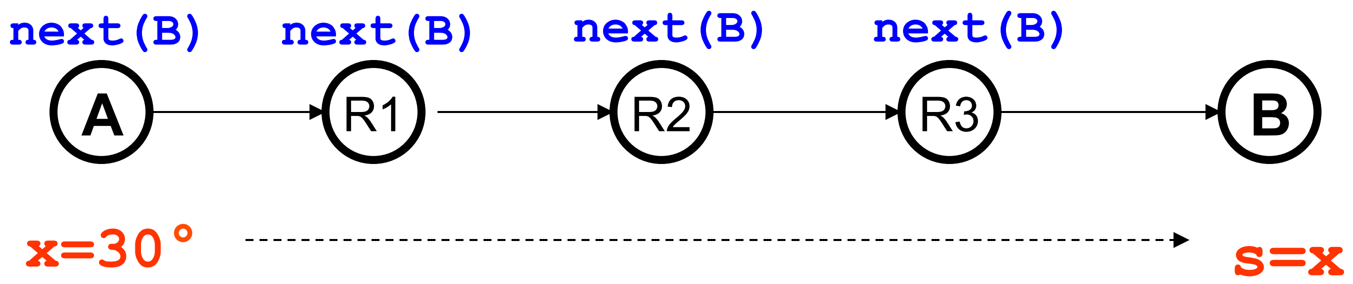
(Obs. 1) Types of Dependencies



s is value dependent on x,
s is transport dependent on next(B) state at A, R1, R2, R3.

v_s = #pieces of state on which s is value dependent
$t_{s \leftarrow x}$ = #pieces of state relied on to transport x to s

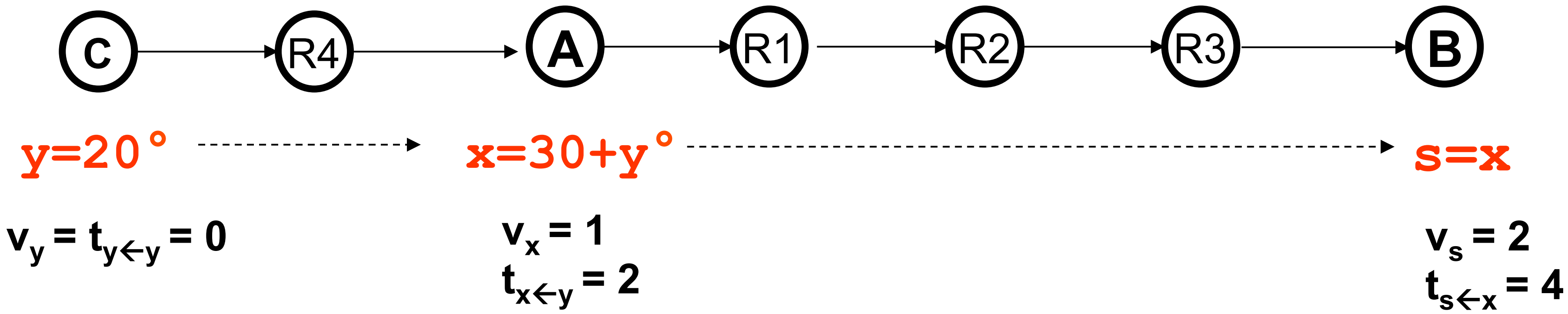
(Obs. 1) Types of Dependencies



$v_x = t_{x \leftarrow x} = 0$

$v_s = 1$; $t_{s \leftarrow x} = 4$

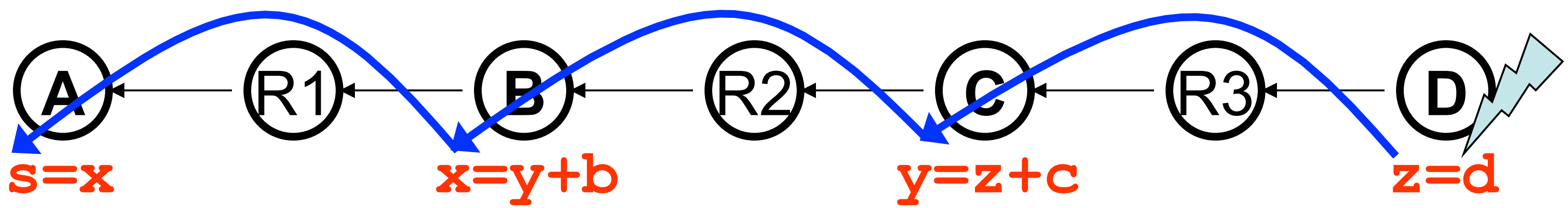
(Obs. 2) Value Dependencies Accumulate



Dependencies → Complexity?

