Theorem

Formally Verified Software Defined Delay-Tolerant Networks

Jan-Paul Ramos-Dávila

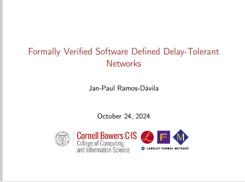
October 24, 2024

Cornell Bowers C·IS
College of Computing
and Information Science

LANGLEY FORMAL METHODS

2024-10-24

Formally Verified Software Defined Delay-Tolerant Networks


Formally Verified Software Defined Delay-Tolerant Networks

Jan-Paul Ramos-Dávila

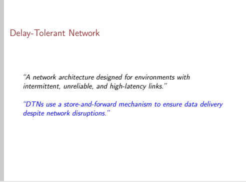
October 24, 2024

Networks
DTN & SDDTN
SDDTN Architecture
Verification
Properties
Algebra
Automated Theorem Proving
Programming Languages and Usability
Why?
NetQIR
Type Preserving Compilation

Delay-Tolerant Network

"A network architecture designed for environments with intermittent, unreliable, and high-latency links."

"DTNs use a store-and-forward mechanism to ensure data delivery despite network disruptions."

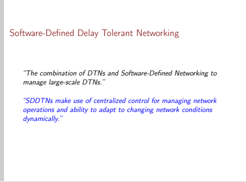


Applications: Useful in space missions, underdeveloped regions, and disaster-stricken areas.
Explanation of Functionality: What is a store-and-forward mechanism? Keep this mechanism in mind because it will come in handy later.
- Storing data happens when a DTN node receives a data packet. This is what we know as a "bundle". The node stores this bundle in its local storage.
- Waiting for an Opportunity happens when a node waits for a communication opportunity, such as a contact with another node or the availability of a network path. During this time, the bundle remains in storage, ensuring ti is not lost even if immediate transmission is impossible.
- Forwarding the Data happens when a connection becomes available, the node forwards the bundle to the next hop (another node closer to the destination). This process may repeat multiple times across different nodes.
- Finally, the deliver occurs when the bundle reaches the destination node and a complete path is formed.

Software-Defined Delay Tolerant Networking

"The combination of DTNs and Software-Defined Networking to manage large-scale DTNs."

"SDDTNs make use of centralized control for managing network operations and ability to adapt to changing network conditions dynamically."



Software-Defined Delay Tolerant Networking

"The combination of DTNs and Software-Defined Networking to manage large-scale DTNs."

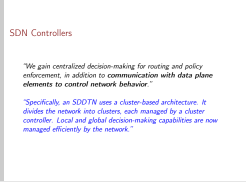
"SDDTNs make use of centralized control for managing network operations and ability to adapt to changing network conditions dynamically."

There are some basic challenges that DTNs suffer from, such as scalability issues in large networks and handling dynamic / unpredictable link conditions. SDDTNs take care of these problems, by making use of SDN Controllers.

SDN Controllers

"We gain centralized decision-making for routing and policy enforcement, in addition to ***communication with data plane elements to control network behavior****."*

"Specifically, an SDDTN uses a cluster-based architecture. It divides the network into clusters, each managed by a cluster controller. Local and global decision-making capabilities are now managed efficiently by the network."


SDN Controllers

"We gain centralized decision-making for routing and policy enforcement, in addition to ***communication with data plane elements to control network behavior****."*

"Specifically, an SDDTN uses a cluster-based architecture. It divides the network into clusters, each managed by a cluster controller. Local and global decision-making capabilities are now managed efficiently by the network."

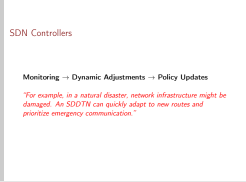
There are some basic challenges that DTNs suffer from, such as scalability issues in large networks and handling dynamic / unpredictable link conditions. SDDTNs take care of these problems, by making use of SDN Controllers.
1. Centralized Controller: The SDN controller acts as the "brain" of the network, making decisions about how data should be routed and handled. 2. Global view: The controller has a comprehensive view of the entire network, including the state of all nodes, available paths, and current network conditions. 3. Centralized management: It centrally manages and configures network policies, such as routing rules and priority handling. This management includes: a. Determining the best paths for data to take through the network. b. Allocating bandwidth and storage resources to optimize network performance. c. Enforcing security rules to protect data.

Moreover, these are ways of making sure our three basic steps from a DTN (Store, Wait, Deliver incrementally) are efficiently and safely produced.

SDN Controllers

Monitoring $\rightarrow$ Dynamic Adjustments $\rightarrow$ Policy Updates

"For example, in a natural disaster, network infrastructure might be damaged. An SDDTN can quickly adapt to new routes and prioritize emergency communication."


SDN Controllers

Monitoring $\rightarrow$ Dynamic Adjustments $\rightarrow$ Policy Updates

"For example, in a natural disaster, network infrastructure might be damaged. An SDDTN can quickly adapt to new routes and prioritize emergency communication."

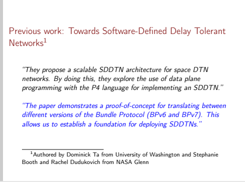
The dynamic details of an SDDTN proceed in these three steps:
1. The SDN controller continuously monitors network conditions, such as link availability, node status, and traffic patterns. 2. Based on the monitored data, the controller can adjust the network configuration in real-time. This includes rerouting, prioritization of critical data, and load balancing. 3. Finally, the controller can update network policies dynamically to respond to changing requirements, such as during network expansion or an emergency situation.

Previous work: Towards Software-Defined Delay Tolerant Networks[1]

"They propose a scalable SDDTN architecture for space DTN networks. By doing this, they explore the use of data plane programming with the P4 language for implementing an SDDTN."

"The paper demonstrates a proof-of-concept for translating between different versions of the Bundle Protocol (BPv6 and BPv7). This allows us to establish a foundation for deploying SDDTNs."

