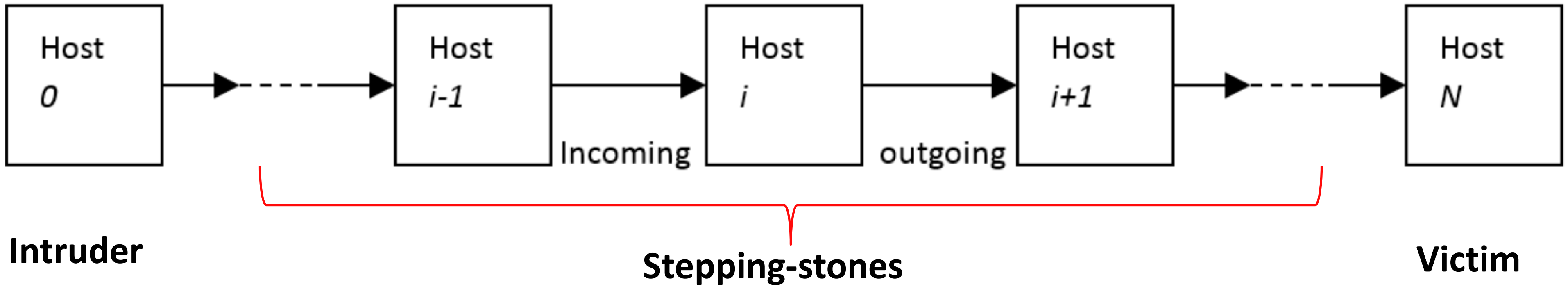
# Stepping-stone Intrusion Detection Using Packet Crossover

Student: Austin Lee
Faculty Advisor: Lixin Wang, Ph.D.
Columbus State University

CAE Cybersecurity Community Symposium
June 9-10, 2022

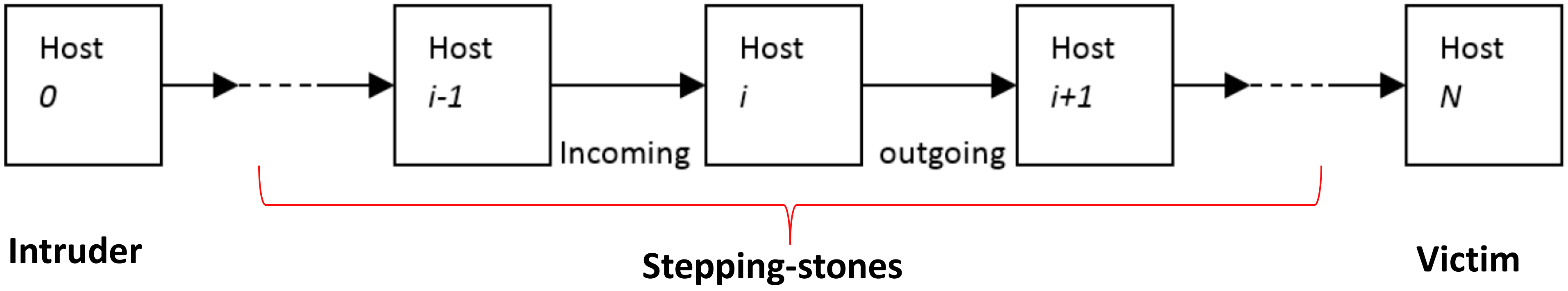
# Stepping-stone Intrusion

- Intruders often launch attacks through compromised hosts, called ***stepping-stones***, in order to reduce the chance of being detected
- In a stepping-stone attack, an intruder uses a chain of hosts as relay machines and remotely login these hosts using tools such as *telnet, rlogin or SSH*



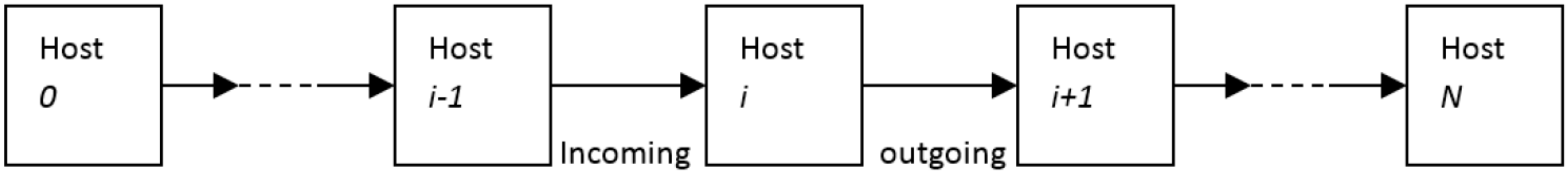
# Stepping-stone Intrusion (2)

