On the Difficulty of Scalably Detecting Network Attacks

Kirill Levchenko
UC San Diego
klevchen@cs.ucsd.edu

Ramamohan Paturi
UC San Diego
paturi@cs.ucsd.edu

George Varghese
UC San Diego
varghese@cs.ucsd.edu

Background

- Traditionally Firewall uses basic ACL rules to control the network traffic
  - Packet filtering : ACL rules based on packet headers
  - Stateful Inspection : ACL rules based on the specific protocol states.
  - Application Proxy : ACL rules based on the application specific context

Firewall and NIDS

- Traditional Firewall is ineffective in preventing attacks hiding in the legitimate form of traffic.
  - SYN Flooding
  - Port Scanning
  - Connection Hijacking
  - ….

Firewall and NIDS

- People start improving the functions of the Firewall. Many of the new functions come from NIDS.
- Ideally, one can combine NIDS and Firewall altogether such that Firewall will be able to prevent all kind of known malicious traffic.
- Practically, Firewall sits on the critical path, and short processing time is important. NIDS, on the other hand, can work asynchronously and is still quite useful in terms of detecting attacks and providing forensics data.

Firewall and NIDS

- People is still trying to find the balance point between the two.
  - If security is the prime concern, financial resource/engineering efforts can be spent on making a powerful Firewall which has full NIDS capability.
  - Most likely, resource is limited. Therefore, a trade-off is there.

Example of Firewalls with partial NIDS functions

- SYN-Flood Attack
  - Cisco PIX firewall
  - Checkpoint Firewall-1 with SYNDefender Equipped

Table B-15 Items Detected by the Cisco PIX Firewall Info Mediator Abnormal Behavior Definition File

Behavior Type	Message	Behavior Definition	Acquiescence Period	Threshold Adjust	Entry Number in File	Example
Source Overuse	source overuse	Hitting 50+ different destination machines.	30	10	1	A single IP address probing a network for machines to attack.
Machine Overuse	machine overuse	1000+ accepts between two machines.	10	10	2	A machine making a large number of connections to another machine, possibly in an attempt to degrade its performance.
Denial of Service	excessive transfer check	100+ Mb transferred by a single machine.	10	10	3	A machine transferring a large amount of data over the network.
	HTTP port overuse	100+ hits on port 80 from one machine to another.	10	10	6	A machine making many connection attempts to a web server to degrade its performance.
	POP3 port overuse	100+ hits on port 110 from one machine to another.	10	10	8	A machine making many connection attempts to a POP3 server to degrade its performance.
	STMP port overuse	100+ hits on port 25 from one machine to another.	10	10	7	A machine making many connection attempts to a mail server to degrade its performance.
	TCP port overuse	50+ TCP hits on any single port from one machine to another.	10	10	9	A machine making many connection attempts to a TCP service running on a machine to degrade its performance.