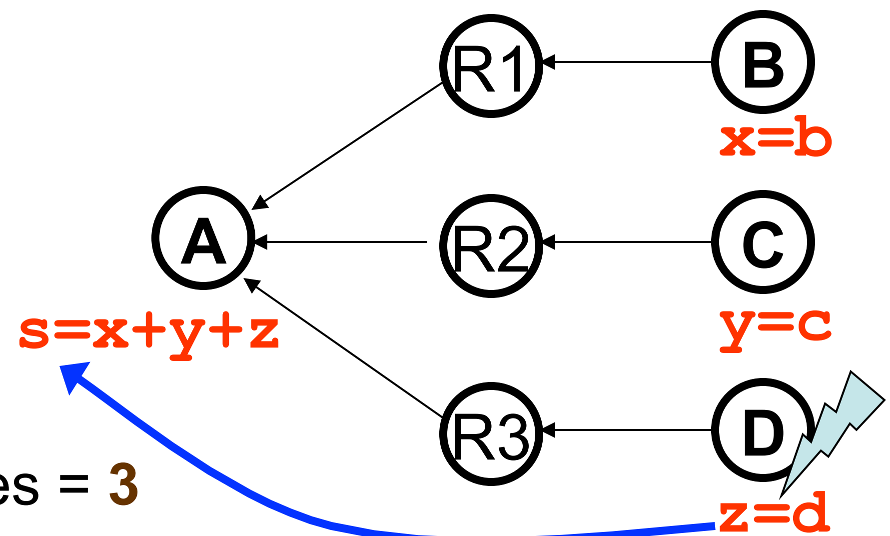
How do the number of value and transport dependencies for a piece of state *s*, translate to the complexity of *s* ?

