Improving TCP/IP Security Through Randomization Without Sacrificing Interoperability

Michael J. Silbersack

November 26th, 2005

http://www.silby.com/eurobsdcon05/

What does that title mean?

- TCP was not designed with an eye towards security
- There are many attacks against TCP which can be prevented without resorting to encryption or keyed hashes
- Sometimes the obvious fix to a TCP security problem leads to interoperability problems

Cases of Security Breaking Interoperability

- Implementation of OpenBSD ISN scheme in FreeBSD
- Implementation of zeroed IP ID values in Linux
- Implementation of port randomization in FreeBSD

Topics to discuss

- IP ID values
- Ephemeral Port Randomization
- TCP Initial Sequence Numbers
- TCP Timestamps

IP ID Values

- IP ID values are used for the purpose of IP fragment reassembly
- If IP ID values are repeated too quickly, two different packets can be reassembled together, creating a corrupt packet
- Operating systems traditionally use a single system-wide counter which increments by one for each packet sent
- This leaks information about a host's level of traffic and a host's identity

IP ID Fixes

- Use a ID value of 0 on fragments with the DF (don't fragment) bit set
  - Tried by Linux, some firewalls / NAT machines were found to strip DF bits, causing a stream of fragmented packets that all had the same ID value
- Store per-IP state and use a separate counter for each IP (Linux)
- Use a LCG to generate psuedo-random ID values that have a relatively long time between repeats (OpenBSD)

IP ID fixes - simpler

- The danger of quickly repeated ID values has been overstated
- Repeated values only cause packet corruption in cases where packets were lost or reordered on the network
- If two packets are misassembled together, the TCP/UDP checksum will detect the corruption and throw the packet away
- Therefore, the worst case is that a single packet drop causes two packets to be dropped

Ephemeral Port Randomization

- Ephemeral ports have traditionally been allocated in a sequential fashion, making it easy for an attacker to figure out the next port to be used
  - One positive property of this behavior is that the period of time before ephemeral port reuse was maximized
- Ephemeral port randomization makes spoofing attacks more difficult, nearly 2^15 times more difficult if a large ephemeral port space is used
  - Ports can be reused a few milliseconds later

Port Randomization Problems

- After FreeBSD enabled port randomization, one user with a FreeBSD machine running squid in front of a FreeBSD machine running Apache started to notice that some connections were failing
- Disabling port randomization solved the problem for him
- One of the failure cases was caught; a port was being reused in 15ms

Port Randomization Problems Continued

- The glitch is almost certainly a bug in the FreeBSD TCP stack – but it is one that would never happen without port randomization
- Do other operating systems have lingering bugs like this that port randomization will expose?
- For now, FreeBSD turns off port randomization when the connection rate exceeds a certain threshold
- A better solution is still being sought

One troubled connection

```
17:31:15.374266 XX.XX.XX.XX.1501 > YY.YY.YY.YY.80: F
4253937378:4253937378(0) ack 1547682423 win 8688
<nop,nop,timestamp 152193515 295129972> (DF)

17:31:15.374537 YY.YY.YY.YY.80 > XX.XX.XX.XX.1501: . ack
4253937379 win 57920 <nop,nop,timestamp 295129972
152193515> (DF)

17:31:15.389416 XX.XX.XX.XX.1501 > YY.YY.YY.YY.80: S
4253971599:4253971599(0) win 8192 <mss 1460,nop,wscale
0,nop,nop,timestamp 152193545 0> (DF)

17:31:15.389598 YY.YY.YY.YY.80 > XX.XX.XX.XX.1501: R
1547682423:1547682423(0) ack 4253937379 win 57920 (DF)

17:31:15.389604 YY.YY.YY.YY.80 > XX.XX.XX.XX.1501: R 0:0(0)
ack 4253971600 win 0 (DF)
```

TCP Connections: A Quick Review

- A TCP connection is identified by a 4-tuple:
  - Source IP
  - Source Port
  - Destination IP
  - Destination Port
- The destination port is usually a well known port such as port 80 on a web server
- The source port is usually chosen from the ephemeral port range by the operating system

TCP Sequence Numbers

- TCP uses 32-bit sequence numbers to track how much data has been transmitted
- Each direction's sequence number is independent, and is chosen by the operating system at that end of the connection
- A sliding window is used, typically around 32K in size. Packets with sequence numbers that fall into this window are accepted.

A Sample Connection

IP 10.1.1.9.65500 > 10.1.1.237.80: S 2766364594:2766364594(0) win 65535 <mss 1460,sackOK,wscale 1,timestamp 146016542 0>
IP 10.1.1.237.80 > 10.1.1.9.65500: S 4027082585:4027082585(0) ack 2766364595 win 5792 <mss 1460,sackOK,timestamp 80799562 146016542,wscale 2>
IP 10.1.1.9.65500 > 10.1.1.237.80: . ack 4027082586 win 33304 <timestamp 146016542 80799562>
IP 10.1.1.9.65500 > 10.1.1.237.80: P 2766364595:2766364664(69) ack 4027082586 win 33304 <timestamp 146016542 80799562>
IP 10.1.1.237.80 > 10.1.1.9.65500: . ack 2766364664 win 1448 <timestamp 80799563 146016542>
IP 10.1.1.237.80 > 10.1.1.9.65500: P 4027082586:4027083050(464) ack 2766364664 win 1448 <timestamp 80799565 146016542>
IP 10.1.1.9.65500 > 10.1.1.237.80: F 2766364664:2766364664(0) ack 4027083050 win 33304 <timestamp 146016542 80799565>
IP 10.1.1.237.80 > 10.1.1.9.65500: F 4027083050:4027083050(0) ack 2766364665 win 1448 <timestamp 80799566 146016542>
IP 10.1.1.9.65500 > 10.1.1.237.80: . ack 4027083051 win 33303 <timestamp 146016542 80799566>

Classes of Initial Sequence Numbers

- Time based
- Random Positive Increments
- Random
- RFC 1948

RFC 1948

- Steven Bellovin describes a near-perfect solution to this problem in RFC 1948
- A system-wide secret is generated and stored at boot time
- A system-wide time counter is incremented at a constant rate
- Initial sequence numbers are generated as follows:
- ISN = time + MD5(srcip, srcport, dstip, dstport, secret)

One Flaw In RFC 1948

- For a certain tuple, sequence numbers are fully predictable until the system reboots
- Example:
  - A SMTP server uses RFC 1948 for all ISNs
  - Spammer uses an AOL account to connect to that SMTP server, records ISN values
  - Spammer can now spoof connections from that AOL IP to the SMTP server until it reboots
- If the hash is rekeyed, then monotonicity is broken – so we can't fix it that way

IP Spoofing

- An exact guess at the ISN in a SYN-ACK allows you to spoof a connection
- As you can only send data, this only serves a purpose against rsh/rlogin
- This attack was easy when time-based sequence numbers were used
- Random positive increments make this attack more difficult, but not impossible

Connection corruption

- Attacks well described in “Slipping in the Window” by Paul Watson
- The following attacks work because TCP stacks generally accept packets that have a seq # value that is anywhere in the sliding window
  - RST attacks
  - SYN attacks
  - Data injection attacks

How to defeat these attacks

- Ensure that the sequence numbers of each connection are entirely independent of one another
  - Attackers will have to spoof the entire sequence space
- Implement the countermeasures described in tcpsecure so that not just any sequence number in the window is accepted

Interoperability concerns

- Initial sequence numbers can be randomized...
  - Except when the same 4-tuple is reused within a short period of time
- Theoretical reasoning: If the same 4-tuple is reused and the same sequence space is overlapped, old duplicate packets may corrupt the connection
- Practical reason: TIME_WAIT socket recycling rules

The Time Wait State

- During a normal TCP socket close, the side of the connection that starts to close the connection will enter the time wait state
- The purpose of the time wait state is to ignore any old (or duplicate) packets still in the network
- BSD-derived TCP/IP stacks will recycle a TIME_WAIT socket only if the ISN in the SYN packet is greater than the sequence number at the end of the previous connection

Empirical TIME_WAIT recycling results

- In order to verify the monotonically increasing sequence number requirement, a FreeBSD machine was modified so that it would generate monotonically decreasing sequence numbers
- The results showed types of behavior that were not expected

Empirical TIME_WAIT results

- Cisco IOS 12.3: All connections accepted
- FreeBSD: All connections delayed
- Linux 2.6.11-FC4: All connections accepted due to a heuristic + tcpsecure behavior
- NetBSD 2.0.2: tcpsecure behavior
- OpenBSD 3.7: tcpsecure behavior
- Windows XP SP2: All connections delayed

The tcpsecure Behavior

59.515622 IP 10.1.1.203.80 > 10.1.1.9.65527: F 993959099:993959099(0) ack 4086058688 win 33580<nop,nop,timestamp 2 146055920>

59.515742 IP 10.1.1.9.65527 > 10.1.1.203.80: . ack 993959100 win 33303 <nop,nop,timestamp 146056026 2>

65.657308 IP 10.1.1.9.65527 > 10.1.1.203.80: S 4078507753:4078507753(0) win 65535 <mss 1460,nop,nop,sackOK,nop,wscale 1,nop,nop,timestamp 146056640 0>

65.657610 IP 10.1.1.203.80 > 10.1.1.9.65527: . ack 4086058688 win 33580 <nop,nop,timestamp 14 146056640>

65.657741 IP 10.1.1.9.65527 > 10.1.1.203.80: R 4086058688:4086058688(0) win 0

68.655831 IP 10.1.1.9.65527 > 10.1.1.203.80: S 4078507753:4078507753(0) win 65535 <mss 1460,nop,nop,sackOK,nop,wscale 1,nop,nop,timestamp 146056940 0>

68.655914 IP 10.1.1.203.80 > 10.1.1.9.65527: S 2006422470:2006422470(0) ack 4078507754 win 32768 <mss 1460,nop,wscale 0,nop,nop,timestamp 0 146056940>

