Stateful Programming of High-Speed Network Hardware

Mina Tahmasbi Arashloo

Final Public Oral Presentation

Advisor: Jennifer Rexford
Readers: David Walker, Arvind Krishnamurthy
Examiners: Nick Feamster, Michael Freedman
Networks of Unprecedented Diversity and Scale

Enterprise Network

Data Center Network

Transit Network

- Email
- Skype

- Hadoop
- Spark
- Amazon S3
Networks of Unprecedented Diversity and Scale

Modern networks must provide:

- **High Performance**
- **Reliability**
- **Security**
The Evolution of Network Hardware
The Evolution of Network Hardware
The Evolution of Network Hardware

End-host CPU
Network Interface Card (NIC)

Switches
Early Networks: Stateful Edge, Stateless Core
Early Networks: Stateful Edge, Stateless Core

Stateless packet processing

End Host
- App
- Transport
- Network
- Link

Switch/Router
- Network
- Link

End Host
- App
- Transport
- Network
- Link
Early Networks: Stateful Edge, Stateless Core

Stateful packet processing

Stateless packet processing

Stateful packet processing

End Host

App

Transport

Network

Link

Switch/Router

Network

Link

Switch/Router

Network

Link

End Host

App

Transport

Network

Link
Early Networks: Stateful Edge, Stateless Core
Early Networks: Stateful Edge, Stateless Core
The Need for Stateful Processing in Hardware

Diagram showing the need for stateful processing in hardware.
The Need for Stateful Processing in Hardware

High-Speed Stateful Packet Processing??
Trend #1: In-Network Stateful Processing
Trend #1: In-Network Stateful Processing

Real-time monitoring (e.g., heavy hitters)
**Trend #1: In-Network Stateful Processing**

- **Real-time monitoring** (e.g., heavy hitters)
- **Efficient load balancing** (e.g., flowlet switching)

Diagram showing the layers of software and hardware in a network, including **End Host**, **Switch/Router**, and **NIC**.
Trend #1: In-Network Stateful Processing

- **Real-time monitoring** (e.g., heavy hitters)
- **Efficient load balancing** (e.g., flowlet switching)
- **Security** (e.g., stateful firewall)
Trend #1: In-Network Stateful Processing

Deploying stateful functionality:
- Adding middleboxes
- Modifying switch hardware
Trend #2: Increasing Link Speeds

10Gbps ⇒ 40Gbps ⇒ 100Gbps ⇒ 400Gbps

Software

End Host

App
Transport
Network
Link

Switch/Router

Network
Link

Middlebox

Hardware

End Host

App
Transport
Network
Link

NIC

NIC
Trend #2: Increasing Link Speeds

10Gbps ⇒ 40Gbps ⇒ 100Gbps ⇒ 400Gbps
Trend #2: Increasing Link Speeds

10Gbps $\Rightarrow$ 40Gbps $\Rightarrow$ 100Gbps $\Rightarrow$ 400Gbps

Network Hardware

High-Speed Stateful Packet Processing
What about Flexibility?
What about Flexibility?

Hard-Coded Vendor-Specific Protocols and Algorithms

Software

Hardware
Requirements of Today’s Network Hardware

- High-Speed Network Hardware
- Stateful Packet Processing
- Programmable End Host
- Link
- Network
- Transport
- App
- Switch/Router
- Middlebox
- NIC

Network Hardware:

- High-Speed
- Stateful Packet Processing
- Programmable
Network Hardware Design Space

- Speed
- Programmability
- Stateful Packet Processing
Network Hardware Design Space

Speed

Programmability

Stateful Packet Processing
Network Hardware Design Space

Speed

Stateful Packet Processing

Programmability
Network Hardware Design Space

- Speed
- Programmability
- Stateful Packet Processing
Network Hardware Design Space

- Speed
- Programmability
- Stateful Packet Processing

Programmable Switches
Network Hardware Design Space

- Programmability
- Speed
- Stateful Packet Processing

Programmable Switches

PISA
Network Hardware Design Space

Programmable NICs

Programmability

Speed

Programmable Switches

Stateful Packet Processing

PISA
Network Hardware Design Space

Programmable NICs

Speed

Stateful Packet Processing

Programmability

Programmable Switches

SoC

FPGA

PISA
Network Hardware Design Space

Programmable Switches

Programmable NICs

Network Hardware Design Space

Notoriously Difficult to Program

Programmability

Speed

Stateful Packet Processing
Network Hardware Design Space

Notoriously Difficult to Program

Interfaces are

Programmability

Stateful Packet Processing

Speed
Network Hardware Design Space

Notoriously Difficult to Program

Interfaces are
- Low-level
Network Hardware Design Space

Notoriously Difficult to Program

Interfaces are

- Low-level
- Tied to each device’s architecture
Network Hardware Design Space

Notoriously Difficult to Program

Interfaces are
• Low-level
• Tied to each device’s architecture
• Only suitable for programming a single device
This Dissertation

