Checking-in on Network Functions

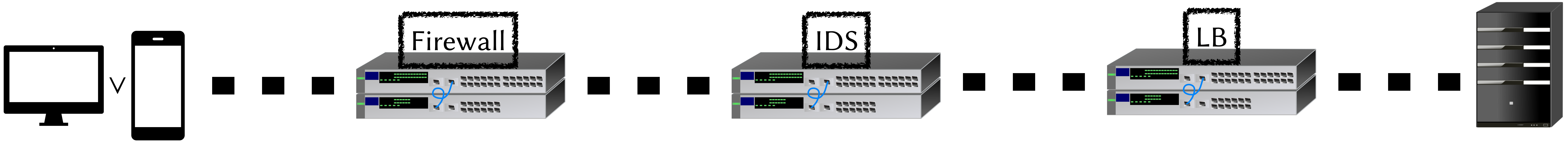
by Zeeshan Lakhani and Heather Miller

@ Carnegie Mellon University

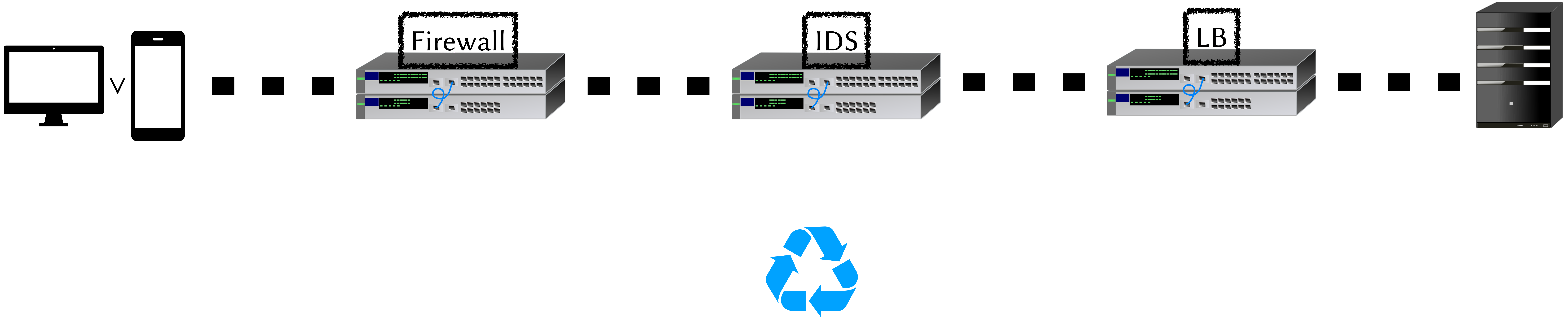
institute for SOFTWARE RESEARCH

The rise of network functions?

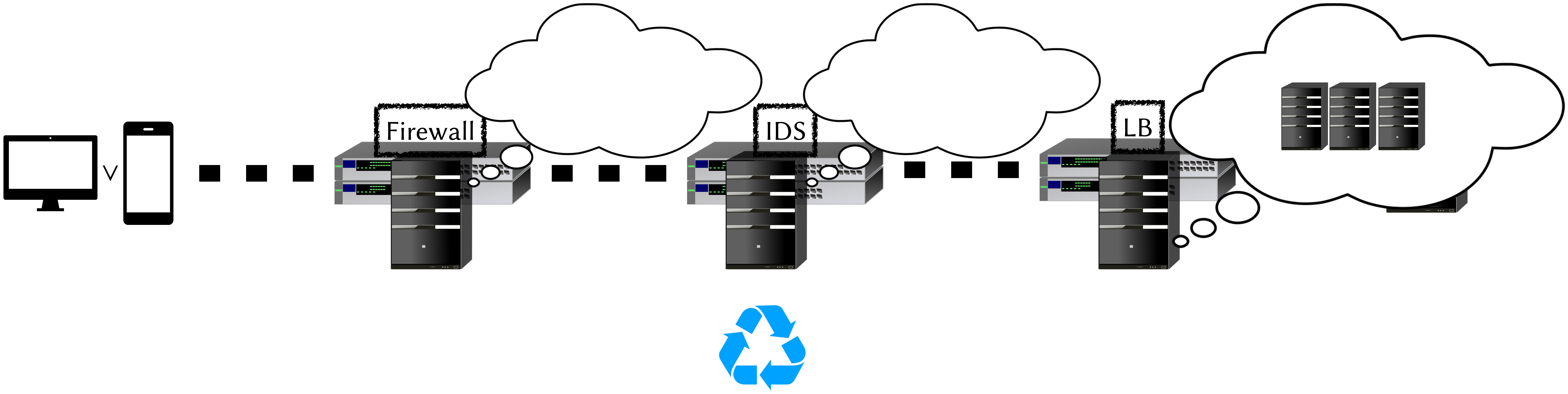
The rise of network functions?