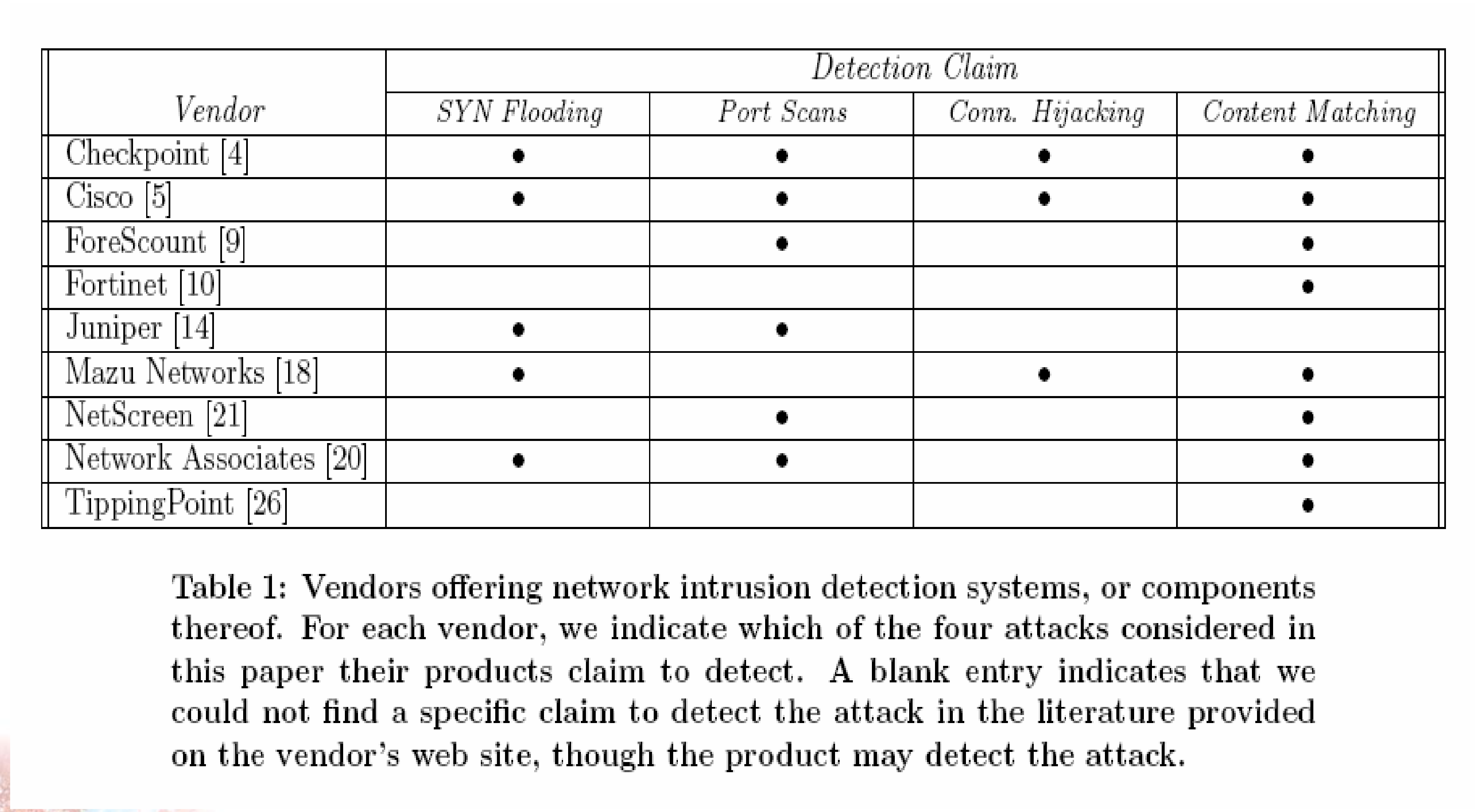
	UDP port overuse	20+ UDP hits on any single port from one machine to another.	10	3	10	A machine making many connection attempts to a UDP service to degrade its performance.
Telnet Overuse	possible password guessing	20+ telnet (port 23) hits from one machine to another.	10	10	11	A user making multiple telnet connection attempts to a machine, possibly attempting to guess the password.
FTP Overuse	possible password guessing	20+ FTP (port 21) hits from one machine to another.	10	10	12	A user making multiple FTP connection attempts, possibly attempting to guess a password.
Port Scanning	TCP port scan	50+ TCP hits on different ports from one machine to another.	10	20	4	Someone using a port scanner (for example, Satan, ISS) to find the TCP services running on a particular machine.
	UDP port scan	50+ UDP hits on different ports from one machine to another.	10	20	5	Someone using a port scanner (for example, Satan, ISS) to find the UDP services running on a particular machine.
Application Overuse	large FTP transfer check	20+ Mb between two machines.	10	10	25	An FTP client transferring a lot of information with a server.
	large HTTP transfer check	10+ Mb between two machines.	10	10	26	A web browser transferring too much data.
	large POP3 transfer check	20+ Mb between two machines.	10	10	27	An e-mail client transferring too much data.
	large SMTP transfer check	50+ Mb between two machines.	10	10	28	Two SMTP servers transferring a large amount of data between themselves.

	large TCP transfer check	10+ Mb between two machines.	10	10	29	A client and a TCP service transferring large amounts of data.
	large UDP transfer check	5+ Mb between two machines.	10	10	30	A client and a UDP service transferring large amounts of data.
ICMP	echo replies	10+ ICMP network echo replies.	10	10	13	Not applicable.
	echo requests	10+ ICMP network echo replies.	10	10	14	Not applicable.
	ICMP network unavailable	5+ ICMP network unreachable events between two machines.	10	10	15	A misconfigured machine trying to access a network that does not exist.
	ICMP port unavailable	5+ ICMP port unreachable events between two machines.	10	10	16	A misconfigured machine trying to access a service (HTTP, SMTP, etc.) on a machine that does not exist.
	ICMP protocol unavailable	5+ ICMP machine unreachable events on a single machine.	10	10	17	A misconfigured machine trying to access a protocol (TCP, UDP, etc.) on a machine that does not exist.
	ICMP machine unavailable	5+ ICMP machine unreachable events on a single machine.	10	10	18	A misconfigured machine trying to access a machine that does not exist.
	ICMP source route failed	2+ ICMP source route events between two machines.	10	10	19	A machine is trying to find its own route through the network, possibly maliciously trying to hide itself.
	ICMP source quench	2+ICMP source quench events between two machines.	10	10	20	A machine is overloading a router and is being instructed to stop.

