Redundant Logic Elimination in Network Functions

Bangwen Deng[1], Wenfei Wu[1], Linhai Song[2]

1: Tsinghua University 2: The Pennsylvania State University

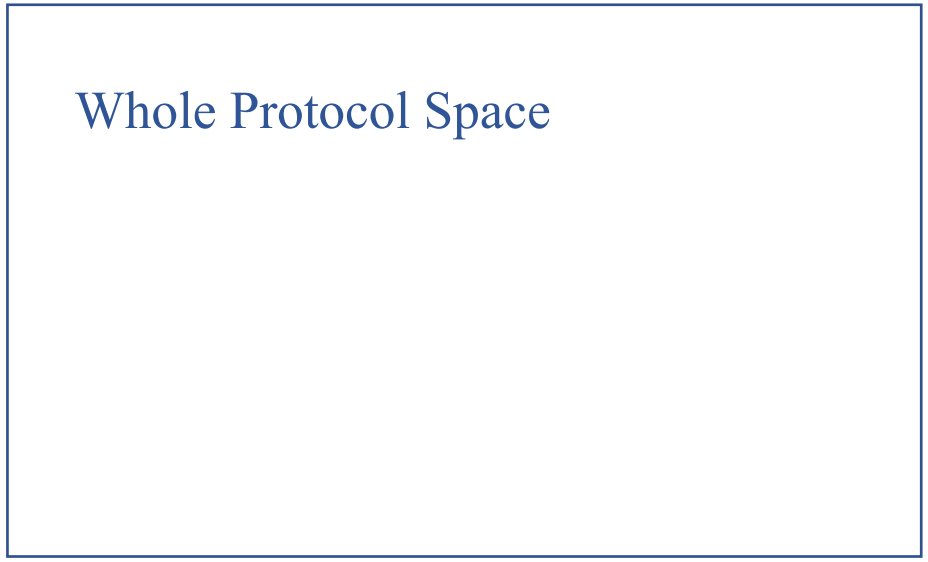
Network Functions: Critical components in network

- Growing impact:
  - Various network scenarios
  - Diverse functions (e.g. , Firewall, NAT, IDS, Load Balancer)

- NF's efficiency in flow processing is critical:
  - Affects network's end-to-end performance in a significant way (e.g., latency accumulation, throughput bottleneck)

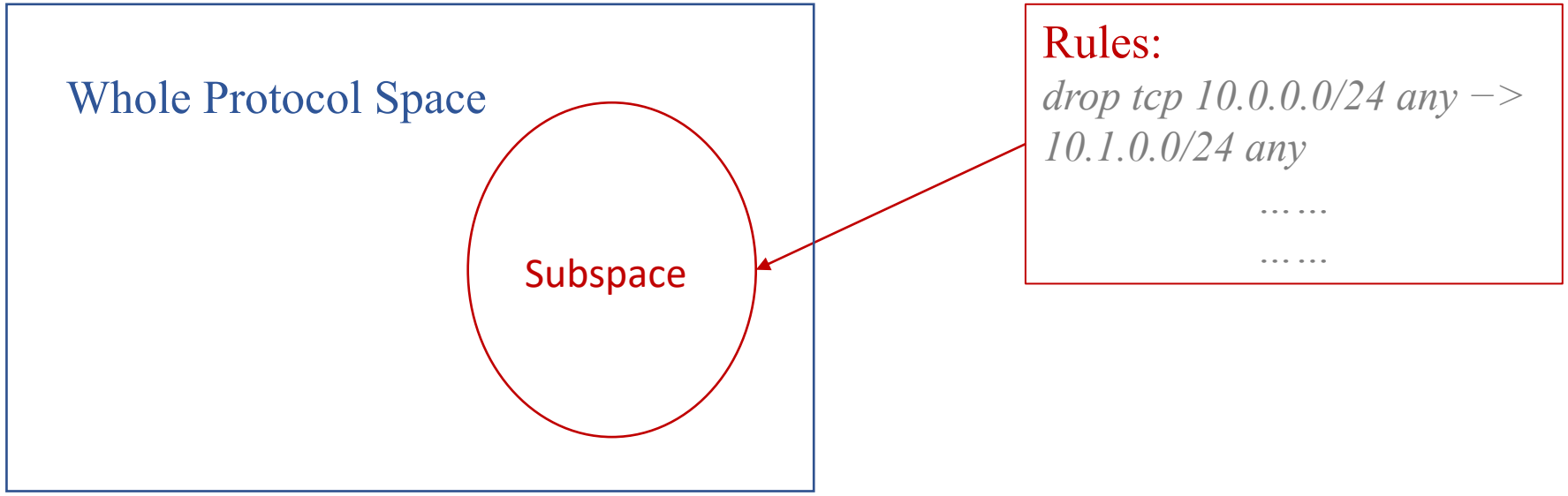
Network Functions: Critical components in network

- Mismatch of the protocol space in the development and that in the deployment leads to redundant logic:
  - Covering a large protocol space in development
  - Configuring a subspace of the entire protocol space in deployment



Network Functions: Critical components in network

- Mismatch of the protocol space in the development and that in the deployment leads to redundant logic:
  - Covering a large protocol space in development
  - Configuring a subspace of the entire protocol space in deployment



Network Functions: Critical components in network

- Mismatch of the protocol space in the development and that in the deployment leads to redundant logic:
  - Covering a large protocol space in development
  - Configuring a subspace of the entire protocol space in deployment

Goal:

To use compiler techniques to optimize away the redundancy.

Outline

- *Introduction*
- Design Intuition
- NFReducer Implementation
- Preliminary Evaluation
- Conclusion

Snort IDS Code(Simplified)

```
1  /* One example Snort rule:
2  drop tcp 10.0.0.0/24 any -> 10.1.0.0/24 any
3  */
4  struct {
5      unsigned long sip, dip;
6      unsigned short sport, dport;
7      ...
8  } net;
9  void main() {
10     LoadRules();
11     while(1) {
12         pkt = ... // get a packet
13         DecodeEthPkt(pkt); // decode a packet
14         ApplyRules(); // match rules
15 } }
16 void DecodeEthPkt(u_char *pkt) {
17     DecodeIPPkt(pkt);
18 }
19 void DecodeIPPkt(u_char *pkt) {
20     net.dip = ...
21     net.sip = ...
22     net.protocol = ...
23     log(net.sip, net.dip, net.protocol);
24     if (net.protocol == TCP)
25         DecodeTCPPkt(pkt);
26     else if(net.protocol == UDP)
27         DecodeUDPPkt(pkt);
28     else if(...) { ... }
29 }
```

```
30 void DecodeTCPPkt(u_char *pkt) {
31     net.dport = ...
32     net.sport = ...
33     log(net.sport, net.dport);
34 }
35 void DecodeUDPPkt(u_char *pkt) {
36     net.dport = ...
37     net.sport = ...
38     log(net.sport, net.dport);
39 }
40 void ApplyRules() {
41     while(...) { //iterate each rule r
42         if(MatchRule(r)) {
43             Action();
44             return;
45 } } }
46 int MatchRule(Rule *r){
47     if(r->sip != net.sip) return 0;
48     if(r->dip != net.dip) return 0;
49     if(r->protocol != net.protocol) return 0;
50     if(r->sport != net.sport) return 0;
51     if(r->dport != net.dport) return 0;
52     return 1;
53 }
```

Snort IDS Code(Simplified)

```
1  /* One example Snort rule:
2  drop tcp 10.0.0.0/24 any -> 10.1.0.0/24 any
3  */
4  struct {
5      unsigned long sip, dip;
6      unsigned short sport, dport;
7      ...
8  } net;
9  void main() {
10     LoadRules();
11     while(1) {
12         pkt = ... // get a packet
13         DecodeEthPkt(pkt); // decode a packet
14         ApplyRules(); // match rules
15 } }
16 void DecodeEthPkt(u_char *pkt) {
17     DecodeIPPkt(pkt);
18 }
19 void DecodeIPPkt(u_char *pkt) {
20     net.dip = ...
21     net.sip = ...
22     net.protocol = ...
23     log(net.sip, net.dip, net.protocol);
24     if (net.protocol == TCP)
25         DecodeTCPPkt(pkt);
26     else if(net.protocol == UDP)
27         DecodeUDPPkt(pkt);
28     else if(...) { ... }
29 }
```

```
30 void DecodeTCPPkt(u_char *pkt) {
31     net.dport = ...
32     net.sport = ...
33     log(net.sport, net.dport);
34 }
35 void DecodeUDPPkt(u_char *pkt) {
36     net.dport = ...
37     net.sport = ...
38     log(net.sport, net.dport);
39 }
40 void ApplyRules() {
41     while(...) { //iterate each rule r
42         if(MatchRule(r)) {
43             Action();
44             return;
45 } } }
46 int MatchRule(Rule *r){
47     if(r->sip != net.sip) return 0;
48     if(r->dip != net.dip) return 0;
49     if(r->protocol != net.protocol) return 0;
50     if(r->sport != net.sport) return 0;
51     if(r->dport != net.dport) return 0;
52     return 1;
53 }
```

Parsing

Snort IDS Code(Simplified)

```
1  /* One example Snort rule:
2  drop tcp 10.0.0.0/24 any -> 10.1.0.0/24 any
3  */
4  struct {
5      unsigned long sip, dip;
6      unsigned short sport, dport;
7      ...
8  } net;
9  void main() {
10     LoadRules();
11     while(1) {
12         pkt = ... // get a packet
13         DecodeEthPkt(pkt); // decode a packet
14         ApplyRules(); // match rules
15 }   }
16 void DecodeEthPkt(u_char *pkt) {
17     DecodeIPPkt(pkt);
18 }
19 void DecodeIPPkt(u_char *pkt) {
20     net.dip = ...
21     net.sip = ...
22     net.protocol = ...
23     log(net.sip, net.dip, net.protocol);
24     if (net.protocol == TCP)
25         DecodeTCPPkt(pkt);
26     else if(net.protocol == UDP)
27         DecodeUDPPkt(pkt);
28     else if(...) { ... }
29 }
```

```
30 void DecodeTCPPkt(u_char *pkt) {
31     net.dport = ...
32     net.sport = ...
33     log(net.sport, net.dport);
34 }
35 void DecodeUDPPkt(u_char *pkt) {
36     net.dport = ...
37     net.sport = ...
38     log(net.sport, net.dport);
39 }
40 void ApplyRules() {
41     while(...) { //iterate each rule r
42         if(MatchRule(r)) {
43             Action();
44             return;
45 }   }   }
46 int MatchRule(Rule *r){
47     if(r->sip != net.sip) return 0;
48     if(r->dip != net.dip) return 0;
49     if(r->protocol != net.protocol) return 0;
50     if(r->sport != net.sport) return 0;
51     if(r->dport != net.dport) return 0;
52     return 1;
53 }
```

Parsing → Match

Snort IDS Code(Simplified)