The rise of network functions?



The rise of network functions?



The rise of writing and modeling network functions?

Pyretic

```
class firewall(DynamicPolicy):

    def __init__(self):
        # Initialize the firewall
        print "initializing firewall"
        self.firewall = {}
        super(firewall,self).__init__(true)
        import threading
        self.ui = threading.Thread(target=self.ui_loop)
        self.ui.daemon = True
        self.ui.start()

    def AddRule (self, mac1, mac2):
        if (mac2,mac1) in self.firewall:
            print "Firewall rule for %s: %s already exists" % (mac1,mac2)
            return
        self.firewall[(mac1,mac2)]=True
        print "Adding firewall rule in %s: %s" % (mac1,mac2)
        self.update_policy()

    def DeleteRule (self, mac1, mac2):
        try:
            del self.firewall[(mac1,mac2)]
            print "Deleting firewall rule in %s: %s" % (mac1,mac2)
            self.update_policy()
        except:
            pass
        try:
            del self.firewall[(mac2,mac1)]
            print "Deleting firewall rule in %s: %s" % (mac1,mac2)
            self.update_policy()
        except:
            pass
```

Slick

```
class BlacklistDropper(Application):
    def init(self, blacklist):
        flow = self.make_wildcard_flow()
        flow['tp_dst'] = 53
        eds = self.apply_elem(flow, ["DnsDpi"])
        if(self.check_elems_installed(eds)):
            self.installed = True
        droppers = list()

    def handle_trigger(self, ed, trigger):
        if(trigger['type'] == 'BlacklistedQuery'):
          src_flow = self.make_wildcard_flow()
          src_flow['nw_src'] = trigger['src_ip']
          eds = apply_elem(src_flow, ["DropAll"])
          if(self.check_elems_installed(eds)):
              droppers.append(eds[0])
```

NetKat

$$
\begin{aligned}
& (\text{if typ} = \textsc{ssh} \text{ then vlan} := W \text{ else } 1) \cdot \\
& (\text{if dst} = A \text{ then pt} := 1 \text{ else if dst} = B \text{ then pt} := 2 \text{ else } 0) \\
= \;& \text{if dst} = A \cdot \text{typ} = \textsc{ssh} \text{ then vlan} := W \cdot \text{pt} := 1 \\
& \text{else if dst} = A \text{ then pt} := 1 \\
& \text{else if dst} = B \cdot \text{typ} = \textsc{ssh} \text{ then vlan} := W \cdot \text{pt} := 2 \\
& \text{else if dst} = B \text{ then pt} := 2 \\
& \text{else } 0
\end{aligned}
$$

The rise of writing and modeling network functions?

Writing network functions is not
“composed of nothing more than algorithms and small programs”[1]

- complex routing and load balancing policies
- traffic monitoring
- experimental/new specifications, protocols, and headers
- computation and aggregation
  (e.g. In-Network Computation is a Dumb Idea Whose Time Has Come)

[1] Cultures of programming: Understanding the history of programming through controversies and technical artifacts by Tomas Petricek, University of Kent, UK, 2019

Motivation

Motivation

If I program in React, can I program a network function?