Design and implementation of

*modular and high-level* programming abstractions

for *stateful* programming of *high-speed* network hardware
This Dissertation
This Dissertation

Tonic
[Under revision for NSDI’20]
This Dissertation

Tonic
[Under revision for NSDI'20]

SNAP
[SIGCOMM'16]

FPGA
PISA
PISA
PISA
PISA
PISA
PISA
PISA
FPGA
This Dissertation

Tonic
[Under revision for NSDI’20]

SNAP
[SIGCOMM’16]
This Dissertation

• With an emphasis on
  • modularity
  • minimizing development effort
Enabling Programmable Transport Protocols on High-Speed NICs

Mina Tahmasbi Arashloo¹, Alexey Lavrov¹, Manya Ghobadi², Jennifer Rexford¹, David Walker¹, and David Wentzlaff¹

¹ Princeton University, ² MIT
TCP 101
TCP 101

- The most common transport protocol
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**
TCP 101

• The most common transport protocol
• Performs **reliable data delivery** and **congestion control**
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**
TCP 101

• The most common transport protocol
• Performs **reliable data delivery** and **congestion control**
TCP 101

• The most common transport protocol
• Performs **reliable data delivery** and **congestion control**

![Diagram of TCP sender and receiver with sliding window](image-url)
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**
TCP 101

- The most common transport protocol
- Performs **reliable data delivery** and **congestion control**

**Stateful Event Processing!**
(Data segments, Acks, Timeouts, …)
Constant Innovation in Transport Protocols
Constant Innovation in Transport Protocols

Homa
NDP
Karuna
PCC
DCTCP
New Reno
Reno
DCQCN
IRN
pHost
CUBIC
D3
TIMELY
QUIC
Sprout
BIC
Tahoe
BBR
TCP Vegas
RCP
PCC Vivace
Network Stacks in Data Centers
Network Stacks in Data Centers

<table>
<thead>
<tr>
<th>Target</th>
<th>CPU Overhead</th>
<th>Transport Programmability</th>
</tr>
</thead>
</table>

29
<table>
<thead>
<tr>
<th>Software</th>
<th>Target</th>
<th>CPU Overhead</th>
<th>Transport Programmability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>kernel user space</td>
<td>30-40%</td>
<td>✔</td>
</tr>
</tbody>
</table>
## Network Stacks in Data Centers

<table>
<thead>
<tr>
<th></th>
<th>Target</th>
<th>CPU Overhead</th>
<th>Transport Programmability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>kernel user space</td>
<td>30-40%</td>
<td>✔</td>
</tr>
<tr>
<td>Fixed-Function Hardware</td>
<td>NIC</td>
<td>~none</td>
<td>❌</td>
</tr>
</tbody>
</table>
# Network Stacks in Data Centers

<table>
<thead>
<tr>
<th></th>
<th>Target</th>
<th>CPU Overhead</th>
<th>Transport Programmability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software</strong></td>
<td>kernel user space</td>
<td>30-40%</td>
<td>✔</td>
</tr>
<tr>
<td>Fixed-Function Hardware</td>
<td>NIC</td>
<td>~none</td>
<td>❌</td>
</tr>
<tr>
<td>Programmable Hardware (Tonic)</td>
<td>NIC</td>
<td>~none</td>
<td>✔</td>
</tr>
</tbody>
</table>
Challenges of Hardware Programming for High-Speed NICs
Challenges of Hardware Programming for High-Speed NICs

• **Timing Constraints**
  • Median packet size in data centers is 200 bytes
  • At 100 Gbps, one 128-byte packet every ~10 ns
    • Back-to-back stateful event processing
Challenges of Hardware Programming for High-Speed NICs

• **Timing Constraints**
  - Median packet size in data centers is 200 bytes
  - At 100 Gbps, one 128-byte packet every ~10 ns
    - Back-to-back stateful event processing

• **Memory Constraints**
  - A few megabytes of high-speed memory
  - More than a thousand active flows
  - A few kilobits of per-flow state
Challenges of Hardware Programming for High-Speed NICs

- **Timing Constraints**
  - Median packet size in data centers is 200 bytes
  - At 100 Gbps, one 128-byte packet every ~10 ns
  - Back-to-back stateful event processing

- **Memory Constraints**
  - A few megabytes of high-speed memory
  - More than a thousand active flows
  - A few kilobits of per-flow state

---

**Tonic**

- A **programmable** hardware architecture
  - running at **100** Gbps
  - within **memory limits of commodity NICs**
  - to implement transport protocols
  - with **modest development effort**
Main Observation
Main Observation

*Common transport patterns as reusable components*
Main Observation

**Common transport patterns as reusable components**

- drive the design of an efficient hardware “template” for transport logic
Main Observation

*Common transport patterns as reusable components*

- drive the design of an efficient hardware “template” for transport logic
- reduce the functionality users must specify
The Transport Layer

