A Type-disciplined Approach to Scalable, Practical Composition of Networked Services

Abraham I. Matta
Visiting Scientist in Mobile Networking Systems Department
amatta@bbn.com

Azer Bestavros, Assaf J. Kfoury
Computer Science Department
Boston University

Adam D. Bradley
Web Services Technologies
amazon.com

BBN Technologies, Cambridge
January 20, 2006

Supported in part by grants from the National Science Foundation (NSF):
ANI-9986397, ANI-0095988,
ANI-0205294, EIA-0202067
Talk Overview

- Agenda
- The Big Picture
- TRAFFIC (Informal Syntax and Semantics)
- TRAFFIC Type Spaces (from 30K feet)
- Future Work
My Agenda...
My Agenda is **NOT**:

- To give a 30-minute refresher course in formal type theory and inference (upon which most of the formalism depends)

- To give a 30-minute crash course in the Network Calculus (upon which we mostly demonstrate our approach here)
My Agenda IS:

- To note the difficulties of analyzing large, complicated networks using most existing approaches to compositional analysis
- To propose a type-theory-inspired framework within which to explore ways of scaling up approaches to compositional analysis
Compositional Analysis

- It’s good to know your router doesn’t crash when it’s disconnected and idle.

- It’s better to know it doesn’t crash when connected with another router.
Compositional Analysis

- **COMPOSITION:**
  X interacts with Y

- **ANALYSIS:**
  Determining properties of a system by applying some formal method(s)

- **COMPOSITIONAL ANALYSIS:**
  Determining properties of X and Y taken together as a system by applying some formal method(s)
Compositional Analysis

- A few potent examples:
  - Networked Sense + Control
    - What happens when TCP is modulated by other rate/drop control schemes?
  - QoS
    - Can a stream bridge two nets w/ similar QoS goals but different mechanisms? Three?
  - Incremental protocol upgrades
    - Regression hell!
  - Routing
    - Does AS1’s BGP policy compose with AS2’s?
    - Does AS1+AS2 compose w/ “the world”?
Composing Models

- Does compositional analysis *scale* vs. system size?
  - Representation size?
  - Representation legibility?
  - Computational tractability?

- Queuing theory?  Scheduling theory?
- Interconnected Finite State Machines?
Composing Models

\[
\begin{align*}
[A] \bullet [B] & \Rightarrow [A \otimes B] \\
[A \otimes B] \bullet [C \otimes D] & \Rightarrow [A \otimes B \otimes C \otimes D] \\
[A \otimes B \otimes C \otimes D] \bullet [E \otimes F \otimes G \otimes H] & \Rightarrow [A \otimes B \otimes C \otimes D \otimes E \otimes F \otimes G \otimes H]
\end{align*}
\]

... for an Internet-scale application ??? ...
Composing Models vs. Composing Components

- To scale up compositional analysis, compositions should produce components

- \([A \otimes B]\) doesn’t look like one component

- Our not-so-outrageous idea:
  - Internal models are the wrong approach to scaling up analysis for non-trivial systems
Types

\[ \text{fix } \lambda^T \sigma \]
Preserving internals $\rightarrow$ NP-hard

- Once you’ve analyzed a fragment, discard everything you know about its internals
- Instead, describe invariants of its interface with other parts of the program

Exchange some expressive precision for highly scalable analysis
Types for Compositional Analysis
(a.k.a. interfaces, models, definitions, specifications, constraints, invariants, etc…)

- Existing analysis techniques establish local invariants of components
- Express those invariants as properties of *interfaces* between components, not of the components themselves

  - “What are the boundaries of the behavior I can expect to see interacting with you?”
  - “What are the boundaries on my behavior for you to act correctly?”
TRAFFIC

Typed Representation and Analysis of Flows For Interoperability Checks
TRAFFIC, Formally (partially...)

\[
\begin{align*}
x, y, z &\in \text{FlowVar} \quad \text{flow variable} \\
A, B, C &\in \text{LocalFlow} \quad \text{local flow} \\
\mathcal{A}, \mathcal{B}, \mathcal{C} &\in \text{GlobalFlow} ::= A \mid x \\
&\quad \mid A; B \quad \text{sequential flow} \\
&\quad \mid A || B \quad \text{parallel flow} \\
&\quad \mid \text{let } x = A \text{ in } B \quad \text{let-binding}
\end{align*}
\]