TCP Security / Interoperability Summary

- For security purposes, sequence numbers must be unrelated to the sequence numbers of any other connection
- For interoperability purposes, ISNs in SYN packets must be monotonically increasing
  - If this principle is violated, connection establishment may stall whenever a TCP connection is reused
  - If port randomization is used, port reuse may be a common occurrence

A Sequence Number Survey

- Many ISN surveys have been done, but they generally do not consider
  - How RFC 1948 works
  - That OSes may generate SYN and SYN-ACK packets in different manners
- This survey focuses on a small range of ephemeral ports and watches how they behave

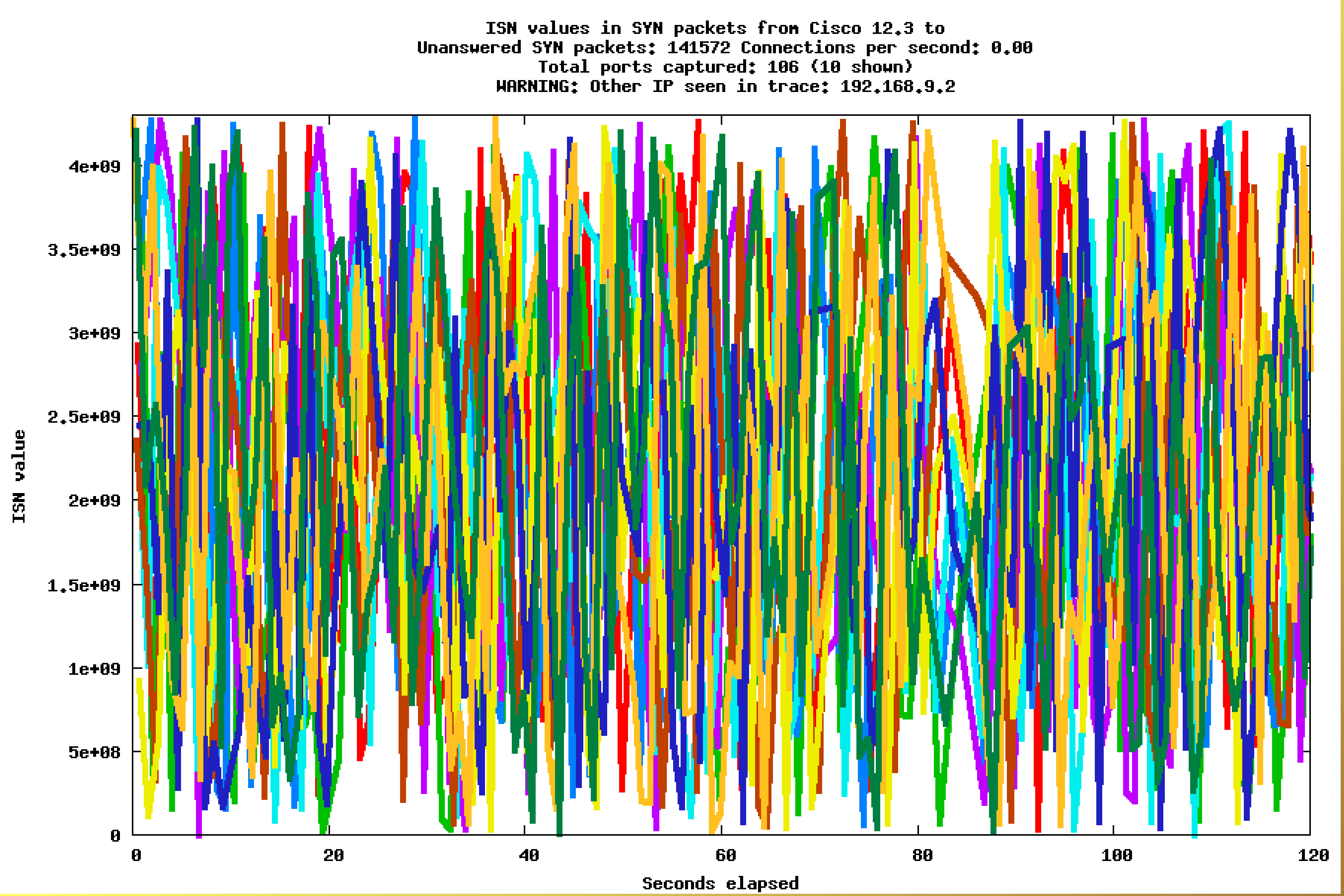
The Graphs

- The graphs you're about to see were generated by running a http benchmark utility against a web server
- Tests were run in both directions so that the ISN values in SYN and SYN-ACK packets could both be observed
- Each line is a series of initial sequence numbers captured in SYN / SYN-ACK packets for a certain sip:sport:dip:dport tuple

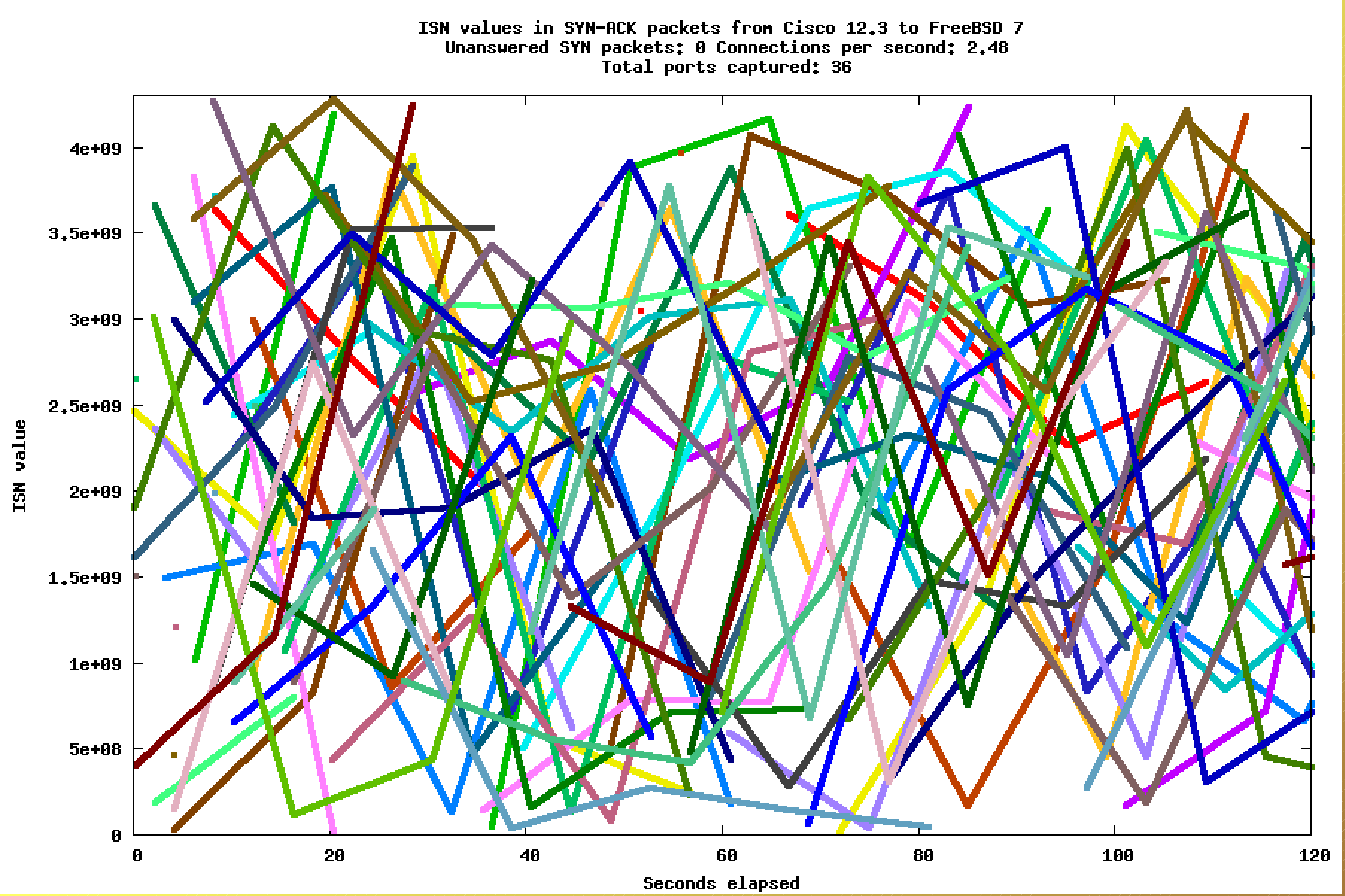
The Graphs (continued)

- Caveat 1: I used different http test tools, and didn't keep the connection rate the same during each test. This should not affect the results...
  - Except for random positive increments, which would change their slope based on the connection rate
- Caveat 2: For OSes that I do not have the source code to, the algorithm could be different than it appears to be.