- On the intruder's local machine, he/she enters commands that are relayed via the stepping-stone hosts in the connection chain till they finally reach the victim machine



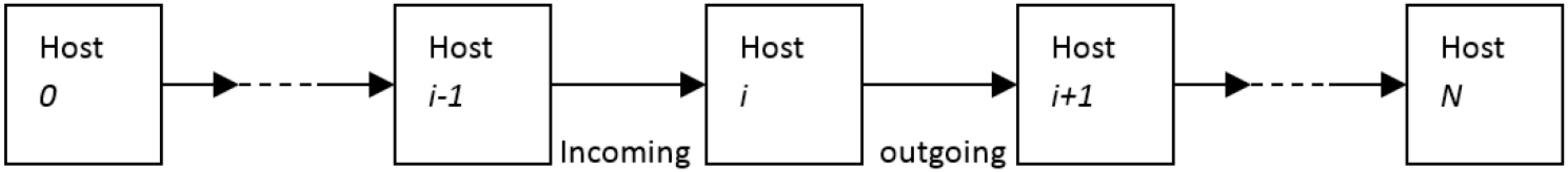
# Benefits of Using Stepping-stones

- Intruders can be hidden by a long interactive session
- Since ***each TCP session*** between a client and a server ***is independent of other sessions*** even though the sessions may be relayed, accessing a server via multiple relayed TCP sessions can make it much harder to tell the intruder's geographical location



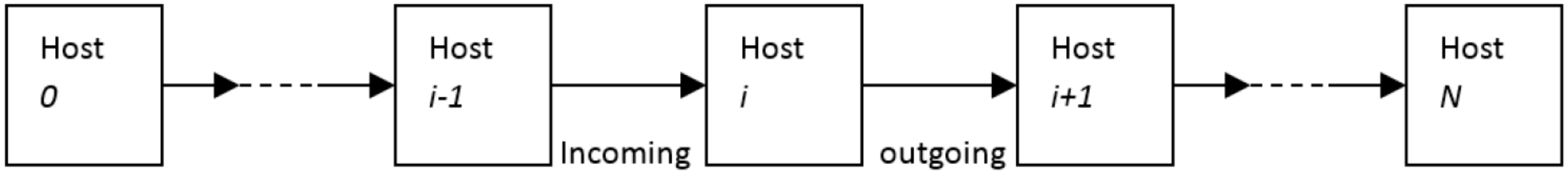
# Benefits of Using Stepping-stones (2)

- The final victim host can only see the traffic from the last connection of the chain, it is extremely difficult for the victim to learn any information about the origin of the attack
- If a stepping-stone intrusion can be detected within the attacking period, the connection can be cut off and the victim can be protected



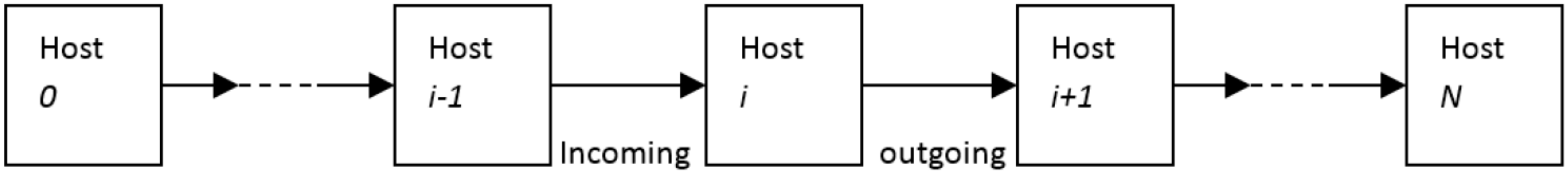
# Stepping-stone Intrusion Detection

- **Host 0**: intruder’s host
- **Host N**: victim’s host
- **Stepping-stones**: Hosts 1 ~ N-1
- Stepping-stone Intrusion Detection (**SSID**) can be performed at one of the stepping-stones
- The detection program resides at Host *i* (called the ***detecting sensor***)



