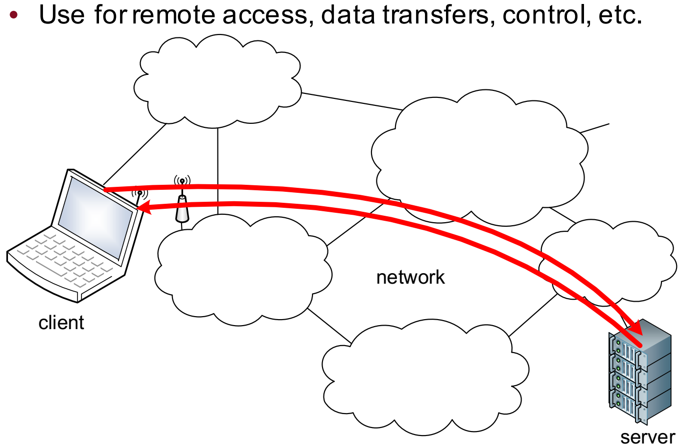
Attacks and Hardware Defenses for Network Infrastructure

October 7, 2016

Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts Amherst

Computer Networks

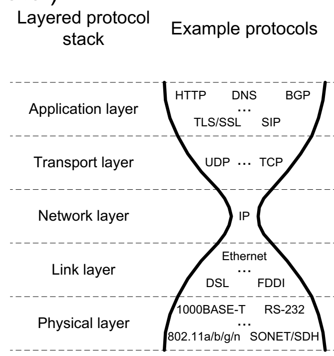
- Networks provide **connectivity** between end-systems
  - Use for remote access, data transfers, control, etc.



- Networking requires **common protocols** for communication

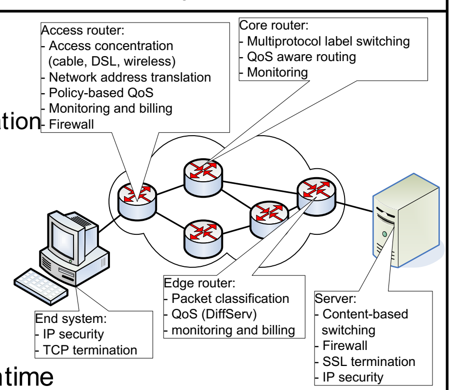
Computer Networks

- Success of the Internet: **hourglass architecture**
  - Very basic services (connectivity, bit pipes, etc.)
  - Highly diverse set of applications "on top"
- Success is also a problem
  - Diverse applications, diverse systems
- **Changing requirements** for network layer
  - New network functionality
    - Security, quality-of-service, multicast, reliability, etc.
  - New communication paradigms
    - Content distribution, content addressable networks, data aggregation, etc.
  - Application-layer processing in network
    - Payload transcoding, content-based load balancing, etc.



Extended Network Functionality

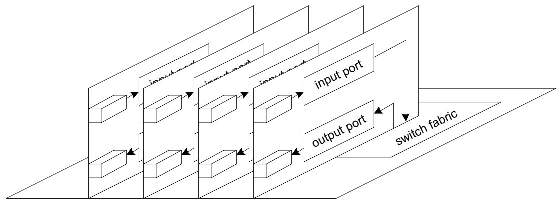
- **Extensions** to current Internet
  - New functionality in routers
    - Firewalls, network address translation
    - Intrusion detection systems
    - Traffic shaping
    - Etc.
- **Customization** of data plane
  - Complex per-packet protocol processing operation
  - Deployment of new features at runtime
  - Vendors may compete on features
- Requires routers with **ability to adapt**
  - Programmability is necessary

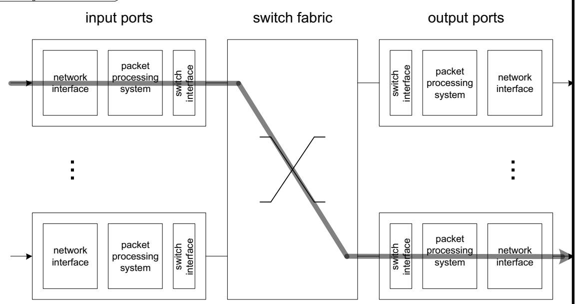


UMassAmherst

Packet Processing on Router

- **Protocol processing** operations implemented on **input port**



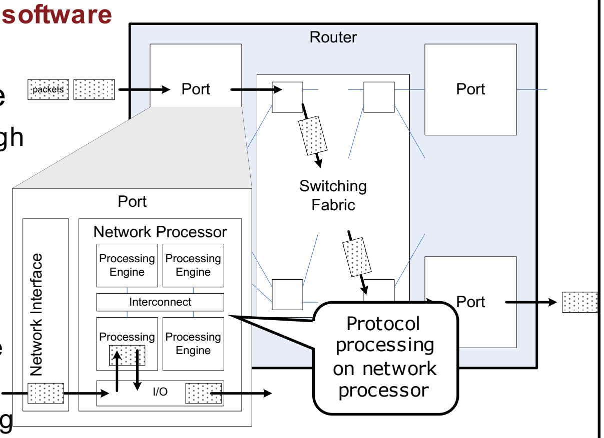


- **Application-Specific Integrated Circuit (ASIC)** for simple protocols
- ASICs is fast, but fixed functionality

UMassAmherst

Programmable Router

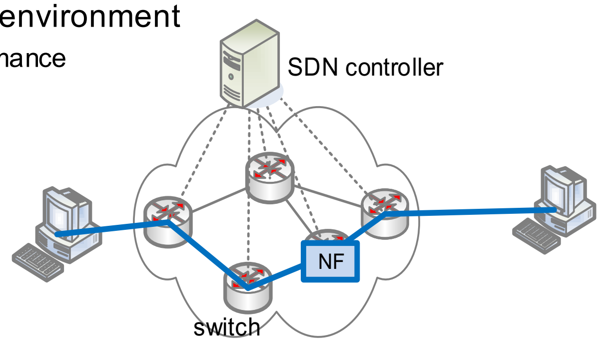
- Flexibility through programmable **network processor**
  - **General-purpose processing** capability in data path
  - Packet processing **in software**
- High-performance processing hardware
  - Scalability through high levels of **parallelism**
  - Example: 40-core network processors
- Key challenge:
  - **Security** is critical for network infrastructure
  - **Vulnerabilities** due to software processing



UMassAmherst

Middleboxes and Network Appliances

- Processing also on **middleboxes** and **network appliances**
  - Standalone nodes (e.g., server-type processor)
- Used with **Software-Defined Networking (SDN)**
  - **Network function virtualization (NFV)**
- Similar processing environment
  - Throughput performance important
  - Limited resources to detect / protect from attacks
- This talk focuses on network processors