[1]Authored by Dominick Ta from University of Washington and Stephanie Booth and Rachel Dudukovich from NASA Glenn


Previous work: Towards Software-Defined Delay Tolerant Networks[1]

"They propose a scalable SDDTN architecture for space DTN networks. By doing this, they explore the use of data plane programming with the P4 language for implementing an SDDTN."

"The paper demonstrates a proof-of-concept for translating between different versions of the Bundle Protocol (BPv6 and BPv7). This allows us to establish a foundation for deploying SDDTNs."

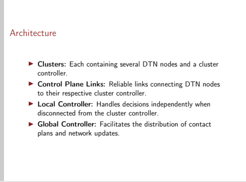
[1]Authored by Dominick Ta from University of Washington and Stephanie Booth and Rachel Dudukovich from NASA Glenn

The P4 programming language is a language for specifying how data plane devices process packets. Remember that the data plane just refers to the part of the network infrastructure that is responsible for the actual forwarding and handling of data packets or bundles as they traverse.

The main motivating factor behind this translation for BPv6 and BPv7 is that while newer deployments are adopting BPv7 due to its enhanced features and improvements, legacy systems still make use of BPv6. This also means that communication can continue uninterrupted.

Architecture

- **Clusters:** Each containing several DTN nodes and a cluster controller.
- **Control Plane Links:** Reliable links connecting DTN nodes to their respective cluster controller.
- **Local Controller:** Handles decisions independently when disconnected from the cluster controller.
- **Global Controller:** Facilitates the distribution of contact plans and network updates.

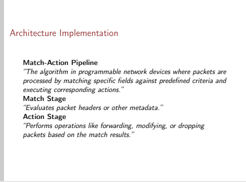

Architecture

- **Clusters:** Each containing several DTN nodes and a cluster controller.
- **Control Plane Links:** Reliable links connecting DTN nodes to their respective cluster controller.
- **Local Controller:** Handles decisions independently when disconnected from the cluster controller.
- **Global Controller:** Facilitates the distribution of contact plans and network updates.

This architecture ensures a reliable communication strategy for volatile environments.

Architecture Implementation

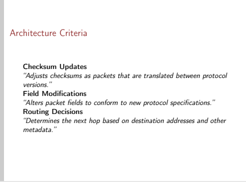
Match-Action Pipeline
"The algorithm in programmable network devices where packets are processed by matching specific fields against predefined criteria and executing corresponding actions."
Match Stage
"Evaluates packet headers or other metadata."
Action Stage
"Performs operations like forwarding, modifying, or dropping packets based on the match results."



So what are the predefined criteria setup in this paper?

Architecture Criteria

Checksum Updates
"Adjusts checksums as packets that are translated between protocol versions."
Field Modifications
"Alters packet fields to conform to new protocol specifications."
Routing Decisions
"Determines the next hop based on destination addresses and other metadata."

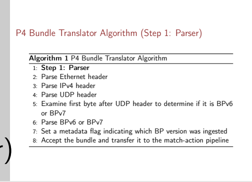


Here, the Routing Decisions aspect of the algorithm informs us of the decisions behind the "Action Stage". These are guarantees that are provided by a working implementation of the algorithm.

P4 Bundle Translator Algorithm (Step 1: Parser)

Algorithm 1 P4 Bundle Translator Algorithm

```
1: Step 1: Parser
2: Parse Ethernet header
3: Parse IPv4 header
4: Parse UDP header
5: Examine first byte after UDP header to determine if it is BPv6
   or BPv7
6: Parse BPv6 or BPv7
7: Set a metadata flag indicating which BP version was ingested
8: Accept the bundle and transfer it to the match-action pipeline
```

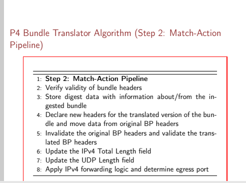

P4 Bundle Translator Algorithm (Step 1: Parser)
Algorithm 1 P4 Bundle Translator Algorithm
1: Step 1: Parser
2: Parse Ethernet header
3: Parse IPv4 header
4: Parse UDP header
5: Examine first byte after UDP header to determine if it is BPv6 or BPv7
6: Parse BPv6 or BPv7
7: Set a metadata flag indicating which BP version was ingested
8: Accept the bundle and transfer it to the match-action pipeline

Step 1 Parser: Focuses on parsing the information from the Ethernet layer and identifying the protocol version. The matching occurs at this stage, by marking the bundle with metadata
Step 2 Match-Action Pipeline: This is the stage we will be focusing on for the rest of this talk.
We start by checking the integrity and correctness of the bundle headers. Then we record information about the bundle for monitoring and tracking. We create headers for the translated version (BPv6 to BPv7 or vice versa) and transfer data. We mark the original BP headers as invalid after translation. We then recalculate the UDP Length field accordingly. We use IPv4 forwarding logic to decide the outgoing port.
Step 3 Deparser: This step transmits collected digest data to the control plane for analysis and logging and forwards the processed bundles through the designated egress port.

Now why do we specifically care about the Match-Action pipeline?

P4 Bundle Translator Algorithm (Step 2: Match-Action Pipeline)

```
1: Step 2: Match-Action Pipeline
2: Verify validity of bundle headers
3: Store digest data with information about/from the in-
   gested bundle
4: Declare new headers for the translated version of the bun-
   dle and move data from original BP headers
5: Invalidate the original BP headers and validate the trans-
   lated BP headers
6: Update the IPv4 Total Length field
7: Update the UDP Length field
8: Apply IPv4 forwarding logic and determine egress port
```

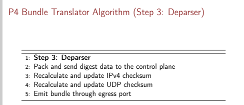

P4 Bundle Translator Algorithm (Step 2: Match-Action Pipeline)
1: Step 2: Match-Action Pipeline
2: Verify validity of bundle headers
3: Store digest data with information about/from the ingested bundle
4: Declare new headers for the translated version of the bundle and move data from original BP headers
5: Invalidate the original BP headers and validate the translated BP headers
6: Update the IPv4 Total Length field
7: Update the UDP Length field
8: Apply IPv4 forwarding logic and determine egress port

Step 1 Parser: Focuses on parsing the information from the Ethernet layer and identifying the protocol version. The matching occurs at this stage, by marking the bundle with metadata
Step 2 Match-Action Pipeline: This is the stage we will be focusing on for the rest of this talk.
We start by checking the integrity and correctness of the bundle headers. Then we record information about the bundle for monitoring and tracking. We create headers for the translated version (BPv6 to BPv7 or vice versa) and transfer data. We mark the original BP headers as invalid after translation. We then recalculate the UDP Length field accordingly. We use IPv4 forwarding logic to decide the outgoing port.
Step 3 Deparser: This step transmits collected digest data to the control plane for analysis and logging and forwards the processed bundles through the designated egress port.