On the Difficulty of Scalably Detecting Network Attacks

- Gives theoretical lower bound on the space complexity required for detecting some well-known network attacks
- Space is an issue for a Firewall (or a NIDS where detection speed is critical)
  - 1Gbit/sec link may see up to 3 million SYN packets per second
  - High speed SRAM is still expensive.
  - The bandwidth of the memory bus and the latency of addressing logic have to grow up as well

Attacks v.s. Vendors



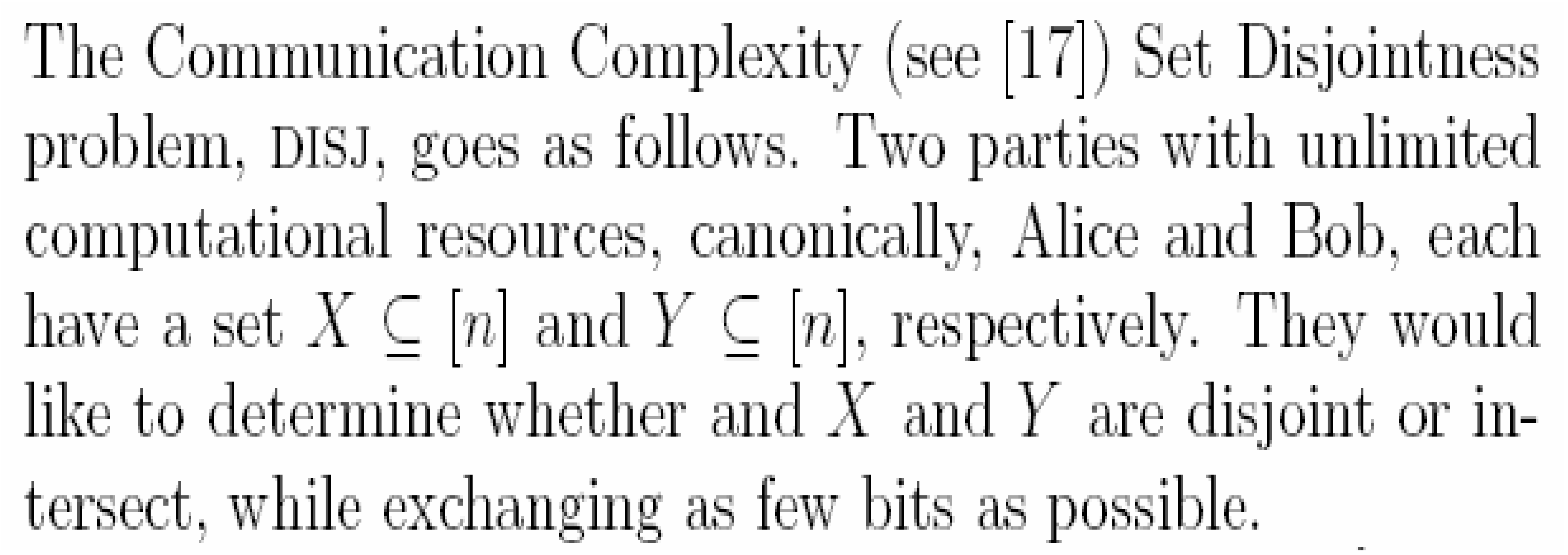
Vendor	*Detection Claim*			
	SYN Flooding	*Port Scans*	*Conn. Hijacking*	*Content Matching*
Checkpoint [4]	●	●	●	●
Cisco [5]	●	●	●	●
ForeScount [9]		●		●
Fortinet [10]				●
Juniper [14]	●	●		
Mazu Networks [18]	●		●	●
NetScreen [21]		●		●
Network Associates [20]	●	●		●
TippingPoint [26]				●

Table 1: Vendors offering network intrusion detection systems, or components thereof. For each vendor, we indicate which of the four attacks considered in this paper their products claim to detect. A blank entry indicates that we could not find a specific claim to detect the attack in the literature provided on the vendor's web site, though the product may detect the attack.

Space Complexity : Is Per-Flow State Necessary?