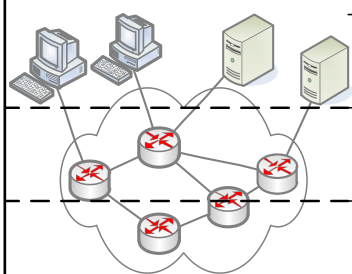
UMassAmherst

Outline

- Introduction
- **Vulnerabilities**
  - **Example attack on network processor**
- Defense mechanism
  - Hardware monitor
- Extensions
  - Multicore hardware monitor and dynamic workloads
  - Secure loading and avoiding homogeneity
  - Operating system support
- Conclusions

Network Attack Classification

- Programmable data plane introduces **new type of attack**
  - Hacking of packet processing engine on router
  - Attack targets **network infrastructure**

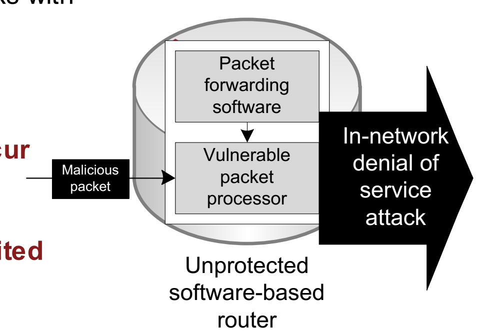


Attack target	Goal of attack	Attack examples	Defenses
End-system	Data access and modification	Hacking, phishing, espionage, etc.	Virus scanner, firewall, network intrusion detection system, etc.
	Denial-of-service	Denial-of-service attack via botnets, etc.	
Control plane	Data access and modificaiton	Malicious route announcement, DNS cache poisoning, etc.	Secure routing protocols (with cryptographic authentication), secure DNS (DNSSEC), etc.
	Denial-of-service	DNS recursion attack, etc.	
Data plane	Data access and modificaiton	Eavesdropping, man-in-the-middle attack, etc.	Secure network protocols (IPSec, TLS), etc.
	Denial-of-service	Exploit of vulnerable packet processing code	Processing monitor, etc.

Focus of this work

Exploit of Vulnerable Network Processor

- Vulnerability can be exploited to launch attack
- **"In-network" denial of service attack**
  - Router has access to many links with high data rates
  - Potentially devastating impact
- Key questions
  - **Can such vulnerabilities occur** in packet processing code? (Yes, we show one example.)
  - **Can vulnerabilities be exploited** to launch DoS attack? (Yes for one processor type; no for another (crashed instead))

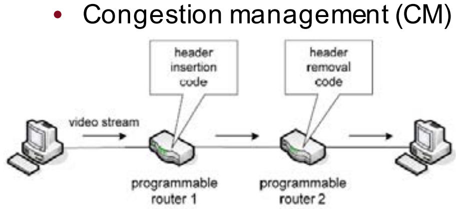


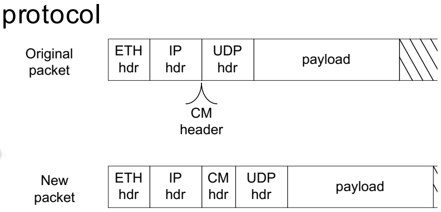
Attack Type

- **Overflow** attacks
  - Malicious data exploits vulnerable code
  - Often leads to attacker executing arbitrary code
  - Can be exploited via network
- **National Vulnerability Database** (late 2014)
  - 66,399 vulnerabilities total
  - 6,518 vulnerabilities that exploit "overflows" via network (approx. 10%)
- End-system vulnerabilities can be detected
  - Virus scanner on end-system
  - Content-inspection firewalls in network
- Packet processors need **custom protection**
  - No processing power for virus scanner
  - No protection from firewall inside network core

Example of Data Plane Vulnerability

- Many different potential vulnerabilities
  - We focus on **one example** to show that it is possible
  - Specific attack depends on system, software, etc.
- Requirements
  - Vulnerability must be **in packet processing code**
  - Vulnerability must be **triggered by data packet**
- Protocol processing: header insertion
  - Congestion management (CM) protocol



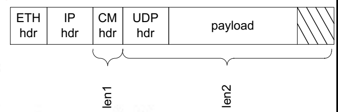


Header Insertion Vulnerability

- Vulnerability based on **stack smashing attack**
  - Exploit of **integer overflow**
    - Sum is 4 (not 65540)

```
unsigned short sum;
unsigned short one = 65532;
unsigned short two = 8;
sum = one + two;
```

- Vulnerable code in **networking context**
  - Carefully crafted shifting of packet content:



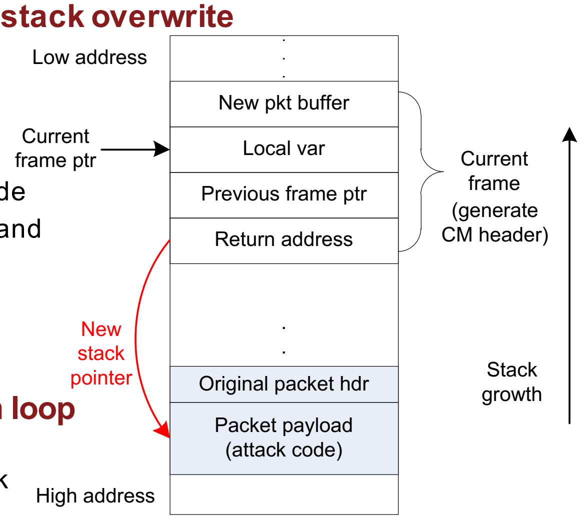
    - Check if enough room for shift
    - Perform memcpy

```
unsigned short sum;
sum= len1+ len2;
if(sum > MAX_PKT) { return -1;}
else {
memcpy((new_pkt_buf+len1), orig_pkt, len2);
```

  - Problem:
    - Attack can send short, malformed UDP packet (length field of 65532)
    - Integer overflow can occur on check
      `CM_hdr_size + UDP_length = 12 + 65532 = 8`
      Thus, `CM_hdr_size + UDP_length < MAX_PKT`
    - Memory copy of 65532 bytes will overwrite outside boundaries of packet

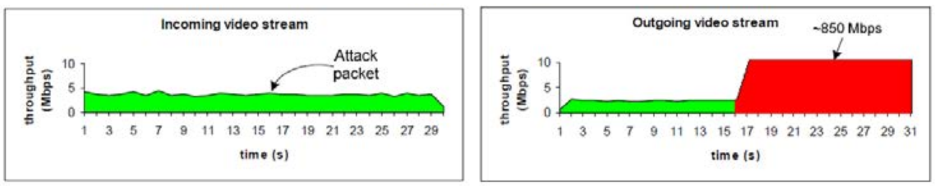
Header Insertion Exploit

- Memory copy causes **stack overwrite**
  - Return address from function call can be changed
  - **Control flow** can be redirected to attack code
  - If separate instruction and data memory (Harvard architecture) use return-to-libc attack
- Example Attack code: **infinite transmission loop**
  - Self-propagating denial-of-service attack



Header Insertion Exploit

- **Prototype** implementation
  - Custom network processor on NetFPGA
  - Packet forwarding with vulnerable code
  - Malicious packet injected into benign background traffic
- **Single malicious packet** triggers **attack at full link rate**!