Network Stack

App 1
send-data (addr,length)

App 2
send-data (addr,length)

... App N
send-data (addr,length)

IP and Below
The Transport Layer

Network Stack

The Transport Layer

IP and Below
The Transport Layer

Network Stack

App 1
send-data (addr,length)
Flow 1
- Byte status
  sent, in-flight, lost, ...
- Credit

App 2
send-data (addr,length)
Flow 2
- Byte status
  sent, in-flight, lost, ...
- Credit

... Flow m
- Byte status
  sent, in-flight, lost, ...
- Credit

App N
send-data (addr,length)

IP and Below
The Transport Layer

App 1
send-data (addr,length)

App 2
send-data (addr,length)

... ...

App N
send-data (addr,length)

Network Stack

The Transport Layer

Flow 1
- Byte status
  sent, in-flight, lost, ...
- Credit

Flow 2
- Byte status
  sent, in-flight, lost, ...
- Credit

... ...

Flow m
- Byte status
  sent, in-flight, lost, ...
- Credit

flow id, segment address

IP and Below
The Transport Layer

- Credit Management:

Network Stack

The Transport Layer

- Flow 1
  - Byte status
    - sent, in-flight, lost, ...
  - Credit

- Flow 2
  - Byte status
    - sent, in-flight, lost, ...
  - Credit

- Flow m
  - Byte status
    - sent, in-flight, lost, ...
  - Credit

flow id, segment address

IP and Below

App 1
send-data (addr,length)

App 2
send-data (addr,length)
The Transport Layer

- Credit Management: How many bytes can I send?

Network Stack

The Transport Layer

- Flow 1
  - Byte status
    - sent, in-flight, lost, ...
  - Credit

- Flow 2
  - Byte status
    - sent, in-flight, lost, ...
  - Credit

- Flow m
  - Byte status
    - sent, in-flight, lost, ...
  - Credit

flow id, segment address

IP and Below
The Transport Layer

**Credit Management:**
How many bytes can I send?

**Segment Selection:**

**Network Stack**

- Flow 1
  - Byte status
    - sent, in-flight, lost, ...
  - Credit

- Flow 2
  - Byte status
    - sent, in-flight, lost, ...
  - Credit

- Flow m
  - Byte status
    - sent, in-flight, lost, ...
  - Credit

**send-data (addr,length)**
The Transport Layer

- Credit Management: How many bytes can I send?
- Segment Selection: Which bytes do I send?

Network Stack

The Transport Layer

Flow 1
- Byte status
  - sent, in-flight, lost, ...
- Credit

Flow 2
- Byte status
  - sent, in-flight, lost, ...
- Credit

Flow m
- Byte status
  - sent, in-flight, lost, ...
- Credit

flow id, segment address

IP and Below
Segment Selection
(reliable delivery)

Update Byte Status

Pick Bytes for Next Segment
Segment Selection Patterns

Segment Selection
(reliable delivery)

Update Byte Status

Pick Bytes for Next Segment

Cannot maintain per-byte state on the NIC
Segment Selection Patterns

Segment Selection
(reliable delivery)

- Pre-Calculate Segment Boundaries
- Update Segment Status
- Select Next Segment
Segment Selection Patterns

Segment Selection
(reliable delivery)

- Pre-Calculate Segment Boundaries
  - Tonic
    - Update Segment Status
    - Select Next Segment
Segment Selection Patterns

1. Only a few bits of state per segment
Segment Selection Patterns

1. Only a few bits of state per segment
   - acked, rtxed, lost
Segment Selection Patterns

1. Only a few bits of state per segment
   - acked, rtxed, lost
   - fixed function modules for common state updates
Segment Selection
(reliable delivery)

1. Only a few bits of state per segment
   - acked, rtxed, lost
   - fixed function modules for common state updates
   - programmable modules only for loss detection
Segment Selection Patterns

1. Only a few bits of state per segment
   - acked, rtxed, lost
   - fixed function modules for common state updates
   - programmable modules only for loss detection

2. Loss detection: acks and timeouts
Segment Selection Patterns

1. Only a few bits of state per segment
   - acked, rtxed, lost
   - fixed function modules for common state updates
   - programmable modules only for loss detection

2. Loss detection: acks and timeouts
   - only two programmable modules
1. Only a few bits of state per segment
   • acked, rtxed, lost
   • fixed function modules for common state updates
   • programmable modules only for loss detection

2. Loss detection: acks and timeouts
   • only two programmable modules
   • mutually exclusive → fewer concurrent state updates
Segment Selection Patterns

1. Only a few bits of state per segment
   - acked, rtxed, lost
   - fixed function modules for common state updates
   - programmable modules only for loss detection

2. Loss detection: acks and timeouts
   - only two programmable modules
   - mutually exclusive → fewer concurrent state updates