Now why do we specifically care about the Match-Action pipeline?

P4 Bundle Translator Algorithm (Step 3: Deparser)

```
1: Step 3: Deparser
2: Pack and send digest data to the control plane
3: Recalculate and update IPv4 checksum
4: Recalculate and update UDP checksum
5: Emit bundle through egress port
```

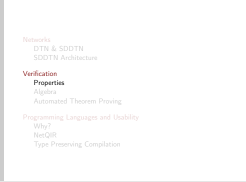

P4 Bundle Translator Algorithm (Step 3: Deparser)
1: Step 3: Deparser
2: Pack and send digest data to the control plane
3: Recalculate and update IPv4 checksum
4: Recalculate and update UDP checksum
5: Emit bundle through egress port

Step 1 Parser: Focuses on parsing the information from the Ethernet layer and identifying the protocol version. The matching occurs at this stage, by marking the bundle with metadata
Step 2 Match-Action Pipeline: This is the stage we will be focusing on for the rest of this talk.
We start by checking the integrity and correctness of the bundle headers. Then we record information about the bundle for monitoring and tracking. We create headers for the translated version (BPv6 to BPv7 or vice versa) and transfer data. We mark the original BP headers as invalid after translation. We then recalculate the UDP Length field accordingly. We use IPv4 forwarding logic to decide the outgoing port.
Step 3 Deparser: This step transmits collected digest data to the control plane for analysis and logging and forwards the processed bundles through the designated egress port.

Now why do we specifically care about the Match-Action pipeline?

2024-10-24

Formally Verified Software Defined Delay-Tolerant Networks
└─Verification
  └─Properties



Properties

Correctness Properties

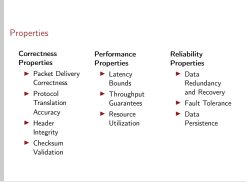
- Packet Delivery Correctness
- Protocol Translation Accuracy
- Header Integrity
- Checksum Validation

Performance Properties

- Latency Bounds
- Throughput Guarantees
- Resource Utilization

Reliability Properties

- Data Redundancy and Recovery
- Fault Tolerance
- Data Persistence



By proving these properties, you can ensure that the SDDTN network is reliable, secure, efficient, and scalable, while also meeting the necessary regulatory and compliance standards. The verification process provides confidence in the network's ability to function correctly under a wide range of conditions and use cases.

Properties

Security Properties

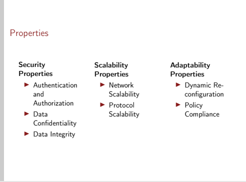
- Authentication and Authorization
- Data Confidentiality
- Data Integrity

Scalability Properties

- Network Scalability
- Protocol Scalability

Adaptability Properties

- Dynamic Re-configuration
- Policy Compliance



By proving these properties, you can ensure that the SDDTN network is reliable, secure, efficient, and scalable, while also meeting the necessary regulatory and compliance standards. The verification process provides confidence in the network's ability to function correctly under a wide range of conditions and use cases.

Properties

Interoperability Properties

- Cross-Protocol Compatibility
- Backwards Compatibility

Safety Properties
Compliance Properties
Quality of Service Properties
Temporal Properties
Concurrency Properties
Etc...

Properties

Interoperability Properties
- Cross-Protocol Compatibility
- Backwards Compatibility

Safety Properties
Compliance Properties
Quality of Service Properties
Temporal Properties
Concurrency Properties
Etc...

By proving these properties, you can ensure that the SDDTN network is reliable, secure, efficient, and scalable, while also meeting the necessary regulatory and compliance standards. The verification process provides confidence in the network's ability to function correctly under a wide range of conditions and use cases.

Properties

Correctness Properties

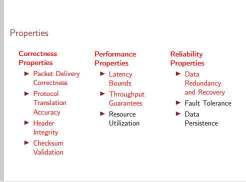
- Packet Delivery Correctness
- Protocol Translation Accuracy
- Header Integrity
- Checksum Validation

Performance Properties

- Latency Bounds
- Throughput Guarantees
- Resource Utilization

Reliability Properties

- Data Redundancy and Recovery
- Fault Tolerance
- Data Persistence


Properties

Correctness Properties
- Packet Delivery Correctness
- Protocol Translation Accuracy
- Header Integrity
- Checksum Validation

Performance Properties
- Latency Bounds
- Throughput Guarantees
- Resource Utilization

Reliability Properties
- Data Redundancy and Recovery
- Fault Tolerance
- Data Persistence

These are the properties we currently care about in respect to the Match-Action Pipeline. As you can see, it's nontrivial to choose the algorithm to prove these a lot of these properties safe in the SDDTN!

Properties

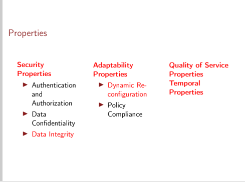
Security Properties

- Authentication and Authorization
- Data Confidentiality
- Data Integrity

Adaptability Properties

- Dynamic Re-configuration
- Policy Compliance

Quality of Service Properties
Temporal Properties


Properties

Security Properties
- Authentication and Authorization
- Data Confidentiality
- Data Integrity

Adaptability Properties
- Dynamic Re-configuration
- Policy Compliance

Quality of Service Properties
Temporal Properties

These are the properties we currently care about in respect to the Match-Action Pipeline. As you can see, it's nontrivial to choose the algorithm to prove these a lot of these properties safe in the SDDTN!