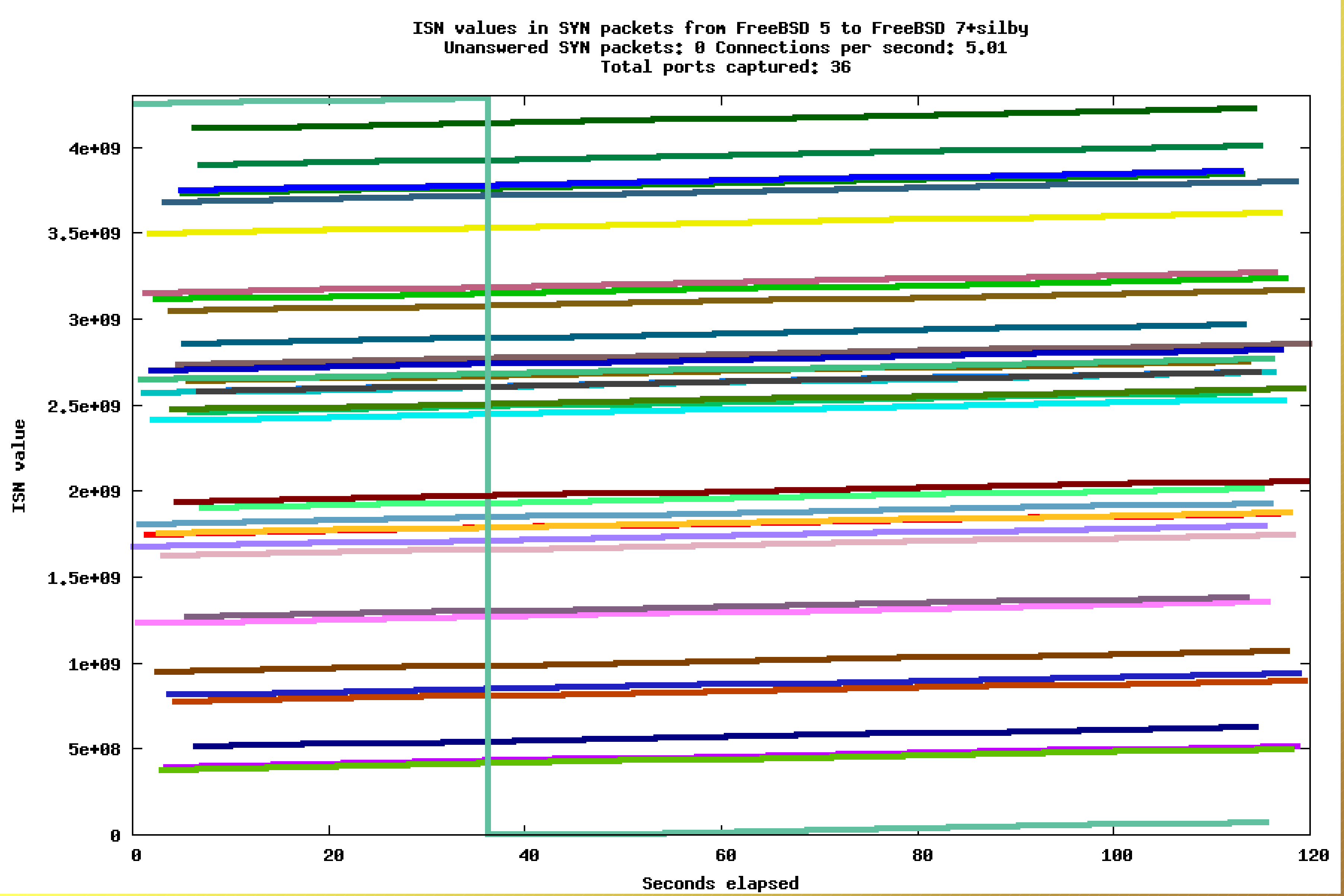
Cisco IOS 12.3 SYN



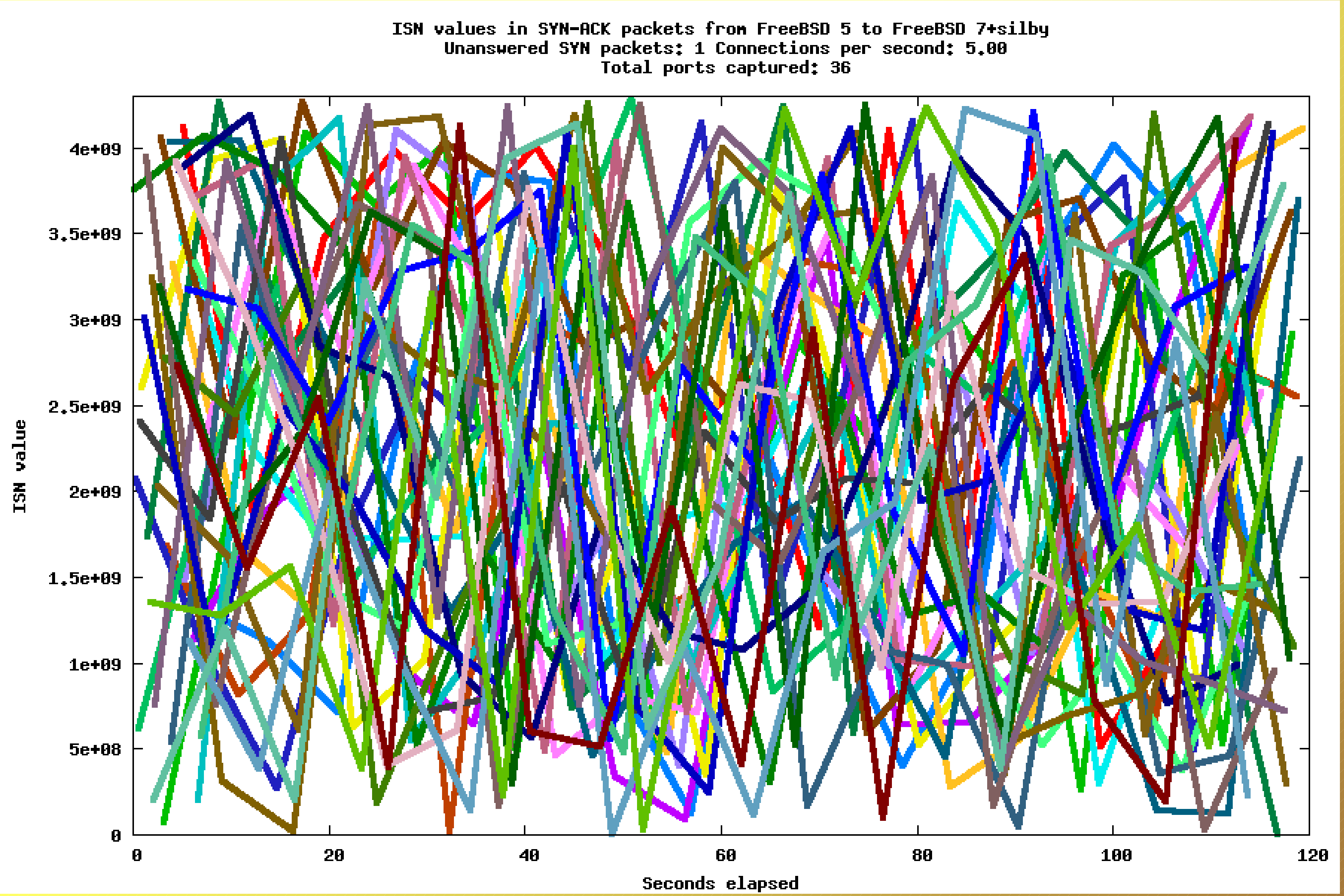
Cisco IOS 12.3 SYN-ACK



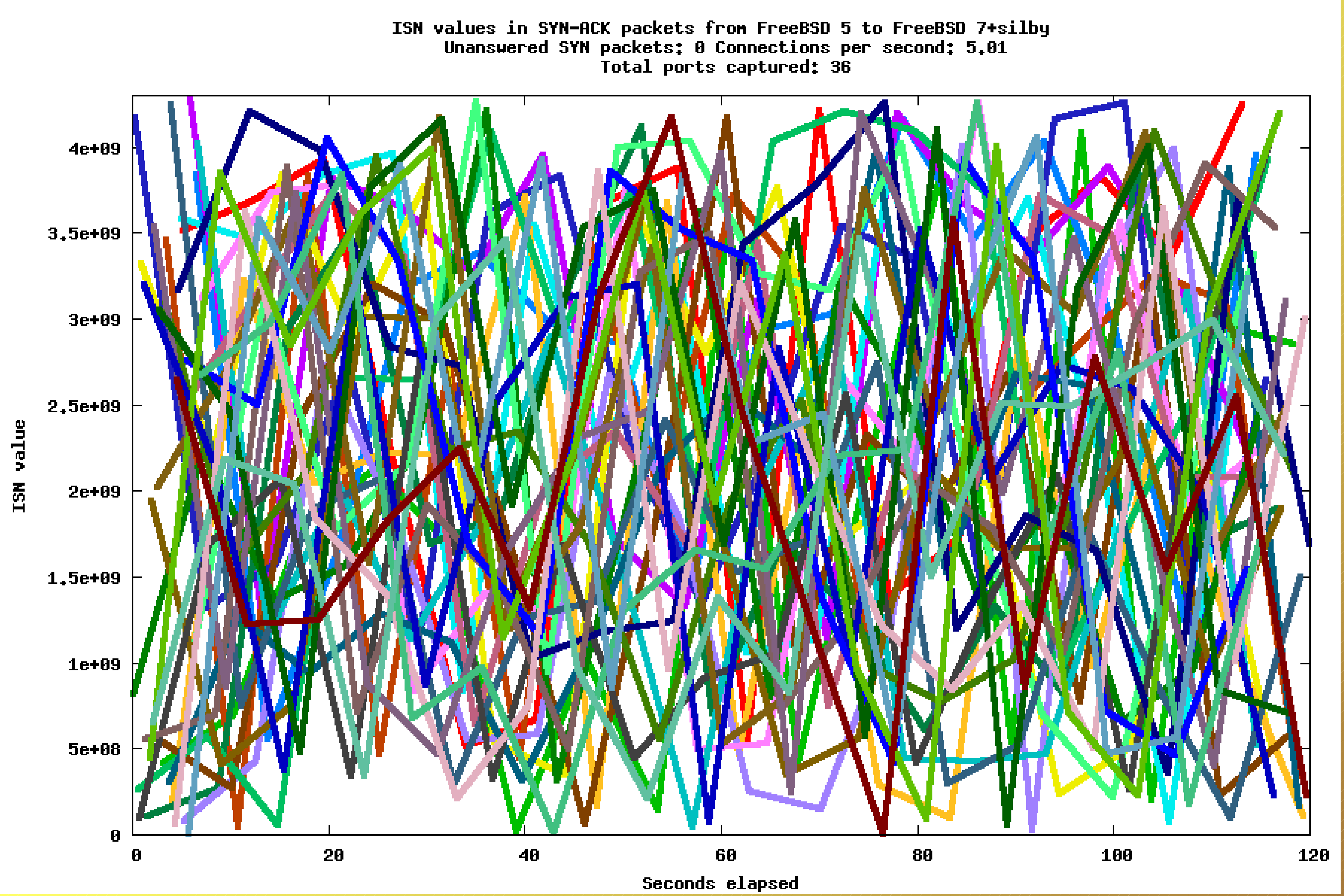
FreeBSD SYN


ISN values in SYN packets from FreeBSD 5 to FreeBSD 7+silby
Unanswered SYN packets: 0 Connections per second: 5.01
Total ports captured: 36
ISN value
4e+09
3.5e+09
3e+09
2.5e+09
2e+09
1.5e+09
1e+09
5e+08
0
0
20
40
60
80
100
120
Seconds elapsed

