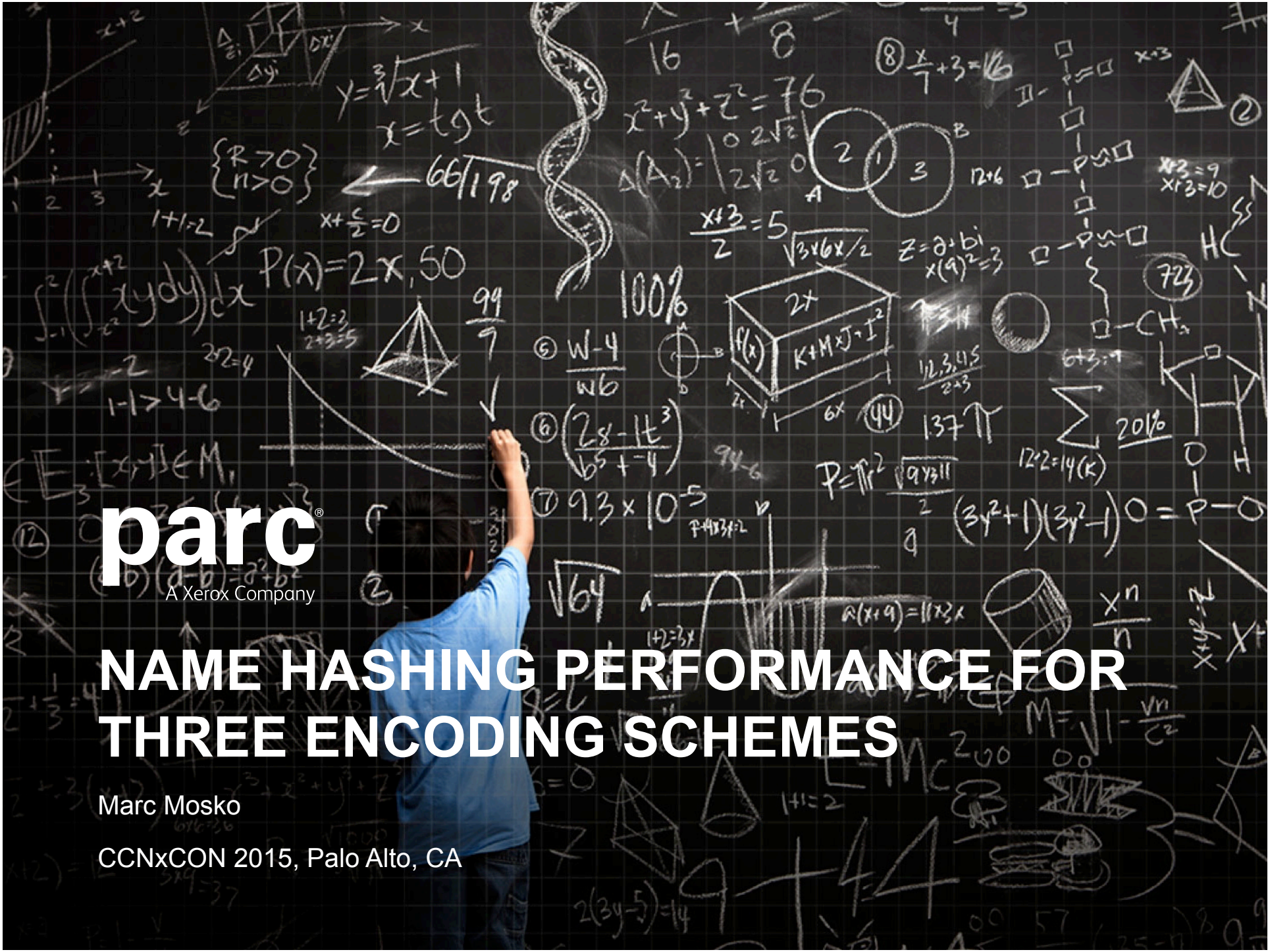


parc®
A Xerox Company

NAME HASHING PERFORMANCE FOR THREE ENCODING SCHEMES

Marc Mosko

CCNxCON 2015, Palo Alto, CA

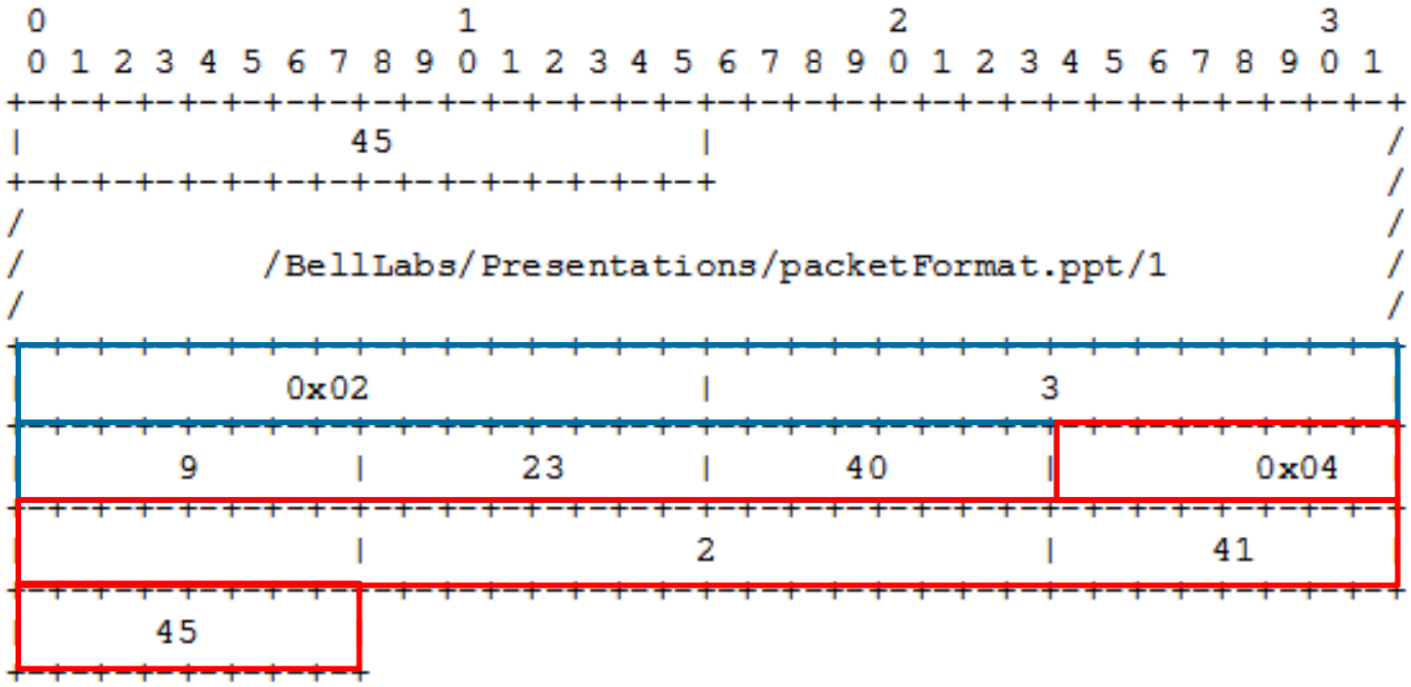
NAME HASHING PERFORMANCE

- At ICNRG Cambridge, ALU presented results of hashing ALU-style names vs 2+2 TLV style names [1].
- We provide a second look at the topic.
- We also include NDN 1,3,5 style names.
- Only evaluate name hashing time, not FIB time.
- We replicate the same CRC32C “hash” without endorsing this as the right thing to use.
- Main result: Unlike ALU evaluation, we find no penalty for using 2+2 encoding, especially for randomly distributed name lengths.