Motivation

If I program in React, can I program a network function?

How do we know what we're doing is right?

Motivation

If I program in React, can I program a network function?

How do we know what we're doing is right?

How can we iterate?

Motivation

- Limits of Correctness
  - e.g. reliance on OpenFlow protocol

Motivation

- Limits of Correctness
  - e.g. reliance on OpenFlow protocol
- Arbitrary (ad-hoc) Logic & Variable-length Data, e.g. Ipv6 Extensions, ndp options
  - packet length
  - failure and reconfiguration

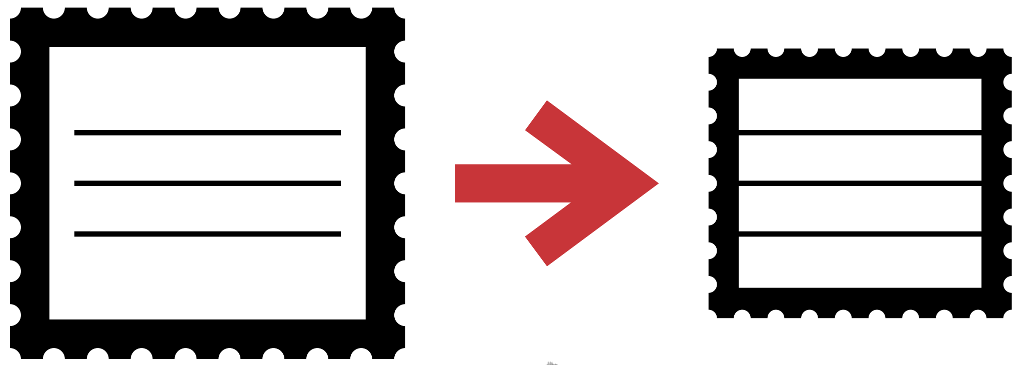
Motivation

```
if(ntohs(ip->ip6_plen) > (plen - 40)) [2]
  goto bad;
```

[2] The Click Modular Router by Eddie Kohler, et. al., Laboratory for Computer Science, MIT, 1999

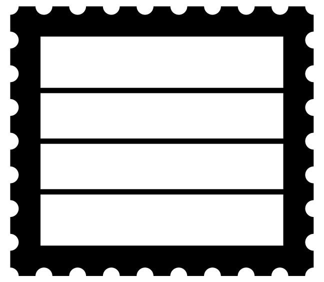
Two examples

MTU: *Send Too Big*



- swap ethernet addresses
- swap src/dst
- change protocol
- set mtu info
- calculate checksum

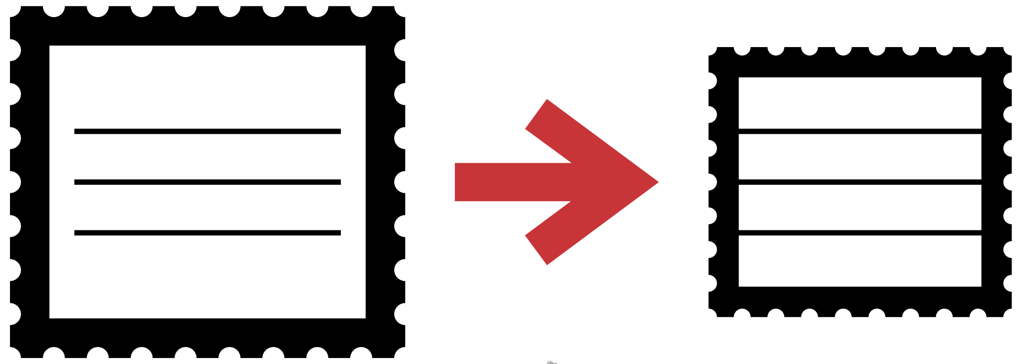
Ipv6 Extension Headers: *SRH*



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Hdr Ext Len  | Routing Type  | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Last Entry   |     Flags     |              Tag              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[0] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
                              ...
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[n] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//                                                             //
//         Optional Type Length Value objects (variable)       //
//                                                             //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