3. Lost segments first, new segments next
Segment Selection Patterns

1. Only a few bits of state per segment
   - acked, rtxed, lost
   - fixed function modules for common state updates
   - programmable modules only for loss detection

2. Loss detection: acks and timeouts
   - only two programmable modules
   - mutually exclusive → fewer concurrent state updates

3. Lost segments first, new segments next
   - fixed-function module for segment generation
Concurrent State Update

Segment Selection
Concurrent State Update

Segment Selection

Memory for per-flow state:
segment status, window size, …
Concurrent State Update

Segment Selection

Memory for per-flow state:
segment status, window size, ...

active flow

Select Next Segment
Concurrent State Update

Segment Selection

Memory for per-flow state:
segment status, window size, ...

active flow

Select Next Segment

Incoming

Common Segment Updates

Loss Detection and Recovery

ACK
Concurrent State Update

Segment Selection

Memory for per-flow state:
segment status, window size, ...

Select Next Segment

Incoming

Common Segment Updates

Loss Detection and Recovery

Periodic Updates
(Timeout-based loss detection and recovery)

ACK

active flow

Timeout
Concurrent State Update

Segment Selection

Memory for per-flow state:
segment status, window size, ...

Select Next Segment

Incoming

Common Segment Updates

Loss Detection and Recovery

ACK

Timeout

active flow

Periodic Updates
(Timeout-based loss detection and recovery)
Concurrent State Update

Memory for per-flow state:
segment status, window size, ...

Incoming
Common Segment Updates
Loss Detection and Recovery

Periodic Updates (Timeout-based loss detection and recovery)

Select Next Segment

Timeout
ACK
active flow
Merge
Concurrent State Update

Memory for per-flow state:
segment status, window size, ...

Segment Selection

Select Next Segment

Incoming

Common Segment Updates

Loss Detection and Recovery

Periodic Updates
(Timeout-based loss detection and recovery)

Merge

ACK

active flow

Timeout
Concurrent State Update

Memory for per-flow state:
segment status, window size, ...

Segment Selection

Select Next Segment

Incoming

Common Segment Updates

Loss Detection and Recovery

Periodic Updates
(Timeout-based loss detection and recovery)

ACK

Timeout

active flow

Merge
Concurrent State Update

Segment Selection

Memory for per-flow state:
segment status, window size, ...

Select Next Segment

Incoming

Common Segment Updates

Loss Detection and Recovery

Periodic Updates
(Timeout-based loss detection and recovery)

ACK

Timeout

active flow

Merge

36
Credit Management Patterns
Credit Management Patterns

Credit Management
(congestion control)
Credit Management
(congestion control)

Control Loop

Monitor

Adjust Params
Credit Management Patterns

Credit Management (congestion control)

Control Loop

Monitor -> Adjust Params

window/rate

Calculate Credit
Credit Management Patterns

Credit Management
(congestion control)

Control Loop

Monitor

Adjust Params

window/rate

Calculate Credit

1. Three common credit calculation schemes
Credit Management Patterns

1. Three common credit calculation schemes
   - congestion window, rate, grant tokens
Credit Management Patterns

1. Three common credit calculation schemes
   - congestion window, rate, grant tokens

2. Two main parameter adjustment signals
Credit Management Patterns

1. Three common credit calculation schemes
   - congestion window, rate, grant tokens

2. Two main parameter adjustment signals
   - external signals, e.g., acks and CNPs
Credit Management Patterns

1. Three common credit calculation schemes
   - congestion window, rate, grant tokens

2. Two main parameter adjustment signals
   - external signals, e.g., acks and CNPs
   - periodic internal signals, e.g., counters
Credit Management Patterns

1. Three common credit calculation schemes
   - congestion window, rate, grant tokens

2. Two main parameter adjustment signals
   - external signals, e.g., acks and CNPs
   - periodic internal signals, e.g., counters
   - aligns with existing programmable modules for segment selection
The Two Engines

Segment Selection

Credit Management

flow ID, segment ID

segment transmitted
Integration into the Network Stack

Host

NIC

Outgoing Link
Integration into the Network Stack

Host
- Memory
- Application Layer

NIC
- IP Layer and Below

Outgoing Link
Integration into the Network Stack

Host
- Memory
- Application Layer
  - Transport Layer - on the host
    - Connection Management
    - Segmentation

NIC
- IP Layer and Below

Outgoing Link
Integration into the Network Stack

Host
- Memory
- Application Layer
  - Transport Layer - on the host
  - Connection Management
  - Segmentation

NIC
- Transport Layer - on the NIC
  - Data Transfer
- IP Layer and Below
Integration into the Network Stack

Host
- Memory
- Application Layer
  - Transport Layer - on the host
    - Connection Management
    - Segmentation

NIC
- Transport Layer - on the NIC
  - Data Transfer

add/remove connection
send N segments from memory address A
Integration into the Network Stack