\[
\begin{align*}
\{ t_1 < t_2 \} &\subseteq \Delta \\
\Delta \vdash t_1 < t_2 &\quad \Delta \vdash \tau < : \tau \\
\Delta \vdash \tau_1 < : \tau_2 &\quad \Delta \vdash \tau_2 < : \tau_3 \\
\Delta \vdash \tau_1 < : \tau_3 &\quad \Delta \vdash \tau < : \tau' \\
\Delta \vdash \tau < : \tau' &\quad \Delta \vdash \rho < : \rho' \\
\Delta \vdash \tau \rho < : \tau' \rho' &\quad \Delta \vdash \rho_1 < : \rho_1, \Delta \vdash \rho_2 < : \rho_2' \\
\Delta \vdash \rho_1 \rho_2 < : \rho_1 \rho_2' &\quad \Delta \vdash \sigma_1 < : \sigma_1' \Delta \vdash \sigma_2 < : \sigma_2 \\
\Delta \vdash \left[ \begin{array}{cc}
\rho_1 & \rho_2 \\
\sigma_1 & \sigma_2
\end{array} \right] &\quad < : \left[ \begin{array}{cc}
\rho_1' & \rho_2' \\
\sigma_1' & \sigma_2'
\end{array} \right]
\end{align*}
\]

\[
\begin{align*}
\Gamma, \Delta \vdash A : T &\quad \Gamma, \Delta \vdash B : T' \\
\Gamma, \Delta \vdash A || B : T \cdot T' \\
\Gamma, \Delta \vdash A : \left[ \begin{array}{cc}
\rho_1 & \rho_2 \\
\sigma_1 & \sigma_2
\end{array} \right], \Gamma, \Delta \vdash B : \left[ \begin{array}{cc}
\rho_3 & \rho_4 \\
\sigma_3 & \sigma_4
\end{array} \right], \\
\Delta \vdash \rho_2 < : \rho_3, \Delta \vdash \sigma_3 < : \sigma_2 \\
\Gamma, \Delta \vdash A ; B : \left[ \begin{array}{cc}
\rho_1 & \rho_4 \\
\sigma_1 & \sigma_4
\end{array} \right]
\end{align*}
\]
TRAFFIC, Informally

- “flows”: Logical components
  - Simple or compound, logical or physical, with arbitrarily detailed models inside
  - “A”

- “flow variables”: Unknown pieces
  - Endpoint, delivery network, interceptor, intermediary service...
  - “X”
TRAFFIC, Informally

- “sockets”: Directed connectors
  - “Forward” and “backward” (but there’s nothing magical about the directions or number...)

- “composition”: Joining sockets
  - Parallel and serial (but there’s nothing magical about these structures...)

\[
A = \begin{bmatrix}
  f_{in} & f_{out} \\
  b_{out} & b_{in}
\end{bmatrix}
\]
TRAFFIC, Informally

- “socket types”: interface constraints
  - Data encoding, control type/frequency/delay, bandwidth envelope, max backlog/latency, traffic processes/shapes...
  - Guarantees, Statistical modes (e.g. 99.9\textsuperscript{th} %ile), Tail distributions (exp vs. heavy)...

![Diagram of network traffic flow](image)
TRAFFIC, Informally

- “socket types” (ct’d):
  - Key to a flexible and scalable type system: SUBTYPE relationships: “t₁ <: t₂” (aka “t₁ ⊆ t₂”)
  - Are all possible outputs described by \( f₂ \) also described by (acceptable to) \( f₃ \)?

- Controlled loss of precision
- M/M/1 <: M/G/1
- “d<5ms” <: “d<10ms”
- “d>2ms” <: “d>1ms”
TRAFFIC Type Inference

Composite flows are still flows
(i.e. compositions produce *simple* components)

\[
\begin{bmatrix}
\tau_1 & \tau_2 \\
\tau_5 & \tau_6 \\
\end{bmatrix} ; \begin{bmatrix}
\tau_3 & \tau_4 \\
\tau_7 & \tau_8 \\
\end{bmatrix} \rightarrow \begin{bmatrix}
\tau_1 & \tau_4 \\
\tau_5 & \tau_8 \\
\end{bmatrix}
\]
TRAFFIC Type Inference

- Limited computation in inference steps
  - First-order functions as output types based upon zero-order input types
  - Higher-order: Why not? (Future work)

\[
\begin{bmatrix}
\tau_1 & \tau_2 \\
\tau_5 & \tau_6 \\
\end{bmatrix} ; \begin{bmatrix}
\tau_3 & \tau_4 \\
\tau_7 & \tau_8 \\
\end{bmatrix} \rightarrow \begin{bmatrix}
\tau_1 & \tau_4(\tau_2) \\
\tau_5(\tau_7) & \tau_8 \\
\end{bmatrix}
\]

- Try to keep complexity quasi-linear (Maintain computability/tractability)
TRAFFIC Type Inference

- A ; (B || C) ; D ; (E || F || G)
  - Fully known network
  - Do the pieces “fit”? Are all requirements satisfied?