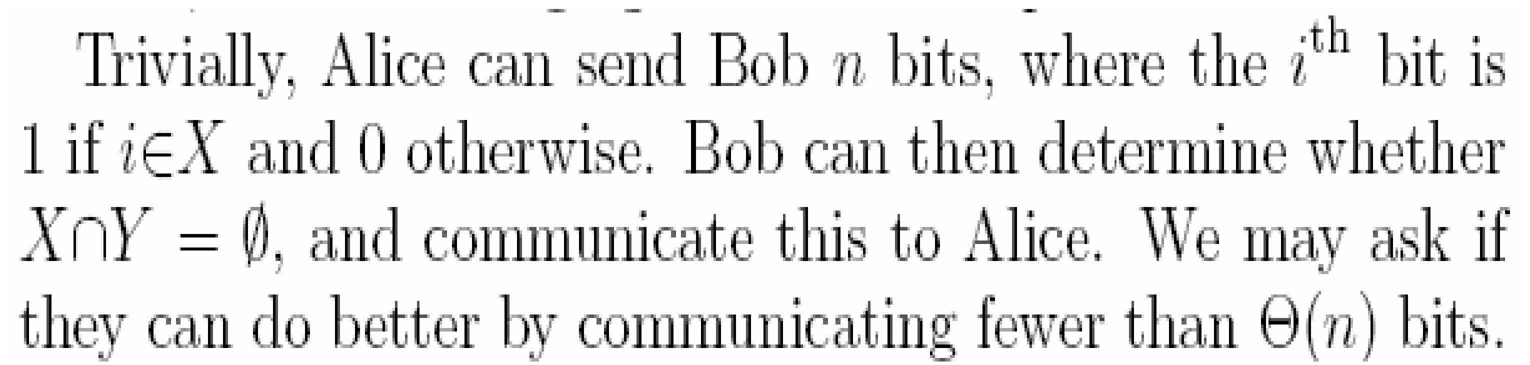
- When Per-Flow State is necessary
  - Means the Firewall (NIDS) has to keep track of m concurrent flows. In terms of space complexity, this means it requires Ω(m) space.
  - Not scalable then.
- When Per-Flow State is not necessary
  - Means the space complexity is o(m).
  - Scalable

Theoretical Tool : Set Disjointness



The Communication Complexity (see [17]) Set Disjointness problem, DISJ, goes as follows. Two parties with unlimited computational resources, canonically, Alice and Bob, each have a set $X \subseteq [n]$ and $Y \subseteq [n]$, respectively. They would like to determine whether and X and Y are disjoint or intersect, while exchanging as few bits as possible.

Set Disjointness Problem



Trivially, Alice can send Bob n bits, where the i^{th} bit is 1 if $i \in X$ and 0 otherwise. Bob can then determine whether $X \cap Y = \emptyset$, and communicate this to Alice. We may ask if they can do better by communicating fewer than $\Theta(n)$ bits.

θ (n) turns out to be exactly the lower bound for this problem. Proof can be found in *E. Kushilevitz and N. Nisan. Communication Complexity, Cambridge University Press, 1997.*

Ingress SYN Flood Detection

- Abstract Problem Formulation
  - Define the packet set P to be [m]×{SYN, FIN}
    - [m] : session identifier. A number in the range [1,m]. In practice, this can be the TCP 4-tuple (src ip/port and dst ip/port).
    - {SYN,FIN} : packet type field, either SYN or FIN
  - Call a packet (x, SYN) in a packet sequence matched if (x, FIN) occurs in the remainder of the sequence.
  - Call a packet unmatched if it is not matched. Intuitively, unmatched SYN packets correspond to unclosed connections.