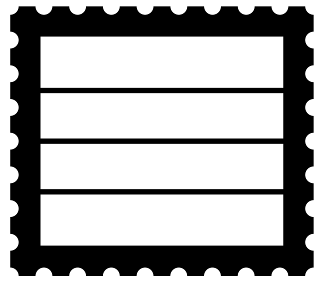
Two examples

MTU: *Send Too Big*



- swap ethernet addresses
- swap src/dst
- change protocol
- set mtu info
- calculate checksum

Ipv6 Extension Headers: *SRH*



Decimal	**Protocol**	RFC	IANA
0	Hop-by-Hop Options	✓	✓
43	Routing	✓	✓
44	Fragment	✓	✓
50	Encapsulating Security Payload	✓	✓
51	Authentication	✓	✓
60	Destination Options	✓	✓
135	Mobility Header		✓
139	Host Identity Protocol		✓
140	Shim6		✓
253	Experiments/testing purposes		✓
254	Experiments/testing purposes		✓

Kinds of Contracts

Kinds of Contracts

Design by Contract

focused on how runtime contracts can be turned on for monitoring and testing situations so that developers can
“sit back, and just watch their contracts be violated”

- erased on release binaries

Kinds of Contracts

Design by Contract

focused on how runtime contracts can be turned on for monitoring and testing situations so that developers can
"sit back, and just watch their contracts be violated"

- erased on release binaries

Static Assertions

compile-time assertions for consts, statics

- remain in release binaries

Kinds of Contracts

Design by Contract

focused on how runtime contracts can be turned on for monitoring and testing situations so that developers can
"sit back, and just watch their contracts be violated"

- erased on release binaries

Static Assertions

compile-time assertions for consts, statics

- remain in release binaries

Static Order-Preserving Headers

```
impl EndOffset for Ipv6Hdr {
    type PreviousHdr=EthHdr;
    fn offset(&self) -> usize { 40 }
}
```

Kinds of Contracts: Design by Contract

dependencies and related components in the system. These contracts are usually separated into *pre* (input/ingress) and *post* conditions (output/egress), where invariants can be asserted on for incoming and outgoing data accordingly.

In our system, design by contract-styled assertions help programmers articulate what the values of fields in a header should be equal to, bound by, approximate to, or how these values may have shifted during packet transformation (e.g. swapping of MAC addresses). From a processing perspective, the input precondition runs when the packet enters a NF and the postcondition runs as the packet is exiting the function.

Kinds of Contracts: Design by Contract

dependencies and related components in the system. These contracts are usually separated into *pre* (input/ingress) and *post* conditions (output/egress), where invariants can be asserted on for incoming and outgoing data accordingly.

In our system, design by contract-styled assertions help programmers articulate what the values of fields in a header should be equal to, bound by, approximate to, or how these values may have shifted during packet transformation (e.g. swapping of MAC addresses). From a processing perspective, the input precondition runs when the packet enters a NF and the postcondition runs as the packet is exiting the function.

Kinds of Contracts: Design by Contract

dependencies and related components in the system. These contracts are usually separated into *pre* (input/ingress) and *post* conditions (output/egress), where invariants can be asserted on for incoming and outgoing data accordingly.

In our system, design by contract-styled assertions help programmers articulate what the values of fields in a header should be equal to, bound by, approximate to, or how these values may have shifted during packet transformation (e.g. swapping of MAC addresses). From a processing perspective, the input precondition runs when the packet enters a NF and the postcondition runs as the packet is exiting the function.