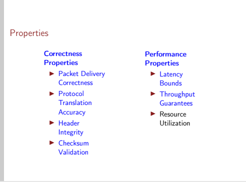
Properties

Correctness Properties

- Packet Delivery Correctness
- Protocol Translation Accuracy
- Header Integrity
- Checksum Validation

Performance Properties

- Latency Bounds
- Throughput Guarantees
- Resource Utilization



Properties

Correctness Properties
- Packet Delivery Correctness
- Protocol Translation Accuracy
- Header Integrity
- Checksum Validation

Performance Properties
- Latency Bounds
- Throughput Guarantees
- Resource Utilization

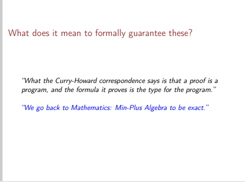
I'll go into detail for the properties that I was able to prove correct. The rest are of course a work in progress and ideally I've provided a framework to allow anyone to work on the more complicated problems...

1. Packet Delivery Correctness The Match-Action Pipeline determines the appropriate actions for each packet, such as forwarding, dropping, or modifying. This pipeline ensures that the packets are handled correctly according to the rules.
2. Protocol Translation Accuracy: The pipeline is responsible for translating between different protocol versions. It ensures that headers and pauloads are correctly translated without data loss or corruption.
3. Header Integrity: It involves checking and potentially modifying packet headers. The pipeline must maintain the integrity of these headers during processing.
4. Checksum Validation: As the packets are processed, the pipeline may need to verify or update checksums, such as the IPv4 and UDP checksums, to ensure data integrity during transit.

What does it mean to formally guarantee these?

"What the Curry-Howard correspondence says is that a proof is a program, and the formula it proves is the type for the program."

"We go back to Mathematics: Min-Plus Algebra to be exact."



This statement is very loaded and I will not be going into the exact details of it. But essentially, we have a framework in which we can encode this computational behavior mathematically. If this is the case, we are able to provide robust guarantees for the mathematical framework via formal proofs. This would lead us to also guarantee these same principles for the computational model we based our mathematical model on.

The mathematical representation for our Match-Action Pipeline is what we will call our Network Calculus. Ideally, a Network Calculus has more substance than modeling a single algorithm, but for the sake of simplicity, we can imagine that this model is a 1 to 1 abstraction from the algorithm we just saw.

2024-10-24

Formally Verified Software Defined Delay-Tolerant Networks
└─Verification
  └─Algebra

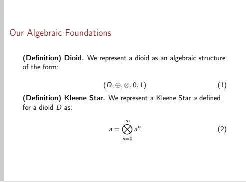
Our Algebraic Foundations

(Definition) Dioid. We represent a dioid as an algebraic structure of the form:

$$(D, \oplus, \otimes, 0, 1) \tag{1}$$

(Definition) Kleene Star. We represent a Kleene Star a defined for a dioid D as:

$$a = \bigotimes_{n=0}^{\infty} a^n \tag{2}$$


Our Algebraic Foundations

(Definition) Dioid. We represent a dioid as an algebraic structure of the form:

$$(D, \oplus, \otimes, 0, 1) \tag{1}$$

(Definition) Kleene Star. We represent a Kleene Star a defined for a dioid D as:

$$a = \bigotimes_{n=0}^{\infty} a^n \tag{2}$$

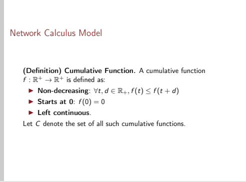
It is a well-known approach to use Tropical algebra (also referred to as the algebra of a min-plus dioid) to model a Network Calculus for analyzing performance bounds like delays and backlogs.
In here, a dioid's oplus is associative, commutative, and idempotent. The otimes is associative, distributes over oplus, and has an absorbing element 0. 1 is the neutral element for otimes.
For Kleene Star, we have that $a^0 = 1$ and $a^{n+1} = a \otimes a^n$.

Network Calculus Model

(Definition) Cumulative Function. A cumulative function $f : \mathbb{R}^+ \to \mathbb{R}^+$ is defined as:

- **Non-decreasing**: $\forall t, d \in \mathbb{R}_+, f(t) \leq f(t+d)$
- **Starts at 0**: $f(0) = 0$
- **Left continuous**.

Let C denote the set of all such cumulative functions.



Network Calculus Model

(Definition) Cumulative Function. A cumulative function $f : \mathbb{R}^+ \to \mathbb{R}^+$ is defined as:

- **Non-decreasing**: $\forall t, d \in \mathbb{R}_+, f(t) \leq f(t+d)$
- **Starts at 0**: $f(0) = 0$
- **Left continuous**.

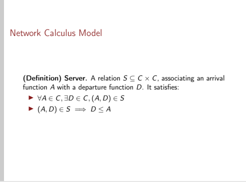
Let C denote the set of all such cumulative functions.

The cumulative function represents the total amount of data that has arrived or departed at a network node over time, providing a way to model data flow in terms of volume and time.

Network Calculus Model

(Definition) Server. A relation $S \subseteq C \times C$, associating an arrival function A with a departure function D. It satisfies:

- $\forall A \in C, \exists D \in C, (A, D) \in S$
- $(A, D) \in S \implies D \leq A$



Network Calculus Model

(Definition) Server. A relation $S \subseteq C \times C$, associating an arrival function A with a departure function D. It satisfies:

- $\forall A \in C, \exists D \in C, (A, D) \in S$
- $(A, D) \in S \implies D \leq A$

Servers are abstract models that describe how data is processed and forwarded from arrival to departure, effectively representing network elements like switches or routers.