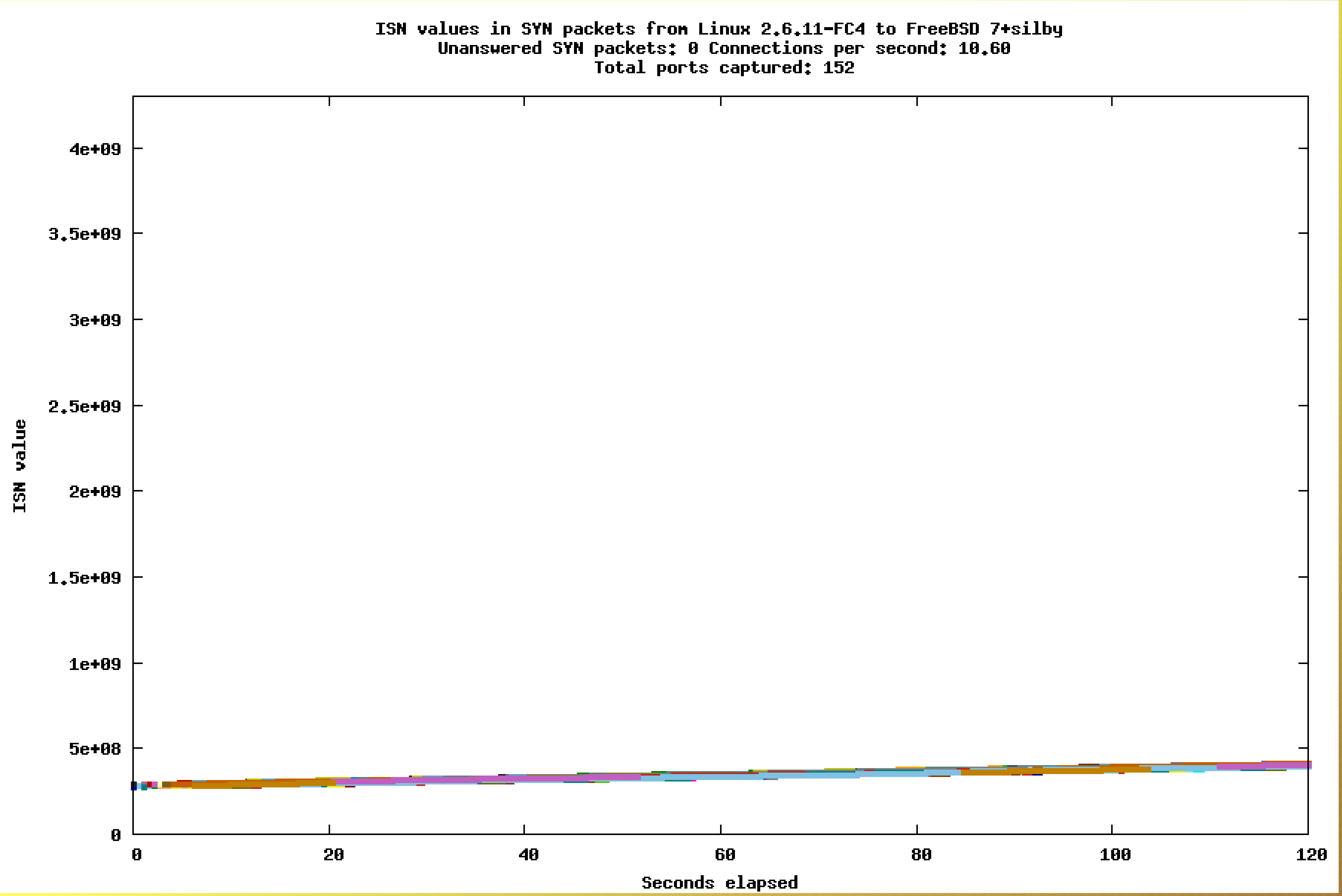
FreeBSD SYN-ACK (no cookies)



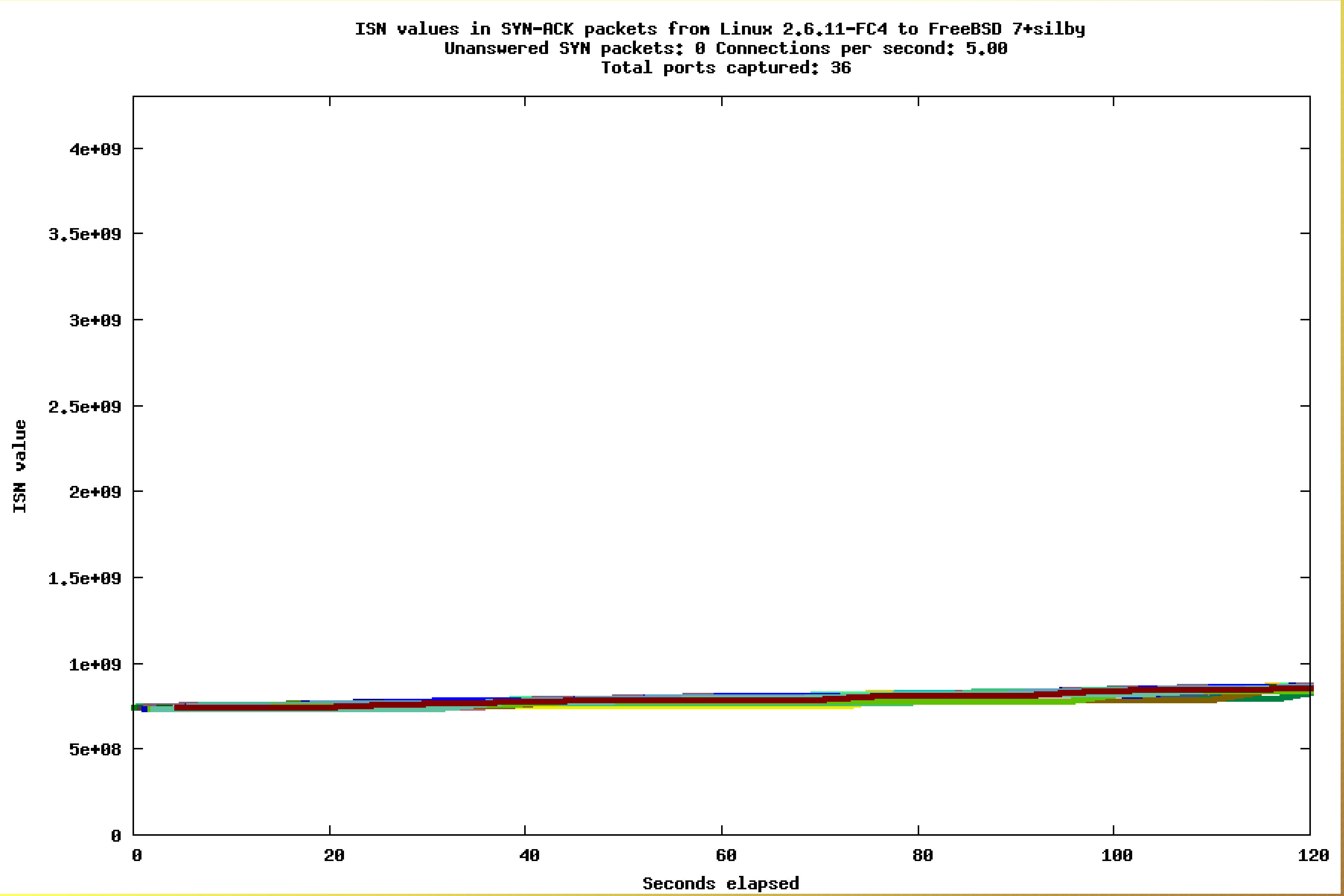
FreeBSD SYN-ACK (cookies!)