Kinds of Contracts: Static Assertions

Static assertions, popularized in the C, C++, and D languages, allow for compile-time assertions of statically defined expressions, e.g. constants, statics. Beyond just checking for specific values, static assertions can be used to enforce fields on *struct* types and check if a pointer's underlying value is the same when coerced to another type. NF programs tend to be comprised of many constants referring to values derived from specifications. For example, the IPv6 minimum MTU value is 1280 [6], but is actually 1294 in practice when the Ethernet header is included. Our approach can check this caveat statically at the call site where the NF is defined—not where it's instantiated—via compile-time assertions in our prototype for constant checking. Additionally, thanks to *conditional compilation* (see 4.1 for more information), static assertions remain in release binaries.

Kinds of Contracts: Static Assertions

Static assertions, popularized in the C, C++, and D languages, allow for compile-time assertions of statically defined expressions, e.g. constants, statics. Beyond just checking for specific values, static assertions can be used to enforce fields on *struct* types and check if a pointer's underlying value is the same when coerced to another type. NF programs tend to be comprised of many constants referring to values derived from specifications. For example, the IPv6 minimum MTU value is 1280 [6], but is actually 1294 in practice when the Ethernet header is included. Our approach can check this caveat statically at the call site where the NF is defined—not where it's instantiated—via compile-time assertions in our prototype for constant checking. Additionally, thanks to *conditional compilation* (see 4.1 for more information), static assertions remain in release binaries.

Kinds of Contracts: Static Assertions

Static assertions, popularized in the C, C++, and D languages, allow for compile-time assertions of statically defined expressions, e.g. constants, statics. Beyond just checking for specific values, static assertions can be used to enforce fields on *struct* types and check if a pointer's underlying value is the same when coerced to another type. NF programs tend to be comprised of many constants referring to values derived from specifications. For example, the IPv6 minimum MTU value is 1280 [6], but is actually 1294 in practice when the Ethernet header is included. Our approach can check this caveat statically at the call site where the NF is defined—not where it's instantiated—via compile-time assertions in our prototype for constant checking. Additionally, thanks to *conditional compilation* (see 4.1 for more information), static assertions remain in release binaries.

Kinds of Contracts: Static Order-Persevering Headers

we leverage this statically-defined order mechanism on headers (4) to ensure that incoming and outgoing packet header ordering is preserved according to encoded expectations.

Kinds of Contracts: Static Order-Persevering Headers

we leverage this statically-defined order mechanism on headers (4) to ensure that incoming and outgoing packet header ordering is preserved according to encoded expectations.

Implementation

Implementation

- prototyped as a **gradual** extension to **NetBricks** (i.e. NetBricks: Taking the V out of NFV, OSDI 2016)

Implementation

Focused on Zero-Copy
Soft Isolation

- prototyped as a **gradual** extension to **NetBricks** (i.e. NetBricks: Taking the V out of NFV, OSDI 2016)

Implementation

Focused on Zero-Copy Soft Isolation

- prototyped as a **gradual** extension to **NetBricks** (i.e. NetBricks: Taking the V out of NFV, OSDI 2016)
- implemented as a small rust library to easily write specifications, which **generates code** for validations and assertions at compile-time

Implementation

- prototyped as a **gradual** extension to **NetBricks** (i.e. NetBricks: Taking the V out of NFV, OSDI 2016)
- implemented as a small rust library to easily write specifications, which **generates code** for validations and assertions at compile-time

Focused on Zero-Copy Soft Isolation

macros turn checks into static and dynamic contracts

Implementation

DPDK
DATA PLANE DEVELOPMENT KIT

```rust
fn install<T, S>(ports: Vec<T>, sched: &mut S)
where
    T: PacketRx + PacketTx + Display + Clone + 'static,
    S: Scheduler + Sized,
{
    for port in &ports {
        println!("Receiving port {}", port);
    }

    let pipelines: Vec<_> = ports
        .iter()
        .map(|port| {
            ReceiveBatch::new(port.clone())
                .map(macswap)
                .send(port.clone())
        })
        .collect();

    println!("Running {} pipelines", pipelines.len());
    for pipeline in pipelines {
        sched.add_task(pipeline).unwrap();
    }
}

fn macswap(packet: RawPacket) -> Result<Ethernet> {
    assert!(packet.refcnt() == 1);
    let mut ethernet = packet.parse::<Ethernet>()?;
    ethernet.swap_addresses();
    Ok(ethernet)
}
```