Host

- Memory
- Application Layer
  - Transport Layer - on the host
    - Connection Management
    - Segmentation

NIC

- Transport Layer - on the NIC
  - Data Transfer
- Transport Logic (Tonic)

Outgoing Link

IP Layer and Below

add/remove connection
send N segments from memory address A
Integration into the Network Stack

Host
- Application Layer
  - Transport Layer - on the host
    - Connection Management
    - Segmentation
- Memory

NIC
- Transport Layer - on the NIC
  - Data Transfer
- Transport Logic (Tonic)
- DMA
- IP Layer and Below
- Outgoing Link

add/remove connection
send N segments from memory address A
Integration into the Network Stack

Host
- Application Layer
  - Transport Layer - on the host
    - Connection Management
    - Segmentation

Memory

Transport Layer - on the host
- Connection Management
- Segmentation

Transport Layer - on the NIC
- Data Transfer

IP Layer and Below

Outgoing Link

Transport Logic (Tonic)
DMA

add/remove connection
send N segments from memory address A
Integration into the Network Stack

Host

Application Layer
- Transport Layer - on the host
  - Connection Management
  - Segmentation

Memory

Transport Layer - on the host

add/remove connection
send N segments from memory address A

NIC

Transport Layer - on the NIC
- Data Transfer

Transport Logic (Tonic)

DMA

Outgoing Link

IP Layer and Below

Next Segment

39
Integration into the Network Stack

Host

- Application Layer
  - Transport Layer - on the host
    - Connection Management
    - Segmentation

- Memory

Transport Layer - on the host
- Connection Management
- Segmentation

add/remove connection
send N segments from memory address A

NIC

- Transport Layer - on the NIC
  - Data Transfer

- Transport Logic (Tonic)
- DMA

IP Layer and Below

Outgoing Link

Next Segment
Evaluation - Programmability
Evaluation - Programmability

• Implemented six representative protocols
Evaluation - Programmability

• Implemented six representative protocols
  • Reno, New Reno
Evaluation - Programmability

• Implemented six representative protocols
  • Reno, New Reno
  • SACK (Selective ACK)
Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
  - DCQCN, IRN (Improved RoCE NIC)
Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
  - DCQCN, IRN (Improved RoCE NIC)

- All meet timing for 100 Gpbs (10-ns clock)
Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
  - DCQCN, IRN (Improved RoCE NIC)

- All meet timing for 100 Gpbs (10-ns clock)

- Implemented within 200 lines of Verilog code
Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
  - DCQCN, IRN (Improved RoCE NIC)

- All meet timing for 100 Gpbs (10-ns clock)

- Implemented within 200 lines of Verilog code
  - uses 0.5% of total logic resources
Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
  - DCQCN, IRN (Improved RoCE NIC)

- All meet timing for 100 Gpbs (10-ns clock)

- Implemented within 200 lines of Verilog code
  - uses 0.5% of total logic resources

- Re-usable modules are 8K lines of Verilog code
Evaluation - Programmability

- Implemented six representative protocols
  - Reno, New Reno
  - SACK (Selective ACK)
  - NDP (Receiver-driven data-center transport)
  - DCQCN, IRN (Improved RoCE NIC)

- All meet timing for 100 Gpbs (10-ns clock)

- Implemented within 200 lines of Verilog code
  - uses 0.5% of total logic resources

- Re-usable modules are 8K lines of Verilog code
  - uses 35% of total logic resources
### Evaluation - Scalability

<table>
<thead>
<tr>
<th>Metric</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity of User-Defined Logic</td>
<td>(0, 31) meets timing</td>
</tr>
<tr>
<td>User-Defined Logic</td>
<td>(31, 42) depends on operations</td>
</tr>
<tr>
<td></td>
<td>(42, 65) violates timing</td>
</tr>
<tr>
<td>User-Defined State</td>
<td>256 bytes grant token</td>
</tr>
<tr>
<td></td>
<td>340 bytes rate</td>
</tr>
<tr>
<td></td>
<td>448 bytes congestion window</td>
</tr>
<tr>
<td>Window Size</td>
<td>segments 256</td>
</tr>
<tr>
<td>Concurrent Flows</td>
<td>count 2048</td>
</tr>
</tbody>
</table>
Evaluation - End-to-End Simulations

- Cycle-accurate hardware simulator for Tonic within NS3
- Compared existing protocols with Tonic implementations
  - TCP New Reno (plots shown below) and DCQCN
Summary
Summary

• Tonic is a **programmable hardware** architecture
Summary

- Tonic is a **programmable hardware** architecture
- Enables implementing transport protocols at high-speed
  - with **modest development effort**
Summary

• Tonic is a **programmable hardware** architecture

• Enables implementing transport protocols at high-speed
  • with **modest development effort**

• Exploits domain-specific optimizations
  • Implementing common transport patterns as re-usable modules
# In-Network Stateful Applications