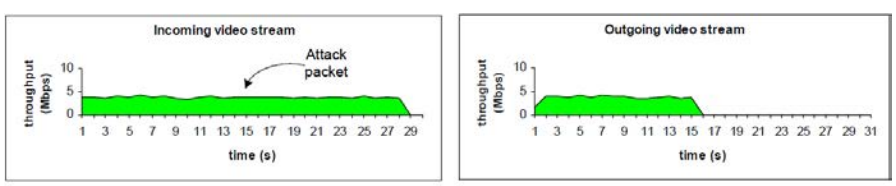


(b) Benign traffic and single attack packet on custom network processor

- Attack has **not** yet been shown on commercial system

Header Insertion Exploit

- Ability to exploit vulnerability depends on processor system
  - Previous result: custom **ARM-based packet processor**
  - Other system: **Click modular** router on Linux system
    - Stack smashing crashes router, but could not create DoS attack



(c) Benign traffic and single attack packet on Click modular router

- Main observation: **software on NP can be attacked**
  - Exploits can happen through data plane only
- Need to develop **defense mechanisms** for router systems

Outline

- Introduction
- Vulnerabilities
  - Example attack on network processor
- **Defense mechanism**
  - **Hardware monitor**
- Extensions
  - Multicore hardware monitor and dynamic workloads
  - Secure loading and avoiding homogeneity
  - Operating system support
- Conclusions

Defense Mechanism for Processors

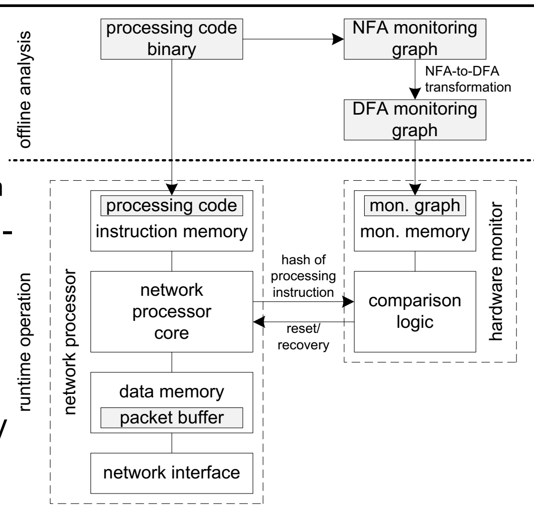
- **Software-based defenses** (e.g., virus scanner)
  - High processing overhead
  - Processing requirement is proportional to input/output operations
  - Scanning for known attacks is reactive, not proactive
- **Hardware-based defenses** more suitable
  - Defense mechanism can be separated from data plane
    - Makes it more difficult to circumvent
  - Performance impact on packet processor is small
  - Challenge is to make it dynamically adaptable
    - Needs to work for new packet processing functions
- We have designed and prototyped **hardware defense for NP**
  - Hardware monitor tracks processor
  - Deviation from “normal behavior” due to attack can be detected
  - Reset operation recovers system

Related Work

- **Monitor-based** defense mechanism for embedded systems
  - Aurora et al., DATE 2005
  - Ragel et al. DAC 2006
  - Zambreno et al., TECS 2005
  - Our monitor uses finer-grained monitoring for faster detection
    - More details in Mao and Wolf, TC 2010
- **Processor-based** defense mechanisms
  - No eXecute (NX) bit (creates virtual Harvard architecture)
  - Depends on processor architecture
- **Network-based** defense mechanisms
  - Attack signature in intrusion-detection systems (e.g., snort, bro)
  - Problem with system homogeneity and IDS only at network edge

System Architecture

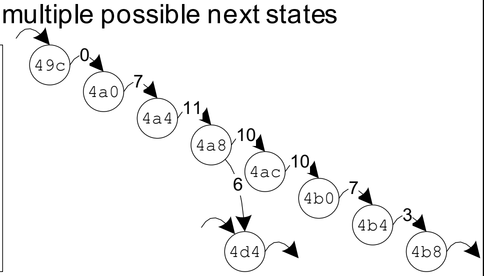
- Hardware monitor co-located with each processor core
  - **Core reports hash** of each executed instruction
- **Monitoring graph** represents correct behavior
  - Obtained from offline analysis of binary
  - Deviations trigger reset
- Change of software easy
  - Just need matching monitoring graph



Offline Analysis of Processing Binary

- Executed instruction reported by core as 4-bit hash
  - **Hash combines address, opcode, registers**
  - Hash allows for **compact representation** of information
- Monitoring graph
  - **Each instruction represented as a state**
  - Edges correspond to execution of instruction
  - Control-flow operations lead to multiple possible next states

```
[…]
49c:     97c20010  lhu      v0,16(s8)
4a0:     00000000  nop
4a4:     2c420033  sltiu    v0,v0,51
4a8:     1440000a  bnez     v0,4d4
4ac:     00000000  nop
4b0:     3c026666  lui      v0,0x6666
4b4:     34430191  ori      v1,v0,0x191
4b8:     97c20010  lhu      v0,16(s8)
[…]
```



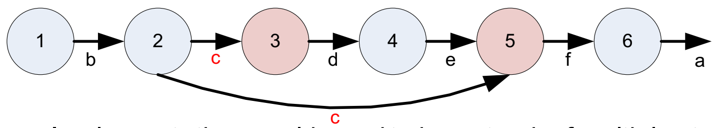
Implementation Cost of Monitor

- Monitor requires additional logic and memory resources
- **Comparison logic** tracks hash value
  - Simple logic to follow control flow in processor
- **Graph memory** stores hash for each instruction
  - Approximately 4 bits for each 32-bit instruction
  - Fraction of size of application binary
- Examples from NpBench
  - Hundreds to thousands of instructions only

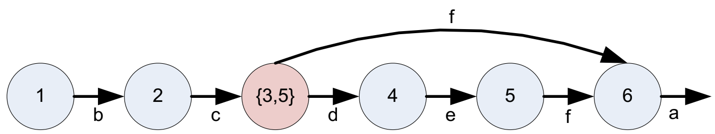
Netw. appli-cation	No. of instr.
crc	276
frag	573
red	802
md5	3,147
ssld	828
wfq	905
mtc	2,427
mpls-upstr.	1,603
mpls-dwnstr.	1,574