Summation isn’t good enough



value dependencies = 3

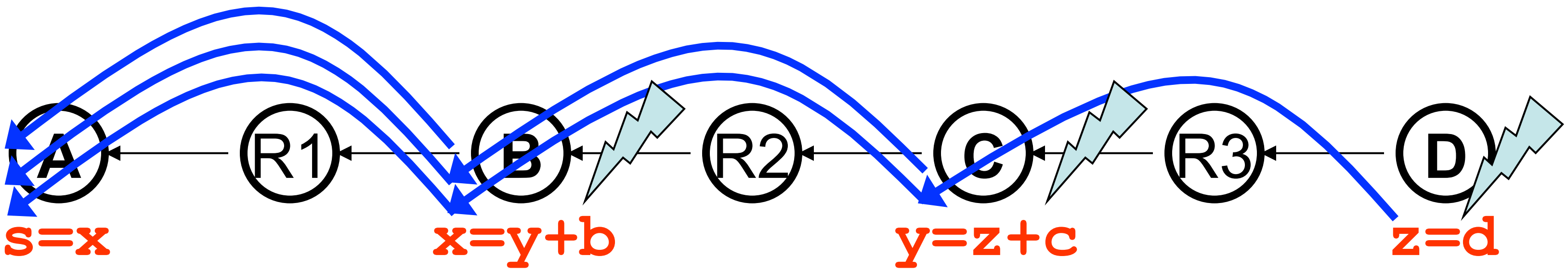
transport dependencies = 6

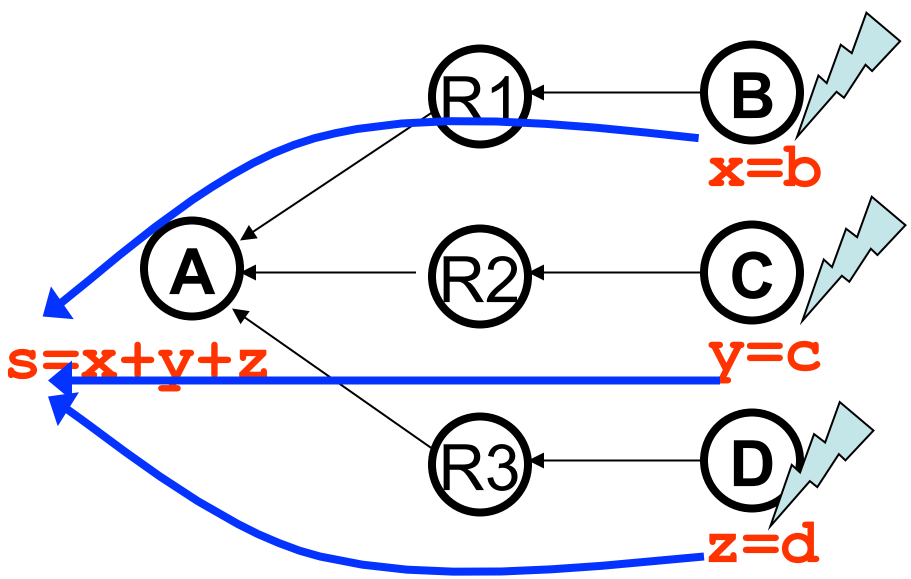


value dependencies = 3

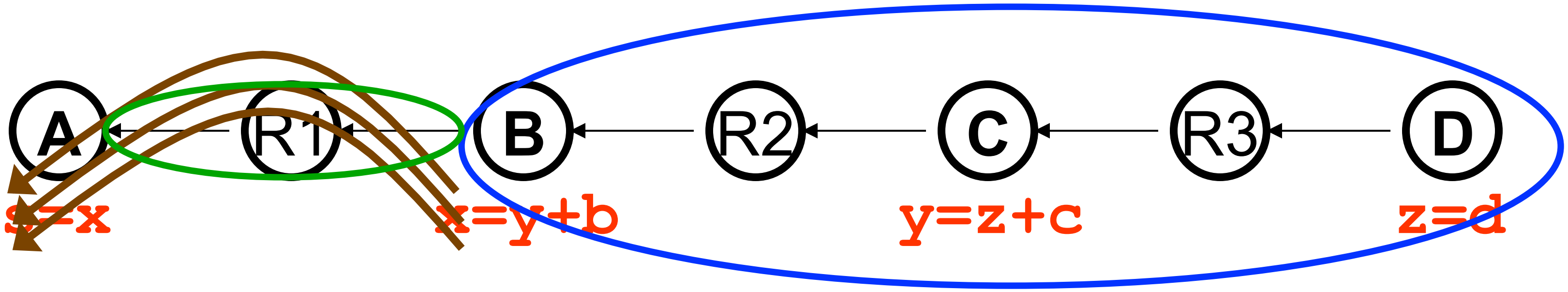
transport dependencies = 6

Must capture the propagation structure of dependencies





Final Metric



<u>Complexity of s:</u> $c_s = u_{s \leftarrow x} + t_{s \leftarrow x} + c_x$

- $u_{s \leftarrow x}$ = number of state changes relayed from x to s
- $t_{s \leftarrow x}$ = complexity of transport states for relaying from x to s
- c_x = complexity of state x

Extensions

1. Complexity of state *s* that is derived from multiple inputs $x_1, x_2, \ldots x_m$
   – distinguish between s derived from ALL, ANY or k-of-m

2. Complexity of a function is derived from the complexity of the state it acts on
   – e.g., “route(msg)” acts on forwarding state at every hop

Recap

- Compute the complexity of each piece of state s
  - if s is local state, $c_s = 0$
  - else identify the inputs $x_1, x_2, \ldots x_m$ that s is derived from and calculate c_s from $c_{x1}, c_{x2} \ldots c_{xm}$ and $t_{s \leftarrow x1} \ldots t_{s \leftarrow xm}$

- Compute the complexity of a function from the complexity of the state it acts on

- System complexity
  - sum of the complexity of its functions and/or
  - sum of the complexity of its states

Outline

- Sketch of a solution
- Sample results
- Limitations and future directions

Evaluation: Routing

(n: number of nodes, f: node degree, d: network diameter = *log* n)

Complex → Simple

Routing scheme	Complexity	State	Message
Compact (AG+04)	$O(nd^2)$	$O(\sqrt{n})$	$O(n\sqrt{n})$
Routing on Flat Labels (data-centric)	$O(d^2 log^2 n)$	$O(log\ n)$	$O(n\, log^2 n)$
Distance Vector	$O(d^3)$	$O(n)$	$O(n^2)$
Link State	$O(d^2)$	$O(nf)$	$O(n^3)$
Centralized (a la SDN)	$O(d^2)$	$O(nf)$; $O(n)$	$O(n^3)$
Centralized + Source Routing	$O(d)$	$O(nf)$; $O(n)$	$O(n^3)$