1. http://www.ietf.org/proceedings/interim/2015/01/13/icnrg/slides/slides-interim-2015-icnrg-1-12.pdf

ALU-STYLE NAMES

NAME ENCODING



Name length = 2 + 45 = 47B plus 13B of index

2+2 TLV STYLE NAMES

```
(2B T, 2B L) {
   (2B T, 2B '8')   { "BellLabs" }
   (2B T, 2B '13') { "Presentations" }
   (2B T, 2B '16') { "packetFormat.ppt" }
   (2B T, 2B '1')   { 0x01 }
}
```

Name length = 4 + 4x4 + 38 = 58B

parc
A Xerox Company

NDN TLV STYLE NAMES

```
(1B T, 1B L) {
   (1B T, 1B '8')   { "BellLabs" }
   (1B T, 1B '13') { "Presentations" }
   (1B T, 1B '16') { "packetFormat.ppt" }
   (1B T, 1B '1')   { 0x01 }
}
```

Name length = 2 + 4x2 + 38 = 48B

parc
A Xerox Company

WHAT ALU MEASURED

- 60 byte names (12B x 5 and 6B x 10)
- "All-in-one" hash
  - Hash all name components (out to 5 or 10), then match backwards in the FIB (e.g. 5, then 4, then 3, then 2, then 1). Saves time if name matches longer name components.
  - ALU names evaluated this way
- "Multiple" hashes
  - Do a component-by-component hash out to 5 (or 10) name components.
  - 2+2 TLV names evaluated this way

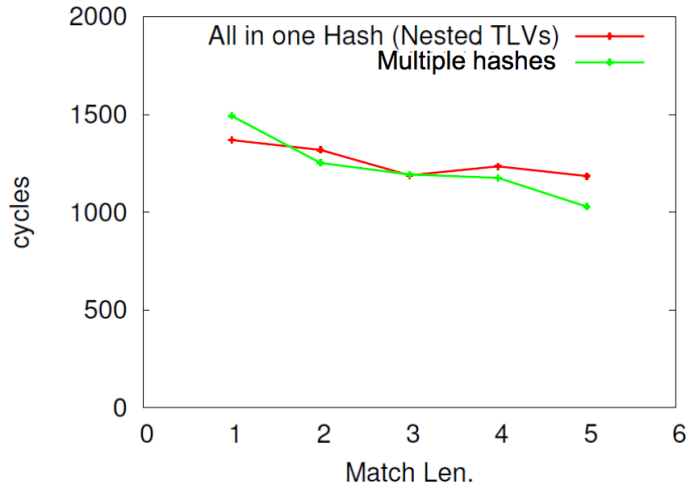
ALU RESULTS FOR 12B X 5 COMPS

SystemX INSTITUT DE RECHERCHE TECHNOLOGIQUE

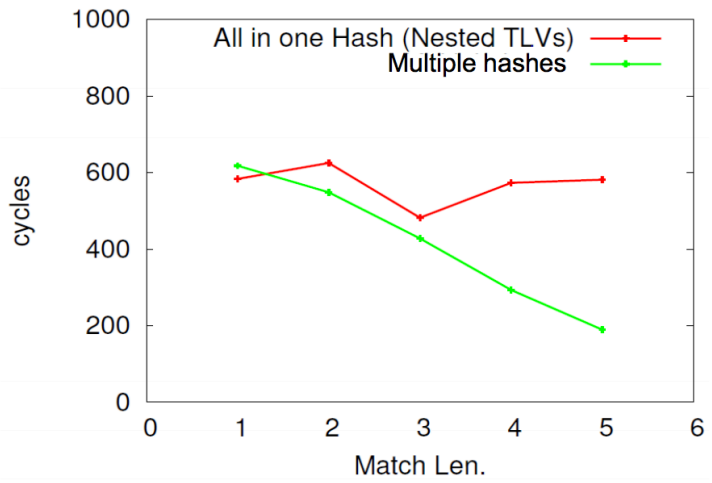
NAME ENCODING EVALUATION

- 60B names with components 5:
- 2 name encoding under test:
  - Name +offsets (1 Byte component separator needed)
    Total length: 65B, Components length: 13B 7,6% overhead
  - Nested TLVs (T+L = 2B+2B)
    Total length: 80B, Components length: 12B (+4B of T+L). 25% overhead

Lookup cost



Hash + parsing cost



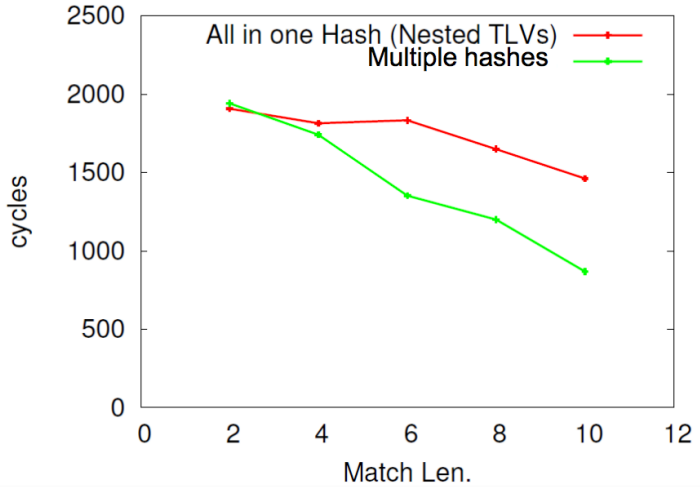
parc A Xerox Company

ALU RESULTS FOR 6B X 10 COMPS

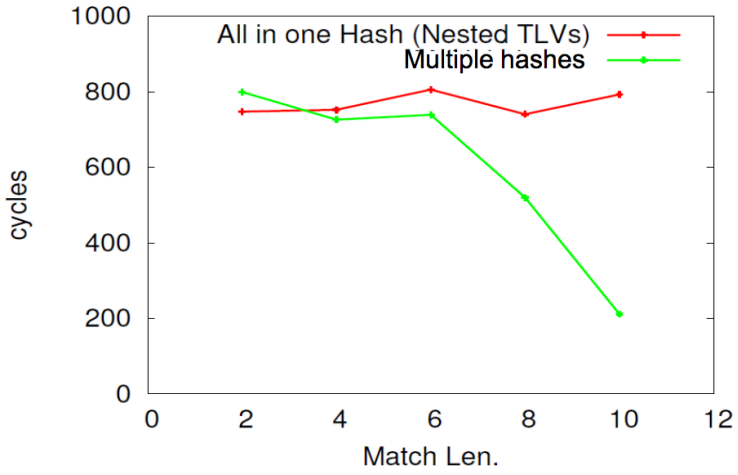
NAME ENCODING EVALUATION

- 60B names with components 10:
- 2 name encoding under test:
  - Name +offsets (1 Byte component separator needed)
    Total length: 70B, Components length: 7B. 12.25% overhead
  - Nested TLVs (T+L = 2B+2B)
    Total length: 100B, Components length: 6B (+4B of T+L) 40% overhead