In Action

```
#[check(IPV6_MIN_MTU = 1280)]
fn send_too_big {
.pre(box pkt {
    ingress_check! {
        input: pkt,
        order: [EthHdr=>Ipv6Hdr=>TcpHdr<Ipv6Hdr>],
        checks: [(payload_len[Ipv6Hdr], >, IPV6_MIN_MTU)]
    }})
...filter/map/group_by operations...
.post(box pkt {
    egress_check! {
        input: pkt,
        order:[EthHdr=>Ipv6Hdr=>Icmpv6PktTooBig<...>],
        checks:[(checksum[Icmpv6PktTooBig], neq, checksum[TcpHdr<Ipv6Hdr>]),
                (payload_len[Ipv6Hdr], ==, 1240),
                (src[Ipv6Hdr], ==, dst[Ipv6Hdr]),
                (dst[Ipv6Hdr], ==, src[Ipv6Hdr]),
                (.src[EthHdr], ==, .dst[EthHdr]),
                (.dst[EthHdr], ==, .src[EthHdr])]
        }})
```

In Action

```
#[ check (IPV6_MIN_MTU = 1280)]
fn send_too_big {
.pre(box pkt {
    ingress_check! {
        input: pkt,
        order : [EthHdr=>Ipv6Hdr=>TcpHdr<Ipv6Hdr>],
        checks: [( payload_len[Ipv6Hdr] , >, IPV6_MIN_MTU )]
    }})
...filter/map/group_by operations...
.post(box pkt {
    egress_check! {
        input: pkt,
        order :[EthHdr=>Ipv6Hdr=>Icmpv6PktTooBig<...>],
        checks:[( checksum[Icmpv6PktTooBig] , neq, checksum[TcpHdr<Ipv6Hdr>] ),
                ( payload_len[Ipv6Hdr] , ==, 1240 ),
                ( src[Ipv6Hdr] , ==, dst[Ipv6Hdr] ),
                ( dst[Ipv6Hdr] , ==, src[Ipv6Hdr] ),
                ( .src[EthHdr] , ==, .dst[EthHdr] ),
                ( .dst[EthHdr] , ==, .src[EthHdr] )]
        }})
```

order is checked statically via a trace of packet contents

In Action

```
#[check(IPV6_MIN_MTU = 1280)]
fn send_too_big {
.pre(box pkt {
    ingress_check! {
        input: pkt,
        order: [EthHdr=>Ipv6Hdr=>TcpHdr<Ipv6Hdr>],
        checks: [(payload_len[Ipv6Hdr], >, IPV6_MIN_MTU)]
    }})
...filter/map/group_by operations...
.post(box pkt {
    egress_check! {
        input: pkt,
        order:[EthHdr=>Ipv6Hdr=>Icmpv6PktTooBig<...>],
        checks:[(checksum[Icmpv6PktTooBig], neq, checksum[TcpHdr<Ipv6Hdr>]),
                (payload_len[Ipv6Hdr], ==, 1240),
                (src[Ipv6Hdr], ==, dst[Ipv6Hdr]),
                (dst[Ipv6Hdr], ==, src[Ipv6Hdr]),
                (.src[EthHdr], ==, .dst[EthHdr]),
                (.dst[EthHdr], ==, .src[EthHdr])]
        }})
```