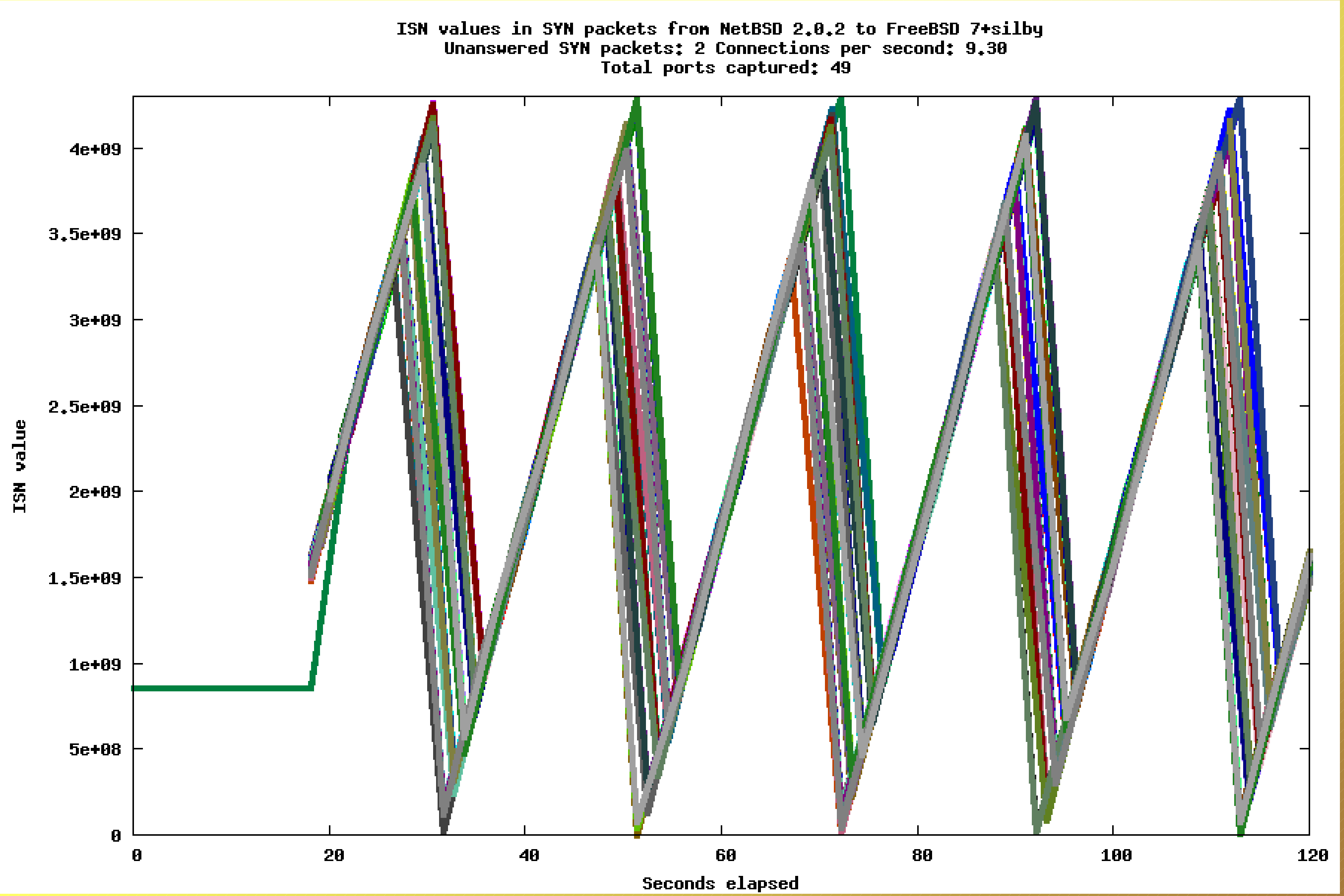
Linux 2.6.11-FC4 SYN



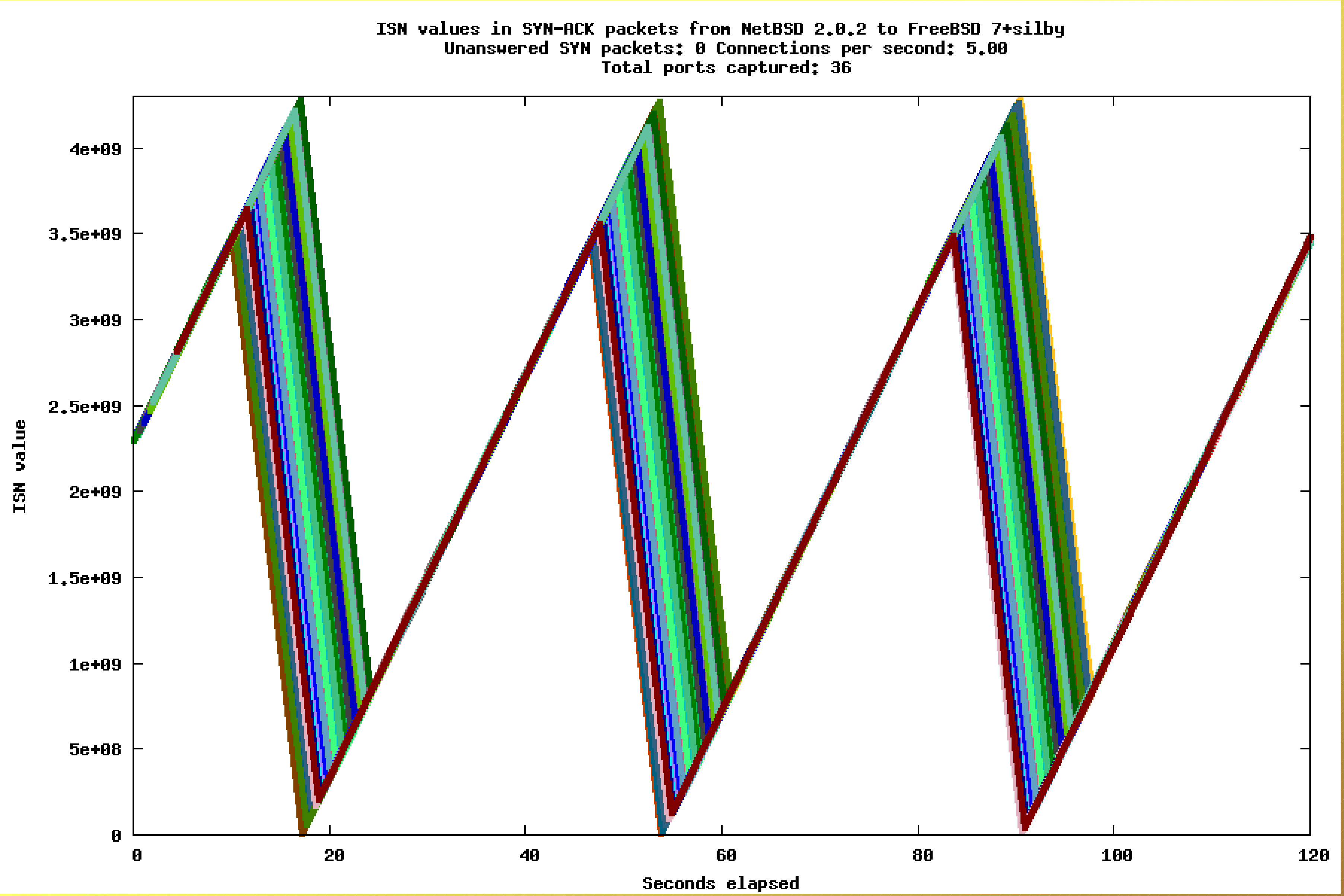
Linux 2.6.11-FC4 SYN-ACK



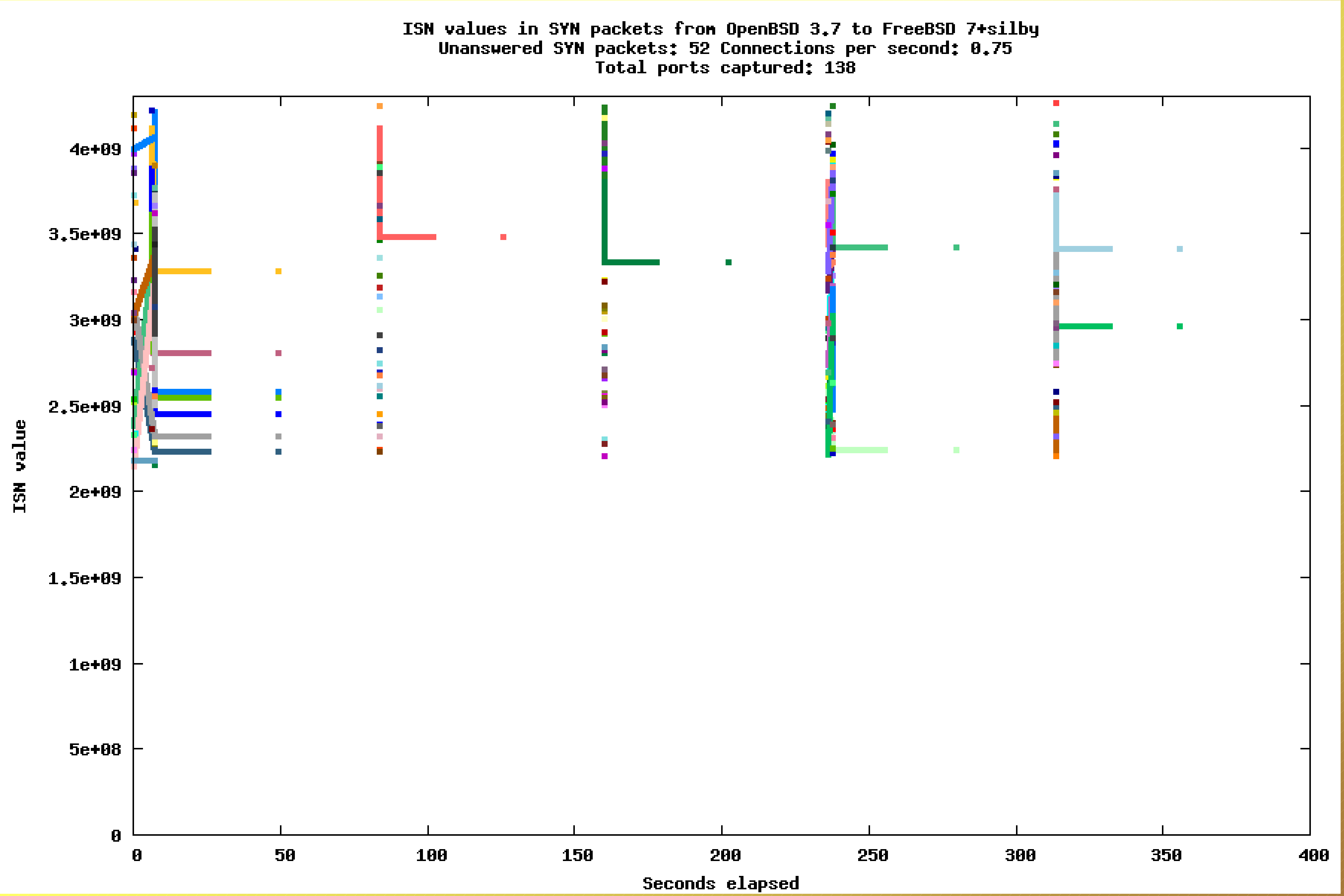
NetBSD 2.0.2 SYN



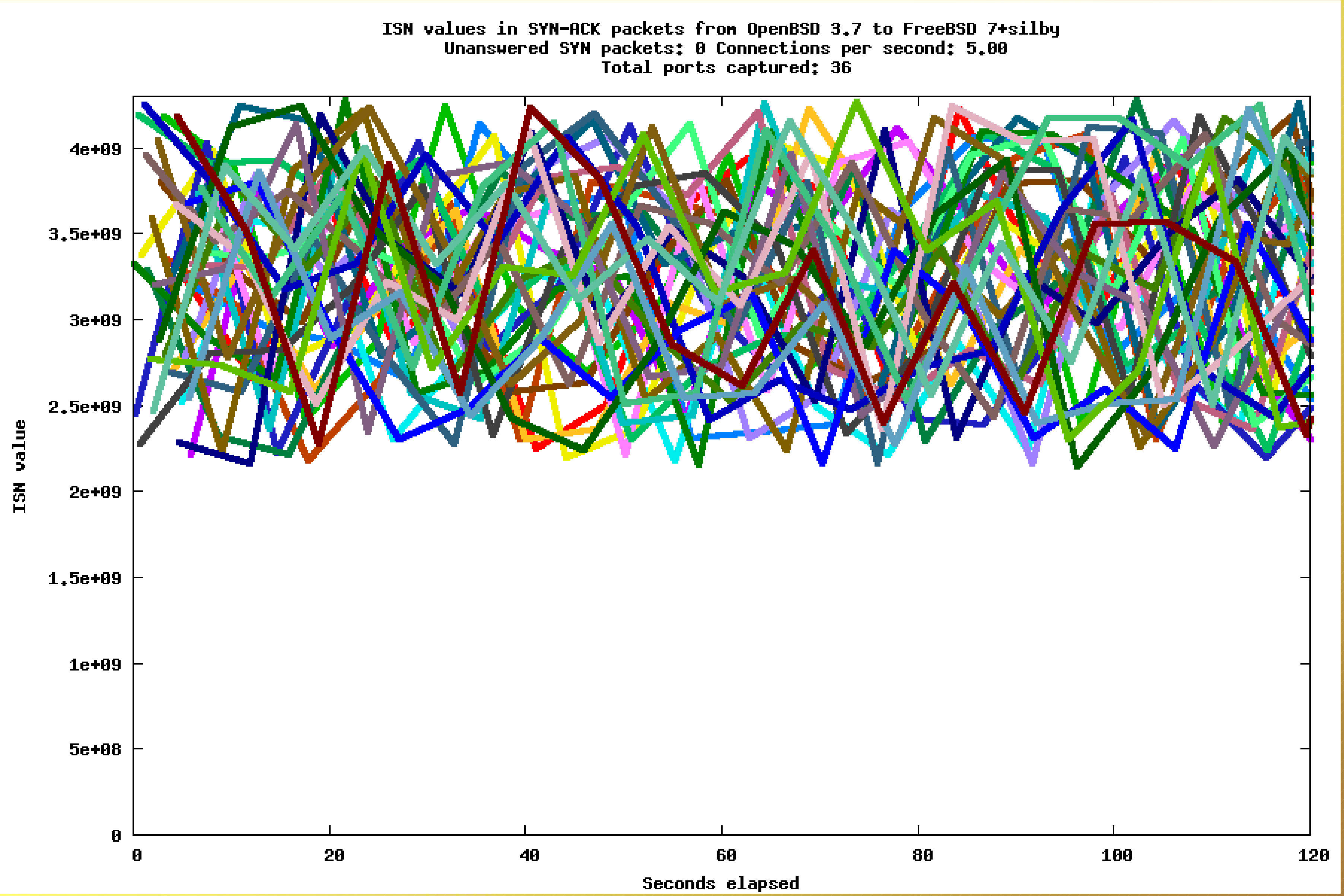