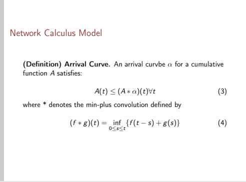
Network Calculus Model

(Definition) Arrival Curve. An arrival curvbe α for a cumulative function A satisfies:

$$A(t) \leq (A * \alpha)(t) \forall t \tag{3}$$

where * denotes the min-plus convolution defined by

$$(f * g)(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\} \tag{4}$$


Network Calculus Model

(Definition) Arrival Curve. An arrival curvbe α for a cumulative function A satisfies:

$$A(t) \leq (A * \alpha)(t) \forall t \tag{3}$$

where * denotes the min-plus convolution defined by

$$(f * g)(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\} \tag{4}$$

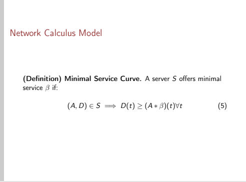
Arrival curves provide an upper-bound on the data flow into a system, describing the maximum amount of data that can arrive within any given time interval.

The min-plus convolution tells that for two functions, their combination determines the worst-case accumulation of delays and data processing in the network. It is doing this by taking the minimum sum of their values shifted over time.

Network Calculus Model

(Definition) Minimal Service Curve. A server S offers minimal service β if:

$$(A, D) \in S \implies D(t) \geq (A * \beta)(t) \forall t \tag{5}$$


Network Calculus Model

(Definition) Minimal Service Curve. A server S offers minimal service β if:

$$(A, D) \in S \implies D(t) \geq (A * \beta)(t) \forall t \tag{5}$$

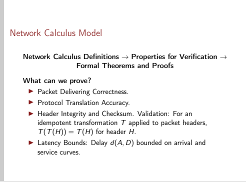
A minimal service curve defines the guaranteed amount of service (i.e., data processing capacity) that a server offers, ensuring a lower bound on the output rate and thus helping to predict delays and backlogs.

Network Calculus Model

Network Calculus Definitions $\rightarrow$ Properties for Verification $\rightarrow$ Formal Theorems and Proofs

What can we prove?

- Packet Delivering Correctness.
- Protocol Translation Accuracy.
- Header Integrity and Checksum. Validation: For an idempotent transformation T applied to packet headers, $T(T(H)) = T(H)$ for header H.
- Latency Bounds: Delay $d(A, D)$ bounded on arrival and service curves.


Network Calculus Model

Network Calculus Definitions $\rightarrow$ Properties for Verification $\rightarrow$ Formal Theorems and Proofs

What can we prove?
- Packet Delivering Correctness.
- Protocol Translation Accuracy.
- Header Integrity and Checksum. Validation: For an idempotent transformation T applied to packet headers, $T(T(H)) = T(H)$ for header H.
- Latency Bounds: Delay $d(A, D)$ bounded on arrival and service curves.

Using these definitions, we can formalize and verify several properties related to the Match-Action pipeline.

Packet Delivering: The Match-Action pipeline must ensure that packets are forwarded to the correct destination. This can be modeled by showing that the server S processing the packets guarantees that for any arrival curve α, the departure curve D satisfies the required bounds.

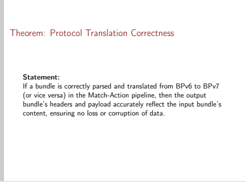
Protocol Translation: Verify that the transformation from BPv6 to BPv7 (or vice versa) adheres to a well-defined bijection, preserving the semantics of the protocols and ensuring no loss of information.

Header Integrity: This involves verifying that the changes made to the packer headers and the recalculated checksums match expected values, ensuring data integrity. This can be modeled as an idempotent transformation T applied to packet headers, where $T(T(H) = T(H)$ for header H.

Latency Bounds: Define and prove bounds on the time taken for a packet to traverse the Match-Action pipeline. The delay $d(A, D)$ can be bounded

Theorem: Protocol Translation Correctness

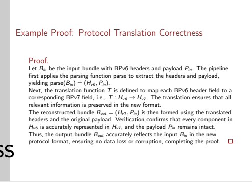
Statement:
If a bundle is correctly parsed and translated from BPv6 to BPv7 (or vice versa) in the Match-Action pipeline, then the output bundle's headers and payload accurately reflect the input bundle's content, ensuring no loss or corruption of data.


Theorem: Protocol Translation Correctness

Statement:
If a bundle is correctly parsed and translated from BPv6 to BPv7 (or vice versa) in the Match-Action pipeline, then the output bundle's headers and payload accurately reflect the input bundle's content, ensuring no loss or corruption of data.

Example Proof: Protocol Translation Correctness

Proof.
Let B_{in} be the input bundle with BPv6 headers and payload P_{in}. The pipeline first applies the parsing function parse to extract the headers and payload, yielding $\text{parse}(B_{in}) = (H_{v6}, P_{in})$.
Next, the translation function T is defined to map each BPv6 header field to a corresponding BPv7 field, i.e., $T : H_{v6} \to H_{v7}$. The translation ensures that all relevant information is preserved in the new format.
The reconstructed bundle $B_{out} = (H_{v7}, P_{in})$ is then formed using the translated headers and the original payload. Verification confirms that every component in H_{v6} is accurately represented in H_{v7}, and the payload P_{in} remains intact.
Thus, the output bundle B_{out} accurately reflects the input B_{in} in the new protocol format, ensuring no data loss or corruption, completing the proof. □



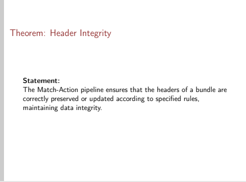
This proof shows that translating protocol headers from BPv6 to BPv7 (or vice versa) correctly preserves the original data's integrity.

Theorem: Header Integrity

Statement:
The Match-Action pipeline ensures that the headers of a bundle are correctly preserved or updated according to specified rules, maintaining data integrity.

2024-10-24

Formally Verified Software Defined Delay-Tolerant Networks
└─Verification
  └─Algebra
    └─Theorem: Header Integrity

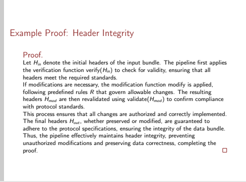

Theorem: Header Integrity

Statement:
The Match-Action pipeline ensures that the headers of a bundle are correctly preserved or updated according to specified rules, maintaining data integrity.