Lookup cost



Hash + parsing cost



parc
A Xerox Company

CURRENT EVALUATION

- Use same evaluations for ALU, 2+2, and NDN names
- Same hash function (CRC32C) using SSE 4.2 intrinsic
- The intrinsics allow 8B, 4B, 2B, and 1B CRC
- Same 6Bx10 and 12Bx5 names.
- Include N(7.3, 11.7)B x 8 names (based on URIs)

EXAMPLE CRC CALCUATION

```
static inline uint32_t
crc32c_UpdateIntel(uint32_t crc, size_t len, const uint8_t p[len])
{
    // The length rounded to 8-bytes
    size_t blocks = len & 0xFFFFFFFFFFFFFFF8ULL;
    uint8_t remainder = len & 0x07;
    size_t offset = 0;

    while (offset < blocks) {
        crc = (uint32_t) _mm_crc32_u64((uint64_t) crc, *(uint64_t *) &p[offset]);
        offset += 8;
    }
    if (remainder & 4) {
        crc = _mm_crc32_u32((uint32_t) crc, *(uint32_t *) &p[offset]);
        offset += 4;
    }
    if (remainder & 2) {
        crc = _mm_crc32_u16((uint16_t) crc, *(uint16_t *) &p[offset]);
        offset += 2;
    }
    if (remainder & 1) {
        crc = _mm_crc32_u8((uint32_t) crc, *(uint8_t *) &p[offset]);
    }
    return crc;
}
```

Due to quantized nature of the CRC32C intrinsic, longer encodings do not necessarily take longer to hash. Any difference only shows up in the last 3 bits of the length.

parc
A Xerox Company

HASH TIMING

- Code uses the Intel recommended benchmarking techniques described in the whitepaper "How to Benchmark Code Execution Times on Intel (R) IA-32 and IA-64 Instruction Set Architectures" [1].
- Uses "CPUID" barriers and "RDTSC" timestamps.

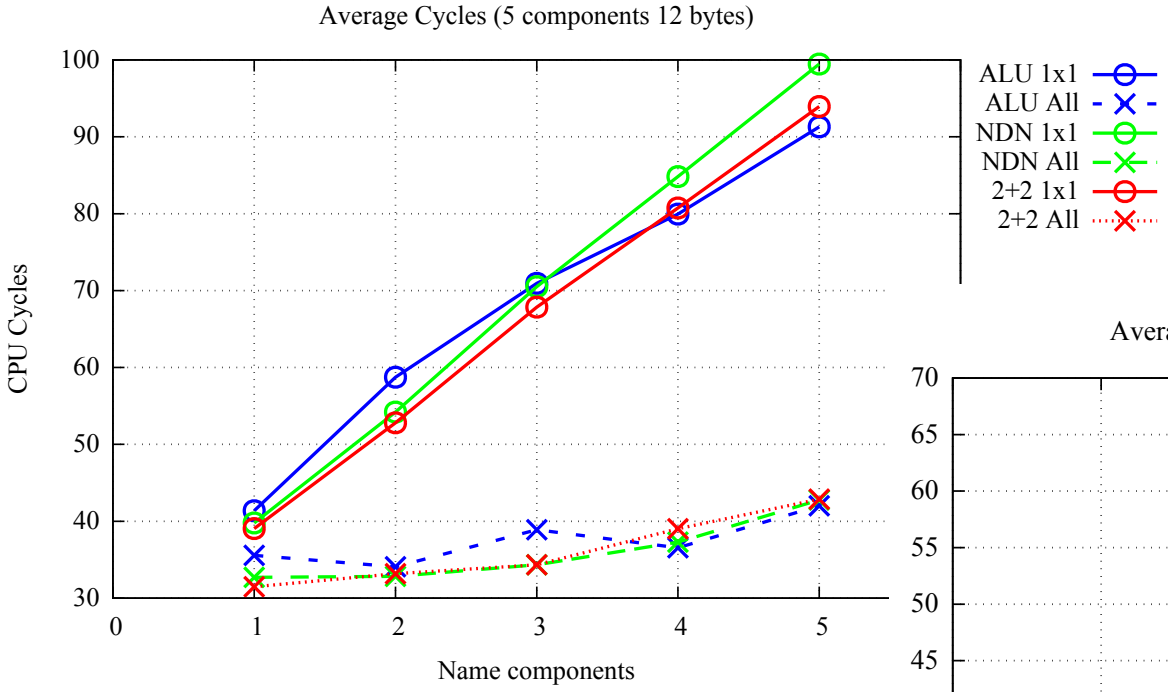
1. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf

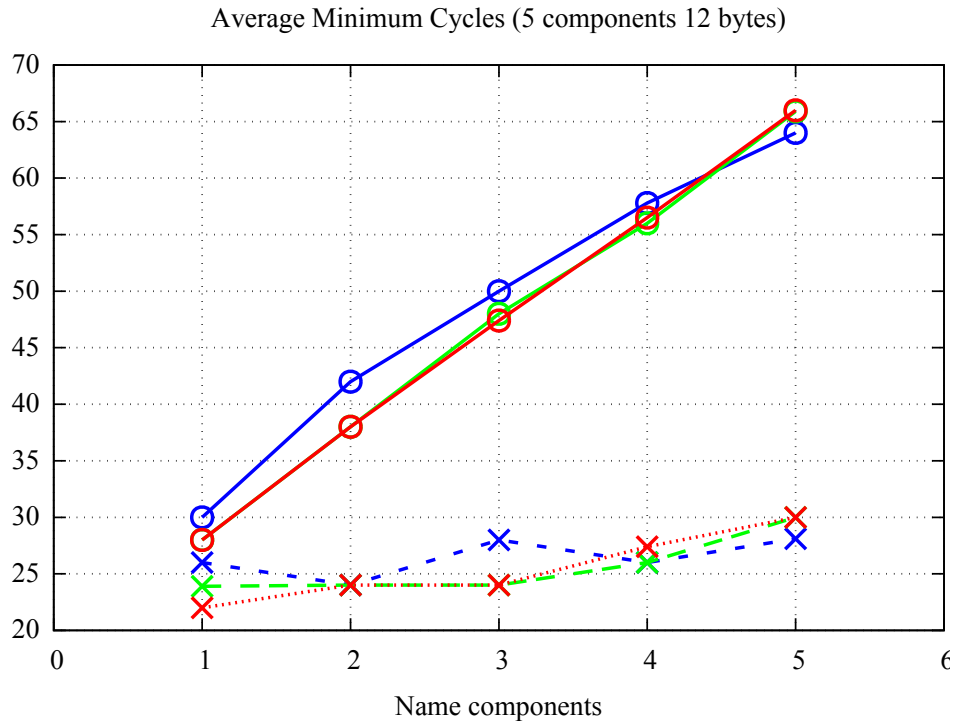
parc
A Xerox Company

METHODOLOGY

- Testset:
  - 1M random pre-generated names, repeated 20 times
  - Same name payload used for each of ALU, 2+2,and NDN
- We measure:
  - Average CPU cycles per name
  - Average minimum cycles per name
  - 20 repetitions keeps standard deviation of averages under 1.0 cycles