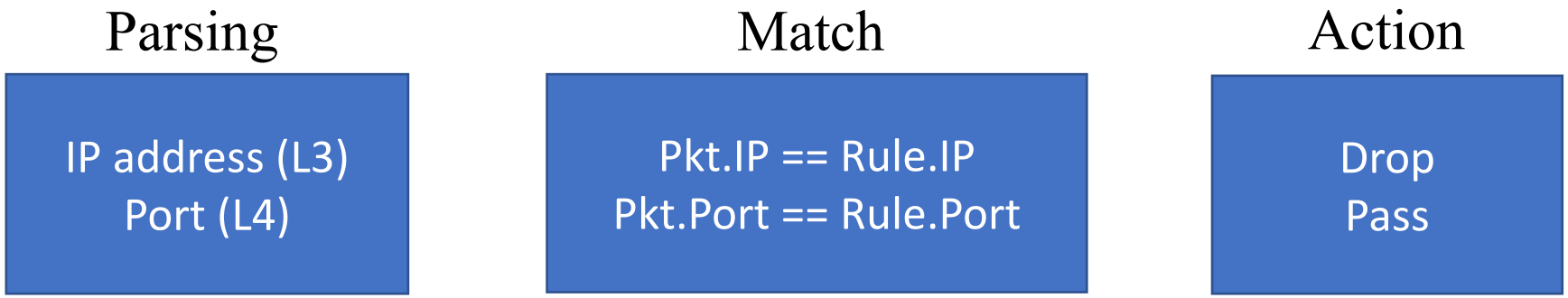
```
1  /* One example Snort rule:
2  drop tcp 10.0.0.0/24 any -> 10.1.0.0/24 any
3  */
4  struct {
5      unsigned long sip, dip;
6      unsigned short sport, dport;
7      ...
8  } net;
9  void main() {
10     LoadRules();
11     while(1) {
12         pkt = ... // get a packet
13         DecodeEthPkt(pkt); // decode a packet
14         ApplyRules(); // match rules
15 }   }
16 void DecodeEthPkt(u_char *pkt) {
17     DecodeIPPkt(pkt);
18 }
19 void DecodeIPPkt(u_char *pkt) {
20     net.dip = ...
21     net.sip = ...
22     net.protocol = ...
23     log(net.sip, net.dip, net.protocol);
24     if (net.protocol == TCP)
25         DecodeTCPPkt(pkt);
26     else if(net.protocol == UDP)
27         DecodeUDPPkt(pkt);
28     else if(...) { ... }
29 }
```

```
30 void DecodeTCPPkt(u_char *pkt) {
31     net.dport = ...
32     net.sport = ...
33     log(net.sport, net.dport);
34 }
35 void DecodeUDPPkt(u_char *pkt) {
36     net.dport = ...
37     net.sport = ...
38     log(net.sport, net.dport);
39 }
40 void ApplyRules() {
41     while(...) { //iterate each rule r
42         if(MatchRule(r)) {
43             Action();
44             return;
45 }   }   }
46 int MatchRule(Rule *r){
47     if(r->sip != net.sip) return 0;
48     if(r->dip != net.dip) return 0;
49     if(r->protocol != net.protocol) return 0;
50     if(r->sport != net.sport) return 0;
51     if(r->dport != net.dport) return 0;
52     return 1;
53 }
```

Parsing → Match → Action

Type-I Redundancy: Unused layer parsing

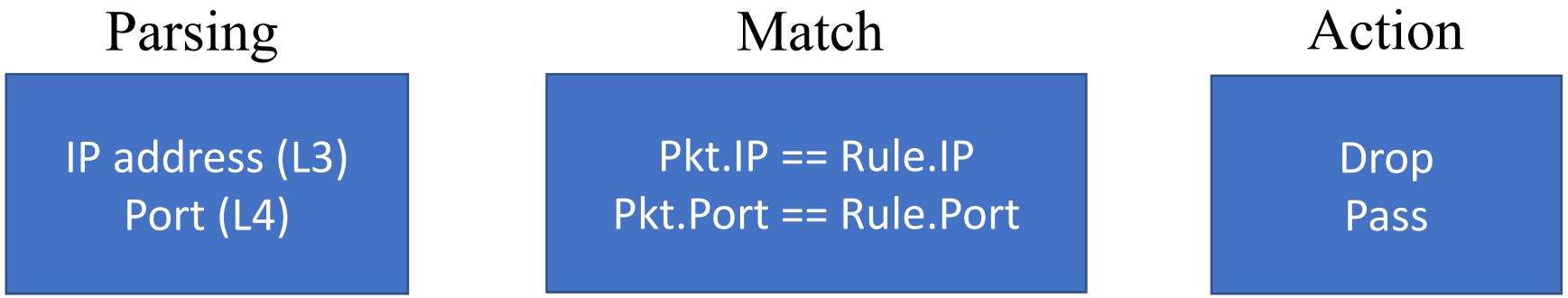
- Example



Type-I Redundancy: Unused layer parsing

- Example

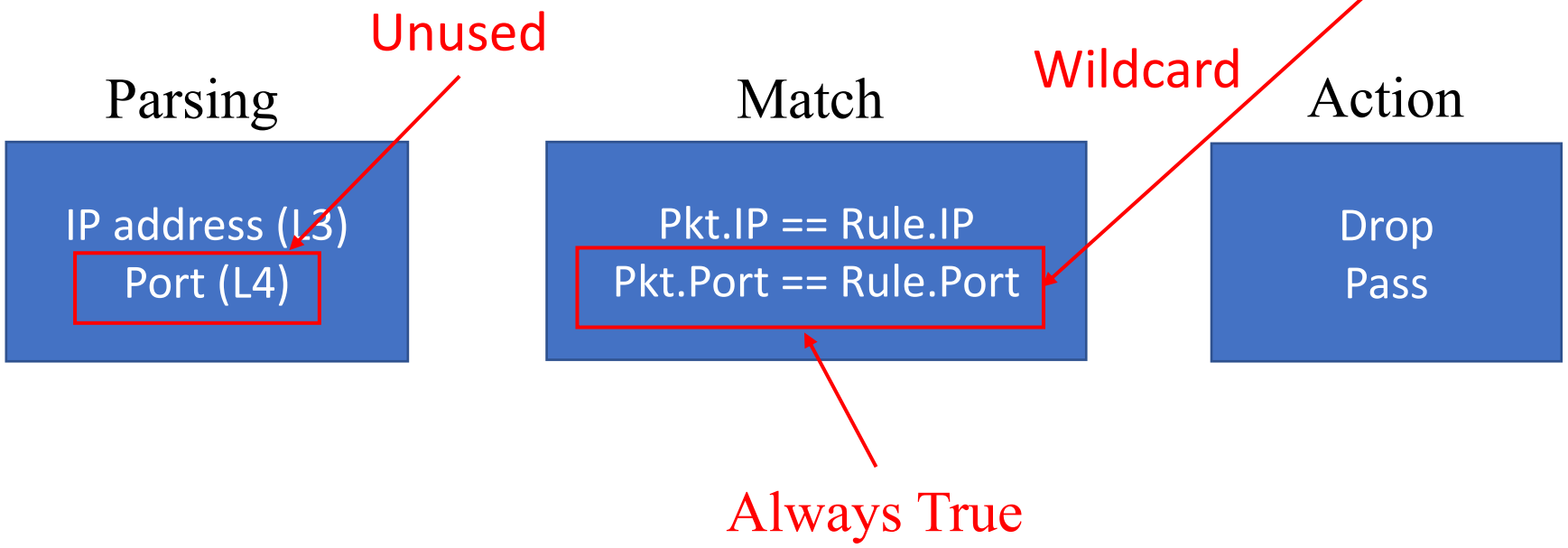
What if only L3 header is used? E.g., *<10.0.0.1->*, s/d port=*, drop>*



