Type-Preserving Compilation for Formally Verified Software-Defined Delay Tolerant Networks

Jan-Paul Ramos-Dávila

Cornell University & NASA Langley Formal Methods Research Center

Software-Defined Delay Tolerant Networks

In space missions, reliable communication is critical yet exceptionally challenging. Traditional networks fail under extreme distances, orbital dynamics, and intermittent connectivity. Software-Defined Delay Tolerant Networks (SDDTNs) address these challenges through:

- Store-and-forward mechanisms for handling disconnections
- Dynamic routing based on predicted contact opportunities
- Centralized control for resource optimization

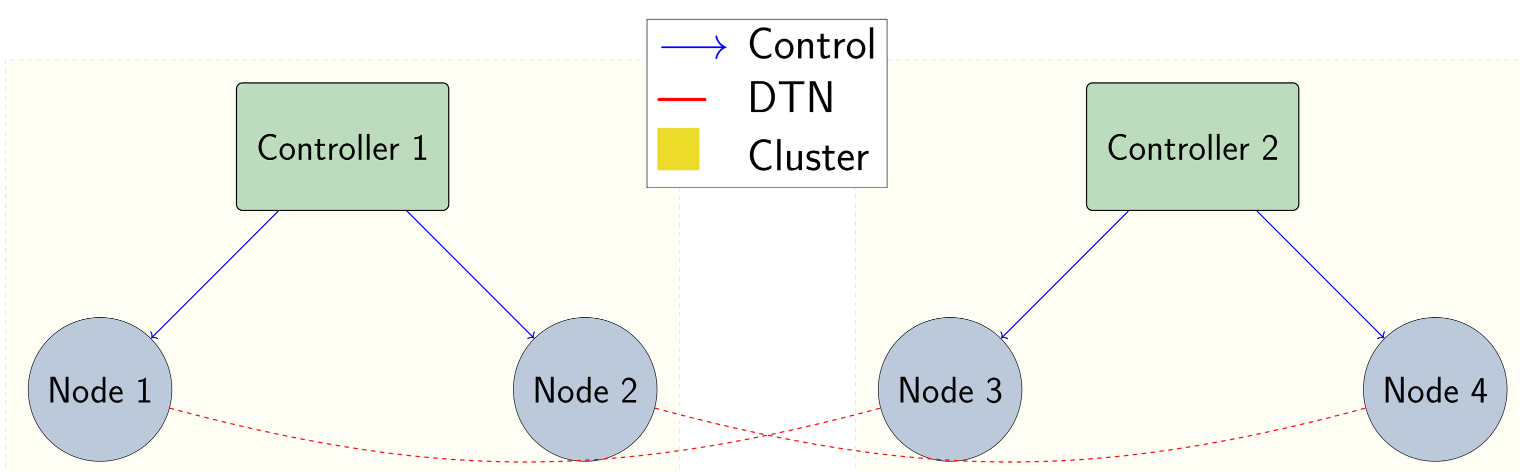

Figure 1. SDDTN Architecture with Cluster Controllers

Mathematical Foundations

Our approach begins with Network Calculus, which provides a mathematical framework for analyzing network behavior:

Core Structures

$(D, \oplus, \otimes, 0, 1)$ forms a dioid where:

- $\oplus$: Alternative behaviors (e.g., routing paths)
- $\otimes$: Sequential composition (e.g., chained operations)
- Distributivity: $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$

Service Curves

$$D(t) \geq (A * \beta)(t) \quad \forall t \geq 0$$

where:

- $D(t)$: Output data at time t
- $A(t)$: Input arrivals by time t
- β: Service guarantee curve
- $*$: Min-plus convolution

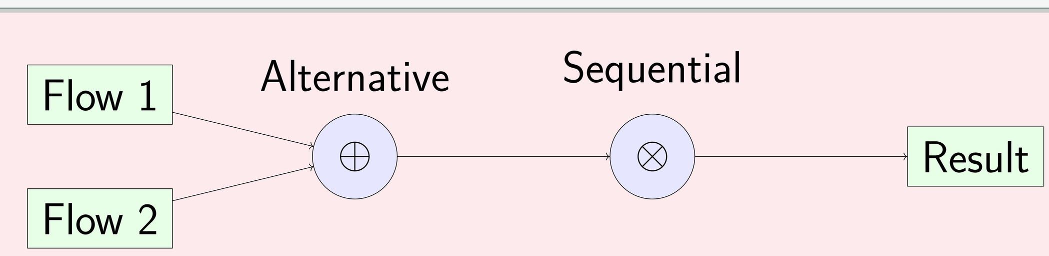

Figure 2. Network Calculus Operations

Coq Verification Framework

Interactive theorem proving in Coq enables mechanized verification of network properties:

Example: Header Validity

```
Theorem header_valid :
  ∀ (h: Header) (p: Packet),
  process_packet p h →
  valid_header h.
```

Formal Verification Guarantees

Property	P4	NetQIR
Type Safety	✗	✓
Header Validity	✗	✓
Flow Conservation	✗	✓
Protocol Compliance	✗	✓
Resource Usage	✗	✓

While P4 offers powerful abstractions for network programming, it lacks formal guarantees. NetQIR introduces static verification to catch errors before deployment.