12B X 5 RESULTS

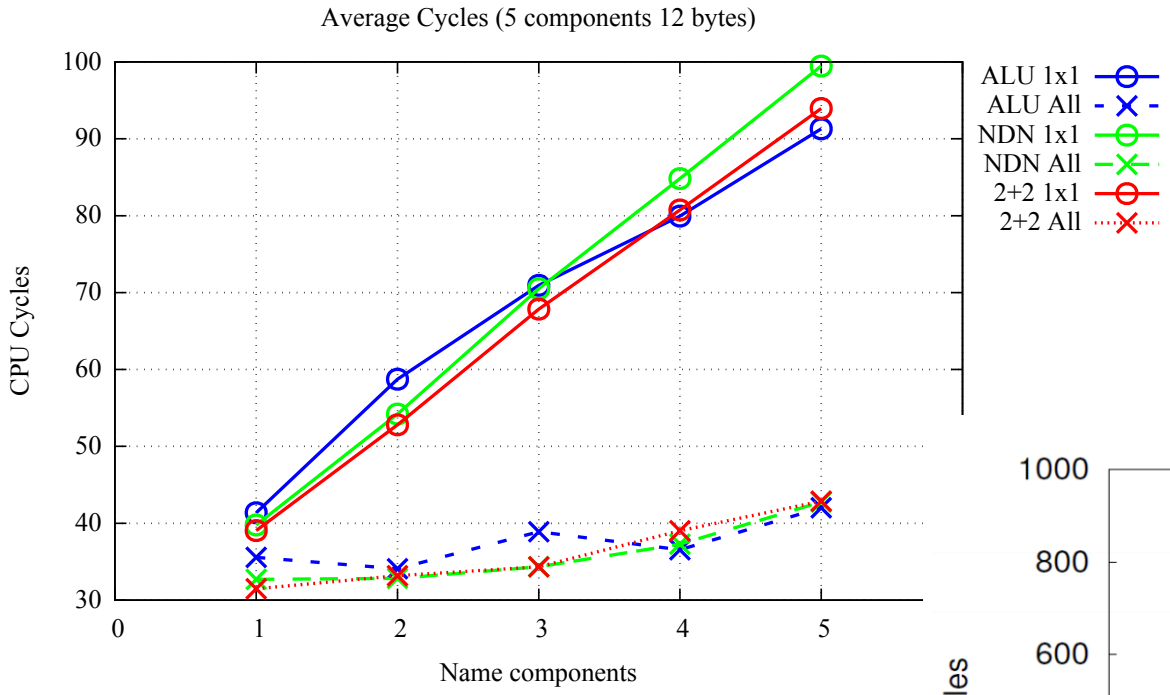


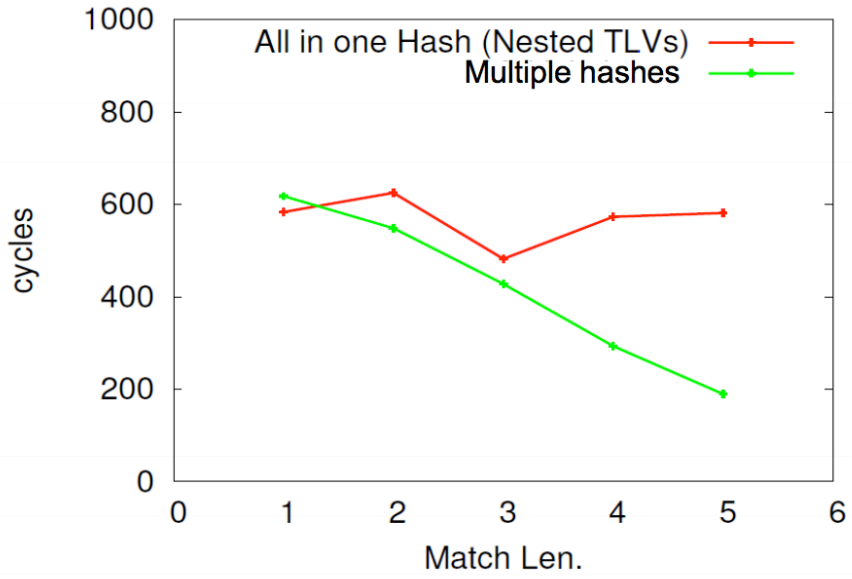


CPU instructions:
ALU = 2 (8 & 4)
NDN = 3 (8 & 4 & 2)
2+2 = 2 (8 & 8)