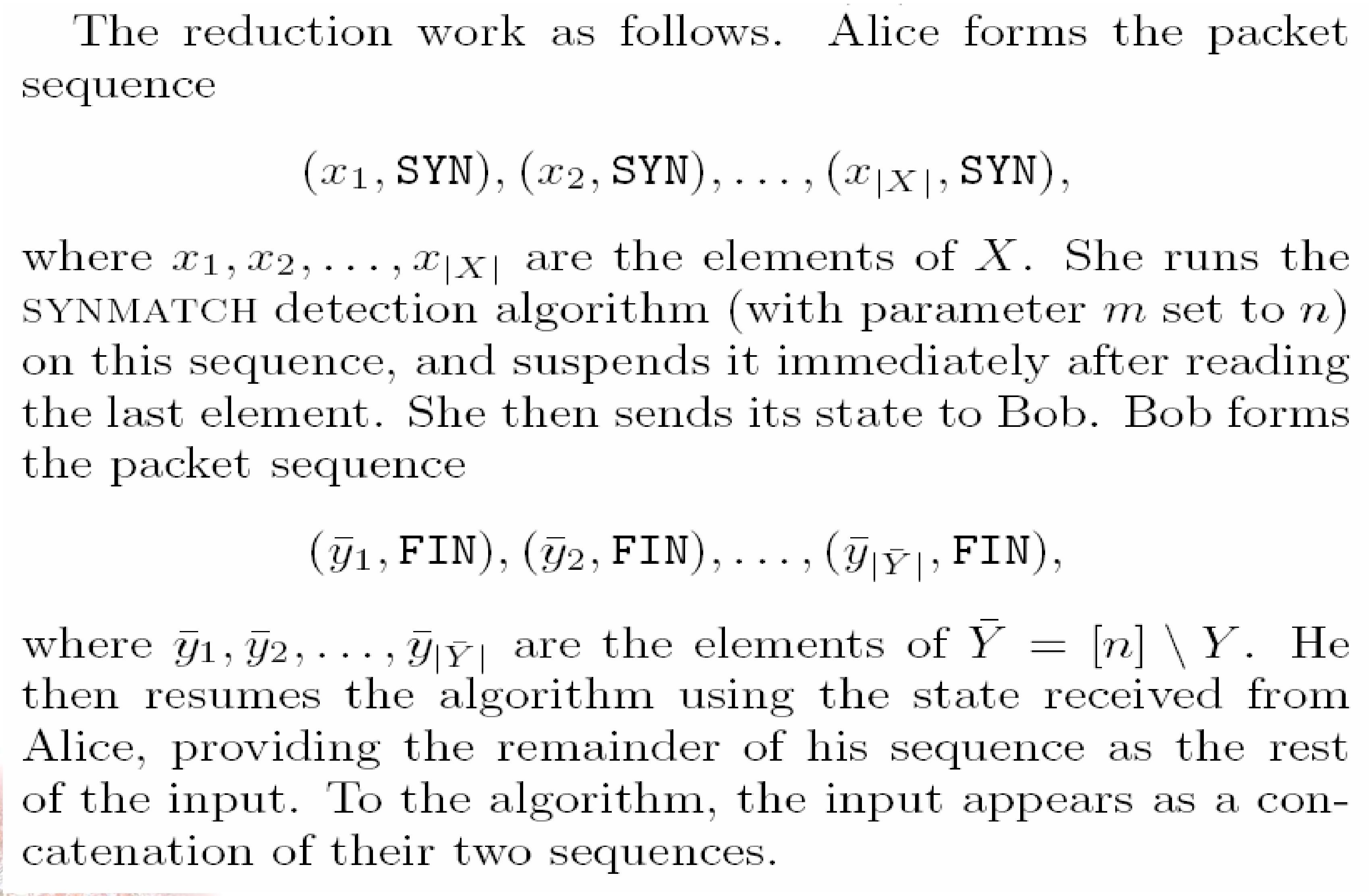
Ingress SYN Flood Detection

- Let *SYNMATCH* be the problem of detecting a packet sequence containing one or more unmatched SYN packets.
- Lower Bound
  – Any algorithm for SYNMATCH must use Ω(m) space.

Ingress SYN Flood Detection : Proof

- By reduction from DISJ (set disjointness problem)
- Show that two parties, Alice and Bob, can decide if two sets, $X \subseteq [n]$ and $Y \subseteq [n]$, held by Alice and Bob respectively, are disjoint using only S bits of communication, where S is the space used by the SYNMATCH detection algorithm.

SYN Flood Detection Proof



The reduction work as follows. Alice forms the packet sequence

$$(x_1, \texttt{SYN}), (x_2, \texttt{SYN}), \ldots, (x_{|X|}, \texttt{SYN}),$$

where $x_1, x_2, \ldots, x_{|X|}$ are the elements of X. She runs the SYNMATCH detection algorithm (with parameter m set to n) on this sequence, and suspends it immediately after reading the last element. She then sends its state to Bob. Bob forms the packet sequence

$$(\bar{y}_1, \texttt{FIN}), (\bar{y}_2, \texttt{FIN}), \ldots, (\bar{y}_{|\bar{Y}|}, \texttt{FIN}),$$

where $\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_{|\bar{Y}|}$ are the elements of $\bar{Y} = [n] \setminus Y$. He then resumes the algorithm using the state received from Alice, providing the remainder of his sequence as the rest of the input. To the algorithm, the input appears as a concatenation of their two sequences.