NetBSD 2.0.2 SYN-ACK



OpenBSD 3.7 SYN



OpenBSD 3.7 SYN-ACK



OpenBSD's algorithm

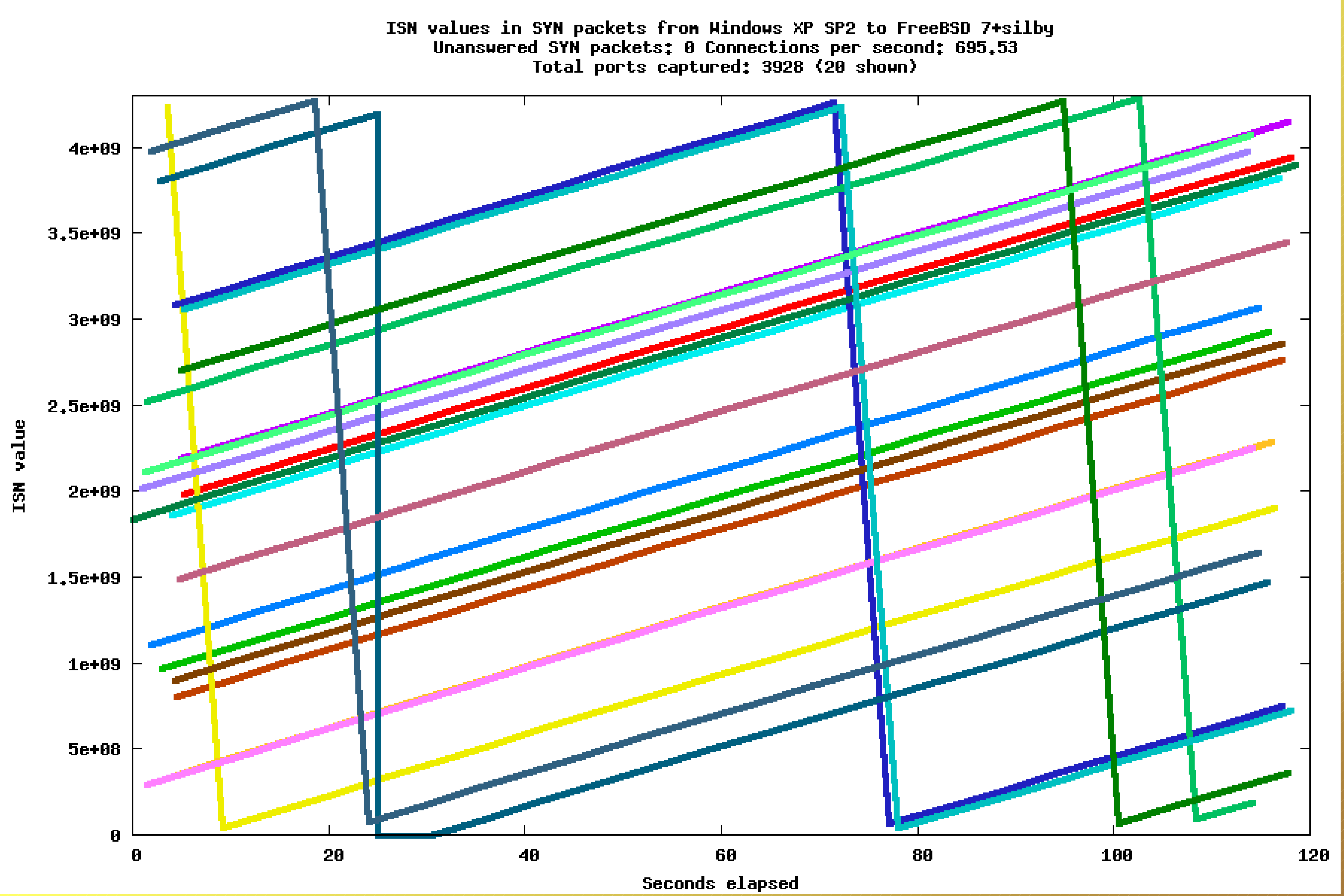
ISN = ((PRNG(t)) << 16) + R(t)

PRNG(t) = a pseudo-randomly ordered list of sequentially-generated 16-bit numbers