Type-I Redundancy: Unused layer parsing

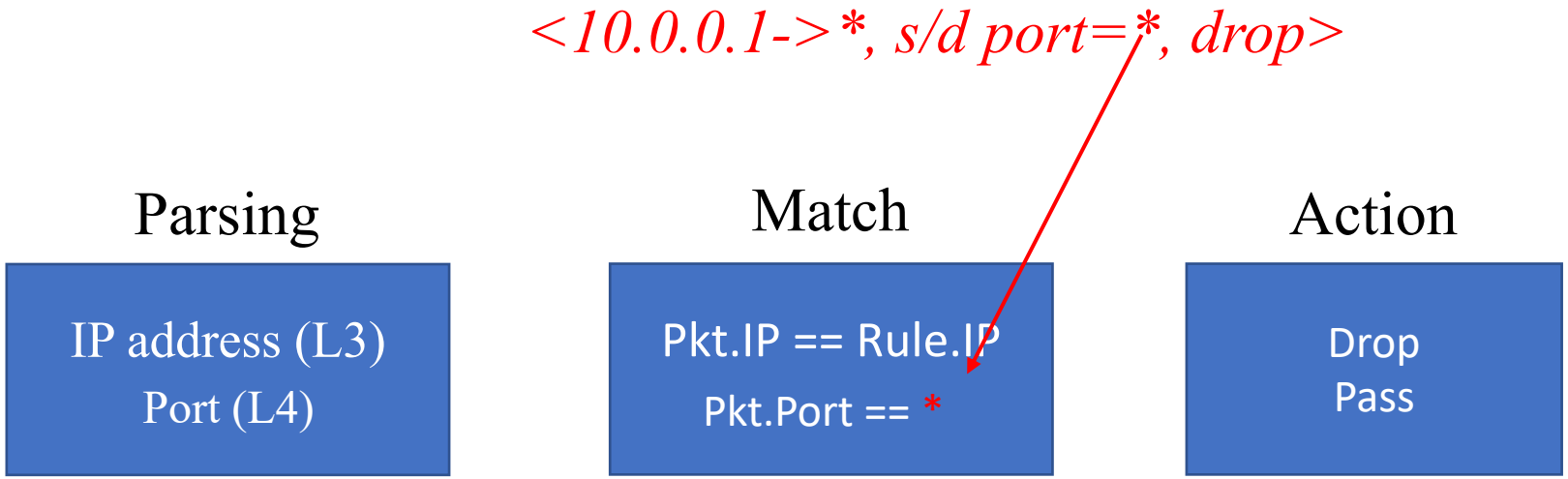
- Example

What if only L3 header is used? E.g., *<10.0.0.1->*, s/d port=*, drop>*



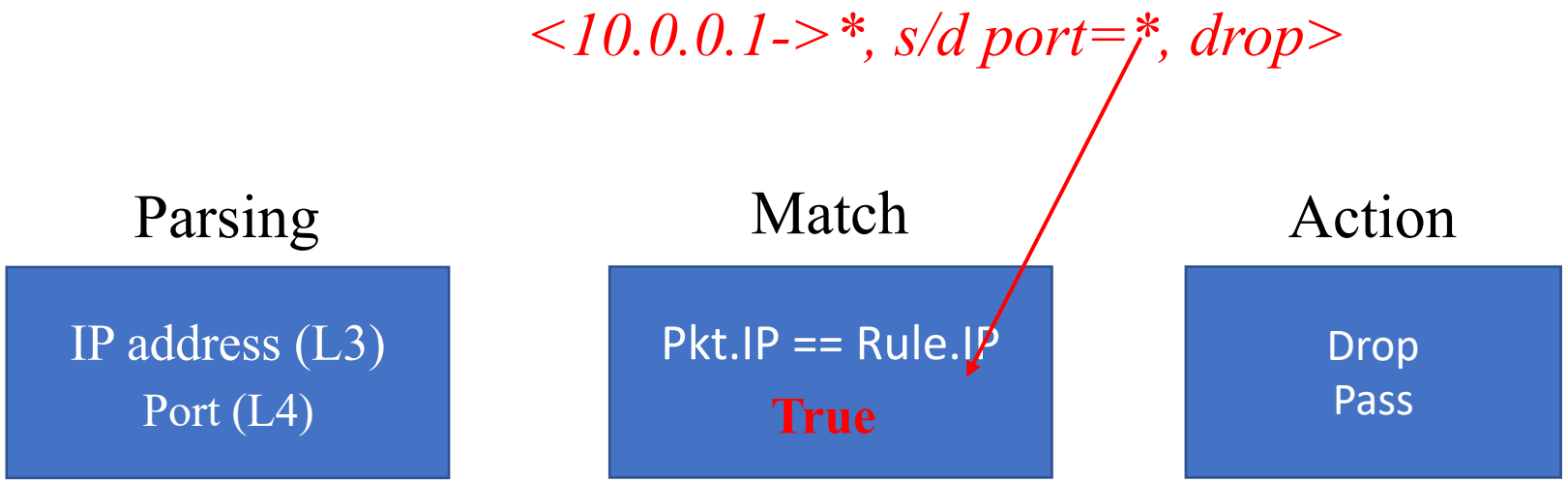
Type-I Redundancy: Method to Solve

- Apply Rules



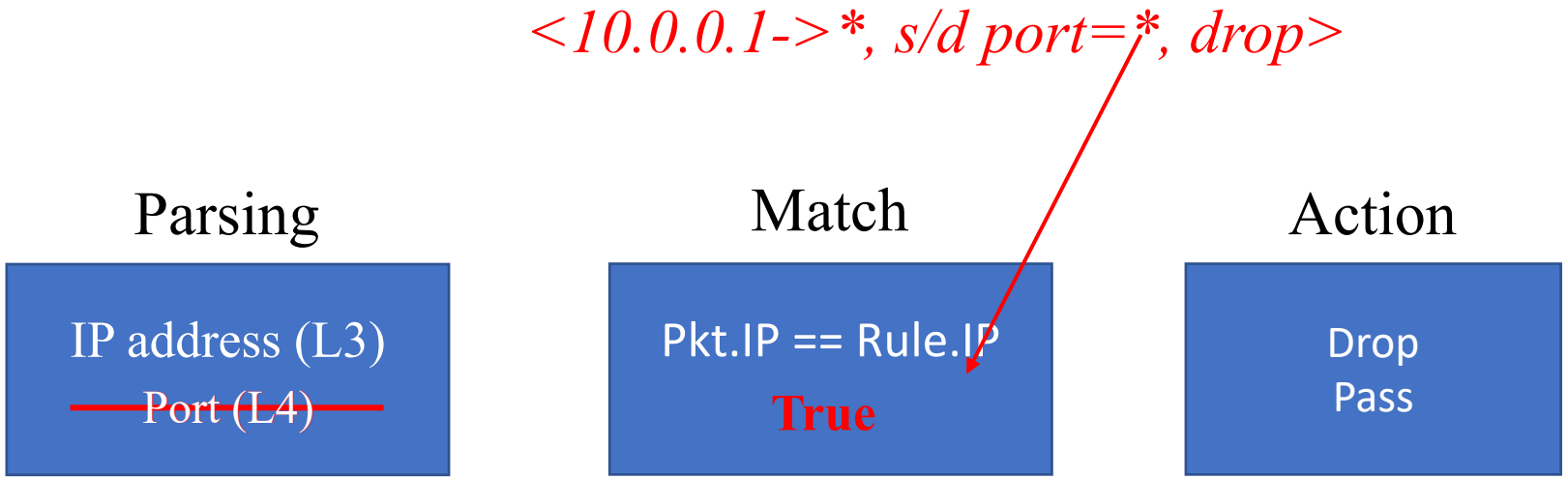
Type-I Redundancy: Method to Solve

- Apply Rules
- Constant Folding and Propagation



Type-I Redundancy: Method to Solve

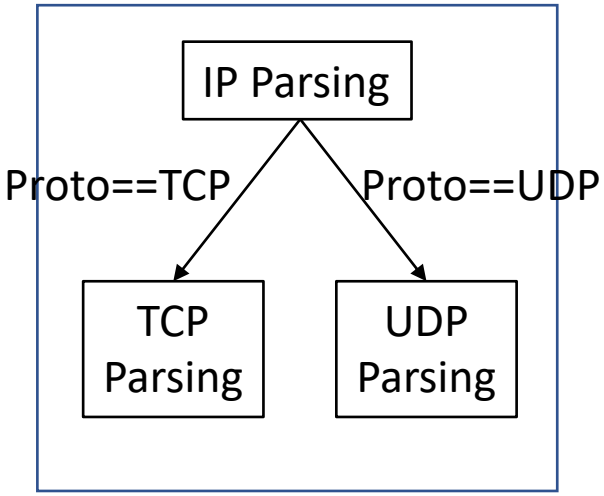
- Apply Rules
- Constant Folding and Propagation
- Dead Code Elimination



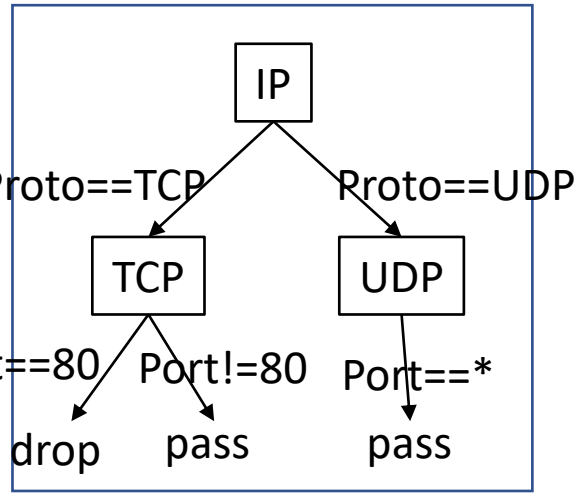
Type-II Redundancy: Unused Protocol (Branch) Parsing

- Branches in Parse and Match

If NF processes TCP packets only, E.g., <10.0.0.0/24, tcp, 80, drop>



Parsing

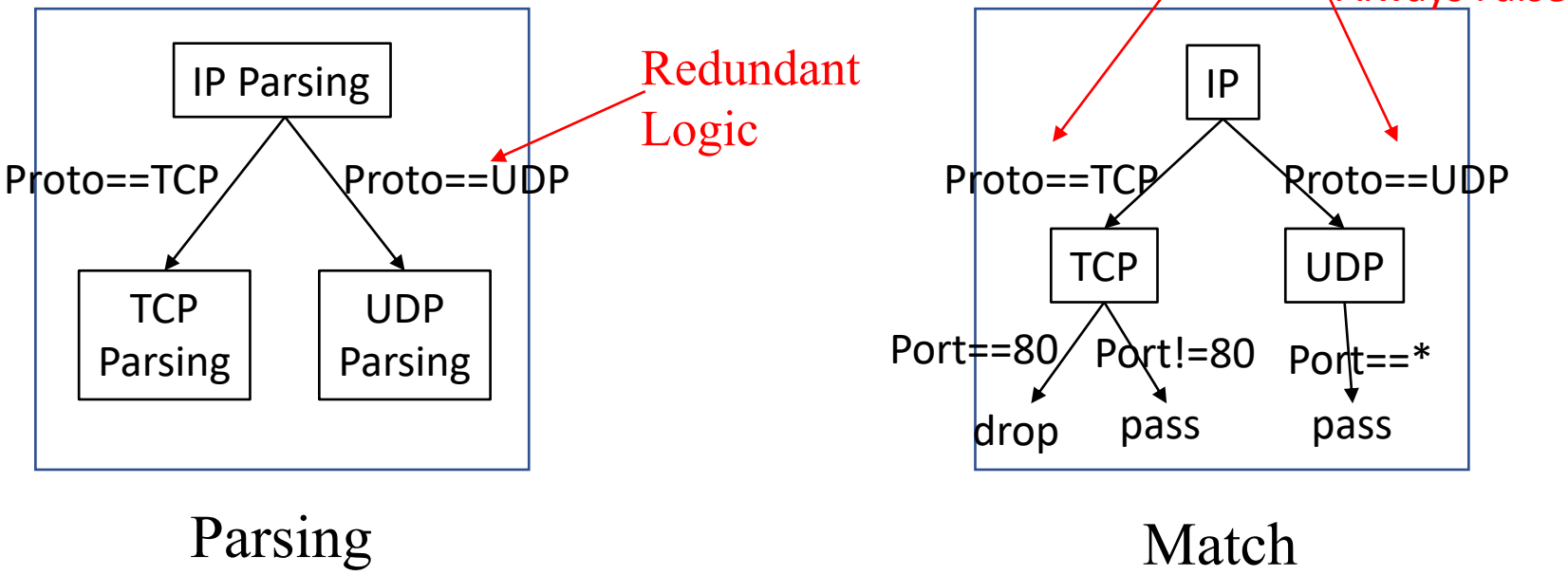


Match

Type-II Redundancy: Unused Protocol (Branch) Parsing

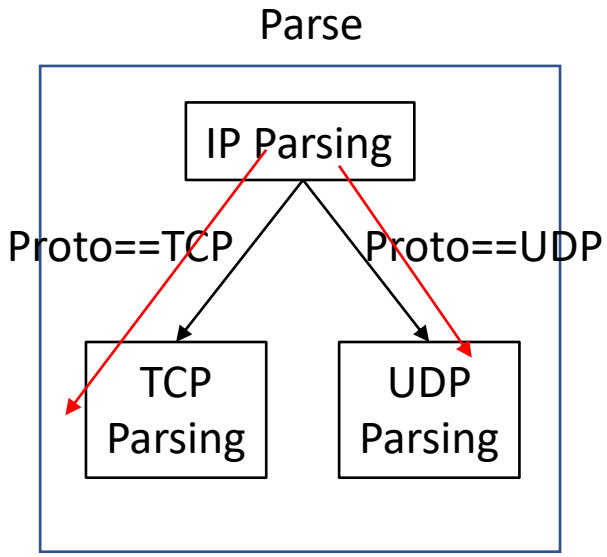
- Branches in Parse and Match

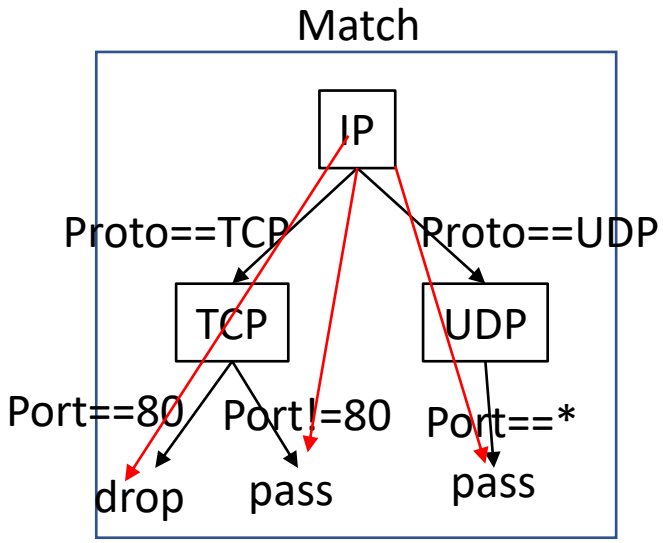
If NF processes TCP packets only, E.g., <10.0.0.0/24, tcp, 80, drop>