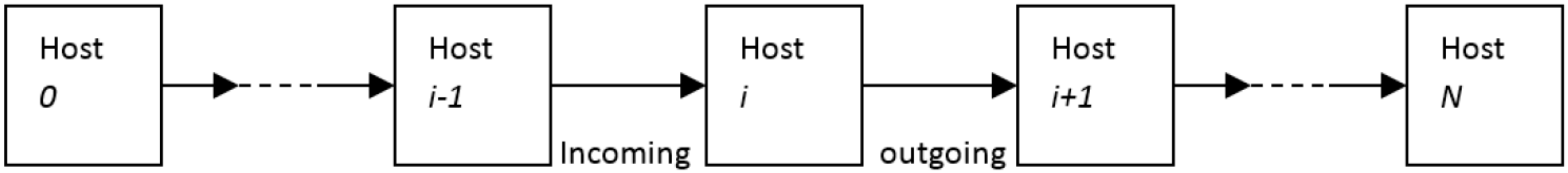
# Stepping-stone Intrusion Detection (2)

- **SSID** is to determine whether the sensor Host *i* is used as a stepping-stone
- The connection from Host *i*-1 to Host *i* is called an ***incoming connection*** to Host *i*
- The connection from Host *i* to Host *i*+1 is called an ***outgoing connection*** from Host *i*



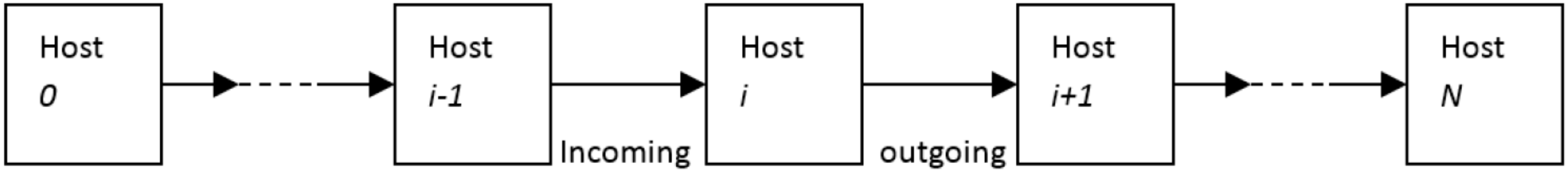
# Host-based SSID

- One type of approach to detect stepping-stone intrusion is to compare all the incoming connections with all the outgoing connections of the same host to see if there exists a relayed pair
- This type of approach is called ***host-based* SSID**
- Some apps may legally use 1 or 2 stepping-stones to access a remote server
  - This type of approach may generate high false-positive errors



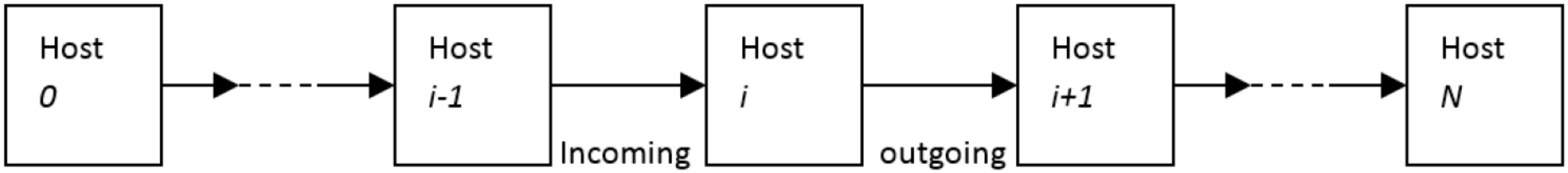
# Network-based SSID

- ***Network-based* SSID -** estimate the number of connections from Host 0 to Host N in the chain
  - length of the connection chain
- The more hosts involved in a session to access a server, the slower network communication
- Accessing a server indirectly via three or more hosts generates lots of network traffic, and makes it very ineffective



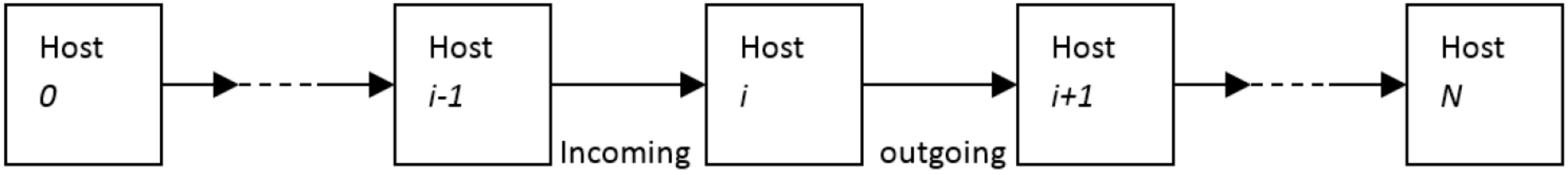
# Network-based SSID (2)