Example Proof: Header Integrity

Proof.

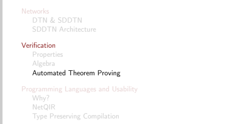
Let H_{in} denote the initial headers of the input bundle. The pipeline first applies the verification function verify(H_{in}) to check for validity, ensuring that all headers meet the required standards.
If modifications are necessary, the modification function modify is applied, following predefined rules R that govern allowable changes. The resulting headers H_{mod} are then revalidated using validate(H_{mod}) to confirm compliance with protocol standards.
This process ensures that all changes are authorized and correctly implemented. The final headers H_{out}, whether preserved or modified, are guaranteed to adhere to the protocol specifications, ensuring the integrity of the data bundle.
Thus, the pipeline effectively maintains header integrity, preventing unauthorized modifications and preserving data correctness, completing the proof. $\square$


Example Proof: Header Integrity
Proof.
Let H_{in} denote the initial headers of the input bundle. The pipeline first applies the verification function verify(H_{in}) to check for validity, ensuring that all headers meet the required standards.
If modifications are necessary, the modification function modify is applied, following predefined rules R that govern allowable changes. The resulting headers H_{mod} are then revalidated using validate(H_{mod}) to confirm compliance with protocol standards.
This process ensures that all changes are authorized and correctly implemented. The final headers H_{out}, whether preserved or modified, are guaranteed to adhere to the protocol specifications, ensuring the integrity of the data bundle.
Thus, the pipeline effectively maintains header integrity, preventing unauthorized modifications and preserving data correctness, completing the proof. $\square$

This proof establishes that headers in a data bundle are handled correctly, ensuring that no unauthorized changes occur, thus maintaining integrity.

2024-10-24

Formally Verified Software Defined Delay-Tolerant Networks
└─Verification
  └─Automated Theorem Proving


Networks
DTN & SDDTN
SDDTN Architecture
Verification
Properties
Algebra
Automated Theorem Proving
Programming Languages and Usability
Why?
NetQIR
Type Preserving Compilation

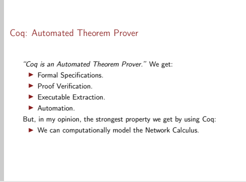
Coq: Automated Theorem Prover

"Coq is an Automated Theorem Prover." We get:

- Formal Specifications.
- Proof Verification.
- Executable Extraction.
- Automation.

But, in my opinion, the strongest property we get by using Coq:

- We can computationally model the Network Calculus.


Coq: Automated Theorem Prover

"Coq is an Automated Theorem Prover." We get:

- Formal Specifications.
- Proof Verification.
- Executable Extraction.
- Automation.

But, in my opinion, the strongest property we get by using Coq:

- We can computationally model the Network Calculus.

Coq provides us with a formal language to write mathematical definitions, executable algorithms, and theorems, together with an environment for semi-interactive development of machine-checked proofs.
Here, we can clearly see that by using Coq, we're able to define the precise properties that the network should uphold. Of course we have the guarantee of our proofs being correct because of the mechanization, but more importantly, we can use Coq as a way to computationally model the Network Calculus, the mathematical representation of our Match-Action Pipeline.

This leads us to the latter part of my project, usability!

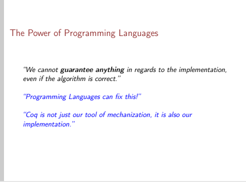
Networks
DTN & SDDTN
SDDTN Architecture
Verification
Properties
Algebra
Automated Theorem Proving
Programming Languages and Usability
Why?
NetQIR
Type Preserving Compilation

The Power of Programming Languages

"We cannot ***guarantee anything*** *in regards to the implementation, even if the algorithm is correct."*

"Programming Languages can fix this!"

"Coq is not just our tool of mechanization, it is also our implementation."



As we have seen thus far, Formal Verification is very useful for providing guarantees when analyzing our algorithms. However, everything we have discussed thus far does not talk about the actual implementation provided in the Glenn paper we have been referencing. More specifically, their Match-Action Pipeline algorithm is not targeted at all by our verification methods. While we can make use of our model to provide guarantees to an abstraction of such, we cannot provide guarantees to the implementation of the algorithm. In fact, we currently can't guarantee any implementation correct!

Using Programming Languages, we can further encode our representation of the Network Calculus in Coq, which is a programming language, to essentially "verify" an implementation of the algorithm.

2024-10-24

Formally Verified Software Defined Delay-Tolerant Networks
└Programming Languages and Usability
└NetQIR

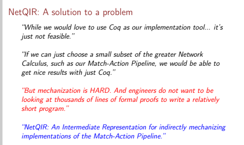
NetQIR: A solution to a problem

"While we would love to use Coq as our implementation tool... it's just not feasible."

"If we can just choose a small subset of the greater Network Calculus, such as our Match-Action Pipeline, we would be able to get nice results with just Coq."

"But mechanization is HARD. And engineers do not want to be looking at thousands of lines of formal proofs to write a relatively short program."

"NetQIR: An Intermediate Representation for indirectly mechanizing implementations of the Match-Action Pipeline."



However, we don't want to actually be using Coq as our implementation tool, mostly because Coq is not Turing-Complete, meaning we cannot arbitrarily encode any program in it. Moreso, it would be incredibly difficult to make a working framework of libraries to support all programs we would like to work with in the Network Calculus. Because of this, we choose to only target the Match-Action pipeline!

Coq is mainly seen in this context as a spec to abide by the engineers who will actually implement the code. But we propose instead a way to indirectly mechanize the code engineers will write (not in Coq).

NetQIR Pipeline Overview

- Type System design in Coq that checks for NetQIR programs
- Parser from P4 to NetQIR and Operational Semantics analyzer in OCaml.
- Serialization from the OCaml generated AST to serve as an input for the Coq type checker.