SYN Flood Detection Proof

- if X and Y are disjoint,
  - For all $x \in X$, there will be a $x \in \overline{Y}$
  - there will be a closed connection consisting of (x, SYN) and (x, FIN) in the aggregate sequence seen by the algorithm.

- If X and Y intersect at some element c
  - there will be a packet (c, SYN) without a matching (c, FIN) packet.

SYN Flood Detection Proof

- Thus, X and Y intersect if and only if the aggregate packet sequence seen by the algorithm contains an unmatched SYN.
- Using the result of the algorithm, Bob can determine if X and Y are disjoint.
- Since the communication complexity is (n) and $m = n$, it follows that $S = \Omega(m)$.

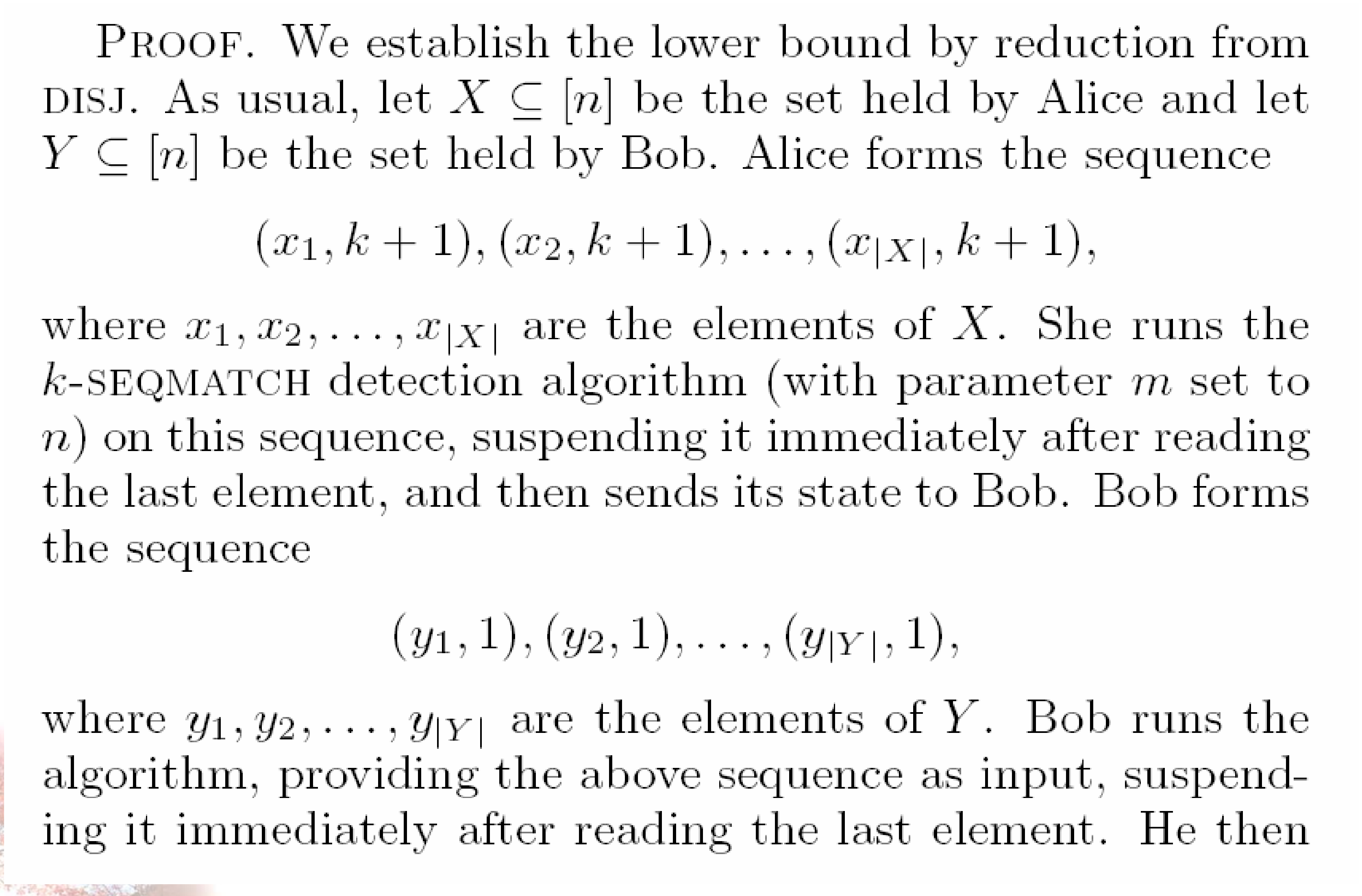
Egress SYN Flood Detection

- SYN Flood can also be detected by considering the difference between the number of SYN packets entering the network and the FIN packets leaving the network.
  - Scalable solution
  - Can we trust all the hosts inside the protected zone?