Type System and Proofs

Core Typing Rules

$$\frac{\Gamma, x_i : \tau_i \vdash s : \sigma \qquad \text{flow-conserved}(s)}{\Gamma \vdash \texttt{action}\ a(\overline{x_i : \tau_i})\{s\} : \overline{\tau_i} \to \sigma}\ \textsc{T-Action}$$

$$\frac{\Gamma \vdash e : \tau_k \qquad \forall i.\ \Gamma \vdash s_i : \tau}{\Gamma \vdash \texttt{match}\ e\ \{\overline{p_i \Rightarrow s_i}\} : \tau}\ \textsc{T-Match}$$

Key Theorems

Type Safety:

$$\forall p, \Gamma.\ \Gamma \vdash p : \tau \implies \exists \sigma', \mathsf{pkt}'.\ \langle p, \sigma, \mathsf{pkt}\rangle \to^* \langle \sigma', \mathsf{pkt}'\rangle$$

Flow Conservation:

$$\mathsf{FlowIn}(\sigma) = \mathsf{FlowOut}(\sigma') + \mathsf{Dropped}(\sigma')$$

From P4 to NetQIR: Example

P4 Implementation

```
action forward(macAddr_t dst, port_t port) {
  std_meta.egress_spec = port;
  hdr.eth.dst = dst;
  hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
}
```

This becomes a dependently-typed operation:

NetQIR Translation

$$\begin{aligned}&\texttt{Forward} : \textstyle\prod(\mathsf{dst} : \mathsf{MacAddr})(\mathsf{port} : \mathsf{Port}).\\ &\quad \{\mathsf{pkt} : \mathsf{Packet} \mid \mathsf{valid}(\mathsf{pkt.ipv4})\} \to\\ &\quad \{\mathsf{pkt}' : \mathsf{Packet} \mid\\ &\qquad \mathsf{pkt}'.\mathtt{ttl} = \mathsf{pkt}.\mathtt{ttl} - 1 \wedge\\ &\qquad \mathsf{valid}(\mathsf{pkt}'.\mathsf{ipv4})\}\end{aligned}$$

The type ensures:

- TTL updates preserve packet validity
- No unexpected packet drops
- Port assignments are type-safe

Mechanized Proof Example

```
Theorem ttl_decrement_valid :
  ∀ (pkt: Packet),
  valid_packet pkt →
  pkt.ttl > 0 →
  valid_packet (decrement_ttl pkt).
Proof.
  intros pkt Hvalid Httl.
  apply packet_validity_preservation; auto.
  apply ttl_positive; auto.
Qed.
```