<table>
<thead>
<tr>
<th>Source</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimera (USENIX Security’12)</td>
<td>Number of domains sharing the same IP address</td>
</tr>
<tr>
<td></td>
<td>Number of distinct IP addresses under the same domain</td>
</tr>
<tr>
<td></td>
<td>DNS TTL change tracking</td>
</tr>
<tr>
<td></td>
<td>DNS tunnel detection</td>
</tr>
<tr>
<td></td>
<td>Sidejack detection</td>
</tr>
<tr>
<td></td>
<td>Phishing/spam detection</td>
</tr>
<tr>
<td>FAST (HotSDN’14)</td>
<td>Stateful firewall</td>
</tr>
<tr>
<td></td>
<td>FTP monitoring</td>
</tr>
<tr>
<td></td>
<td>Heavy-hitter detection</td>
</tr>
<tr>
<td></td>
<td>Super-spreader detection</td>
</tr>
<tr>
<td></td>
<td>Sampling based on flow size</td>
</tr>
<tr>
<td></td>
<td>Selective packet dropping (MPEG frames)</td>
</tr>
<tr>
<td></td>
<td>Connection affinity</td>
</tr>
<tr>
<td>Bohatei (USENIX Security’15)</td>
<td>SYN flood detection</td>
</tr>
<tr>
<td></td>
<td>DNS reflection (and amplification) detection</td>
</tr>
<tr>
<td></td>
<td>UDP flood mitigation</td>
</tr>
<tr>
<td></td>
<td>Elephant flows detection</td>
</tr>
<tr>
<td>Others</td>
<td>Bump-on-the-wire TCP state machine</td>
</tr>
<tr>
<td></td>
<td>Snort flowbits</td>
</tr>
</tbody>
</table>
In-Network Stateful Applications

<table>
<thead>
<tr>
<th>Source</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chimera (USENIX Security’12)</td>
<td>Number of domains sharing the same IP address</td>
</tr>
<tr>
<td></td>
<td>Number of distinct IP addresses under the same domain</td>
</tr>
<tr>
<td></td>
<td>DNS TTL change tracking</td>
</tr>
<tr>
<td></td>
<td>DNS tunnel detection</td>
</tr>
<tr>
<td>(H)</td>
<td></td>
</tr>
<tr>
<td>Bohatei (USENIX Security’15)</td>
<td>SYN flood detection</td>
</tr>
<tr>
<td></td>
<td>DNS reflection (and amplification) detection</td>
</tr>
<tr>
<td></td>
<td>UDP flood mitigation</td>
</tr>
<tr>
<td></td>
<td>Elephant flows detection</td>
</tr>
<tr>
<td>Others</td>
<td>Bump-on-the-wire TCP state machine</td>
</tr>
<tr>
<td></td>
<td>Snort flowbits</td>
</tr>
</tbody>
</table>

Programmable switches expose **data plane state** through a programming interface!
Distributed Stateful Programming is Challenging
Distributed Stateful Programming is Challenging
Distributed Stateful Programming is Challenging

Use one switch?
Distributed Stateful Programming is Challenging

Use one switch?

Shard/Distribute across multiple switches?
Distributed Stateful Programming is Challenging

Use one switch?  
Shard/Distribute across multiple switches?  
which switches to use?
Distributed Stateful Programming is Challenging

- Use one switch?
- Shard/Distribute across multiple switches?
- which switches to use?
- How to coordinate between them for correct stateful processing?
Distributed Stateful Programming is Challenging

- Use one switch?
- Shard/Distribute across multiple switches?
- which switches to use?
- How to coordinate between them for correct stateful processing?
SNAP: Stateful Network-Wide Abstractions for Packet Processing
SNAP: Stateful Network-Wide Abstractions for Packet Processing
SNAP: Stateful Network-Wide Abstractions for Packet Processing
SNAP: Stateful Network-Wide Abstractions for Packet Processing
Example - Detecting DNS Reflection Attacks

Attacker

Attacker-controlled botnet

Small spoofed DNS request

DNS Resolvers

Amplified DNS response from open resolver

Victim

http://ddosandbotnets.blogspot.com
Example - Detecting DNS Reflection Attacks
Example - Detecting DNS Reflection Attacks