Type-II Redundancy: Method to Solve

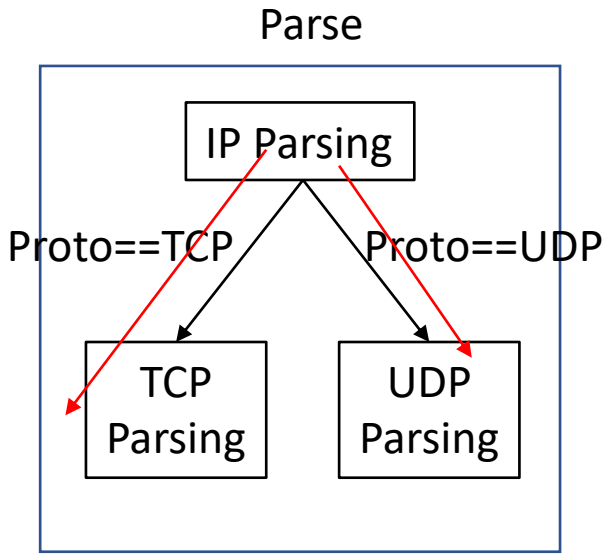
- Extract Feasible Execution Path

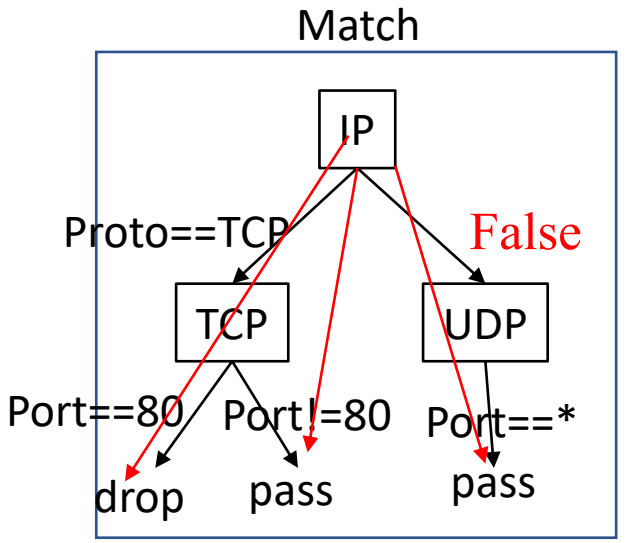




Type-II Redundancy: Method to Solve

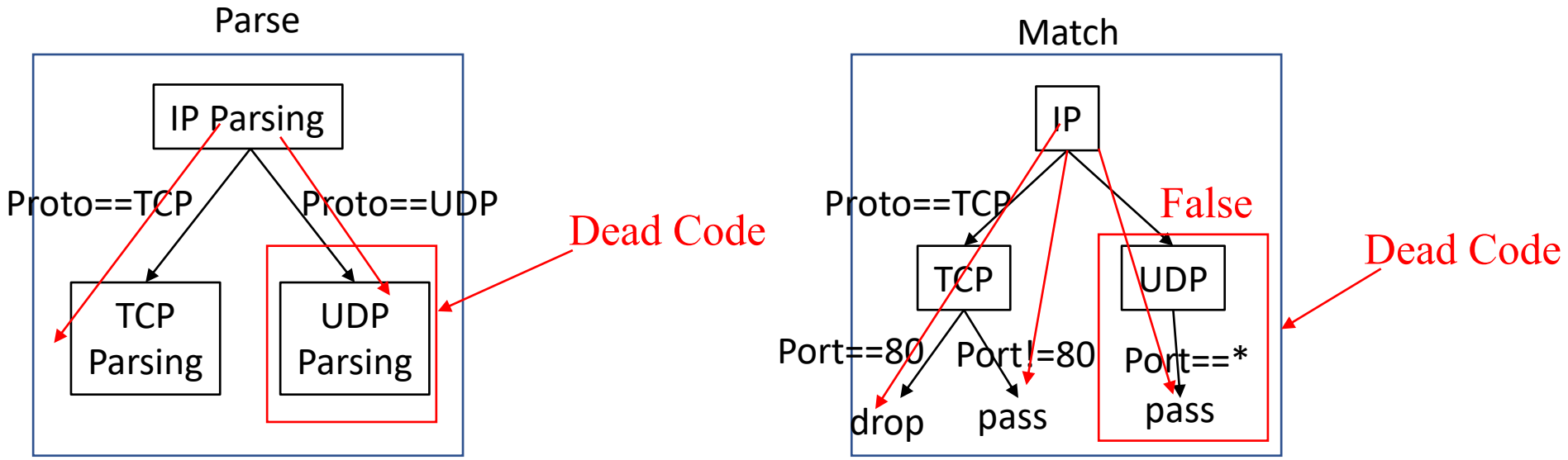
- Extract Feasible Execution Path
- Constant Folding and Propagation





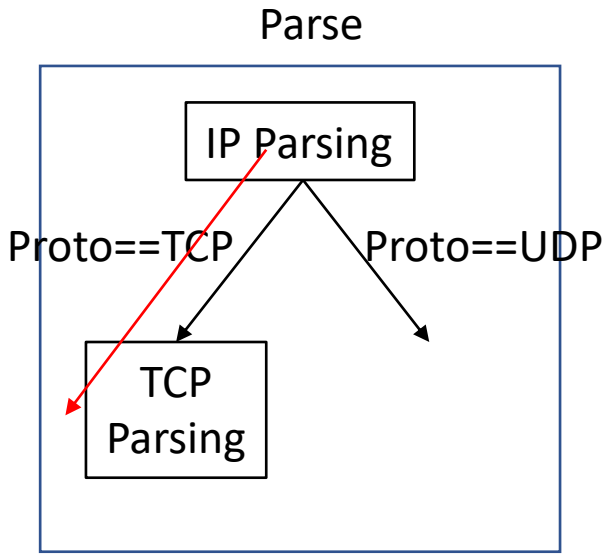
Type-II Redundancy: Method to Solve

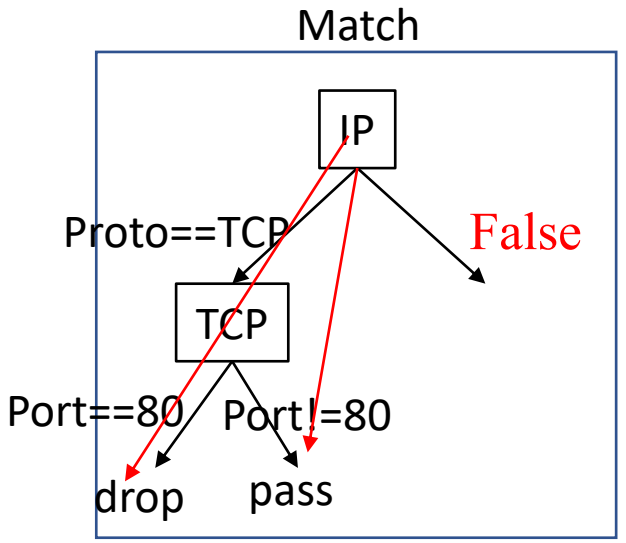
- Extract Feasible Execution Path
- Constant Folding and Propagation
- Dead Code Elimination



Type-II Redundancy: Method to Solve

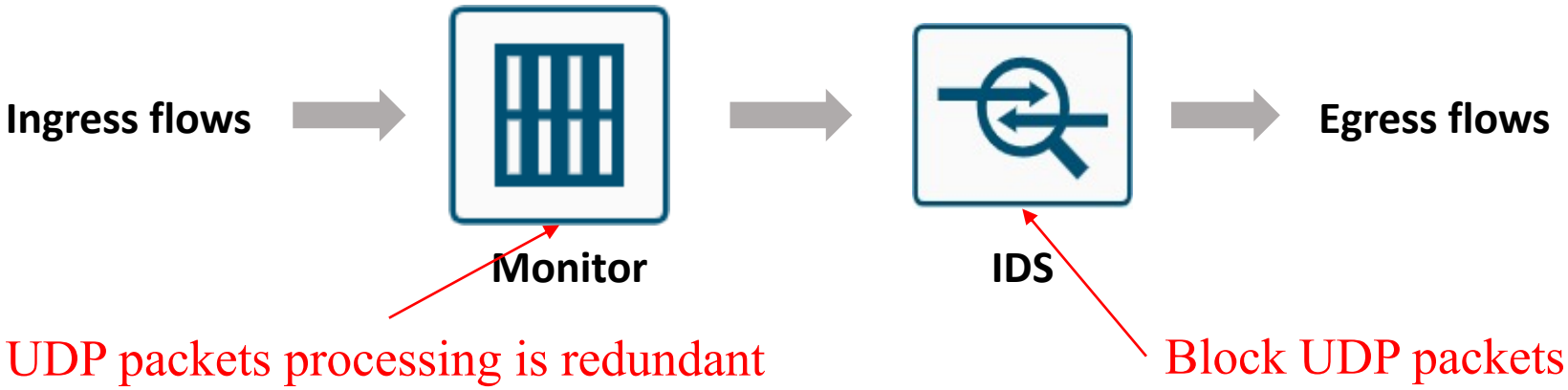
- Extract Feasible Execution Path
- Constant Folding and Propagation
- Dead Code Elimination





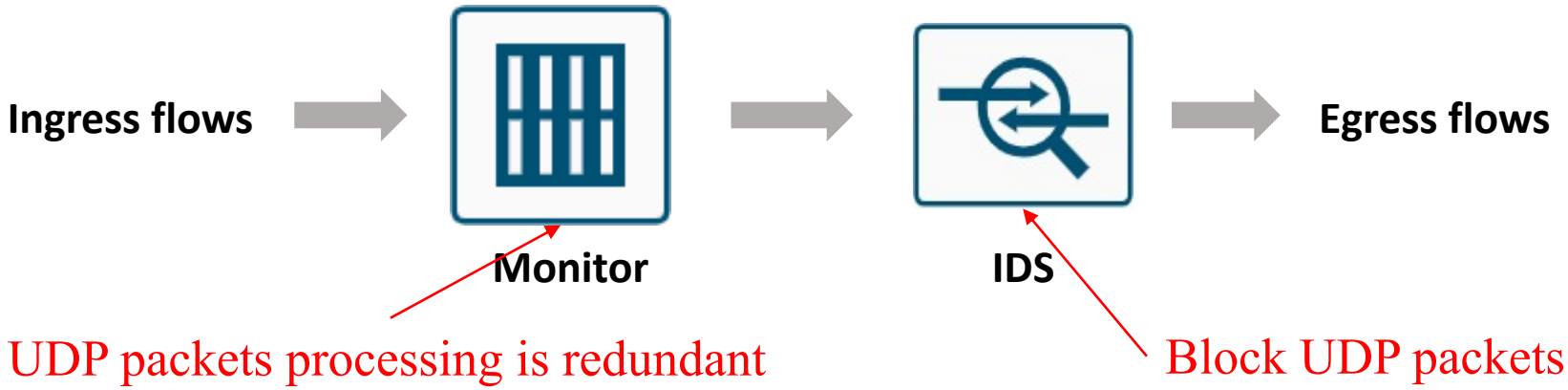
Type-III Redundancy: Cross-NF Redundancy

- If a monitor deployed before an IDS instance who blocks UDP packets, all the parsing and counting for UDP packets in the monitor is redundant.