NFA-to-DFA conversion

- Problem: **non-deterministic finite automaton** (NFA)
  - State may have two next states with same edge value due to hash



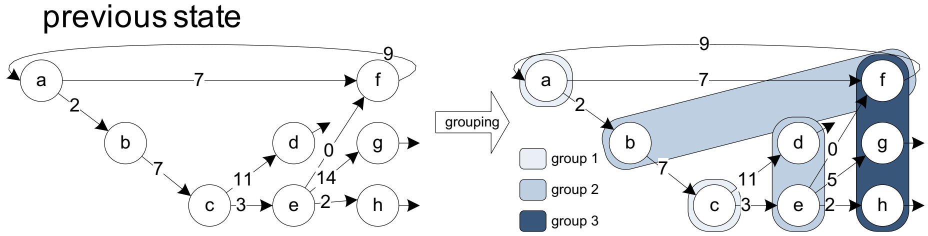
  - Implementation would need to keep track of multiple states
- Solution: **NFA-to-DFA conversion** (powerset construction)
  - Well-known algorithm [Hopcroft and Ullman, 1976]



  - **Deterministic finite automaton** (DFA) requires only one state

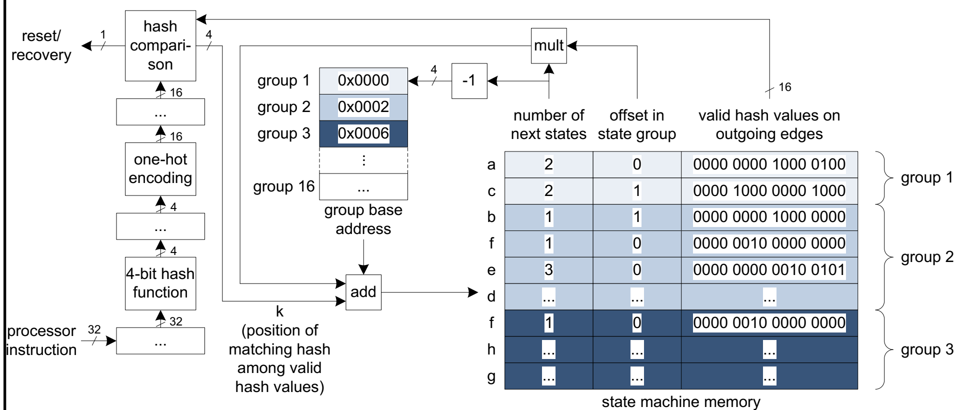
Implementation of DFA Monitor

- Need to design effective **DFA traversal** mechanisms
- Requirements
  - **Compact representation**
  - **Fast processing** of each hash value (single memory access)
- Each state may have up to 16 next states
  - Most states only have one or two next states
- Idea: **grouping** of states by number of outgoing edges of previous state



Implementation of DFA Monitor

- DFA monitor system
  - Memory keeps all valid hash values in one **bit vector**
  - Next state based on **offset** of group and position of hash value



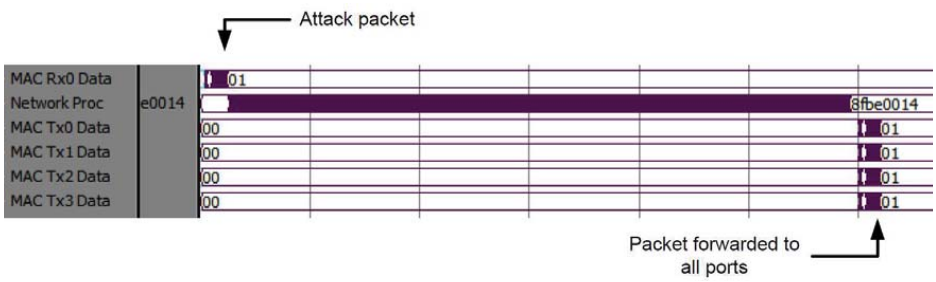
Evaluation

- Monitoring **lookup speed**
  - Single memory access plus lookup into fixed-size register file
- **Memory size** of monitor
  - More states due to NFA-to-DFA conversion
  - More states due to multiple entries in memory for certain states
  - In practice, **overhead is below 10%**

Netw. appli-cation	No. of instr.	NFA states	Max. mem. access	DFA states	Mem. entries	Mem. over-head
crc	276	276	2	276	282	2.2%
frag	573	573	3	592	622	8.6%
red	802	802	2	805	847	5.6%
md5	3,147	3,147	8	3,173	3,228	2.6%
ssld	828	828	5	829	854	3.1%
wfq	905	905	2	914	953	5.3%
mtc	2,427	2,427	3	2,460	2572	6.0%
mpls-upstr.	1,603	1,603	10	1,621	1,753	9.4%
mpls-dwnstr.	1,574	1,574	12	1,582	1,706	8.4%

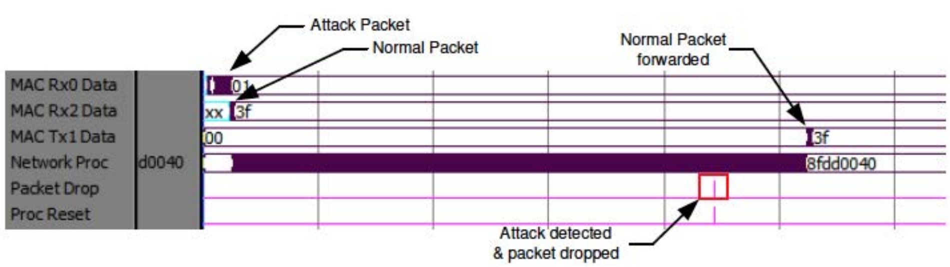
Timing Diagram

- Attack **without monitor**
  - Attack packet is **forwarded on all ports**



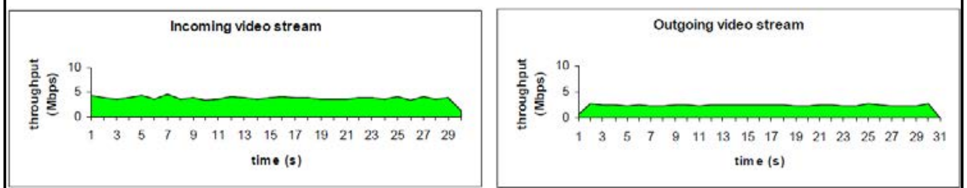
Timing Diagram

- Monitor works as expected
  - **Attack packet is detected** and dropped
  - Later **normal packet is forwarded**