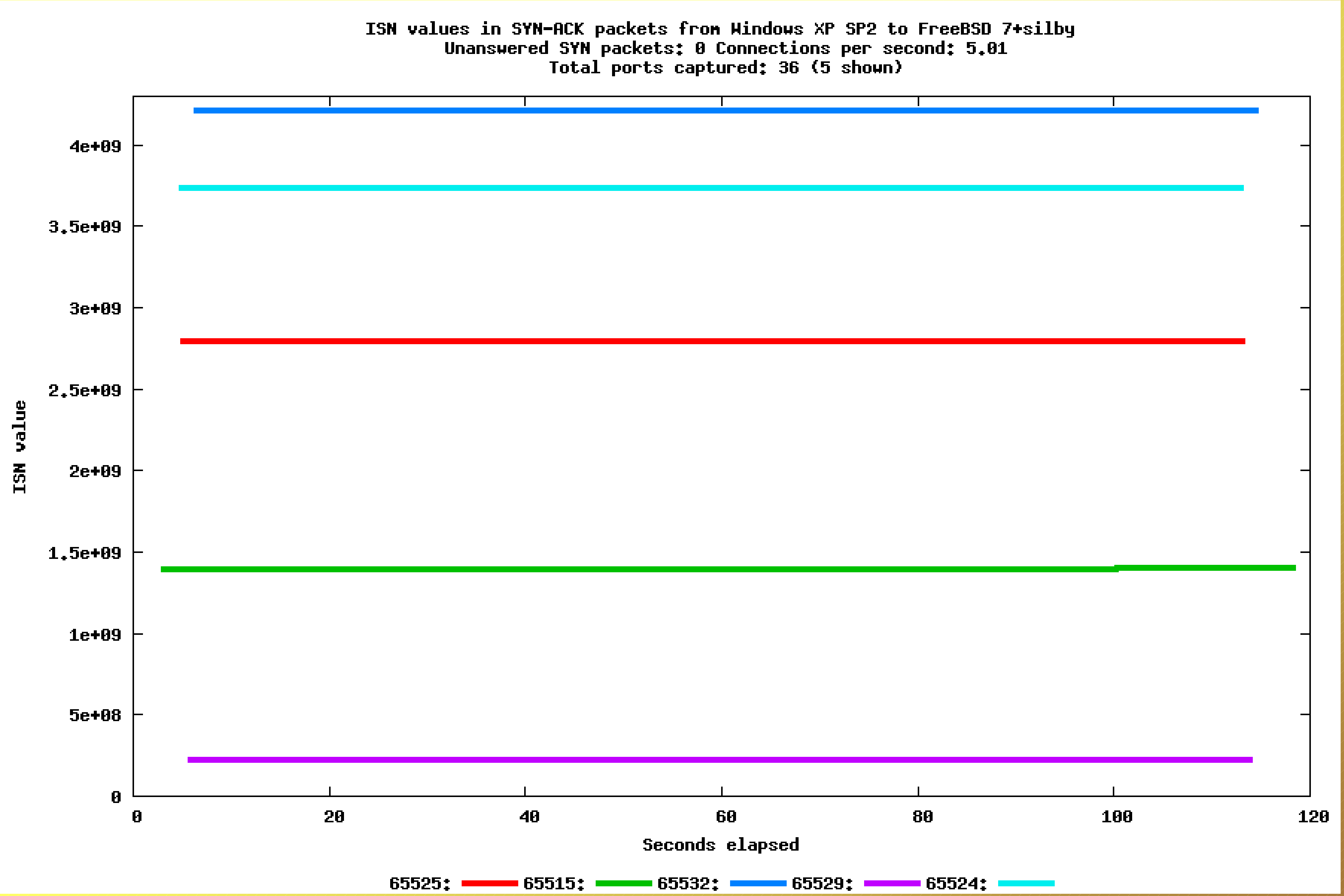
R(t) = a 16-bit random number generator with its msb always set to zero

(this analysis by Bindview in cert-2001-09)

Windows XP SP2 SYN



Windows XP SP2 SYN-ACK



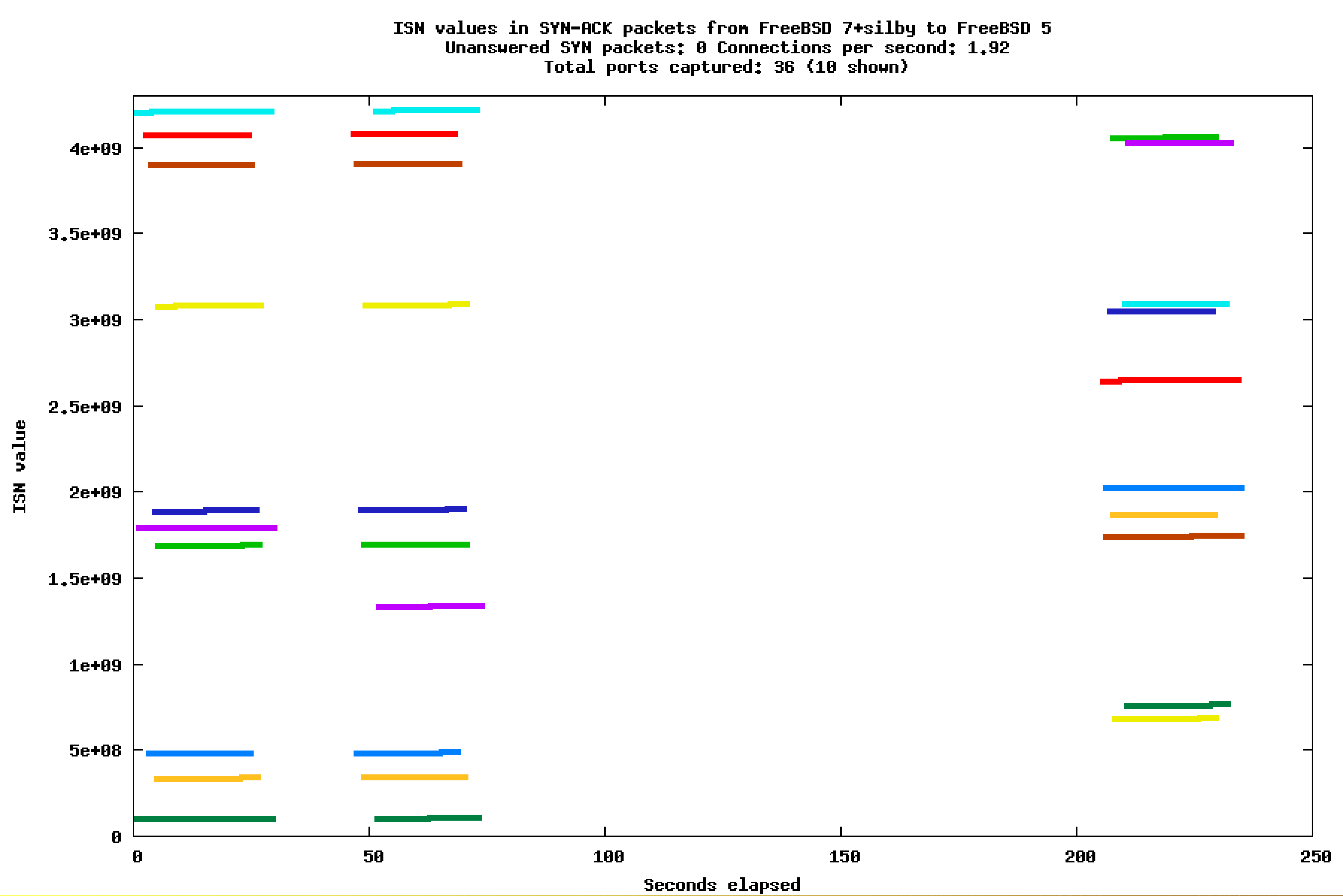
ISN Summary

- No two OSes are the same
  - Why?
- The FreeBSD way best meets the conflicting requirements of security and interoperability, but it is not perfect

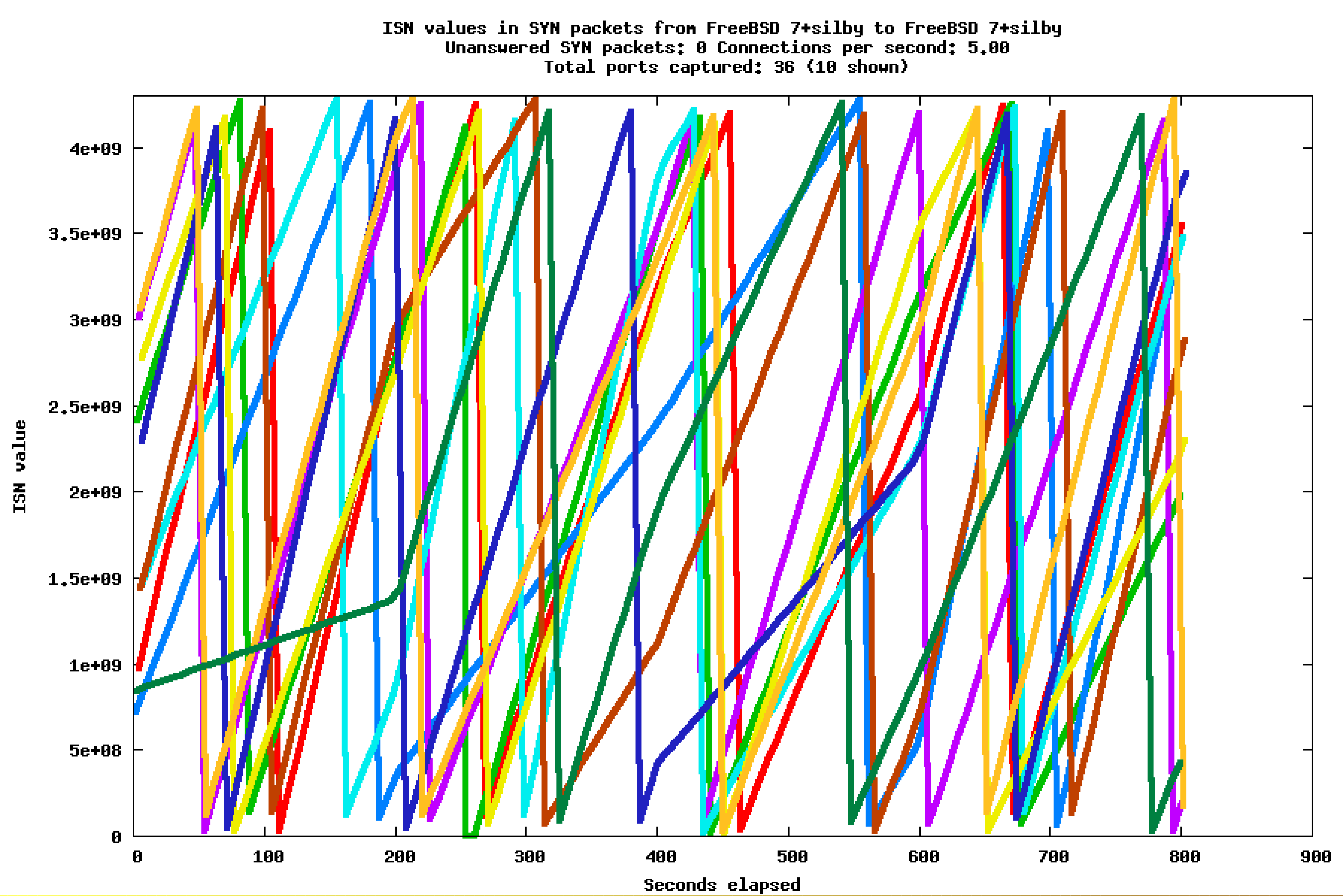
Improving the FreeBSD algorithm

- Flaws in the FreeBSD algorithm:
  - As the ISN values in SYN-ACK packets are randomized, there exists the possibility that the same sequence space will be used and a duplicate packet from the previous incarnation of the connection will cause problems
  - The RFC 1948 generated values in SYN packets exhibit the inherent weakness in RFC 1948