Type-III Redundancy: Cross-NF Redundancy

- If a monitor deployed before an IDS instance who blocks UDP packets, all the parsing and counting for UDP packets in the monitor is redundant.

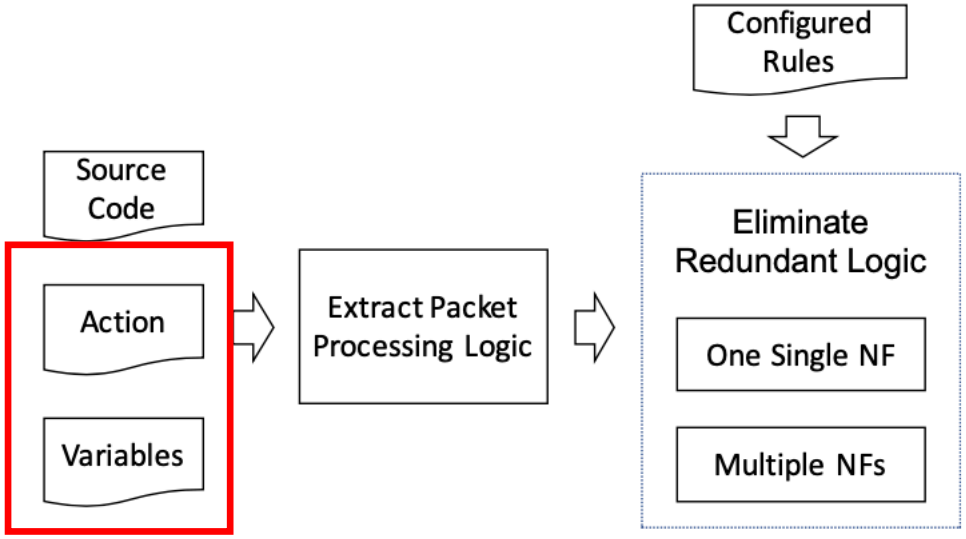


- *Method to Solve:*
  - Consolidate
  - Eliminate *type-I* and *type-II* redundancy
  - Decompose

Outline

- *Introduction*
- *Design Intuition*
- NFReducer Implementation
- Preliminary Evaluation
- Conclusion

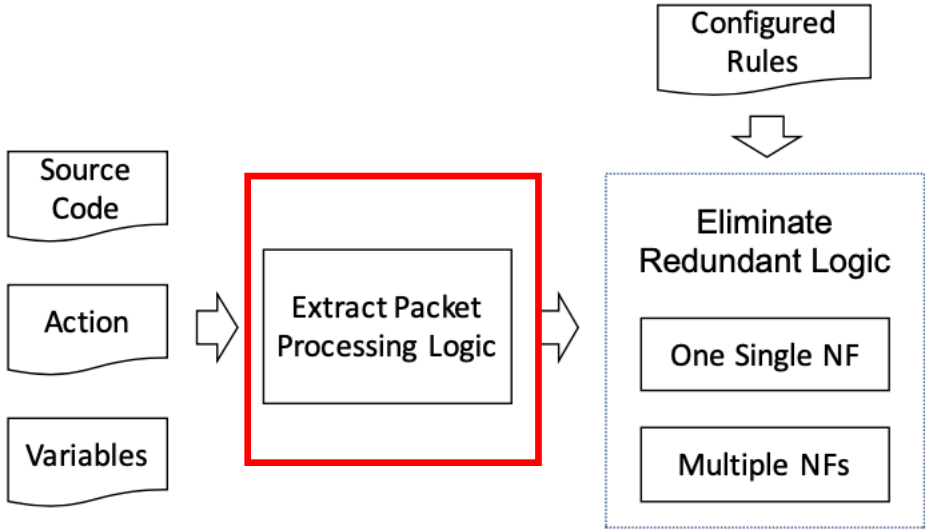
NFReducer Architecture



The architecture of NFReducer

- Labeling Critical Variables and Actions

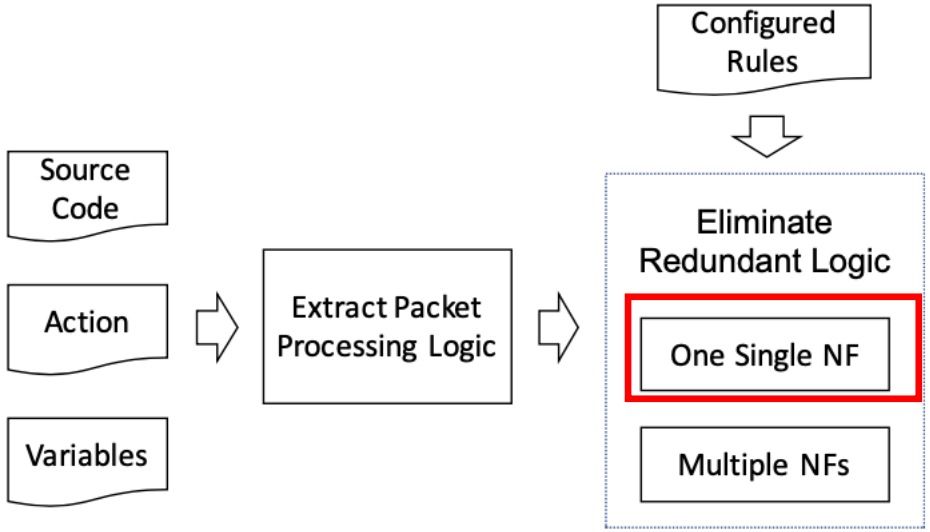
NFReducer Architecture



The architecture of NFReducer

- Labeling Critical Variables and Actions
- Extracting Packet Processing Logic

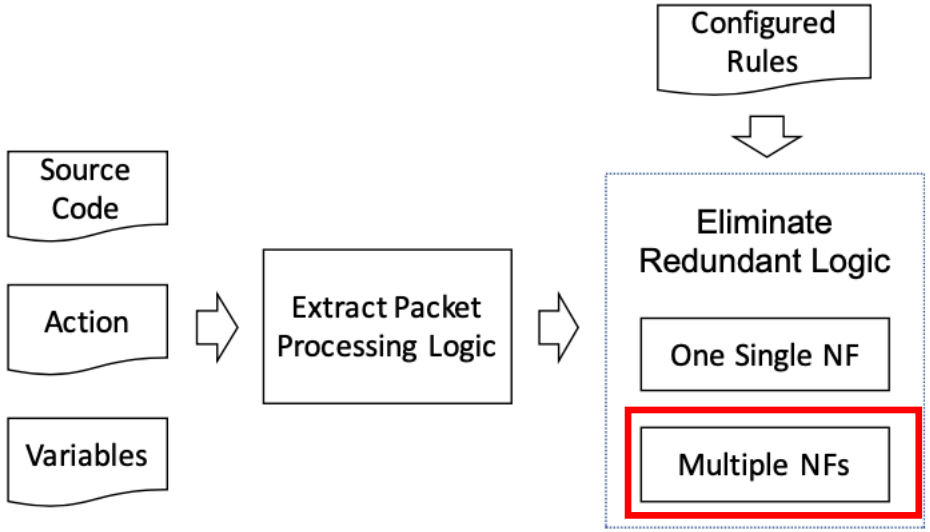
NFReducer Architecture



The architecture of NFReducer

- Labeling Critical Variables and Actions
- Extracting Packet Processing Logic
- Individual NF Optimization

NFReducer Architecture



The architecture of NFReducer

- Labeling Critical Variables and Actions
- Extracting Packet Processing Logic
- Individual NF Optimization
- Cross-NF Optimization

NFReducer Architecture

- Labeling Critical Variables and Actions
  - Critical Variables
    - Packet Variables: Holding the packet raw data.
    - State Variables: Maintaining the NF states. (e.g., counter)
    - Config Variables: Maintaining the config info. (e.g., rules)
  - NF Actions:
    - External Actions (e.g., replying, forward, drop packets)
    - Internal Actions (e.g., updating state variables)

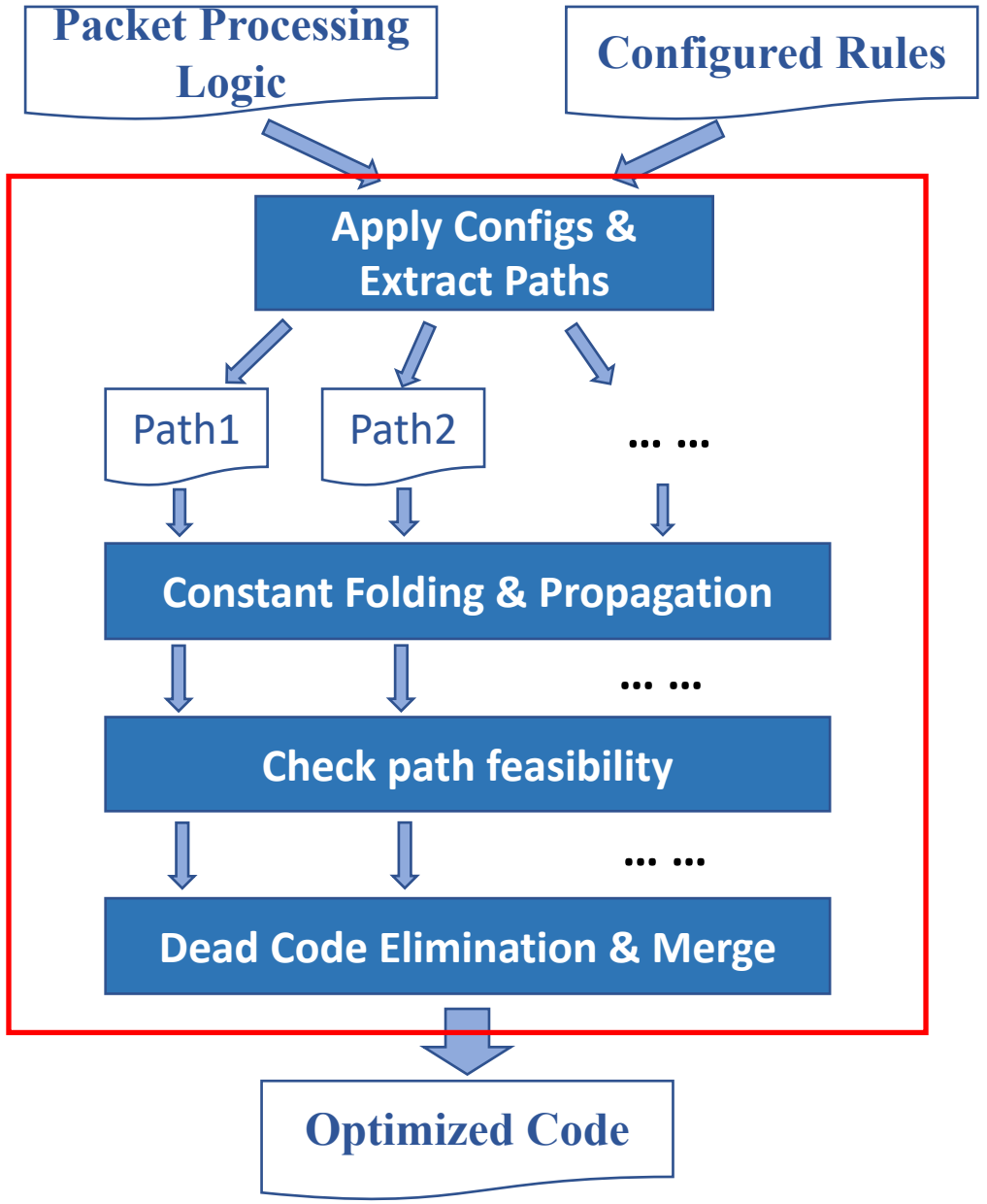
NFReducer Architecture

- Labeling Critical Variables and Actions

- Extracting Packet Processing Logic
  - Removing functionalities unrelated to packet processing (e.g., log).
  - Facilitate the compiler techniques applied later (e.g., symbolic execution).