NetQIR Pipeline Overview

- Type System design in Coq that checks for NetQIR programs
- Parser from P4 to NetQIR and Operational Semantics analyzer in OCaml.
- Serialization from the OCaml generated AST to serve as an input for the Coq type checker.

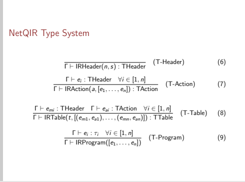
NetQIR Type System

$$\frac{}{\Gamma \vdash \mathrm{IRHeader}(n, s) : \mathrm{THeader}} \quad \text{(T-Header)} \tag{6}$$

$$\frac{\Gamma \vdash e_i : \mathrm{THeader} \quad \forall i \in [1, n]}{\Gamma \vdash \mathrm{IRAction}(a, [e_1, \ldots, e_n]) : \mathrm{TAction}} \quad \text{(T-Action)} \tag{7}$$

$$\frac{\Gamma \vdash e_{mi} : \mathrm{THeader} \quad \Gamma \vdash e_{ai} : \mathrm{TAction} \quad \forall i \in [1, n]}{\Gamma \vdash \mathrm{IRTable}(t, [(e_{m1}, e_{a1}), \ldots, (e_{mn}, e_{an})]) : \mathrm{TTable}} \quad \text{(T-Table)} \tag{8}$$

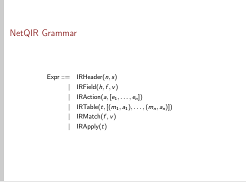
$$\frac{\Gamma \vdash e_i : \tau_i \quad \forall i \in [1, n]}{\Gamma \vdash \mathrm{IRProgram}([e_1, \ldots, e_n])} \quad \text{(T-Program)} \tag{9}$$



The type system defines the rules that specify how expressions and components of the intermediate representation (IR) are classified. It ensures that the components of a network program are used consistently and correctly. For instance, it guarantees that headers are treated as headers, actions as actions, and so on.

NetQIR Grammar

$$\begin{aligned}\text{Expr} ::= \;& \text{IRHeader}(n, s)\\ \mid\;& \text{IRField}(h, f, v)\\ \mid\;& \text{IRAction}(a, [e_1, \ldots, e_n])\\ \mid\;& \text{IRTable}(t, [(m_1, a_1), \ldots, (m_n, a_n)])\\ \mid\;& \text{IRMatch}(f, v)\\ \mid\;& \text{IRApply}(t)\end{aligned}$$



The grammar defines the syntactical structure of the IR expressions. It specifies how different elements like headers, fields, actions, and tables can be composed to form valid programs. The grammar includes definitions for headers, match conditions, actions, and table applications.

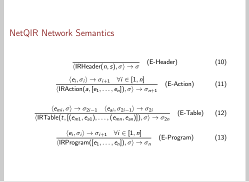
NetQIR Network Semantics

$$\frac{}{\langle \text{IRHeader}(n, s), \sigma \rangle \rightarrow \sigma} \quad \text{(E-Header)} \tag{10}$$

$$\frac{\langle e_i, \sigma_i \rangle \rightarrow \sigma_{i+1} \quad \forall i \in [1, n]}{\langle \text{IRAction}(a, [e_1, \ldots, e_n]), \sigma \rangle \rightarrow \sigma_{n+1}} \quad \text{(E-Action)} \tag{11}$$

$$\frac{\langle e_{mi}, \sigma \rangle \rightarrow \sigma_{2i-1} \quad \langle e_{ai}, \sigma_{2i-1} \rangle \rightarrow \sigma_{2i}}{\langle \text{IRTable}(t, [(e_{m1}, e_{a1}), \ldots, (e_{mn}, e_{an})]), \sigma \rangle \rightarrow \sigma_{2n}} \quad \text{(E-Table)} \tag{12}$$

$$\frac{\langle e_i, \sigma_i \rangle \rightarrow \sigma_{i+1} \quad \forall i \in [1, n]}{\langle \text{IRProgram}([e_1, \ldots, e_n]), \sigma \rangle \rightarrow \sigma_n} \quad \text{(E-Program)} \tag{13}$$



NetQIR Network Semantics

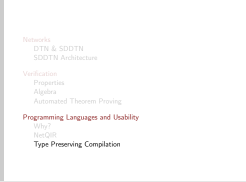
$$\frac{}{\langle \text{IRHeader}(n, s), \sigma\rangle \to \sigma} \quad \text{(E-Header)} \tag{10}$$

$$\frac{\langle e_i, \sigma_i\rangle \to \sigma_{i+1} \quad \forall i \in [1, n]}{\langle \text{IRAction}(a, [e_1, \dots, e_n]), \sigma\rangle \to \sigma_{n+1}} \quad \text{(E-Action)} \tag{11}$$

$$\frac{\langle e_{mi}, \sigma\rangle \to \sigma_{2i-1} \quad \langle e_{ai}, \sigma_{2i-1}\rangle \to \sigma_{2i}}{\langle \text{IRTable}(t, [(e_{m1}, e_{a1}), \dots, (e_{mn}, e_{an})]), \sigma\rangle \to \sigma_{2n}} \quad \text{(E-Table)} \tag{12}$$

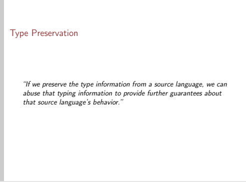
$$\frac{\langle e_i, \sigma_i\rangle \to \sigma_{i+1} \quad \forall i \in [1, n]}{\langle \text{IRProgram}([e_1, \dots, e_n]), \sigma\rangle \to \sigma_n} \quad \text{(E-Program)} \tag{13}$$

The network semantics describe how the execution of IR expressions affects the network state. It defines the transition rules that specify how headers, actions, and tables are evaluated, and how they modify the state represented by the symbol $\sigma. Thesemanticsensurethateachoperationhasawell - definedeffectonthenetwork' sdata.$



Type Preservation

"If we preserve the type information from a source language, we can abuse that typing information to provide further guarantees about that source language's behavior."