1. Log DNS requests

DNS Resolver

DNS Resolvers

Victim
Example - Detecting DNS Reflection Attacks

1. Log DNS requests
2. Match responses

DNS Resolver
DNS Resolvers
Victim
Example - Detecting DNS Reflection Attacks

1. Log DNS requests

2. Match responses

DNS Resolver

DNS Resolvers

Victim
Example - Detecting DNS Reflection Attacks

1. Log DNS requests
2. Match responses
3. Check unmatched count

Victim

DNS Request

DNS Response

DNS Resolver

DNS Resolvers
SNAP Language
Stateful Packet Processing Functions

• A function specifying
  • how to process each packet
  • based on its fields and the program state
if (scrip in CSNET) & (dstport = DNS) then
  seen[srcip][dns.id] ← True
else if (dstip in CSNET) & (srcport = DNS) then
  if ~seen[dstip][dns.id] then
    unmatched[dstip]++
  if unmatched[dstip] = THRESH then
    susp[dstip] ← True
  else id
else id
DNS Reflection Detection in SNAP

```plaintext
if (scrip in CSNET) & (dstport = DNS) then
    seen[srcip][dns.id] ← True
else if (dstip in CSNET) & (srcport = DNS) then
    if ~seen[dstip][dns.id] then
        unmatched[dstip]++
        if unmatched[dstip] = THRESH then
            susp[dstip] ← True
    else id
else id
```
DNS Reflection Detection in SNAP

```plaintext
if (scrip in CSNET) & (dstport = DNS) then
  seen[srcip][dns.id] ← True
else if (dstip in CSNET) & (srcport = DNS) then
  if ~seen[dstip][dns.id] then
    unmatched[dstip]++
    if unmatched[dstip] = THRESH then
      susp[dstip] ← True
  else id
else id
```
if (srcip in CSNET) & (dstport = DNS) then
    seen[srcip][dns.id] ← True

else if (dstip in CSNET) & (srcport = DNS) then
    if ~seen[dstip][dns.id] then
        unmatched[dstip]++
    if unmatched[dstip] = THRESH then
        susp[dstip] ← True

else id
else id
if (srcip in CSNET) & (dstport = DNS) then
    seen[srcip][dns.id] ← True
else if (dstip in CSNET) & (srcport = DNS) then
    if ~seen[dstip][dns.id] then
        unmatched[dstip]++
        if unmatched[dstip] = THRESH then
            susp[dstip] ← True
    else id
else id
if (scrip in CSNET) & (dstport = DNS) then
  seen[srcip][dns.id] ← True
else if (dstip in CSNET) & (srcport = DNS) then
  if ~seen[dstip][dns.id] then
    unmatched[dstip]++
    if unmatched[dstip] = THRESH then
      susp[dstip] ← True
  else id
else id
else id
Program Composition

( + ++ + ) ; ( ; ) + ( + + + ) ; ( + )
SNAP Compiler
Where to Place State Variables?

```
if srcip in CSNET & dstport = 53 then
    seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
    if ~seen[dstip][dns.id] then
        unmatched[dstip]++;
        if unmatched[dstip] = threshold then
            susp[dstip] ← True
    else id
else id
```

```
if dstip = CSNET then outport ← CS
else if dstip = EENET then outport ← EE
else if dstip = ISP1NET then outport ← ISP1
else if dstip = ISP2NET then outport ← ISP2
else drop
```
How to Forward Packets through State Variables?

```
if srcip in CSNET & dstport = 53 then
    seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
    if ~seen[dstip][dns.id] then
        unmatched[dstip]++;
        if unmatched[dstip] = threshold then
            susp[dstip] ← True
    else id
else id

if dstip = CSNET then outport ← CS
else if dstip = EENET then outport ← EE
else if dstip = ISP1NET then outport ← ISP1
else if dstip = ISP2NET then outport ← ISP2
else drop
```
How to Forward Packets through State Variables?

if srcip in CSNET & dstport = 53 then
  seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
  if ~seen[dstip][dns.id] then
    unmatched[dstip]++;
    if unmatched[dstip] = threshold then
      susp[dstip] ← True
  else id
else id

if dstip = CSNET then outport ← CS
else if dstip = EENET then outport ← EE
else if dstip = ISP1NET then outport ← ISP1
else if dstip = ISP2NET then outport ← ISP2
else drop

DNS Response to: H1 in CS
ID: 200
Program Analysis
Program Analysis

• For each flow, find
  • all the state variables that it needs
  • the order in which the state variables should be visited
Program Analysis

• For each flow, find
  • all the state variables that it needs
  • the order in which the state variables should be visited

• A flow can be defined at any granularity
  • As long as its traffic statistics is available
  • E.g., All DNS packets to the CS subnet need all three state variables
  • E.g., All packets from the CS subnet need seen.
Mixed-Integer Linear Program (MILP)

Program Analysis Results

Traffic Matrix

CS
Mixed-Integer Linear Program (MILP)
How to distribute a SNAP program?
How to distribute a SNAP program?
Extended Forwarding Decision Diagrams (xFDDs)