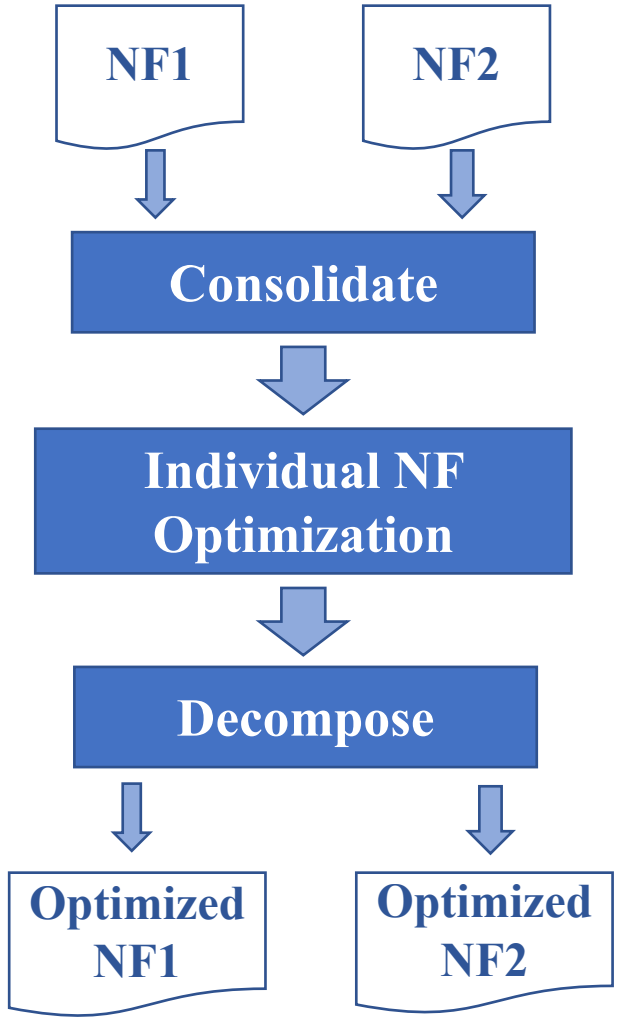
NFReducer Architecture

- Labeling Critical Variables and Actions
- Extracting Packet Processing Logic
- Individual NF Optimization
  - Apply Configs
  - Extract Paths
  - Constant Folding and Propagation
  - Check Path Feasibility
  - Dead Code Elimination

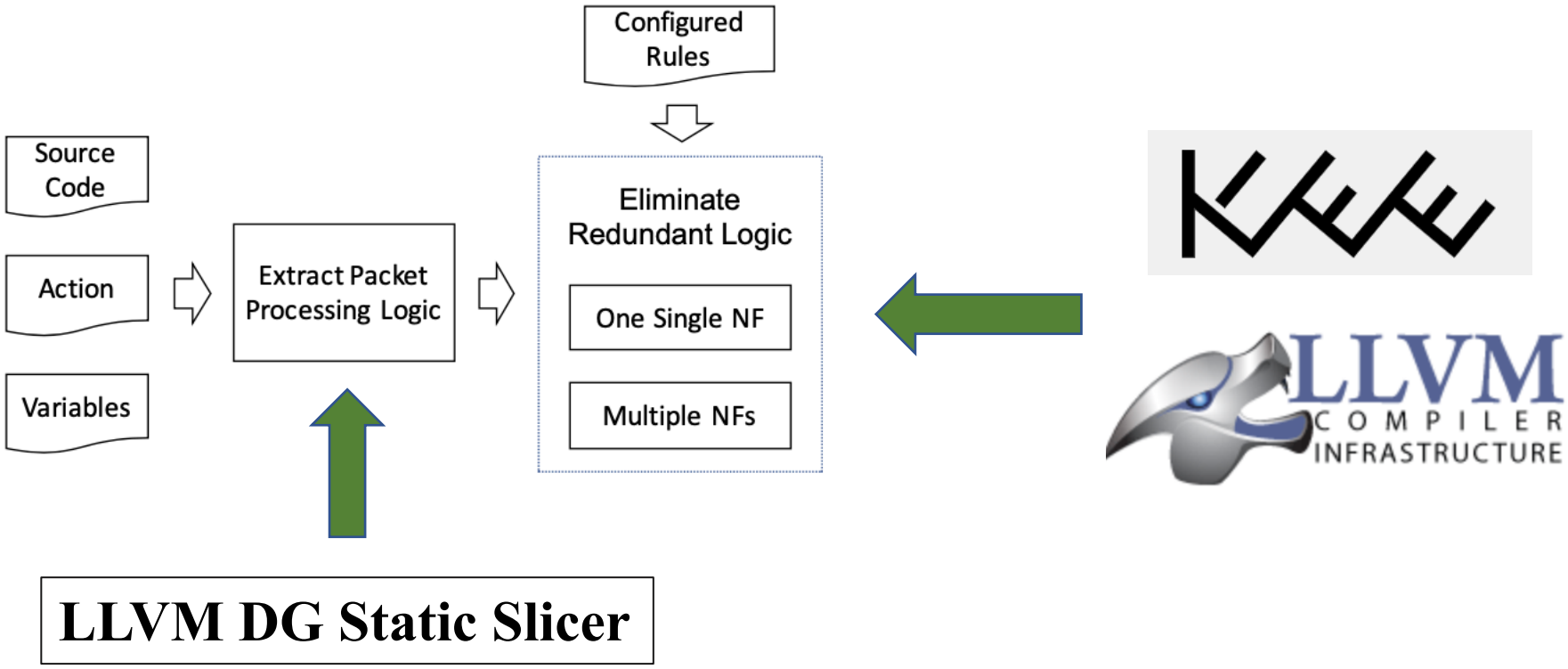


NFReducer Architecture

- Labeling Critical Variables and Actions
- Extracting Packet Processing Logic
- Individual NF Optimization
- Cross-NF Optimization
  - Preliminary discussion on the optimization of different NF chain execution models.



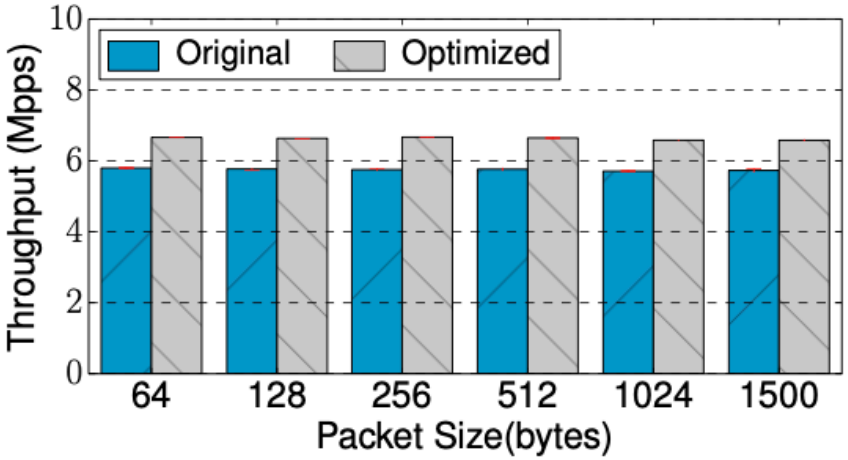
Implementation



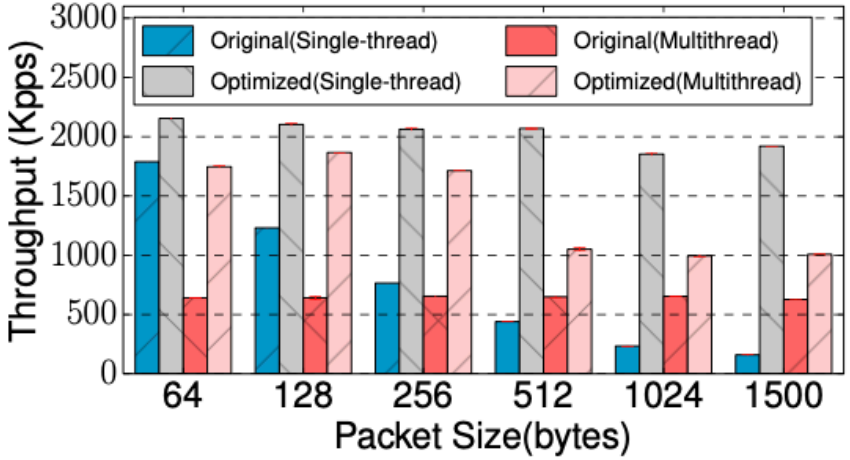
Outline

- *Introduction*
- *Design Intuition*
- *NFReducer Implementation*
- Preliminary Evaluation
- Conclusion