parc
A Xerox Company

12B X 5 COMPARE TO ALU RESULTS

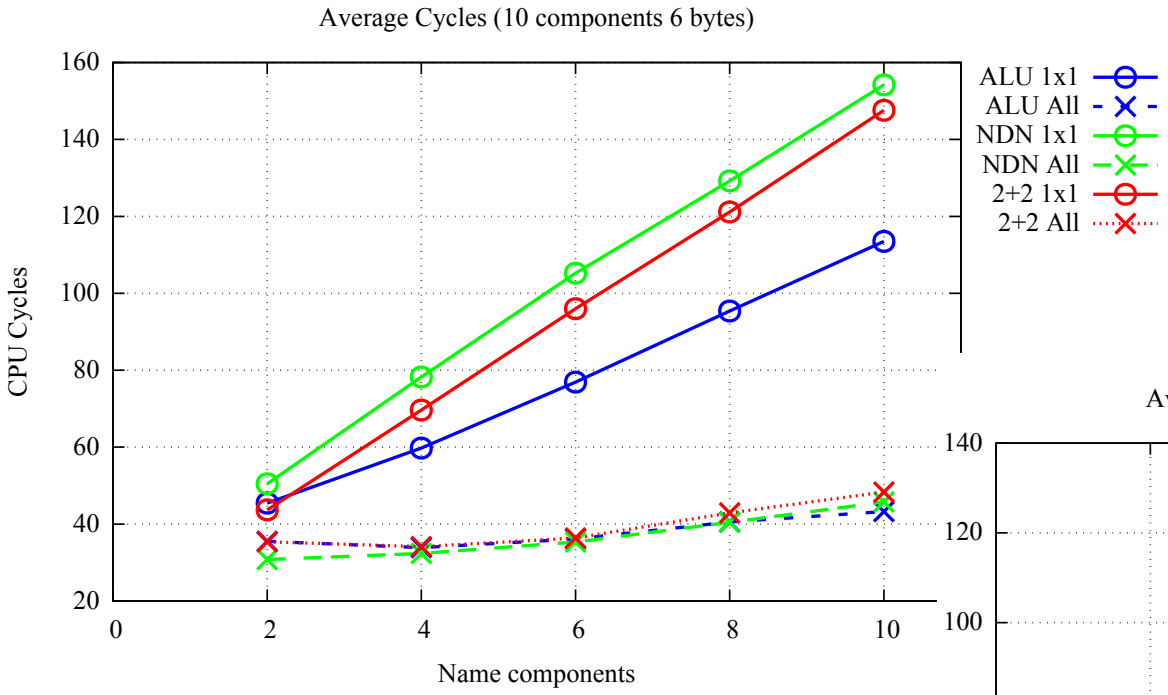


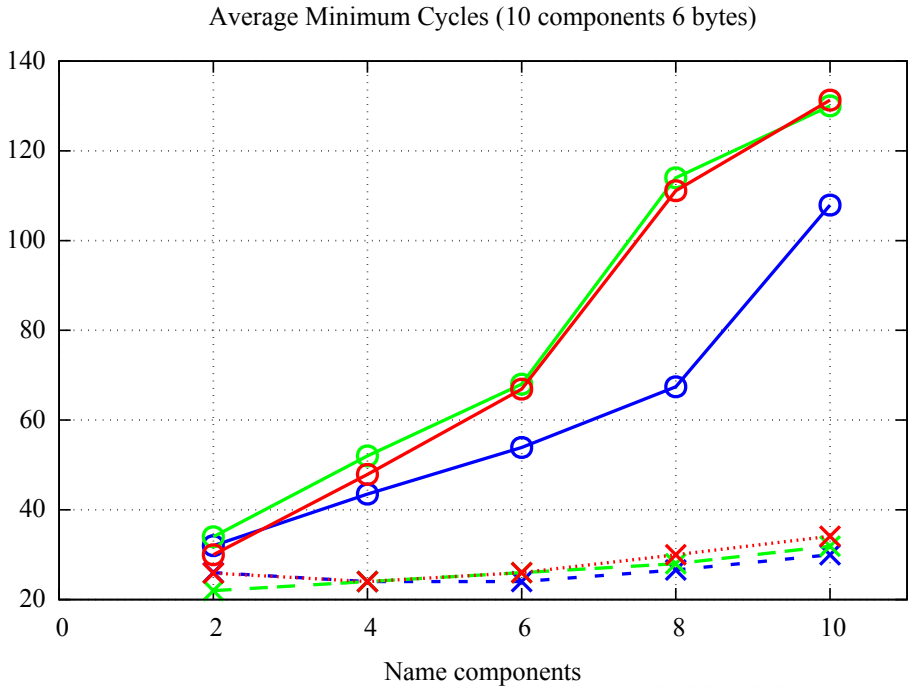


CPU instructions:
ALU = 2 (8 & 4)
NDN = 3 (8 & 4 & 2)
2+2 = 2 (8 & 8)

parc
A Xerox Company

6B X 10 RESULTS

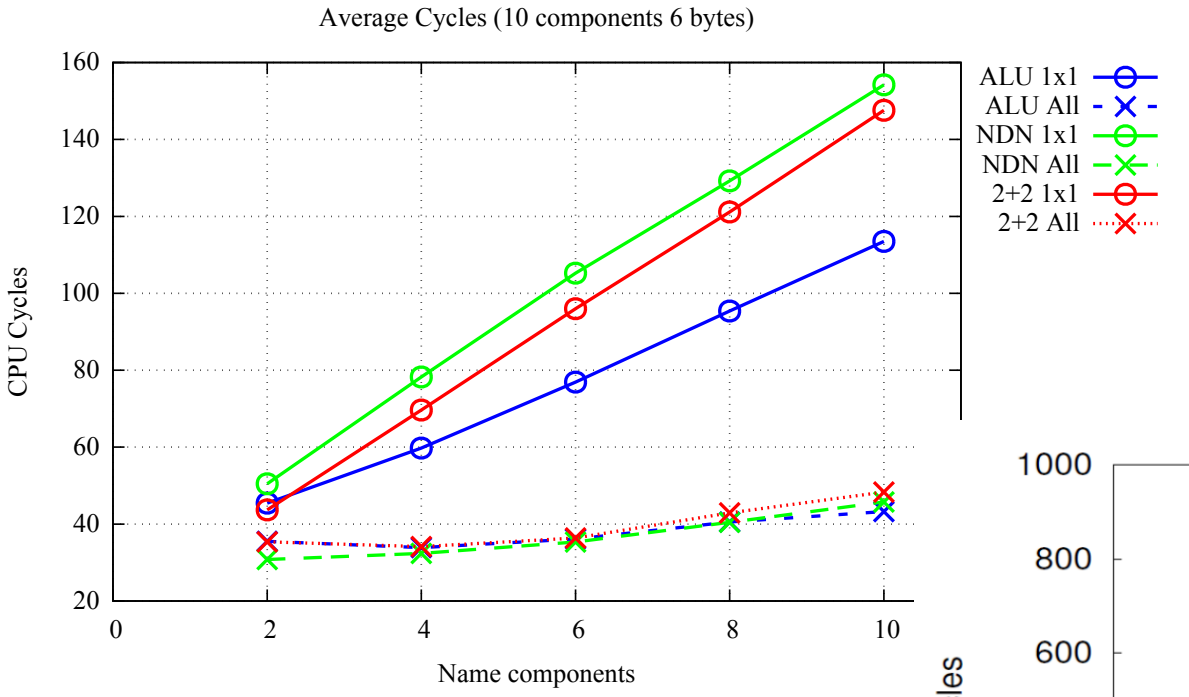


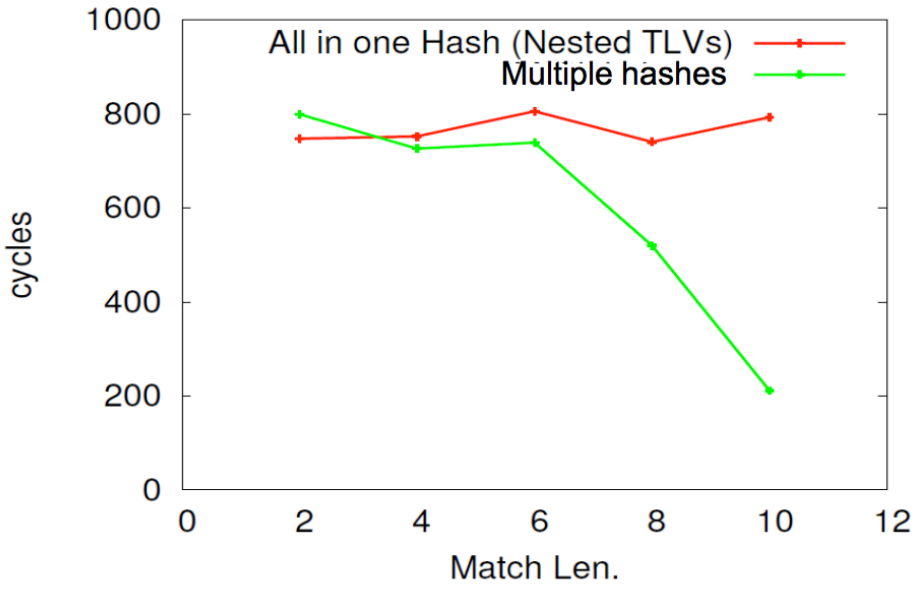


CPU instructions:
ALU = 2 (8 & 2)
NDN = 2 (8 & 4)
2+2 = 3 (8 & 4 & 2)