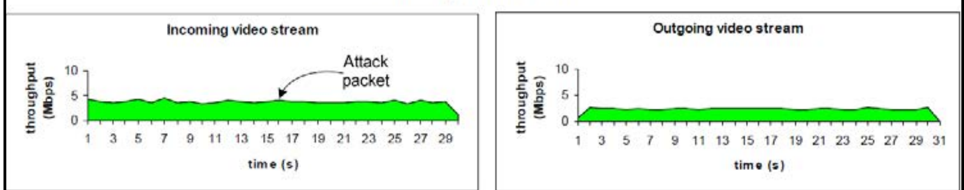


Attack with Defense in Place

- Attack packet dropped, **router continues to operate**



(a) Benign network traffic



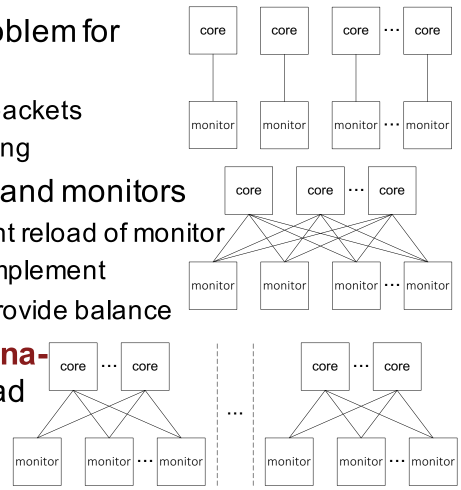
(b) Benign traffic and single attack packet

Outline

- Introduction
- Vulnerabilities
  - Example attack on network processor
- Defense mechanism
  - Hardware monitor
- **Extensions**
  - **Multicore hardware monitor and dynamic workloads**
  - Secure loading and avoiding homogeneity
  - Operating system support
- Conclusions

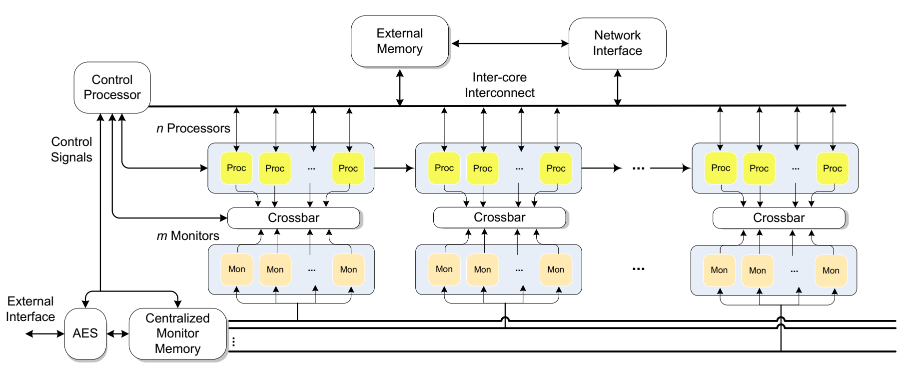
Multicore Monitor

- **Dynamic workloads** pose problem for hardware monitor
  - Processing may differ between packets
  - Monitors need to match processing
- Mapping between processors and monitors
  - **1-to-1 mapping** requires frequent reload of monitor
  - **Any-to-any mapping** costly to implement
  - Clusters with **n-to-m mapping** provide balance
- Interconnect is **configured dynamically** depending on workload
  - Mapping between core and monitor



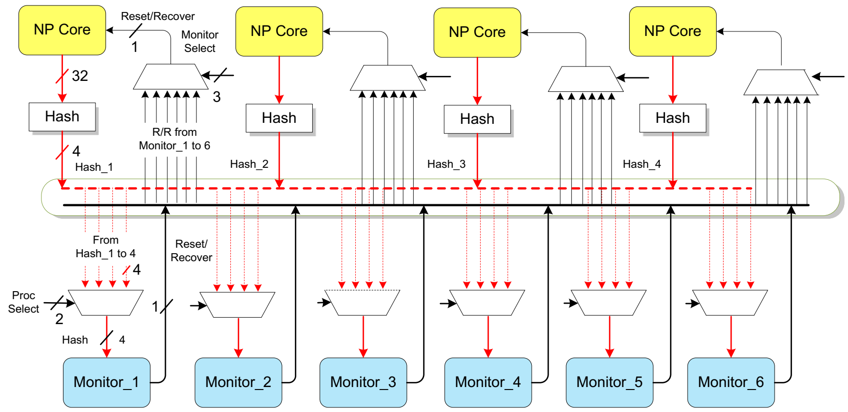
System Architecture of Clustered System

- Multiple cores can access multiple monitors
  - Dynamic configuration of **crossbar**
- **Secure loading** of monitors through external interface



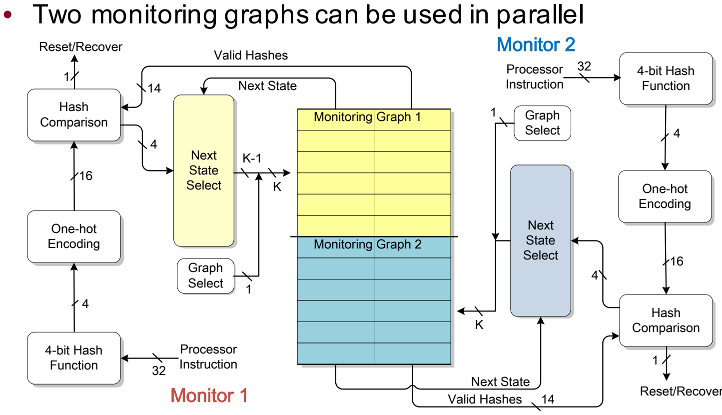
Cluster Design

- Simple implementation of **clustered monitor**
  - Dynamic configuration through programming of demultiplexers



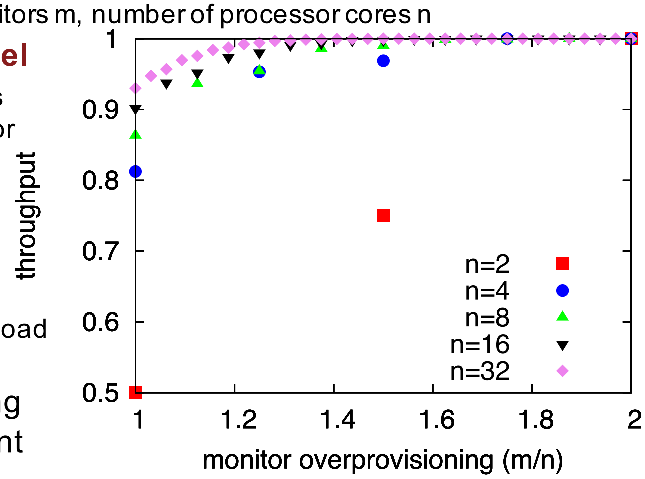
Dual-Ported Monitor Implementation

- Memory of **monitor can be shared** between two monitors
  - Effective use of dual-ported memory
  - Two monitoring graphs can be used in parallel