Type Preservation

"If we preserve the type information from a source language, we can abuse that typing information to provide further guarantees about that source language's behavior."

Type preserving compilation is a compilation process where the source program's type information is maintained throughout the compilation process, ensuring that the compiled program adheres to the same type constraints as the original program. This process involves translating a well-typed source program into a target program that is also well-typed, according to the type system of the target language.

Type preserving compilation is crucial for ensuring that the semantic properties guaranteed by the type system of the source language are preserved in the compiled code. This means that if the source program is type-safe, the compiled program will also be type-safe, preventing certain classes of runtime errors.

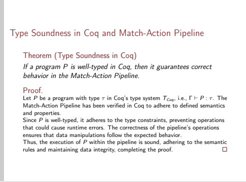
Type Soundness in Coq and Match-Action Pipeline

Theorem (Type Soundness in Coq)

If a program P is well-typed in Coq, then it guarantees correct behavior in the Match-Action Pipeline.

Proof.

Let P be a program with type τ in Coq's type system T_{Coq}, i.e., $\Gamma \vdash P : \tau$. The Match-Action Pipeline has been verified in Coq to adhere to defined semantics and properties.
Since P is well-typed, it adheres to the type constraints, preventing operations that could cause runtime errors. The correctness of the pipeline's operations ensures that data manipulations follow the expected behavior.
Thus, the execution of P within the pipeline is sound, adhering to the semantic rules and maintaining data integrity, completing the proof. □



Type Soundness in Coq and Match-Action Pipeline
Theorem (Type Soundness in Coq)
If a program P is well-typed in Coq, then it guarantees correct behavior in the Match-Action Pipeline.
Proof.
Let P be a program with type τ in Coq's type system T_{Coq}, i.e., $\Gamma \vdash P : \tau$. The Match-Action Pipeline has been verified in Coq to adhere to defined semantics and properties.
Since P is well-typed, it adheres to the type constraints, preventing operations that could cause runtime errors. The correctness of the pipeline's operations ensures that data manipulations follow the expected behavior.
Thus, the execution of P within the pipeline is sound, adhering to the semantic rules and maintaining data integrity, completing the proof. □

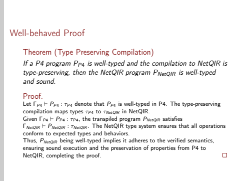
Context: A program P in Coq's type system $T_{Coq} iswell - typedif \Gamma \vdash P : \tau$. Goal: Show that well-typed programs ensure correct behavior in the Match-Action Pipeline. Proof Outline: Coq enforces strict type safety, preventing runtime type errors. The Match-Action Pipeline's operations are verified within Coq to adhere to defined semantics. Thus, if P is well-typed, it will operate correctly within the pipeline, ensuring sound execution.

Well-behaved Proof

Theorem (Type Preserving Compilation)

If a P4 program P_{P4} is well-typed and the compilation to NetQIR is type-preserving, then the NetQIR program P_{NetQIR} is well-typed and sound.

Proof.
Let $\Gamma_{P4} \vdash P_{P4} : \tau_{P4}$ denote that P_{P4} is well-typed in P4. The type-preserving compilation maps types τ_{P4} to τ_{NetQIR} in NetQIR.
Given $\Gamma_{P4} \vdash P_{P4} : \tau_{P4}$, the transpiled program P_{NetQIR} satisfies $\Gamma_{NetQIR} \vdash P_{NetQIR} : \tau_{NetQIR}$. The NetQIR type system ensures that all operations conform to expected types and behaviors.
Thus, P_{NetQIR} being well-typed implies it adheres to the verified semantics, ensuring sound execution and the preservation of properties from P4 to NetQIR, completing the proof. □


Well-behaved Proof
Theorem (Type Preserving Compilation)
If a P4 program P_{P4} is well-typed and the compilation to NetQIR is type-preserving, then the NetQIR program P_{NetQIR} is well-typed and sound.
Proof.
Let $\Gamma_{P4} \vdash P_{P4} : \tau_{P4}$ denote that P_{P4} is well-typed in P4. The type-preserving compilation maps types τ_{P4} to τ_{NetQIR} in NetQIR.
Given $\Gamma_{P4} \vdash P_{P4} : \tau_{P4}$, the transpiled program P_{NetQIR} satisfies $\Gamma_{NetQIR} \vdash P_{NetQIR} : \tau_{NetQIR}$. The NetQIR type system ensures that all operations conform to expected types and behaviors.
Thus, P_{NetQIR} being well-typed implies it adheres to the verified semantics, ensuring sound execution and the preservation of properties from P4 to NetQIR, completing the proof. □

- Context: A P4 program $P_{P4} iswell - typedinP4'stypesystem, \Gamma_{P4} \vdash P_{P4} : \tau_{P4}$. - Goal: Prove that type preservation in compilation ensures soundness in NetQIR. - Proof Outline: - Type-preserving compilation translates $P_{P4} into P_{NetQIR} whilemaintainingtypeinformation. - TheNetQIRtypesystem T_{NetQIR} enforcessimilarorstricterrules, ensuringcorrectbehav$
$Therefore, if P_{P4} iswell - typed, P_{NetQIR} willalsobewell - typedandadheretocorrectsemantics, guaranteeingsoundexecution.$

End.

→ <shapr> I'm addicted to arrows.

→ * shapr begins his own paper "Generalizing Arrows to Spears".

→ <shapr> Spears can do anything efficiently, but they have sixty thousand laws they must satisfy, and we haven't actually found one yet.

→ <raphael> maybe "Generalizing Arrows to Nuclear Weapons" would simply be: `unsafeInterleaveIO`.

End.

→ <shapr> I'm addicted to arrows.

→ * shapr begins his own paper "Generalizing Arrows to Spears".

→ <shapr> Spears can do anything efficiently, but they have sixty thousand laws they must satisfy, and we haven't actually found one yet.

→ <raphael> maybe "Generalizing Arrows to Nuclear Weapons" would simply be: `unsafeInterleaveIO`.