parc
A Xerox Company

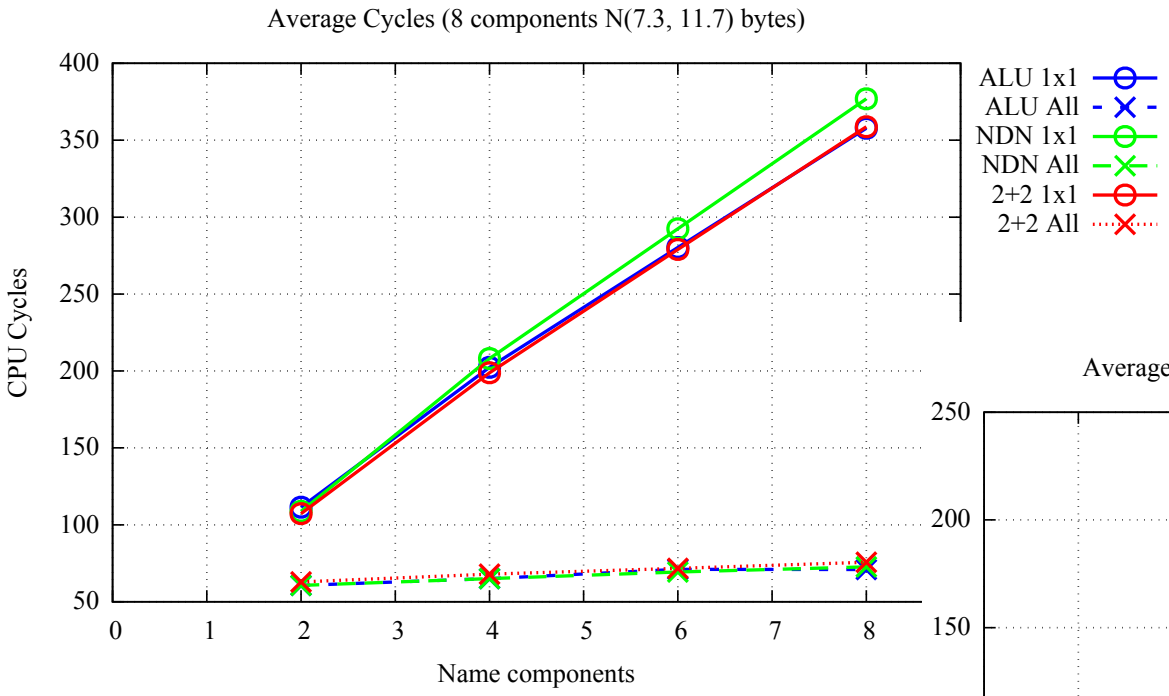
6B X 10 COMPARED TO ALU RESULTS

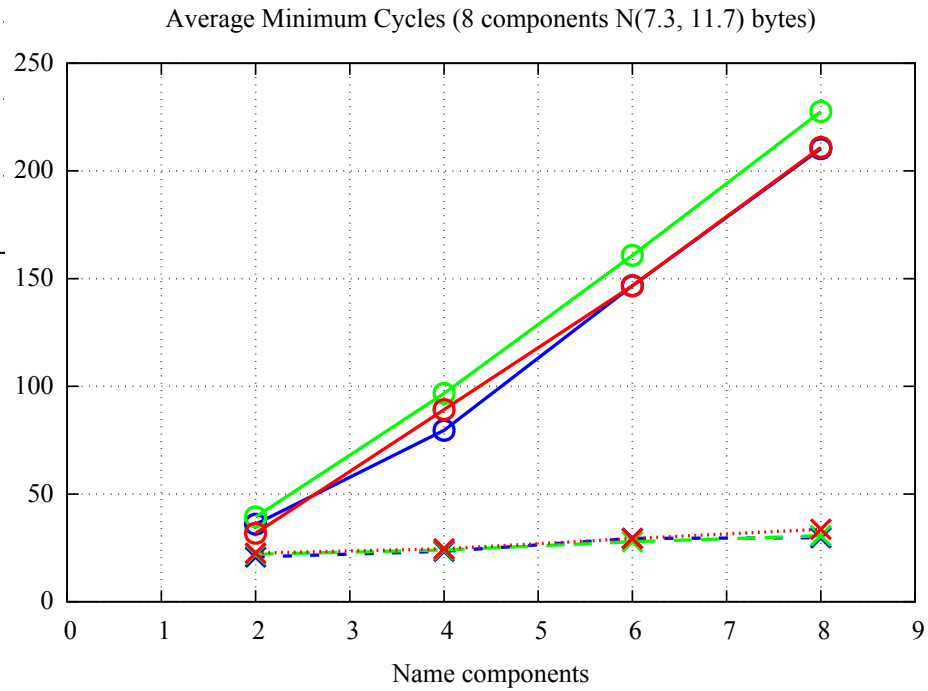




parc
A Xerox Company

N(7.3, 11.7)B X 8 RESULTS





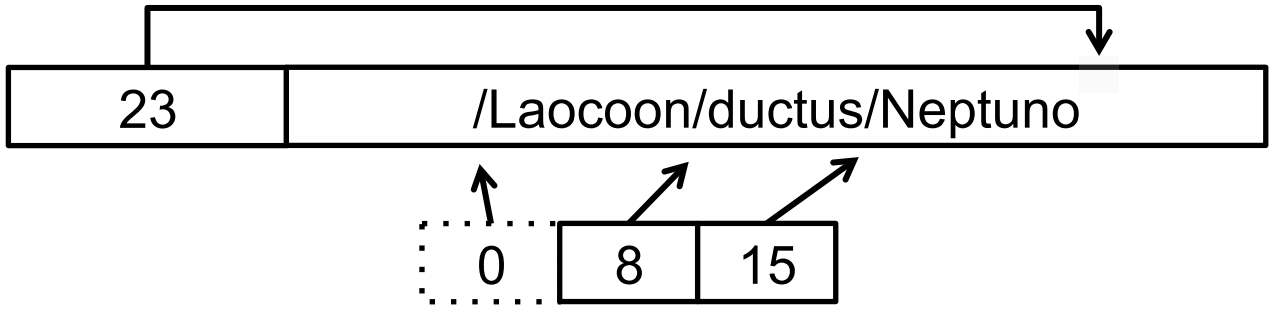
CPU instructions:
CRC32C cycle counts should washout between encodings because of random lengths

parc
A Xerox Company

REVISED RESULTS

- Based on feedback from Massimo Gallo (ALU)
  - He noted in our results that we used names of different lengths to measure the “all at once” time rather than use a shorter prefix of a longer name. This allowed us to use the overall Name TLV length rather than have to compute it.
- Revised details:
  - For CCNx, we do not have SegmentOffset array, so to hash “all at once” to a name component *i*, we need to iterate over components 1 … *i* to calculate the length to hash.
  - Because we do longest first, we can cache this dynamic SegmentOffset Array.
  - Likewise, for 1x1 hashing, we cache intermediate results.