Evaluation: Eliminating Type-I Redundancy



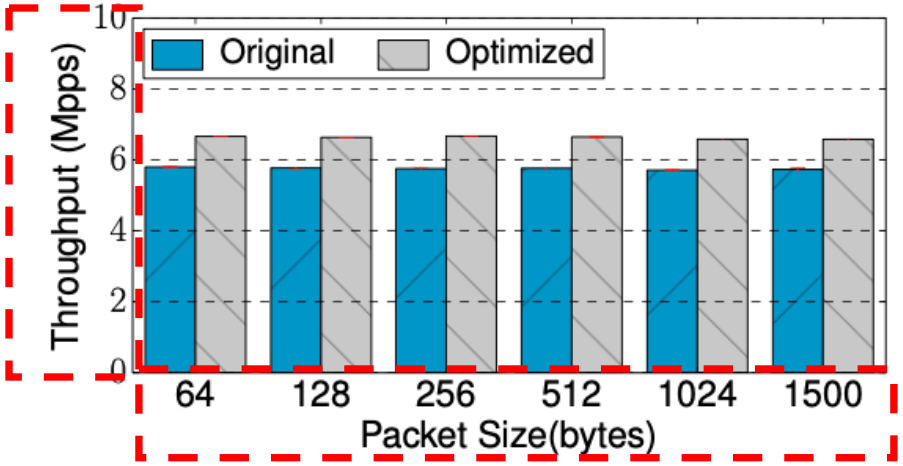
Throughput of Snort



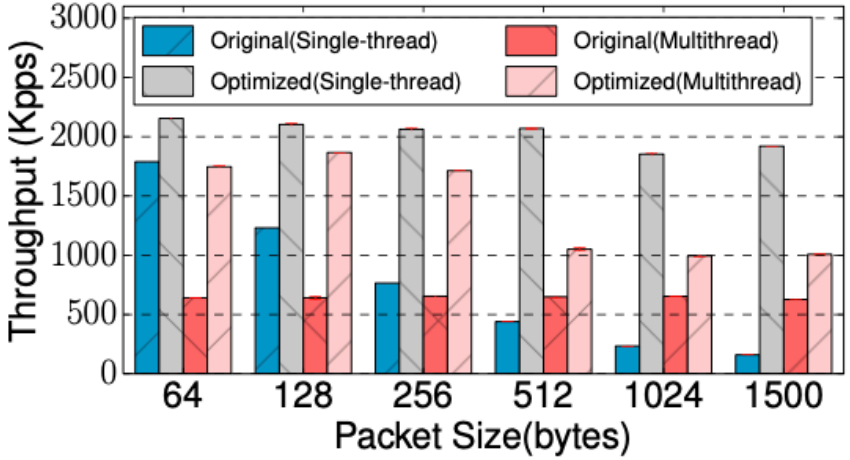
Throughput of Suricata

- Setting: Configured with layer-3 rules.
- Increase by nearly 15% for Snort and by 15% to 10X for Suricata (single thread).
- Suricata is more significant
  - inspects packets deeper in payload than Snort.

Evaluation: Eliminating Type-I Redundancy



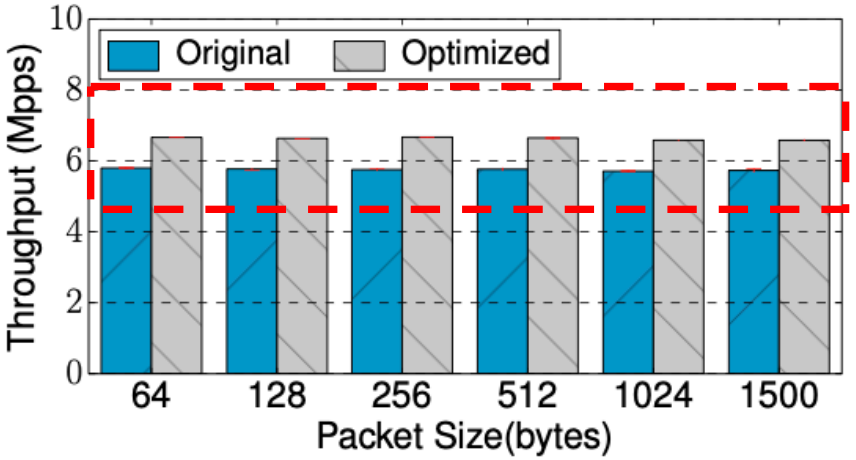
Throughput of Snort



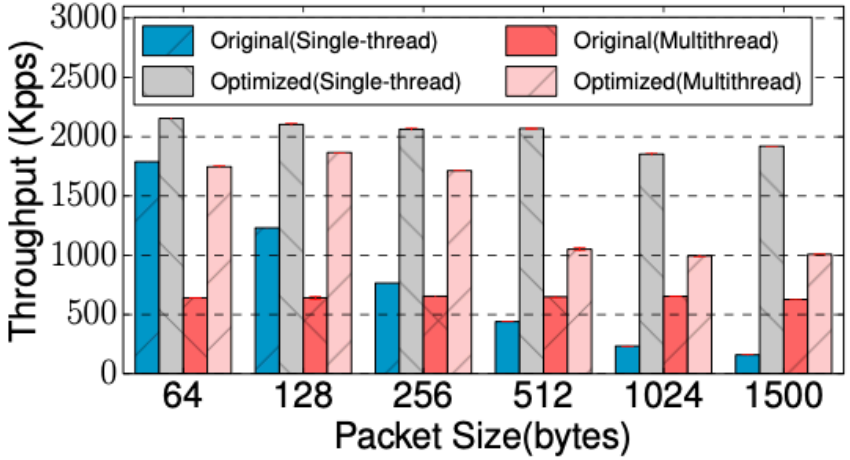
Throughput of Suricata

- Setting: Configured with layer-3 rules.
- Increase by nearly 15% for Snort and by 15% to 10X for Suricata (single thread).
- Suricata is more significant
  - inspects packets deeper in payload than Snort.

Evaluation: Eliminating Type-I Redundancy



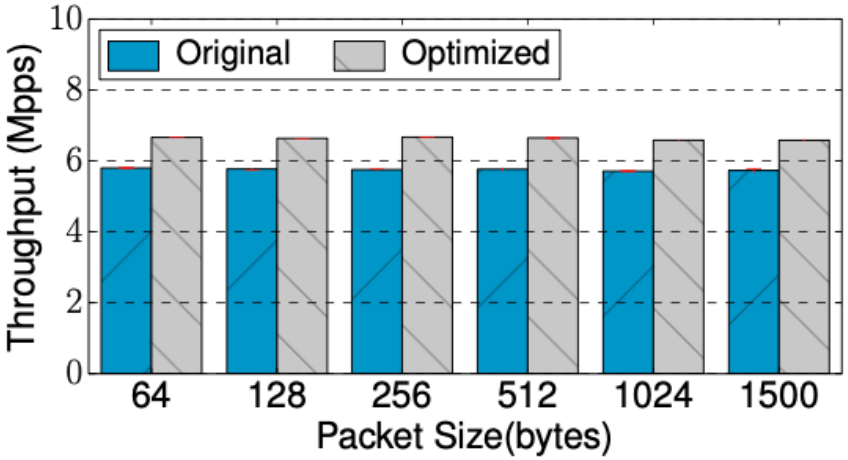
Throughput of Snort



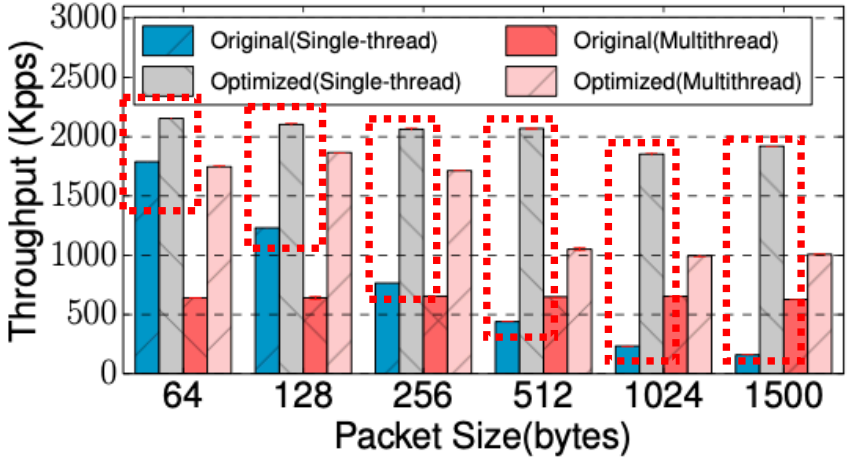
Throughput of Suricata

- Setting: Configured with layer-3 rules.
- Increase by nearly 15% for Snort and by 15% to 10X for Suricata (single thread).
- Suricata is more significant
  - inspects packets deeper in payload than Snort.

Evaluation: Eliminating Type-I Redundancy



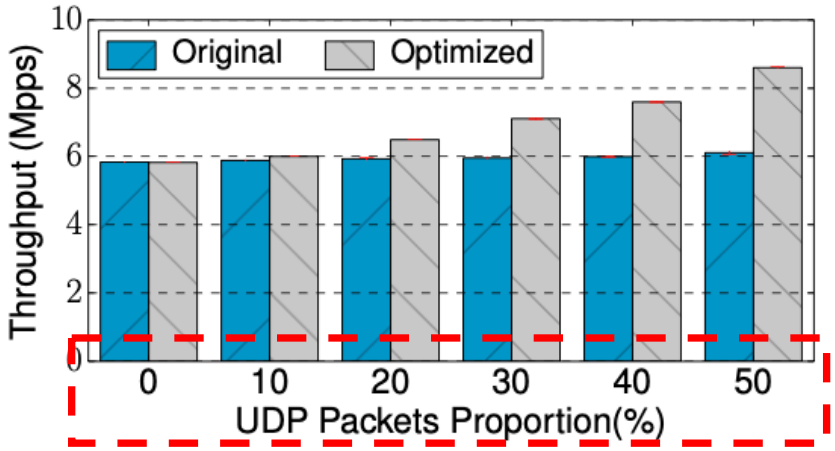
Throughput of Snort



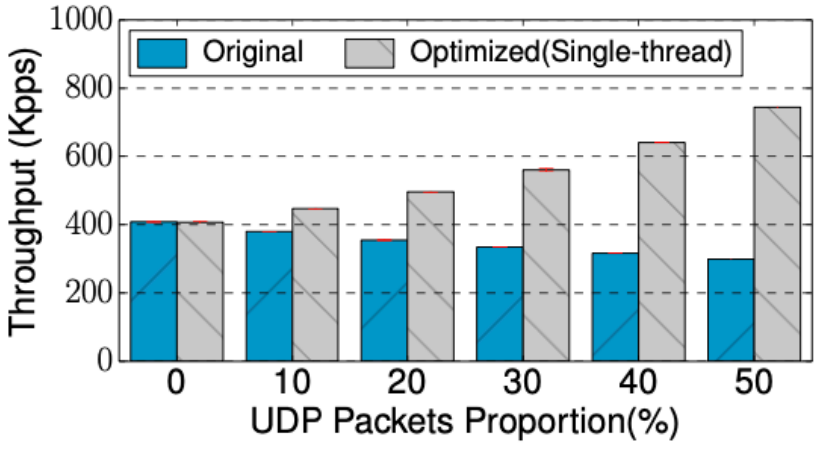
Throughput of Suricata

- Setting: Configured with layer-3 rules.
- Increase by nearly 15% for Snort and by 15% to 10X for Suricata (single thread).
- Suricata is more significant
  - inspects packets deeper in payload than Snort.