- A ; (x || C) ; D ; (E || y || G)
  - Partially known network
  - Do the known pieces “fit”? What is required of unknown pieces?
    - Work Forward: engineer to meet specs
    - ...or Backward: which extant pieces will fit?
TRAFFIC in Action

- TRAFFIC Type Space:
  - Set of possible types
  - Set of subtyping rules
  - Rules for assigning types to flows

- Our Example: Network Calculus
A Network Calculus Type Space

- Network Calculus [Boudec+Thiran ’04]
- Reason about bounds on capacity, demand, utilization, etc... with bounding functions over Δt’s
A Network Calculus Type Space

☐ This Type Space is not TRAFFIC!
  ■ It is a particular application of TRAFFIC

☐ We are not making the Network Calculus more powerful or expressive
  ■ Structured loss of precision is the goal!
  ■ A “by hand” analysis would produce more refined results

☐ Type expressions require some working familiarity w/ NC to be intelligible, but ultimately could be more user-friendly
Varieties of Types

- **Data Flow**
  - Bits seen in \([0,t)\)
  - Rate \((dR/dt)\) is secondary, need not be defined!

- **TRAFFIC:**
  - Define classes w/ upper, lower bounds
  - \(\exists R_1(t) \sqsubseteq_R \llbracket R(t) \rrbracket \sqsupseteq_R R_0(t)\)
  - \(R(t) \subseteq \exists R_1(t) \sqsubseteq_R \llbracket R_0(t) \rrbracket \sqsupseteq_R \)
  - \(\exists 5t \sqsubseteq_R < \exists 5t + 1 \sqsubseteq_R \)
Varieties of Types

- Arrival Curves $\underline{[A]}_\alpha$
  - Defined over marginal (vs. absolute) time
  - “Over any window $[s,s+t)$, how much data will arrive or be served?”
- Service Curves $R^*(t) \geq \min_{s \leq t} \left( R(s) + \beta(t - s) \right)$
- Shapers $\underline{[G]}_\sigma$
  - Same subtype structure: upper+lower bounds
- Losses $\underline{[Q(t)]}_L$ $\underline{[Q(t)]}_L$
  - Convolution using min and addition, rather than multiplication and addition
- Derivatives
  - Backlog, Delay

Same subtype structure: upper+lower bounds
## Additional Inferences

<table>
<thead>
<tr>
<th>Incoming</th>
<th>Outgoing</th>
</tr>
</thead>
<tbody>
<tr>
<td>$|F|_R$</td>
<td>$|F \otimes \beta|_R$</td>
</tr>
<tr>
<td>$|F'|_R$</td>
<td>$|G|_\sigma \cap |F \otimes G|_R$</td>
</tr>
<tr>
<td>$|F|_R$</td>
<td>$|Q|_L \cap |F - Q|_R$</td>
</tr>
<tr>
<td>$|F|_R$</td>
<td>$|Q|_L \cap |(F - Q) \otimes \beta|_R$</td>
</tr>
<tr>
<td>$|F|_R$</td>
<td>$|Q|<em>L \cap |G|</em>\sigma \cap |(F - Q) \otimes G|_R$</td>
</tr>
</tbody>
</table>
Example Application
Example Application

- Total video traffic type checks with input to shaper
- Subtyping is reflexive
Example Application

Assume fair scheduler, i.e. each stream served at minimum rate $t$.
Example Application

- Looser upper bounds ➔ precision / simplicity tradeoff
- Lower bound finally type checks with client’s input

\[
\begin{bmatrix}
\tau_{d,1} \\
\tau_{d,3}
\end{bmatrix} \cap \begin{bmatrix}
\|0.15t + 1\|_O \\
\|0.05t\|_O
\end{bmatrix}
\]

\[
\| (t + 5) - 0.05t \|_R \cap \| (2t) - 0.05t \|_R \text{ on each channel}
\]

\[
\| (t - 5) - (0.15t + 1) \|_R
\]
Future Directions

- **Type Spaces**
  - (snBench, itmBench)
    - Coding & Scheduling Theory
      - 3 fps <::: 1 fps (element can accept 3 fps by dropping 2 out of 3 frames every 1 sec.)
    - Control Theory
      - PI <::: P (element can accept higher quality control signal from PI controller)
  - BGP & VPN/IPSec Policies

![Diagram of framerate types]
Future Directions

- Programming Language Theory:
  - Multiple-choice let-binding
    
    \[
    \text{let } x \in \{A_1, \ldots, A_n\} \text{ in } B
    \]
  - Precise (but expensive) type checkers
  - Inference of interesting TRAFFIC types directly from program code
Thanks for Listening!

Supported in part by grants from the National Science Foundation (NSF):
ANI-9986397, ANI-0095988,
ANI-0205294, EIA-0202067
A Type-disciplined Approach to Scalable, Practical Composition of Networked Services

Abraham I. Matta
Visiting Scientist in Mobile Networking Systems Department
amatta@bbn.com

Azer Bestavros, Assaf J. Kfoury
Computer Science Department
Boston University

Adam D. Bradley
Web Services Technologies

Supported in part by grants from the National Science Foundation (NSF):
ANI-9986397, ANI-0095988, ANI-0205294, EIA-0202067

BBN Technologies, Cambridge
January 20, 2006