Runtime Monitor Allocation

- **How many monitors per cluster?**
  - Number of monitors m, number of processor cores n
- **Analytical model**
  - Blocking occurs when no monitor is available for given packet processing
  - Two programs with equal traffic and workload assumed
- Overprovisioning of 1.5 is sufficient



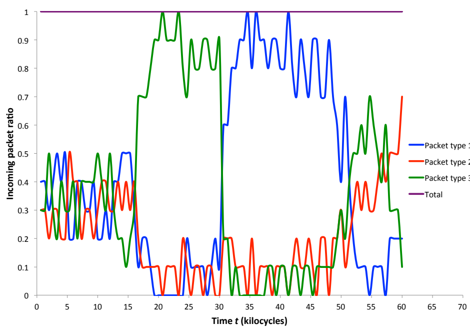
Prototype Implementation on FPGA

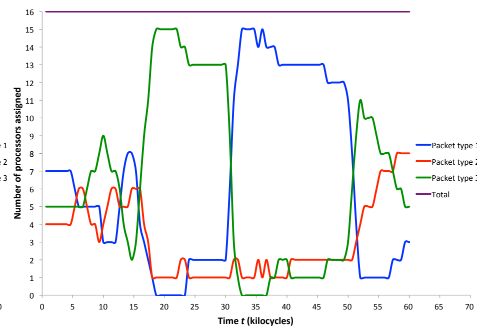
- **Multi-core** system (4 cores, 6 monitors)
  - Monitor logic very simple
  - Interconnect uses very little resources
  - Monitors require about **1/3 of memory** of processors
  - Monitors require about **1/8 of power** of processors

	Available in FPGA	DE4 interface	Network processors	SHMG monitors	SHMG intrcon.
LUTs	182,400 -	33,427 67.8%	15,025 30.4%	816 1.7%	96 0.1%
FFs	182,400	36,467	8,367	147	0
Bits	14,625,792 -	2,263,888 44.0%	2,097,134 40.7%	786,432 15.3%	0 0%
Pwr (mW)	-	1490.83	388.6	41.76	5.30

Runtime Operation

- Adaptation based on **threshold in queue** for application
- Simulation results
  - **Monitor allocation adapts** to dynamics in traffic

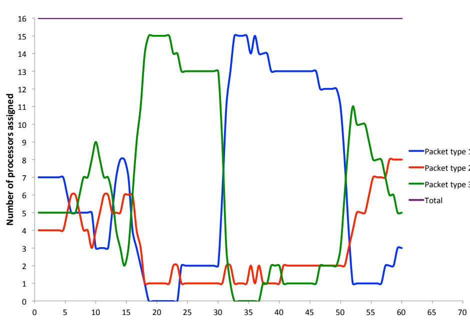


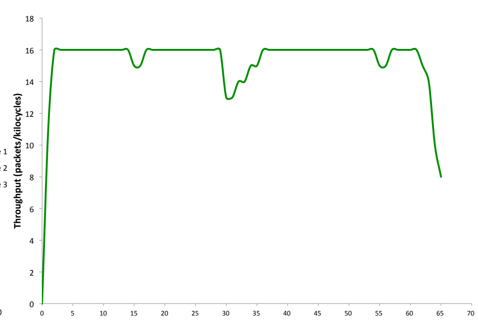


UMassAmherst

Runtime Operation

- Simulation results
  - **Throughput variation** due to adaptation
  - Small inefficiencies during workload change





Graph Loading Times

- Time to load graph depends on **application size**
- Results from NpBench

Network benchmark	Memory graph size (bits)	Graph reload time (cycles)	Graph reload time (μs)
crc	8,460	529	2.64
frag	18,660	1,166	5.83
red	25,410	1,588	7.94
md5	96,840	6,052	30.26
ssld	25,620	1,601	8.01
wfq	28,590	1,787	8.93
mtc	77,160	4,822	24.11
mpls (up)	52,590	3,287	16.43
mpls (down)	51,180	3,199	15.99

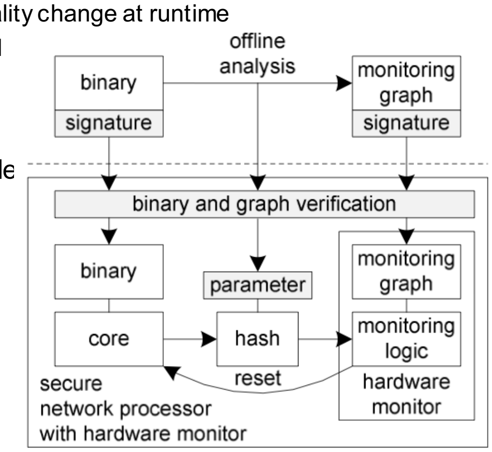
Outline

- Introduction
- Vulnerabilities
  - Example attack on network processor
- Defense mechanism
  - Hardware monitor
- **Extensions**
  - Multicore hardware monitor and dynamic workloads
  - **Secure loading and avoiding homogeneity**
  - Operating system support
- Conclusions

UMassAmherst

Extension: Infrastructure Diversity

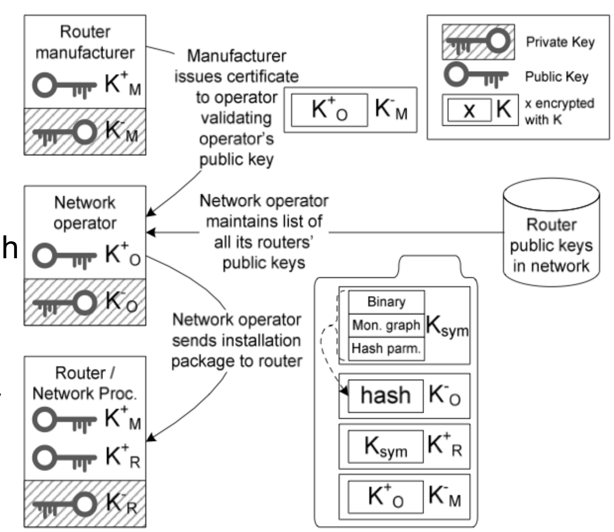
- **System-level challenges**
  - Dynamics: **runtime verification** of monitoring graphs
    - Network traffic and functionality change at runtime
    - Multiple processor cores and their monitors need to be reprogrammed based on the traffic
  - Homogeneity: parameterizable hashing for **heterogeneity**
    - Practical networks use large numbers of identical router devices
    - A successful attack on one device can lead to Internet-scale failures