Ingress TCP Connection Hijacking Detection

- When U sends data to V via TCP, each of the packet contains a monotonically increasing sequence number.
- V accepts only those packets within a certain window [t+1,t+w). A packet with a sequence number out of this range is considered as ‘out-of-order’ and is considered as an indication of a connection hijacking attack.
- Let the k-SEQMATCH be the problem of determining if the packet stream contains a session with k or more out-of-order packets.
- Lower Bound
  - Any algorithm for K-SEQMATCH must use Ω(m/k) space

Ingress TCP Connection Hijacking Detection - Proof



PROOF. We establish the lower bound by reduction from DISJ. As usual, let $X \subseteq [n]$ be the set held by Alice and let $Y \subseteq [n]$ be the set held by Bob. Alice forms the sequence

$$(x_1, k+1), (x_2, k+1), \ldots, (x_{|X|}, k+1),$$

where $x_1, x_2, \ldots, x_{|X|}$ are the elements of X. She runs the k-SEQMATCH detection algorithm (with parameter m set to n) on this sequence, suspending it immediately after reading the last element, and then sends its state to Bob. Bob forms the sequence

$$(y_1, 1), (y_2, 1), \ldots, (y_{|Y|}, 1),$$

where $y_1, y_2, \ldots, y_{|Y|}$ are the elements of Y. Bob runs the algorithm, providing the above sequence as input, suspending it immediately after reading the last element. He then sends the state of the k-SEQMATCH algorithm back to Alice, who resumes it on the sequence

$$(x_1, k+2), (x_2, k+2), \ldots, (x_{|X|}, k+2),$$

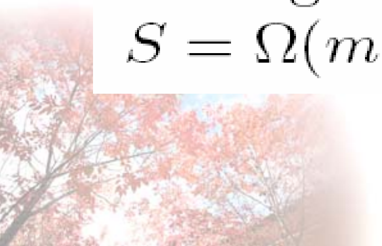
again suspending it immediately after reading the last element, and then sends the state to Bob. Bob resumes th! e simulation on the sequence

$$(y_1, 2), (y_2, 2), \ldots, (y_{|Y|}, 2).$$

They do this k times altogether, with Bob finishing the execution of the algorithm and determining its output. If X and Y are disjoint, then the aggregate packet sequence provided to the algorithm as input contains sessions whose sequence numbers are strictly increasing. However if X and Y intersect, say at an element c, then the packet sequence will contain the session