Evaluation: classical distributed systems

(n: number of nodes, k: quorum size)

Category	**System**	**Complexity**
Shared read/ write variable	Quorum	$O(k^2)$
	Read one/write all available	O(1)

Stronger consistency has higher complexity

	Two phase commit	$O(n^2)$

Stronger fault tolerance has higher complexity

propagation	Gossip	$O(\textit{log}\ n)$

Efficiency need not be congruent with complexity

consistency	TTL-based	ε

Soft-state is viewed as simpler than hard-state

Open Questions

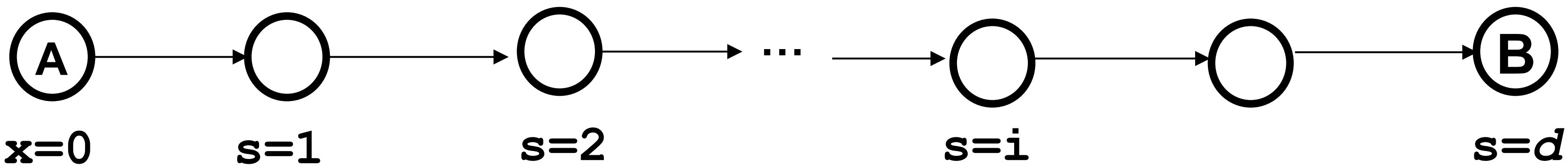
- Extending netcomplex to capture:
  - correctness/quality
  - robustness
  - correlated inputs
- Beyond algorithmic design complexity
  - configuration complexity
  - software complexity
  - complexity in the service model

Summary

- Design simplicity: valued, but poorly measured
- Our question: is design complexity measurable?
- Conjecture: capturing dependencies in state is key (w.r.t. algorithmic component of a system)
- Open: metrics that capture other aspects to system complexity (configuration, code, service model)

Thanks!
Questions?

complexity of operations

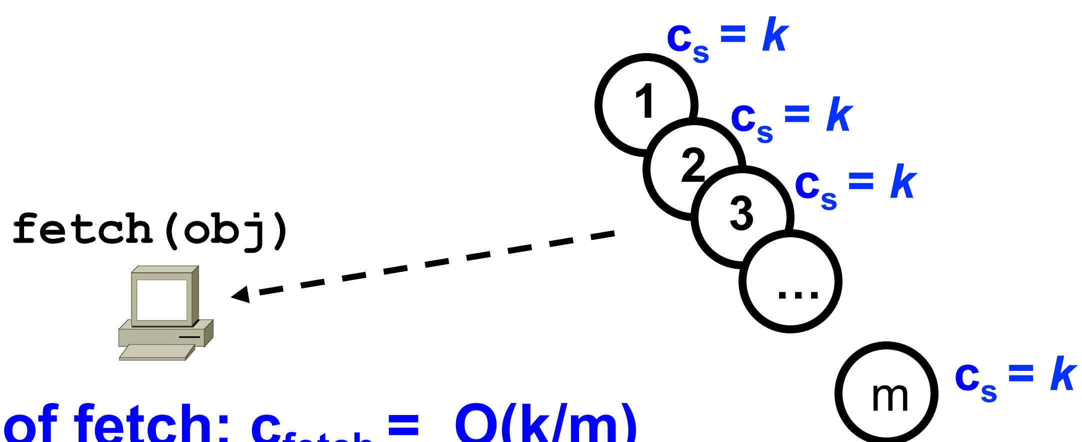


$c_s = 0$ **1** **...** $O((d-1)^2)$ $O(d^2)$

complexity of forwarding operation: $c_{fwd} = O(d^3)$

- an operation acts on different pieces of state
- each piece of state has a complexity
- compute complexity of an operation by (appropriately) combining complexity of state it acts on

complexity of operations



complexity of fetch: $c_{fetch} = O(k/m)$

- an operation acts on different pieces of state
- each piece of state has a complexity
- compute complexity of an operation by (appropriately) combining complexity of state it acts on