order is checked statically via a trace of packet contents

pre-checks validate incoming contents and store contents @ runtime

In Action

```
#[ check (IPV6_MIN_MTU = 1280)]
fn send_too_big {
.pre(box pkt {
    ingress_check! {
        input: pkt,
        order : [EthHdr=>Ipv6Hdr=>TcpHdr<Ipv6Hdr>],
        checks: [( payload_len[Ipv6Hdr] , >, IPV6_MIN_MTU )]
    }})
...filter/map/group_by operations...
.post(box pkt {
    egress_check! {
        input: pkt,
        order :[EthHdr=>Ipv6Hdr=>Icmpv6PktTooBig<...>],
        checks:[( checksum[Icmpv6PktTooBig] , neq, checksum[TcpHdr<Ipv6Hdr>] ),
                ( payload_len[Ipv6Hdr] , ==, 1240 ),
                ( src[Ipv6Hdr] , ==, dst[Ipv6Hdr] ),
                ( dst[Ipv6Hdr] , ==, src[Ipv6Hdr] ),
                ( .src[EthHdr] , ==, .dst[EthHdr] ),
                ( .dst[EthHdr] , ==, .src[EthHdr] )]
        }})
```

order is checked statically via a trace of packet contents

pre-checks validate incoming contents and store contents @ runtime

post-checks validate transformed contents against pre-check contents

Evaluation

Setup In our experimental setup, we ran NetBricks within an Ubuntu Docker container on a local VirtualBox VM. NetBricks uses DPDK [29] for fast packet I/O, which we have properly set up within the VM and container. We used MoonGen [10] to generate varying packet captures (pcaps) for our testing and evaluation harness. We looked at three factors in evaluating our technique for the design of NFs: (*i.*) **additional syntax** (*LoC*—lines of code); (*ii.*) **compilation-time** added to our two example NFs; (*iii.*) and **runtime overhead** of ingress and egress contract generation.

Evaluation

Design Phase

Setup In our experimental setup, we ran NetBricks within an Ubuntu Docker container on a local VirtualBox VM. NetBricks uses DPDK [29] for fast packet I/O, which we have properly set up within the VM and container. We used MoonGen [10] to generate varying packet captures (pcaps) for our testing and evaluation harness. We looked at three factors in evaluating our technique for the design of NFs: (*i.*) **additional syntax** (*LoC*—lines of code); (*ii.*) **compilation-time** added to our two example NFs; (*iii.*) and **runtime overhead** of ingress and egress contract generation.

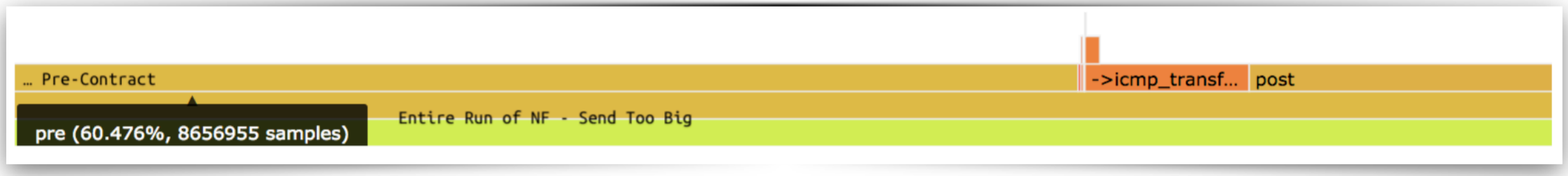
Evaluation: Syntax Added

LoC run	lang	files	lines	code
mtu-too-big: Contracts ON	rust	2	214	183
mtu-too-big: Contracts OFF	rust	2	189	158
mtu-too-big: Contracts ON	toml	1	19	16
mtu-too-big: Contracts OFF	toml	1	16	13
mtu-too-big: Contracts ON	total	3	233	199
mtu-too-big: Contracts OFF	total	3	205	171
Change		0	+28	+28