$$(c, k+1), (c, 1), (c, k+2), (c, 2), \ldots, \\ (c, k+k), (c, k),$$

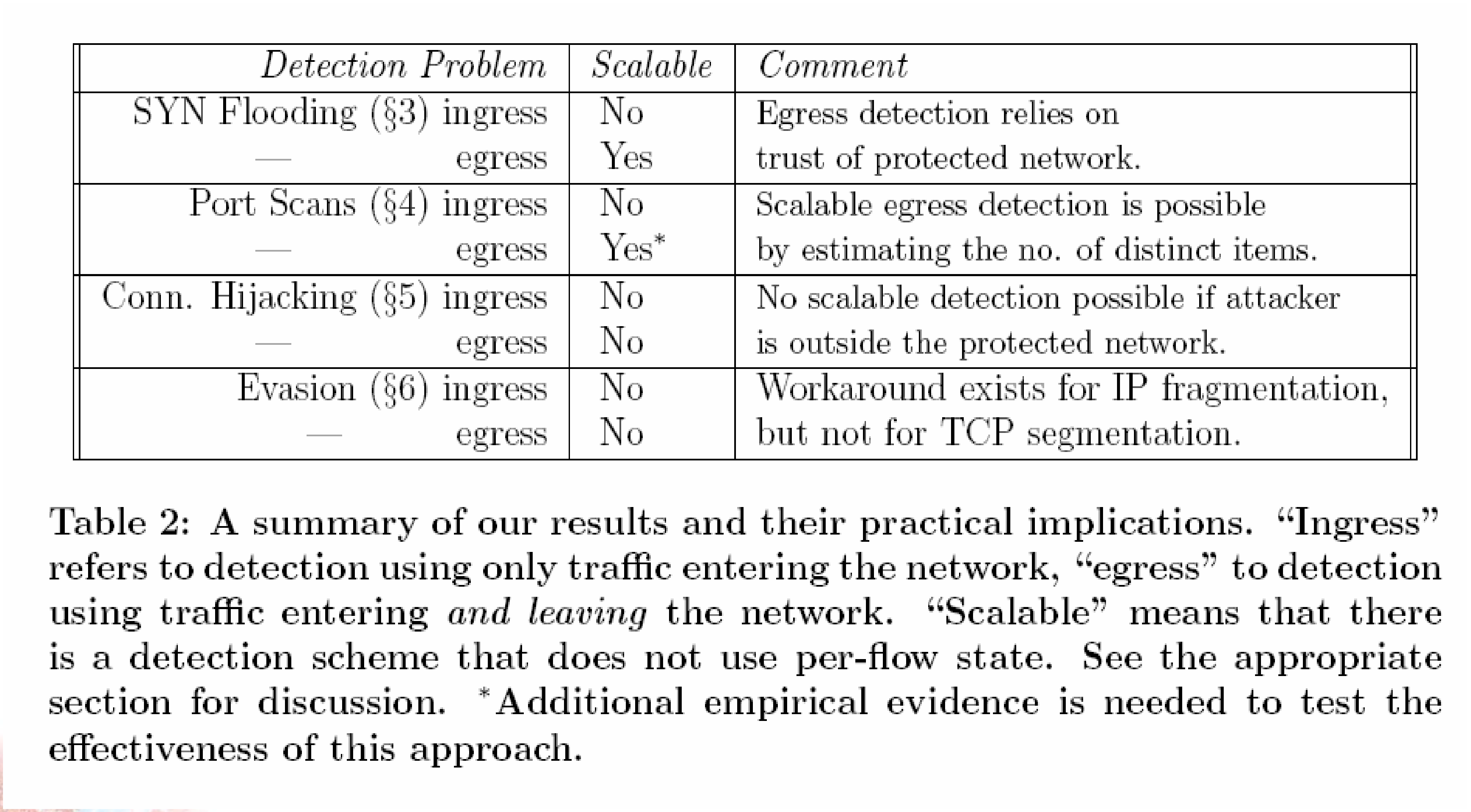
which contains exactly k out-of-order packets. Thus, Bob can use the result of the k-SEQMATCH to determine if X and Y are disjoint. Since Alice and Bob exchanged the state of the algorithm $2k - 1$ times, we have $S(2k - 1) = \Omega(n)$, so $S = \Omega(m/k)$. $\square$



Egress TCP Connection Hijacking Detection

- TCP specification specifies that the appropriate response to a segment sequence number outside the receiver’s window to be an acknowledgement of the last valid packet payload.
- Thus, if sender U sees an excessive number of ACKs from V compared to the number of packets he sent, it might be an indication of a hijacking attack.
- This requires a per-flow ‘counter’ to track. Not scalable.

Summary



Detection Problem	*Scalable*	*Comment*
SYN Flooding (§3) ingress	No	Egress detection relies on
— egress	Yes	trust of protected network.
Port Scans (§4) ingress	No	Scalable egress detection is possible
— egress	Yes*	by estimating the no. of distinct items.
Conn. Hijacking (§5) ingress	No	No scalable detection possible if attacker
— egress	No	is outside the protected network.
Evasion (§6) ingress	No	Workaround exists for IP fragmentation,
— egress	No	but not for TCP segmentation.

Table 2: A summary of our results and their practical implications. "Ingress" refers to detection using only traffic entering the network, "egress" to detection using traffic entering *and leaving* the network. "Scalable" means that there is a detection scheme that does not use per-flow state. See the appropriate section for discussion. *Additional empirical evidence is needed to test the effectiveness of this approach.

Conclusion

- Based on existing architectures, ingress flow based detection of some of the well-known network attacks proves to be expensive (non scalable)
  - This is an analysis under worst case conditions !
  - However, in the war against cyber attacks, worst case does happen.
- NIDS/Firewall design can be changed to overcome the fundamental obstacles
  - A distributed architecture design
  - Exploit more features from current infrastructure

- http://www.theregister.co.uk/2001/08/25/defending_against_synflood_dos_attacks/
- http://www.cisco.com/en/US/products/sw/netmgtsw/ps5477/products_administration_guide_chapter09186a0080238c00.html