- Unless there are something hidden, otherwise, it doesn’t make sense to access a remote sever via **three or most stepping-stones**
- The number “**three**” is used because most legal apps rarely used three or more stepping-stones to access a remote server
- Therefore, if a user accesses a remote server thru three or more stepping-stones, it is highly suspicious that it is intrusion



# RTTs used to represent the length of a chain

- An ***RTT*** of a packet is its ***round-trip time*** between the source host and the destination host
- There might be many other hosts existing in between the two hosts, routers ignored



## Existing approaches of estimating the length of a connection chain

- Yung’s approach (by K. Yung et al. in 2002)
- Step-function approach (by J. Yang et al. in 2004)
- Clustering-partitioning data mining approach (by J. Yang et al. in 2007)
- ***k***-Means approach (by L. Wang et al. in 2021)

## Our Approach

- Observe the behavior of packets crossover at the hosts in a connection chain
- Verify that the number of crossover packets is proportional to the length of a connection chain
- Use this observation to design a new network-based detection method for SSI
  - **SSID using packet crossover**

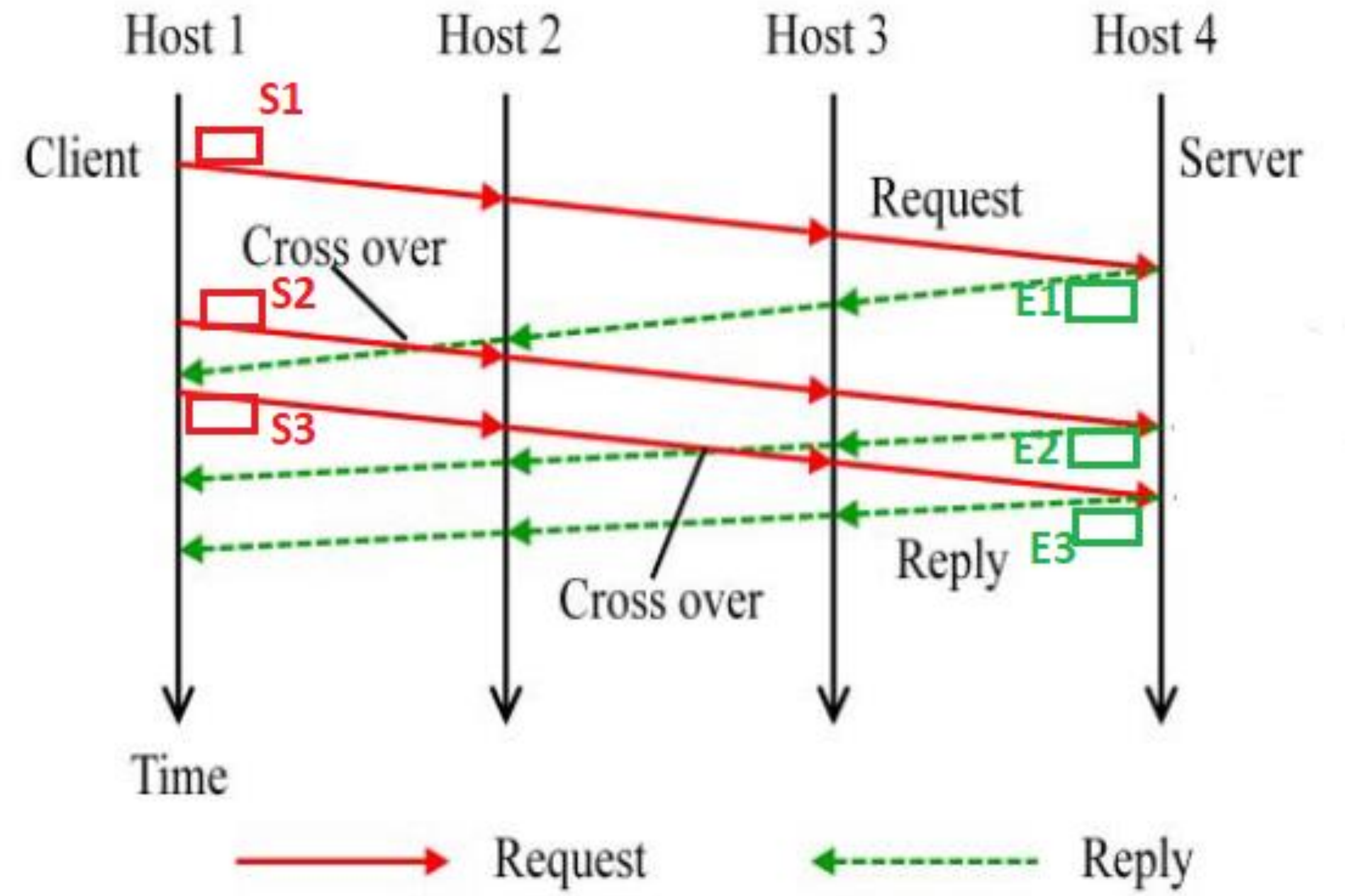
## Send/Echo Packets

- A ***Send packet*** is defined as a TCP data packet sent from Host 0 (attacker host) to Host N (victim host), with the flag bit TCP.Flag.PSH set
- An ***Echo packet*** is defined as a TCP data packet sent from Host N (victim host) to Host 0 (attacker host), with the flag bit TCP.Flag.PSH flag set

# A Matched Pair of Send/Echo Packets

- When a user types a command on a command line in Host 0 (Linux), such as “ls”, it might be sent to Host N (Linux) in 1 or 2 packets
- Suppose that the command “ls” is sent from Host 0 to Host N in 2 separate Send packets: “l” and “s”