Evaluation: Compilation Time

compile times / cargo build	example	mean (s)	stddev (s)	user (s)	system (s)	min (s)	max (s)
Contracts - Off	srv6-change-pkt	26.039	3.286	0.631	10.715	22.330	33.230
Contracts - On	srv6-change-pkt	25.099	2.398	0.549	11.697	20.238	28.220
Effect		**-0.94**	**-0.888**	**-0.082**	**+0.982**	**-2.092**	**-5.01**
Contracts - Off	mtu-too-big	21.652	2.202	0.537	9.201	18.528	25.191
Contracts - On	mtu-too-big	26.052	1.858	0.650	10.851	22.165	28.346
Effect		**+4.4**	**-0.344**	**+0.113**	**+1.65**	**+3.637**	**+3.155**

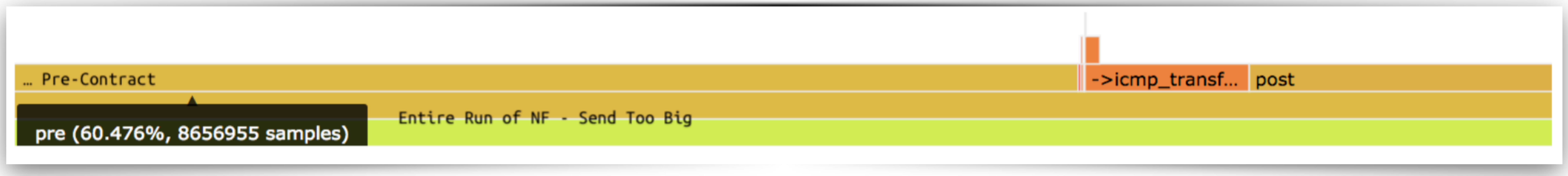
Evaluation: Runtime Cost



Due to:

- mirroring and tracing packet contents
- runtime checks
- storage overhead

Evaluation: Runtime Cost



Design Phase

Due to:

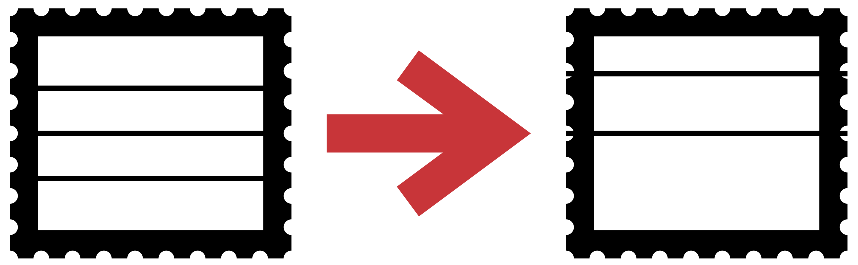
- mirroring and tracing packet contents
- runtime checks
- storage overhead

Future Work

- deployment models / running contracts in simulation / CI
  - e.g. via Mininet / Containernet
- (further) leverage static analysis of input programs
- interactive feedback (many examples in UI tooling and langs like Elm and Rust)
  - program slicing
  - refinement via domain-specific heuristics and constraint solving

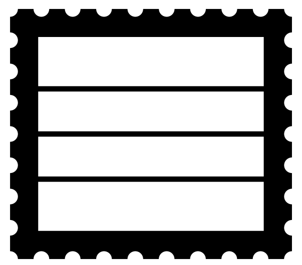
In **Practice**

Scoped Side Effects



- *cascade*
  - packet length
  - checksum
  - etc...

Typed Packets



$Envelope : T < T : IpPacket >$
$Header : TCP$

$\lambda \to \lambda$

Takeaways

- we need better approaches to **VERIFY** and **INTERACT** with network functions and packet processing program properties
- here, we provide a **HYBRID-APPROACH** and implementation for **GRADUALLY** checking and validating the arbitrary logic and side effects by
  - **COMBINING** design by contract, static assertions and type-checking, and code generation via macros
  - all without **PENALIZING** programmers at development time