UMassAmherst

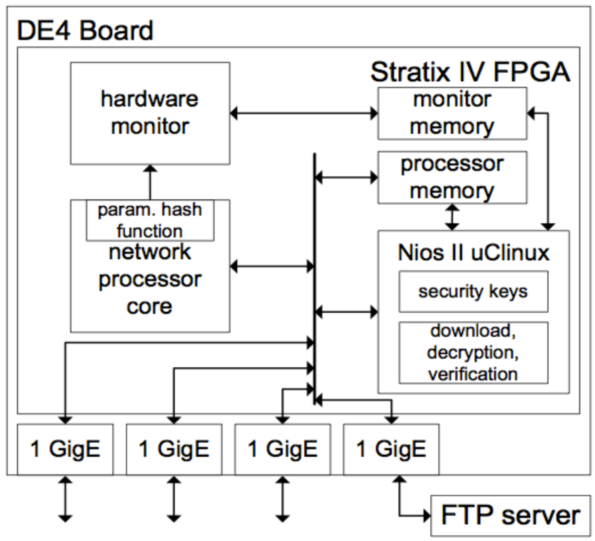
Security Loading of Monitoring Graph

- Three entities:
  - Router **manufacturer**
  - **Network operator**
  - Router/**network processor**
- Signatures on graph establish **chain of trust**
  - Network processor verifies authenticity
  - Network operator can install new graph



Prototype Implementation on FPGA

- **Prototype system**
  - Altera Stratix IV FPGA on a DE4 board
  - Nios II connects to a FTP server through OpenSSL
  - Parameterizable hash function in hardware monitor



Security Operations Evaluation on Nios II

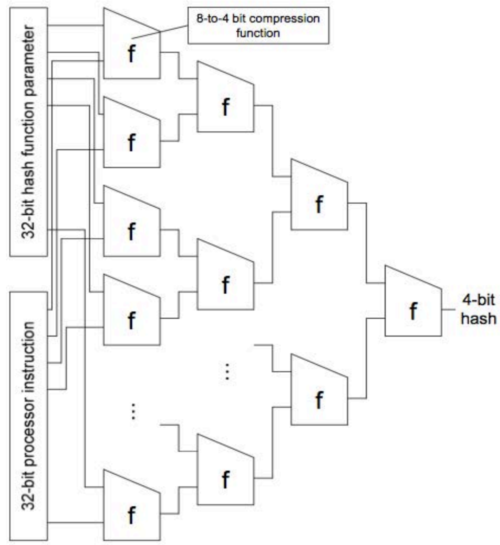
- Secure download, decryption, and verification times
  - IPv4 with congestion management application
  - Verification **takes several sections**

Step	Time (s)
Download data from FTP server	1.90
Check manufacturer certificate of network operator's public key K_O^+	3.33
Decrypt AES key K_{sym} using router's private key K_R^-	8.74
Decrypt package with AES key K_{sym}	7.73
Verify packet signature with network operator's public key K_O^+	3.92
Total	25.62
Total (no networking or certificate check)	20.39

UMassAmherst

Parameterizable Hash Function

- **Merkle tree** for hash function
  - Can be parameterized
  - High performance implementation in hardware
  - Low resource overhead
- Each network processor can use a **different parameter** value
  - Resulting monitoring graph has different hash values



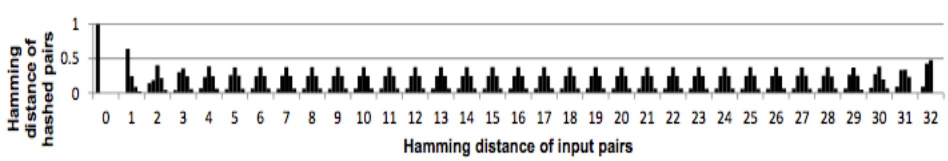
UMassAmherst

Hash Function Evaluation

- Resource cost for **hash function**
  - Compared to non-parameterizable hash function

	Bitcount hash	Merkle tree hash
LUTs	103	95
FFs	61	61
Memory bits	0	32

- **Distribution of hash values** in Merkle tree
  - Random distribution of Hamming distance for almost all inputs
  - Hash function requires zero Hamming distance for same inputs



UMassAmherst

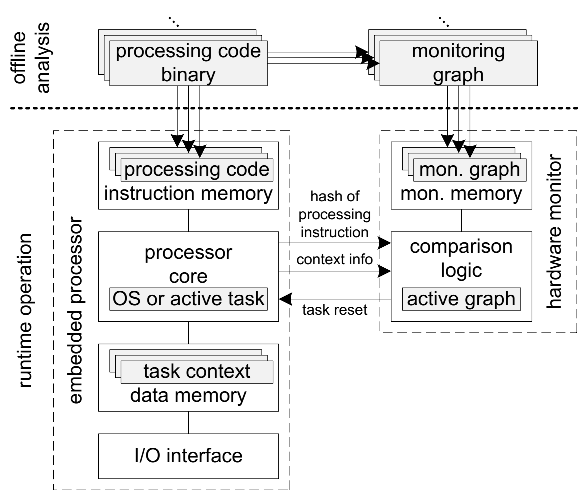
Outline

- Introduction
- Vulnerabilities
  - Example attack on network processor
- Defense mechanism
  - Hardware monitor
- **Extensions**
  - Multicore hardware monitor and dynamic workloads
  - Secure loading and avoiding homogeneity
  - **Operating system support**
- Conclusions

UMassAmherst

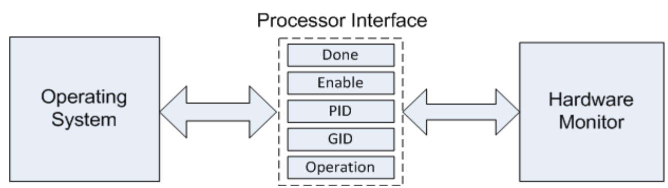
Extension: Operating System in ES

- **Coordination between embedded OS and monitor**
  - Multiple active processes in OS, multiple active monitoring graphs
  - Monitor switches monitoring graphs in sync with OS processes
  - Requires minor extension to OS
- Prototype:
  - NIOS II processor
  - µC/OS-II operating system