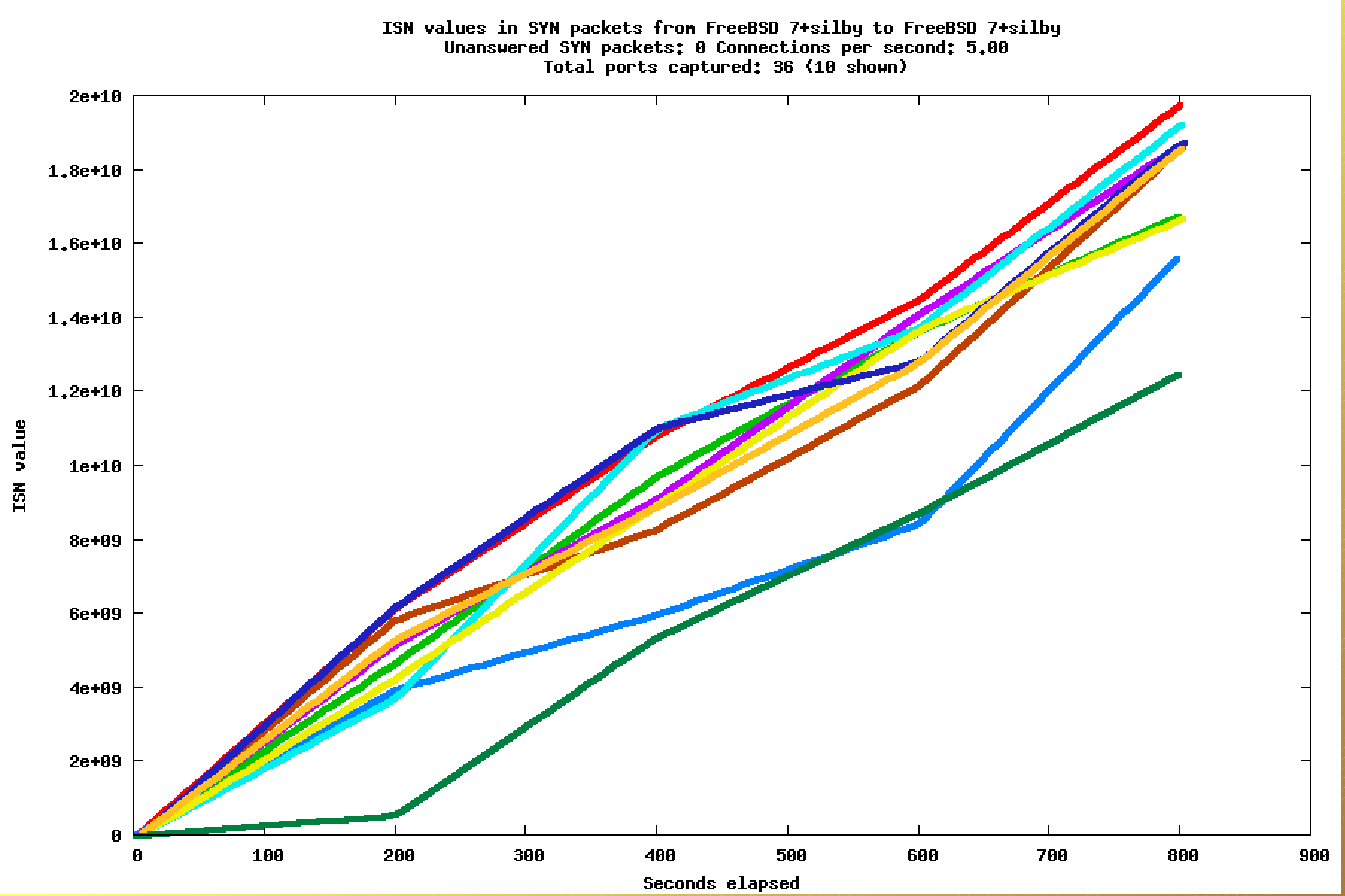
Improving FreeBSD SYN-ACK ISNs



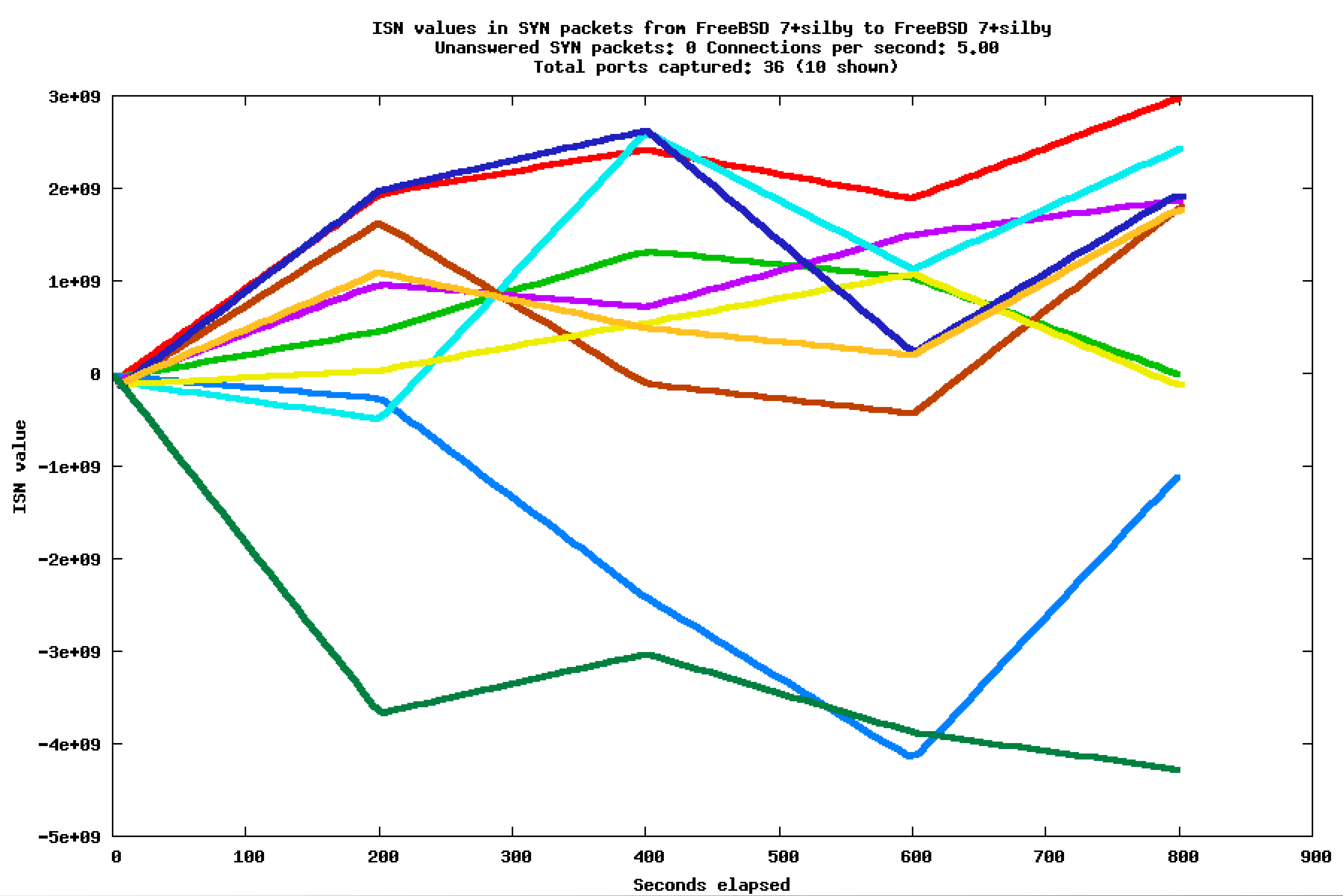
The dual-hash RFC 1948 variant



Another view of dual hashing



A View With Time Removed



TCP Timestamps

- The TCP Timestamp option was introduced in RFC 1323
- Timestamps serve two main purposes:
  - To allow for more accurate RTT calculations
  - For Protection Against Wrapped Sequence numbers (PAWS)
- All popular Operating Systems implement Timestamps, although Windows does not like to use them by default.

Timestamp Information Leakage

- Using a system-wide timestamp counter reveals a host's uptime
- Using a system-wide timestamp counter reveals which connections from a NAT machine originate from the same machine behind NAT.

Quick Fixes to Timestamps

- NetBSD: Start each connection's timestamp at zero
- OpenBSD: Start each connection's timestamp randomized
- The problem:
  - Timestamps are no longer useful for the purposes of PAWS
  - Linux makes the (reasonable) assumption that timestamps are monotonic over connection recycling in a few places

A Better Improvement For Timestamps

- Use the RFC 1948 algorithm, but use only the two IP addresses and the system-wide secret as input.
- Preserves PAWS usage
- Generally obscures uptime
- Does not solve the NAT issue entirely
- Allows for an important security improvement (next slide)

RFC 1948 Timestamp Security

- When timestamps are generated using RFC 1948, they will be predictable only on a per-IP basis.
- Hosts can check 32-bit timestamps as well as 32-bit sequence numbers
- Assume that a 16-bit sliding window of acceptable timestamps is used
- Spoofing packets is now 2^16 times as difficult
- Such a verification algorithm will still work if the other host does not use RFC 1948 timestamps, it will just not improve security.

Summary

- Security and Interoperability can coexist
- Significant testing is necessary to make this happen
- Interoperability is more important than security to some vendors