Evaluation: Eliminating Type-II Redundancy



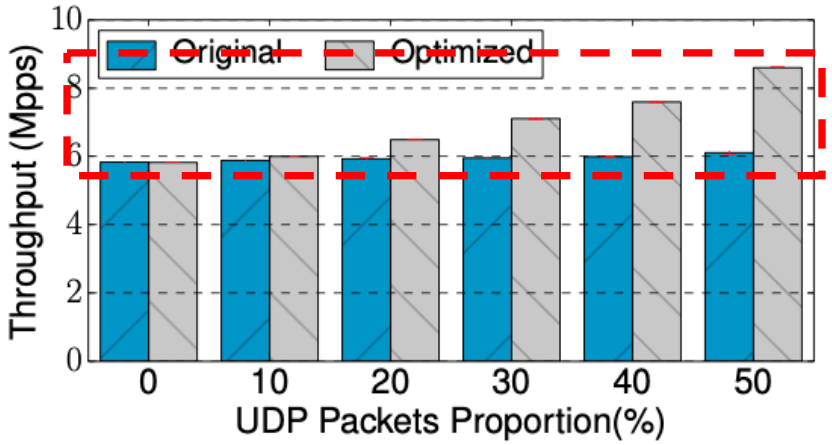
Throughput of Snort



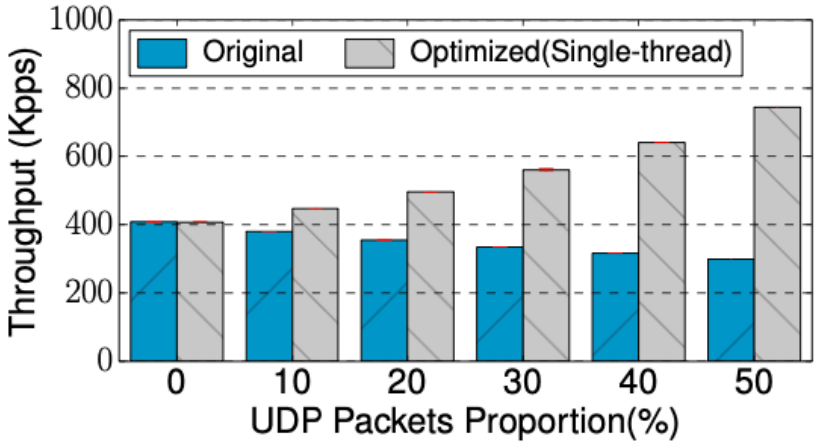
Throughput of Suricata

- Setting: Configured with TCP rules only.
- The larger proportion of UDP packets, the larger performance gain.
- 40% performance gain for Snort and 2.5× for Suricata

Evaluation: Eliminating Type-II Redundancy



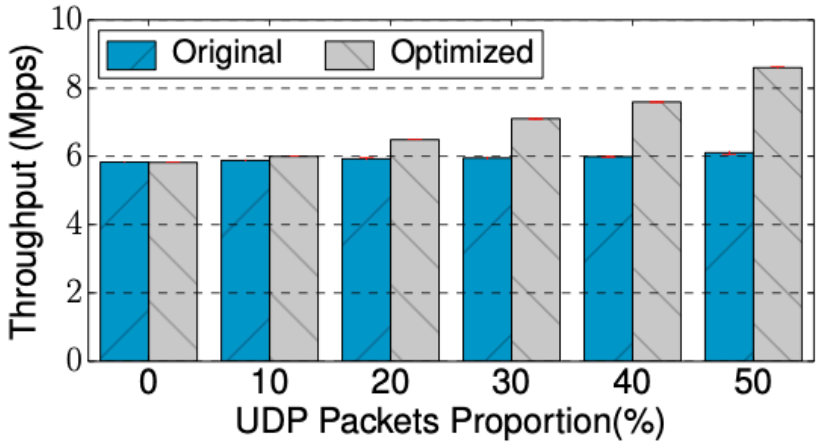
Throughput of Snort



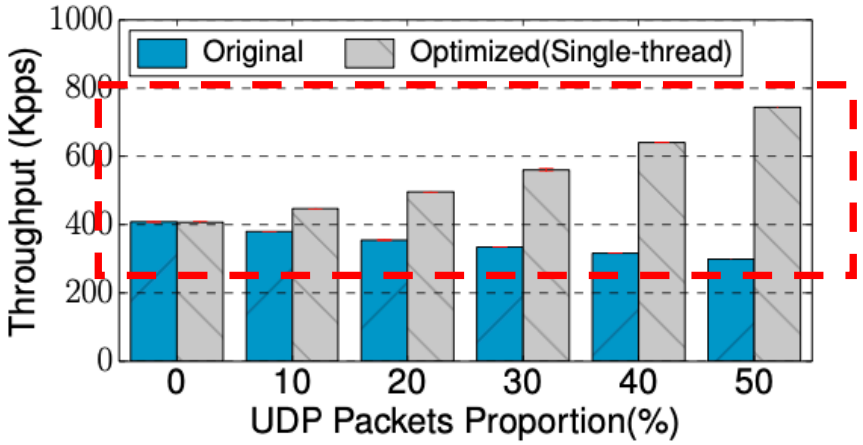
Throughput of Suricata

- Setting: Configured with TCP rules only.
- The larger proportion of UDP packets, the larger performance gain.
- 40% performance gain for Snort and 2.5× for Suricata

Evaluation: Eliminating Type-II Redundancy



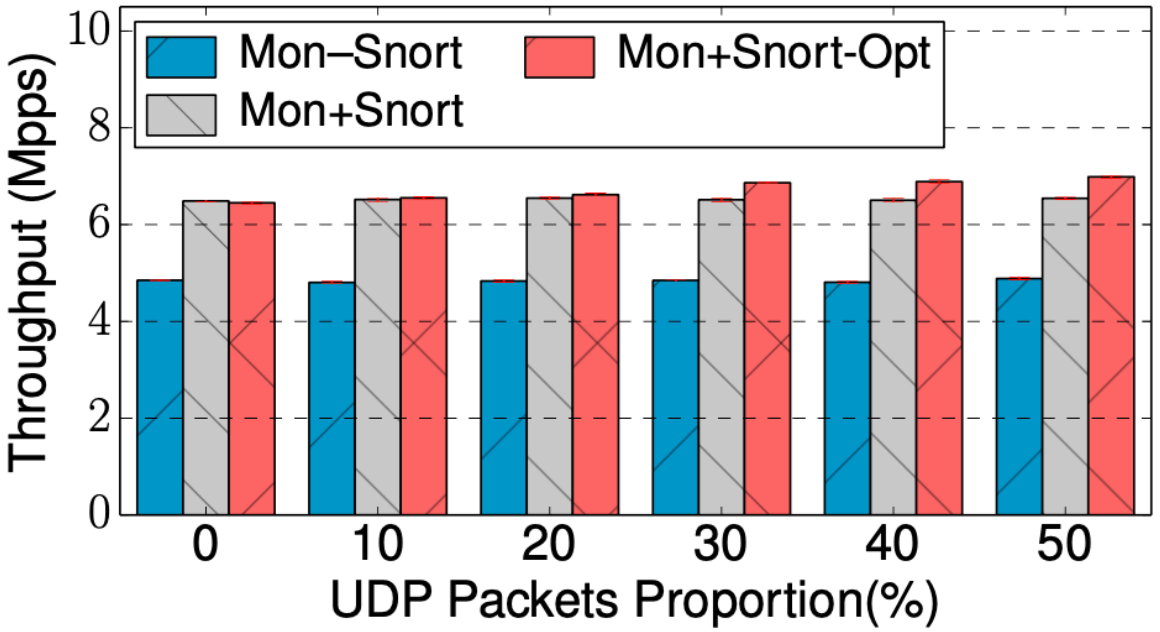
Throughput of Snort



Throughput of Suricata

- Setting: Configured with TCP rules only.
- The larger proportion of UDP packets, the larger performance gain.
- 40% performance gain for Snort and 2.5× for Suricata

Evaluation: Eliminating Type-III Redundancy

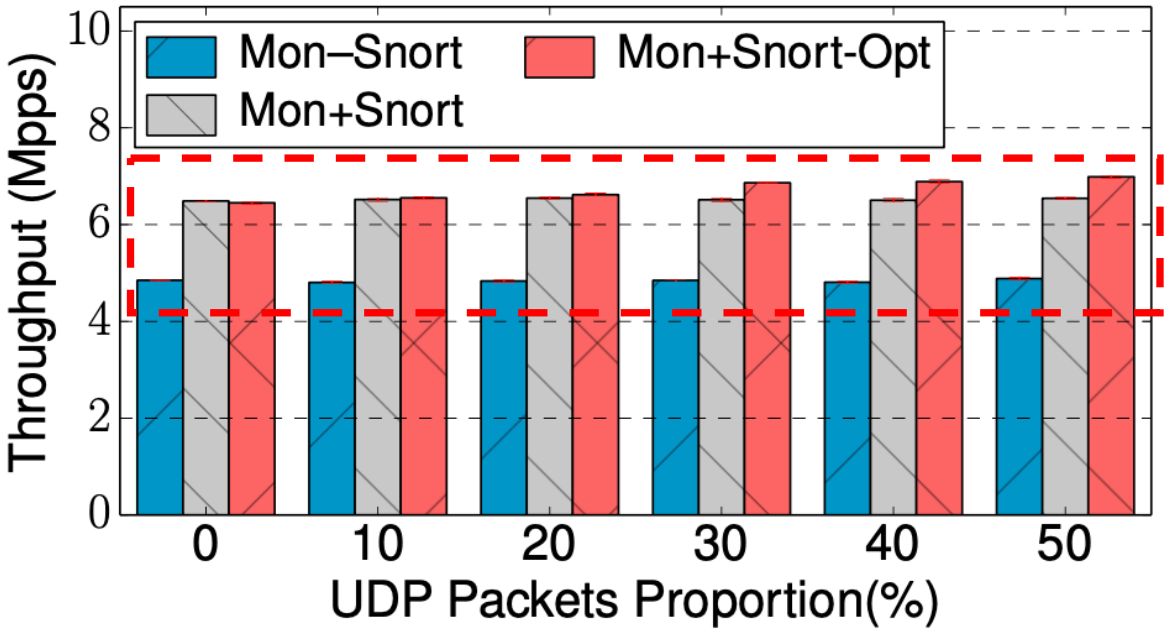


- Setting:
  - Mon—Snort: executed in two processes
  - Mon+Snort: consolidated
  - Mon+Snort-Opt: consolidated and optimized
  - Configured with TCP rules only for Snort

- Consolidation and Redundancy Elimination help improve:
  - By more than 30%

- Performance gain increases as the UDP proportion increases.

Evaluation: Eliminating Type-III Redundancy



- Setting:
  - Mon—Snort: executed in two processes
  - Mon+Snort: consolidated
  - Mon+Snort-Opt: consolidated and optimized
  - Configured with TCP rules only for Snort

- Consolidation and Redundancy Elimination help improve:
  - By more than 30%

- Performance gain increases as the UDP proportion increases.

Evaluation: Overhead

- Labeling Variables and Actions manually:
  - Operator-involved
  - Once for an NF
- Extracting the packet processing logic:
  - 7.2s for Snort and 1.2s for Suricata
- Eliminating Redundancy:
  - 26.8s for Snort and 83.6s for Suricata (mainly cost by symbolic execution).
- Rebuilding:
  - 0.126s for Snort and 2.753s for Suricata

Conclusion

- Show the existence of the redundant logic in NF programs
- Propose NFReducer to eliminate the redundancy.
  - Takes user labeled information
  - Applies compiler techniques
- Performance gain and overhead of the two example NFs.

- In future, we will:
  - Complete and automate the whole workflow process further.
  - Apply NFReducer to more NFs.
  - Make complete tests on NFReducer.