Compilation Pipeline

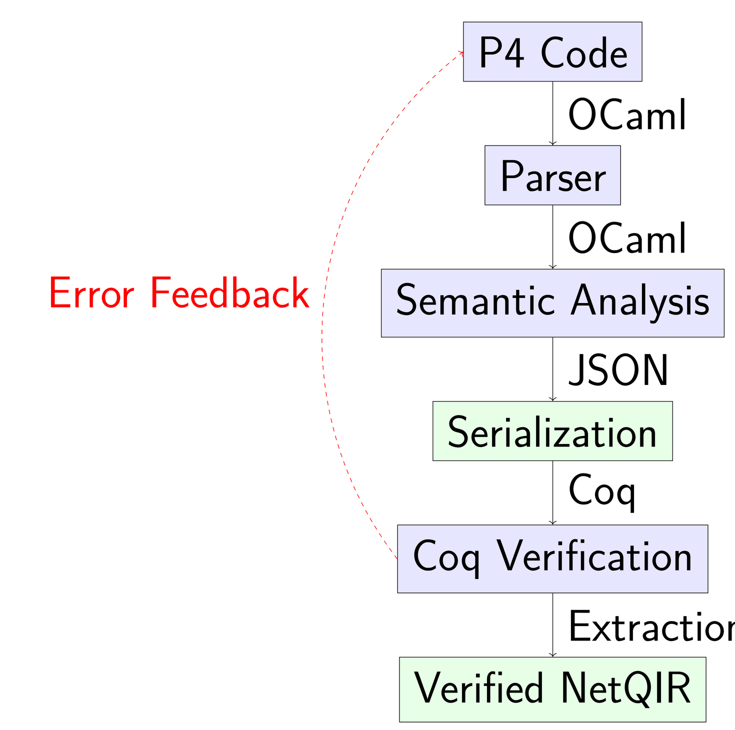

Figure 3. NetQIR Pipeline

- The P4 code is first parsed using an OCaml-based parser, which constructs an AST representing the program's structure.
- A semantic analyzer then processes the AST to check for correctness and annotate the tree with information.
- The annotated AST is then translated into NetQIR code. This translation maps P4 constructs to their NetQIR equivalents while preserving the program's semantics.
- The NetQIR code is serialized into a JSON representation. The JSON format serves as an intermediary that is both machine-readable and suitable for input into the Coq proof assistant.
- The JSON representation is then parsed within Coq to reconstruct the NetQIR constructs using Coq's data types. Any type errors or inconsistencies detected during this phase are reported back, providing feedback for correction.

Match-Action Pipeline Integration

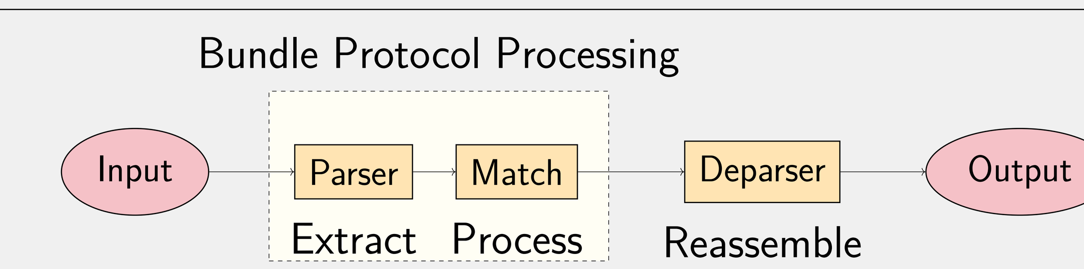

Figure 4. MAP Architecture with Bundle Protocol Integration

Verified Properties

The Match-Action Pipeline guarantees:

$\forall$ packet p, state σ :

- Header validity: $\mathsf{valid}(p) \to \mathsf{valid}(\mathsf{process}(p))$
- Flow conservation: $\mathsf{flow}(\sigma) = \mathsf{flow}(\mathsf{next}(\sigma))$
- Protocol compliance: $\mathsf{bp\_valid}(p) \to \mathsf{bp\_valid}(\mathsf{process}(p))$

Network Flow Properties

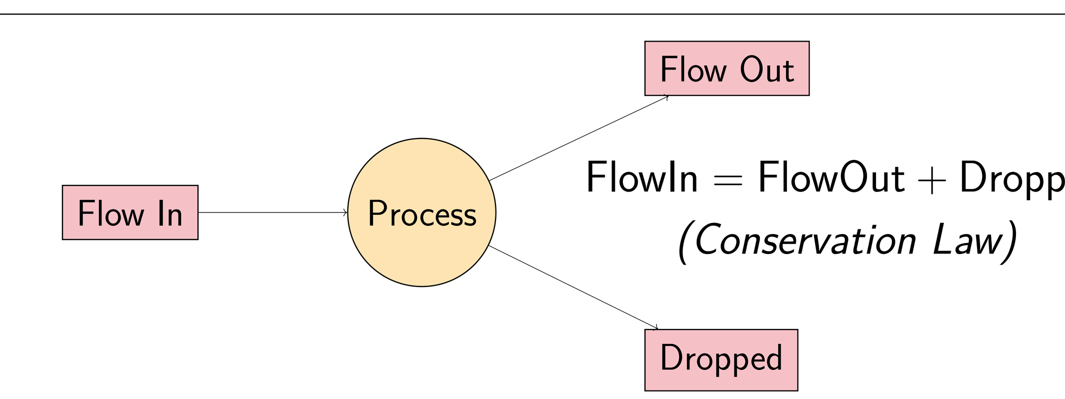

Figure 5. Flow Conservation in NetQIR

Future Directions

Our framework opens several promising research directions:

- Extended protocol verification for complete BP support
- Real-time verification of dynamic network properties
- Automated proof generation for common patterns

Github: janpaulpl

jvr34@cornell.edu