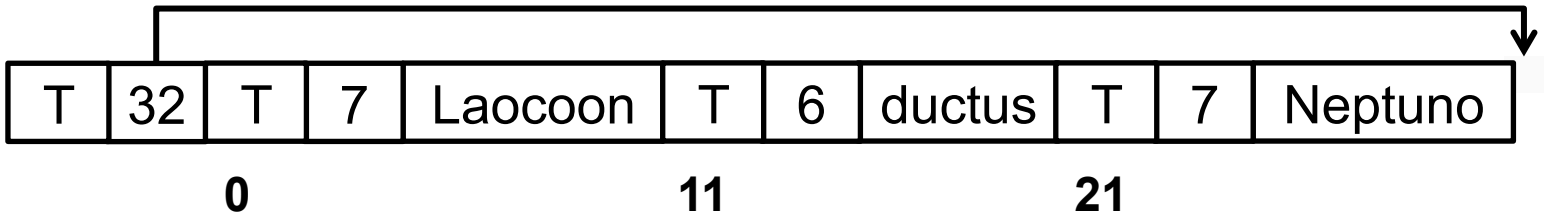
ALU EXAMPLE



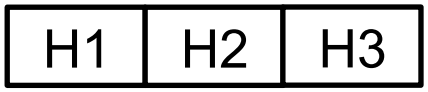
- 1x1
  - H1 = Hash(0-7)
  - H2 = Hash(H1, 8-14)
  - H3 = Hash(H2, 15-22)
- All at once
  - H3 = Hash(0 – 22)
  - H2 = Hash(0 – 14)
  - H1 = Hash(0 – 7)

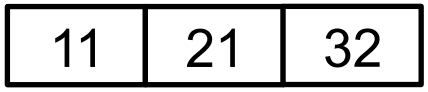
parc
A Xerox Company

2+2 EXAMPLE



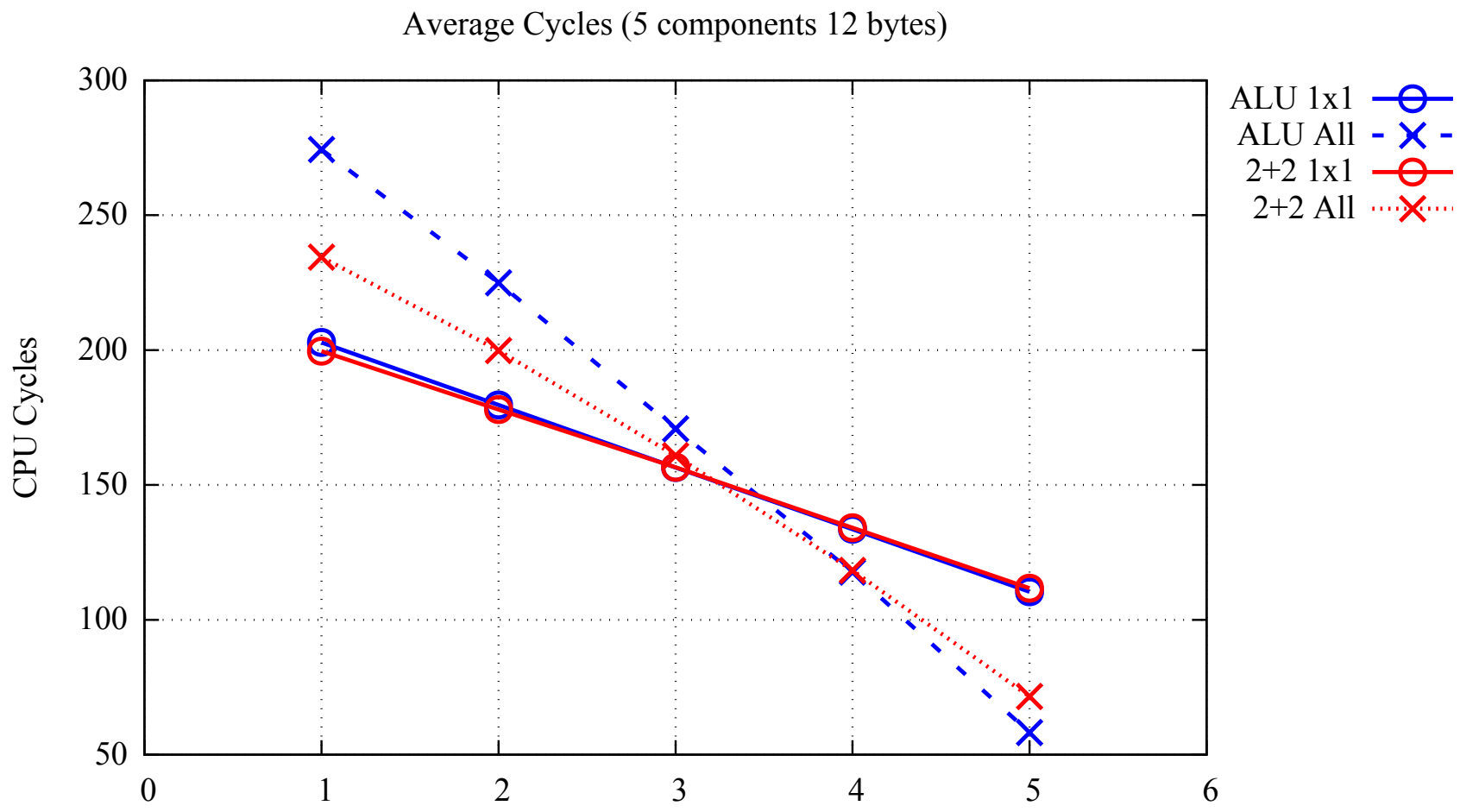
- 1x1
  - H3 = Hash(H1, H2, 21-31)
  - Save intermediate results
  - H2 = Array Lookup
  - H1 = Array Lookup
- All at once
  - Walk name and save lengths
  - H3 = Hash(0 – 31)
  - H2 = Hash(0 – 20)
  - H1 = Hash(0 – 10)