- When “l” is typed on the user’s command line, the packet will be sent to Host N. Once this Send packet is echoed, an Echo packet sent back to Host 0, letter “l” will be shown on the terminal of Host 0. Such Send and Echo packets are called ***a matched pair***
- For the other letter “s”, a matched pair can be similarly obtained: a Send “s” and an Echo “s”

# Packet Crossover



## Our Proposed Approach for SSID

- SSID using packet crossover
- **Observation**: the length of a connection chain is strictly increasing with the packet crossover ratio
- This observation is verified by our well-designed network experiment
- Our proposed approach for SSID is based on the above observation
- This approach is referred to as Packet Crossover Detection

**Algorithm 1 (Compute Packet Crossover Ratio)**

**Input**: a TXT file containing packet timestamp, packet type (Send or Echo), and index of Send or Echo
**Output**: Packet Crossover Ratio

```
sendIndex, echoIndex, crossoverCount = 0
while more packets in data capture file:
        if currentPacket is Acknowledgement:
                discard packet
                break

        else if currentPacket is Echo:
                if echoIndex less than sendIndex:
                        crossoverCount+=(sendIndex–echoIndex)
                echoIndex += 1

        else if (currentPacket is Send):
                sendIndex += 1

PacketCrossoverRatio = crossoverCount / (2 * echoIndex)
Print PacketCrossoverRatio
```

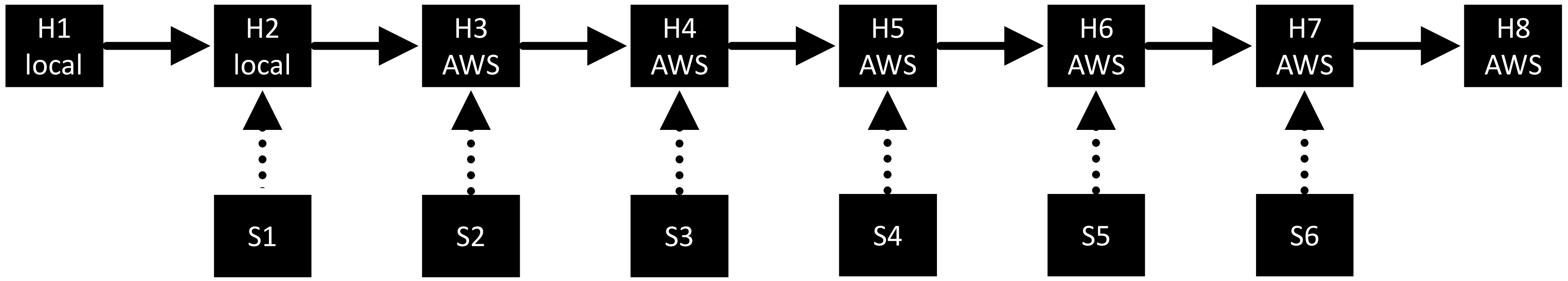
## Packet Crossover Detection

- We first calculate the packet crossover ratio for a connection chain of length 2
- Use this packet crossover ratio as a threshold value for the downstream connections
- Next, we capture network traffic at the sensor from the downstream connections and compute the packet crossover ratio based on the captured data
- It is highly suspicious to have intrusion if the newly obtained packet crossover ratio is greater than the above threshold value