- dstip = 10.0.0.1
- srcip = dstip
- s[srcip] = 2
- {s[dstip] ← 2}
- {drop}

Intermediate Nodes
Tests on header fields and state

Leaves
Sets of action sequences
xfdd for DNS Reflection Detection

if srcip in CSNET & dstport = 53 then
    seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
    if ~seen[dstip][dns.id] then
        unmatched[dstip]++;
        if unmatched[dstip] = threshold then
            susp[dstip] ← True
    else id
else id

srcip in CSNET

dstport = 53

{seen[srcip][dns.id] ← True}

srcport = 53

dstip in CSNET

seen[dstip][dns.id]

unmatched[dstip] = threshold - 1

id

id

{unmatched[dstip]++,
susp[srcip][dstip] ← True}

{unmatched[dstip]++}
xfD for DNS Reflection Detection

if srcip in CSNET & dstport = 53 then
    seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
    if ~seen[dstip][dns.id] then
        unmatched[dstip]++;
        if unmatched[dstip] = threshold then
            susp[dstip] ← True
    else id
else id
xFDD for DNS Reflection Detection

```
if srcip in CSNET & dstport = 53 then
    seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
    if ¬seen[dstip][dns.id] then
        unmatched[dstip]++;
        if unmatched[dstip] = threshold then
            susp[dstip] ← True
    else id
else id
```
xFDD for DNS Reflection Detection

```plaintext
if srcip in CSNET & dstport = 53 then
    seen[srcip][dns.id] ← True
else if dstip in CSNET & srcport = 53 then
    if ~seen[dstip][dns.id] then
        unmatched[dstip]++;
        if unmatched[dstip] = threshold then
            susp[dstip] ← True
    else id
else id
```
Partitioning and Distribution of the xFDD
Partitioning and Distribution of the xFDD

- The stateless tests and actions are at the top.
Partitioning and Distribution of the xFDD

- The stateless tests and actions are at the top.
- Stateful tests and actions on the same variable form subtrees.
Partitioning and Distribution of the xFDD

- The stateless tests and actions go to the edge.
- Subtrees of a stateful variable go to the switch storing it.
Partitioning and Distribution of the xFDD

- The stateless tests and actions go to the edge.
- Subtrees of a stateful variable go to the switch storing it.
Putting It All Together

ISP1

ISP2

CS

dstip in CSNET

srcip in CSNET

srcport = 53

dstip in CSNET

seen[dstip] [dns.id]

unmatched[dstip] = threshold - 1

{unmatched[dstip]++;
susp[srcip][dstip] ← True;
outport ← CS}

{unmatched[dstip]++;
outport ← CS}

outport ← EE

dstip in EENET
Putting It All Together

ISP1 → CS

ISP2 → EE

srcip in CSNET

dstip in CSNET

seen[dstip] [dns.id]

outport ← CS

{unmatched[dstip]++;
susp[srcip][dstip] ← True;
outport ← CS}

{unmatched[dstip]++;
outport ← EE}
Putting It All Together

ISP1

ISP2

CS

dstip in CSNET

srcip in CSNET

srcport = 53

dstip in EENET

seen.dstip [dns.id]

unmatched.dstip++;
susp.srcip.dstip ← True;
outport ← CS

outport ← EE

unmatched.dstip = threshold - 1

{unmatched.dstip++;
outport ← CS}
Putting It All Together

ISP1 → CS
ISP2 → EE

srcip in CSNET

dstip in CSNET

srcport = 53

dstip in EENET

seen[dstip] [dns.id]

unmatched[dstip] = threshold - 1

{unmatched[dstip]++; susp[srcip][dstip] ← True; outport ← CS}

outport ← CS

outport ← EE
Putting It All Together

ISP1
ISP2
CS
EE

srcip in CSNET

dstip in CSNET

dstip in EENET

1
2
3

srcport = 53

4

seen[dstip]
[dns.id]

5

outport ← CS

6

unmatched[dstip] = threshold - 1

7

{unmatched[dstip]++; susp[srcip][dstip] ← True; outport ← CS}

8

{unmatched[dstip]++; outport ← CS}

9

10

outport ← EE
Compiler Evaluation
Compiler Evaluation

• 7 Campus and ISP topologies
Compiler Evaluation

• 7 Campus and ISP topologies

• Order of 100s of switches and links
Compiler Evaluation

- 7 Campus and ISP topologies
- Order of 100s of switches and links
- Scenarios
  - Cold start (freq. weeks)
  - Policy change (freq. days)
  - Topology/TM change (freq. minutes)
Compiler Evaluation - Results

Time (sec.)

Topology/TM Change
Policy Change
Cold Start

Stanford
Berkley
Purdue
ISP 1755
ISP 1221
ISP 6461
ISP 3257
Summary
Summary

• SNAP is a network-wide programmable platform for stateful packet processing
Summary

• SNAP is a network-wide programmable platform for stateful packet processing

• SNAP Language
  • One big stateful switch abstraction
  • Intuitive and flexible composition
Summary

• SNAP is a **network-wide programmable** platform for **stateful** packet processing

• SNAP Language
  • One big stateful switch abstraction
  • Intuitive and flexible composition

• SNAP Compiler
  • Decides state placement and routing
  • Distributes an intermediate representation of the program across the network
Stateful Programming of High-Speed Network Hardware

Tonic
[Under revision for NSDI’19]

SNAP
[SIGCOMM’16]
Thank You!