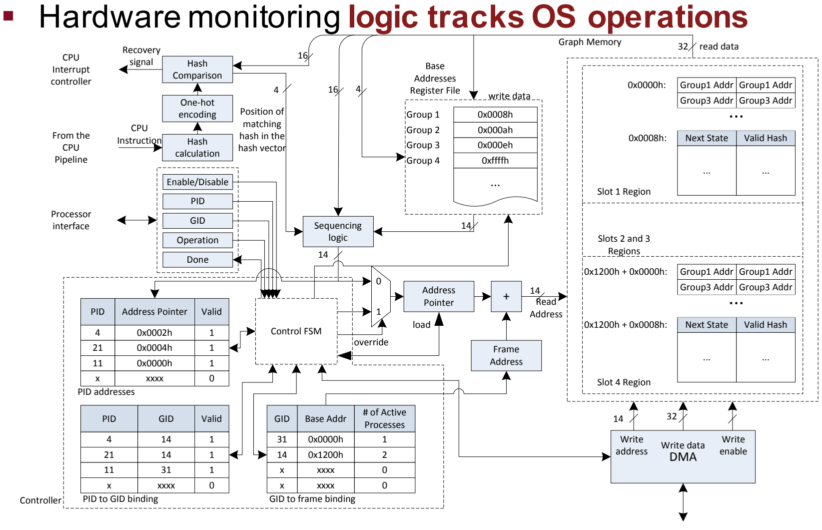
Processor-to-Monitor Interface

- OS on processor needs to coordinate with monitors
  - **Process creation** (ensure monitoring graph is ready)
  - **Context switch** between processes (switch monitoring graph)
  - **Process deletion** (remove monitoring state)
  - **Reset signal** from monitor
- A set of five registers to communicate with the processor



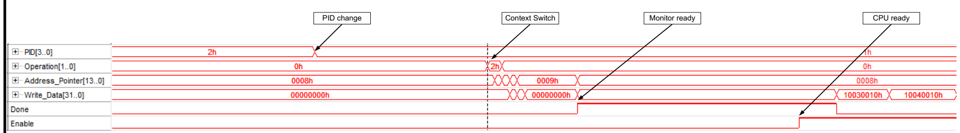
Operating System Support

- Hardware monitoring **logic tracks OS operations**

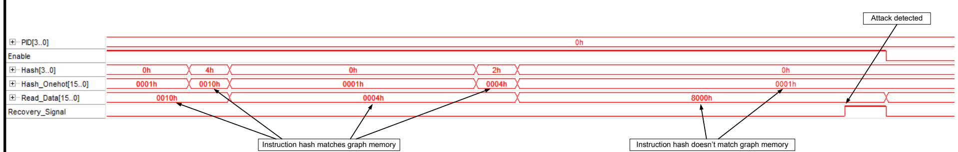


Operating System Support

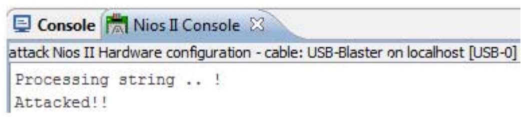
- **Context switch** interactions:



- **Attack detection**:




```
void process_input(char *stringpassed) {
    char name[90];
    strcpy(name,stringpassed);
    printf("Processing string .. !\n");
    return;
}
```



Operating System Support

- **Implementation cost** on Stratix IV FPGA

	Available on FPGA	Nios II with no HW monitor	HW monitor and controller
LUTs	182,400	1,341	406
FFs	182,400	1,166	522
Mem. bits	14,625,792	2,108,416	524,512
Pwr (mW)	-	105.97	41.83

- Hardware monitoring can be used for **embedded systems**
  - Embedded systems are similarly performance constrained

UMassAmherst

Outline

- Introduction
- Vulnerabilities
  - Example attack on network processor
- Defense mechanism
  - Hardware monitor
- Extensions
  - Multicore hardware monitor and dynamic workloads
  - Secure loading and avoiding homogeneity
  - Operating system support
- **Conclusions**

UMassAmherst

Conclusions

- **Current and future Internet** needs to meet new demands
  - Flexibility is key to avoid ossification
  - Deployment of new edge services requires programmable data plane
- Programmable routers provide packet processing platform
  - Systems problem: **security vulnerabilities**
  - **Attacks can be launched within data plane** (i.e., not control access)
  - **Monitor-based hardware defense** mechanism is effective
- Our work has addressed many **practical concerns**
  - Workload **dynamics** and **secure installation** of monitoring graphs
  - System **heterogeneity**
  - Extension to **general embedded systems** with operating systems
- **Exciting research area** that spans computer networking, embedded systems, and system security

Acknowledgements

- **Graduate students**
  - Kekai Hu (now: Intel)
  - Arman Pouraghily
  - Harikrishnan Chandrikakutty (now: Juniper Networks)
  - Danai Chasaki (now: Villanova Univ.)
  - Shufu Mao (now: WorldQuant)
  - Thiago Teixeira
- **Collaborators**
  - Russell Tessier
- **Sponsors**
  - National Science Foundation
  - Altera Corporation

Selected Publications

- **Data plane attack**:
  - Danai Chasaki and Tilman Wolf. Attacks and defenses in the data plane of networks. *IEEE Transactions on Dependable and Secure Computing*, 9(6)798–810, November 2012.
  - Danai Chasaki and Tilman Wolf. Design of a secure packet processor. In *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, San Diego, CA, October 2010.
- **Hardware monitors for network processors**:
  - Shufu Mao and Tilman Wolf. Hardware support for secure processing in embedded systems. *IEEE Transactions on Computers*, 59(6):847–854, June 2010.
  - Harikrishnan Kumarapillai Chandrikakutty, Deepak Unnikrishnan, Russell Tessier, and Tilman Wolf. High-performance hardware monitors to protect network processors from data plane attacks. In *Proc. of 50th Design Automation Conference (DAC)*, Austin, TX, June 2013.
  - Kekai Hu, Harikrishnan Chandrikakutty, Russell Tessier, and Tilman Wolf. Scalable Hardware Monitors to Protect Network Processors from Data Plane Attacks. In *Proc. of First IEEE Conference on Communications and Network Security (CNS)*, Washington, DC, October 2013. (Best Paper Award)
  - Kekai Hu, Tilman Wolf , Thiago Teixeira, and Russell Tessier. System-level security for network processors with hardware monitors. In *Proc. of 51st Design Automation Conference (DAC)*, San Francisco, CA, June 2014.
- **Hardware monitors for embedded systems**:
  - Tedy Thomas, Arman Pouraghily, Kekai Hu, Russell Tessier, and Tilman Wolf. Multi-task support for security-enabled embedded processors. In *Proc. of 26th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 136–143, Toronto, ON, July 2015.

Tilman Wolf
wolf@umass.edu
http://www.ecs.umass.edu/ece/wolf/