# Packet Crossover Detection (2)

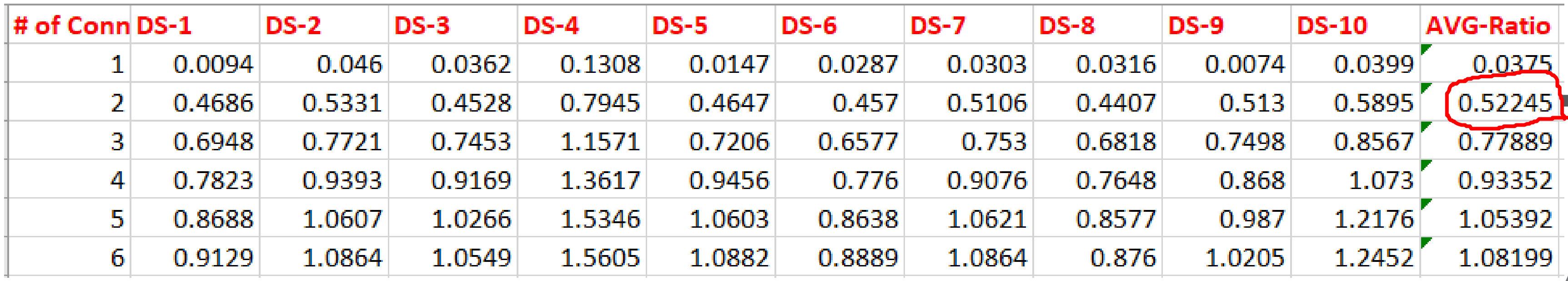
- We perform similar steps in upperstream connections
- First calculate the packet crossover ratio for a connection chain of length 2
- Use this packet crossover ratio as a threshold value for the upperstream connections
- Next, we capture network traffic at the sensor from the upperstream connections and compute the packet crossover ratio based on the captured data
- It is also highly suspicious to have intrusion if the newly obtained packet crossover ratio is greater than the above threshold value

## Experiment setup to verify the observation



- 8 hosts in the connection chain in total
  - 2 local hosts on the same LAN followed by 8 geographically dispersed AWS Servers
  - All accessed from one physical host running 7 terminal windows
  - SSH used for all remote access
  - All packets captured using TCPdump and saved to respective sensor host
  - Process the captured data file for a connection with a specific length

# Experiment Results



| # of Conn | DS-1 | DS-2 | DS-3 | DS-4 | DS-5 | DS-6 | DS-7 | DS-8 | DS-9 | DS-10 | AVG-Ratio |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.0094 | 0.046 | 0.0362 | 0.1308 | 0.0147 | 0.0287 | 0.0303 | 0.0316 | 0.0074 | 0.0399 | 0.0375 |
| 2 | 0.4686 | 0.5331 | 0.4528 | 0.7945 | 0.4647 | 0.457 | 0.5106 | 0.4407 | 0.513 | 0.5895 | 0.52245 |
| 3 | 0.6948 | 0.7721 | 0.7453 | 1.1571 | 0.7206 | 0.6577 | 0.753 | 0.6818 | 0.7498 | 0.8567 | 0.77889 |
| 4 | 0.7823 | 0.9393 | 0.9169 | 1.3617 | 0.9456 | 0.776 | 0.9076 | 0.7648 | 0.868 | 1.073 | 0.93352 |
| 5 | 0.8688 | 1.0607 | 1.0266 | 1.5346 | 1.0603 | 0.8638 | 1.0621 | 0.8577 | 0.987 | 1.2176 | 1.05392 |
| 6 | 0.9129 | 1.0864 | 1.0549 | 1.5605 | 1.0882 | 0.8889 | 1.0864 | 0.876 | 1.0205 | 1.2452 | 1.08199 |

**Packet Crossover Ratio for 2 connections: 0.52245 (intrusion threshold)**

# Our Contributions

- Previously known network-based SSID approaches are either not effective, not efficient since a large number of TCP packets must be captured and processed, or with very limited performance as the length of a connection chain must be pre-determined
- Our proposed Packet Crossover Detection Algorithm can be easily implemented to effectively determine the length of a downstream or upperstream connection chain without
  - any pre-assumption about the length of a connection chain
  - Or requiring a large number of TCP packets to be captured and processed, and thus it is efficient

## Thanks, and Questions!