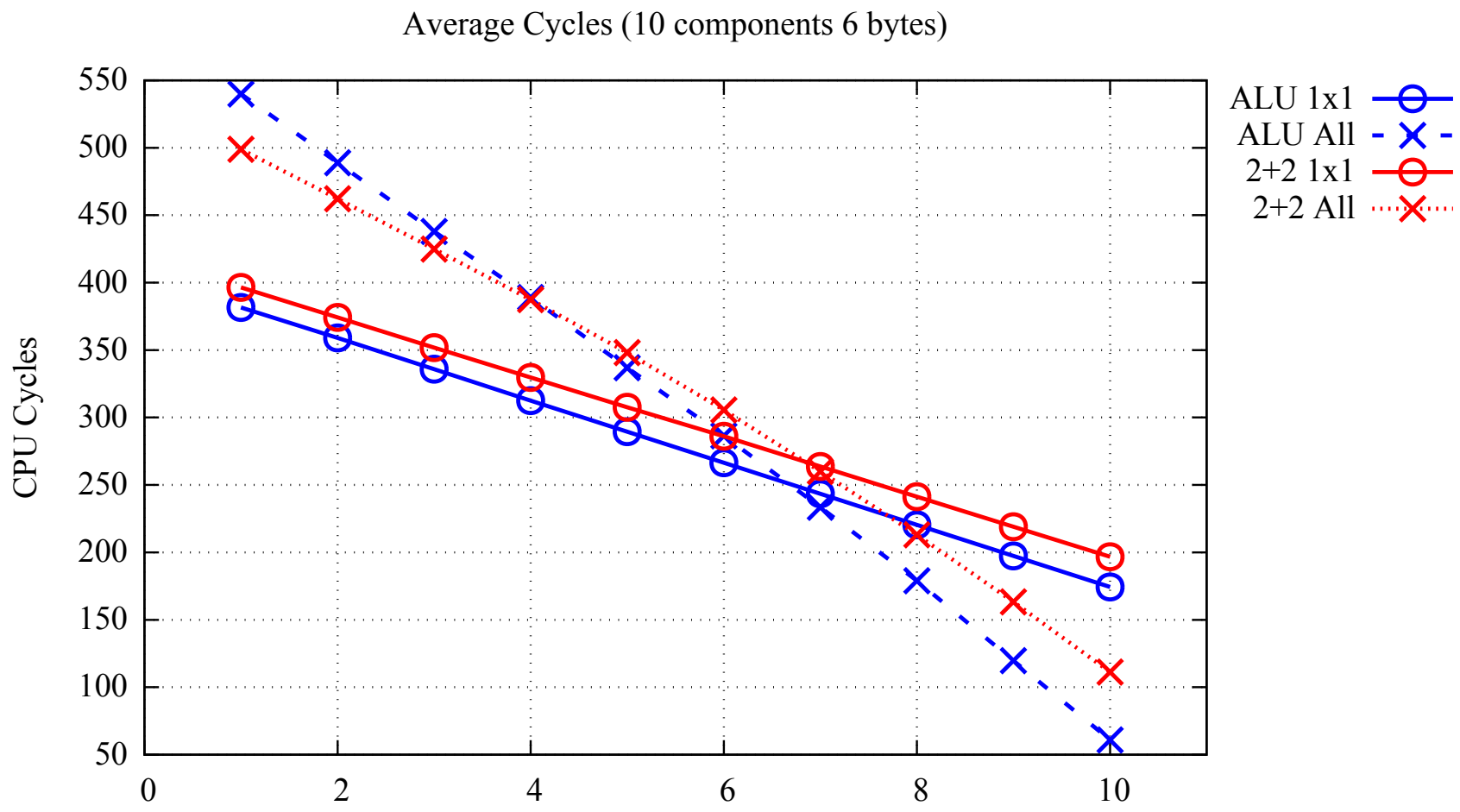
parc
A Xerox Company

5X12 RESULTS (LONGEST FIRST)



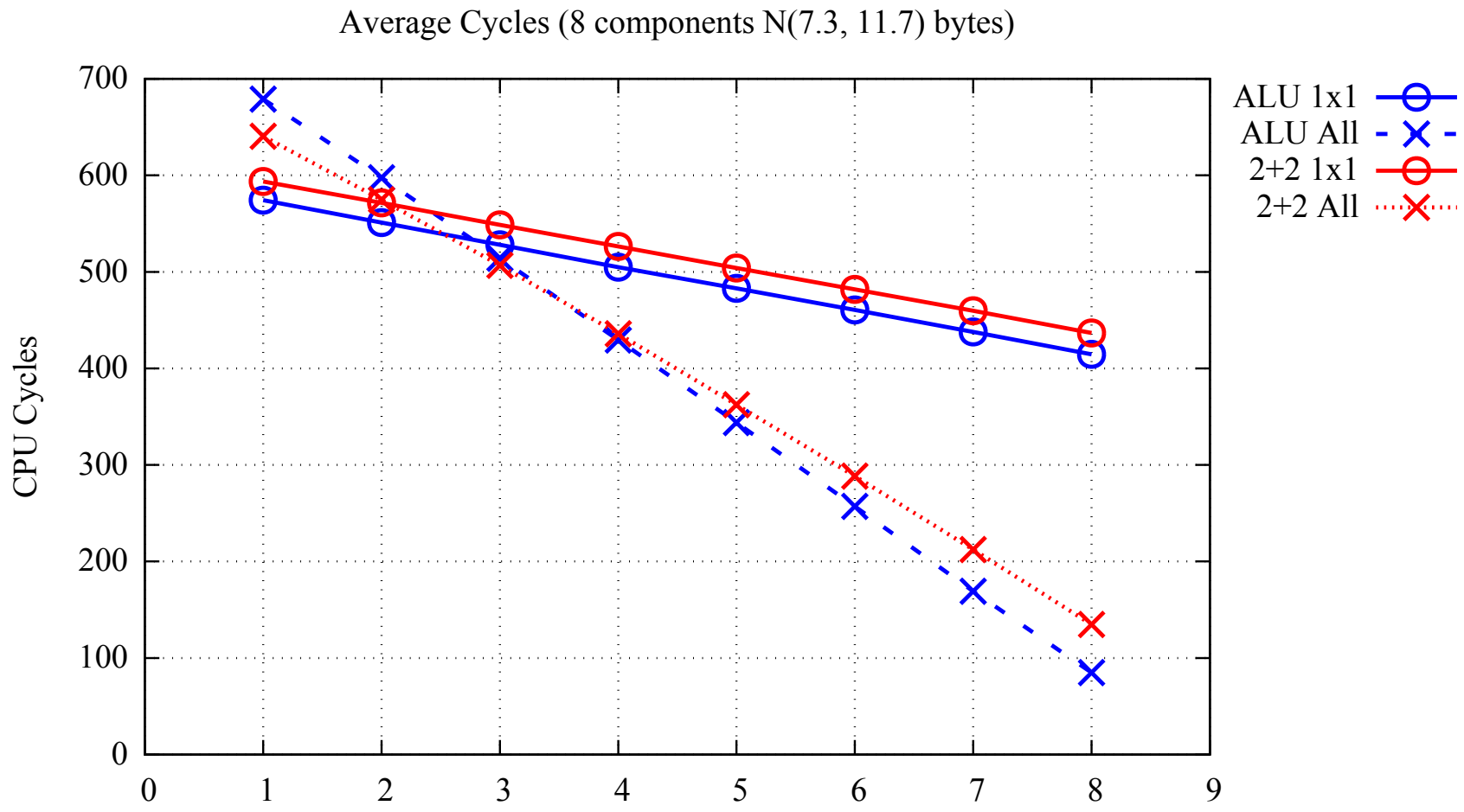
parc
A Xerox Company

10X6 RESULTS (LONGEST FIRST)



parc
A Xerox Company

N(7.3, 11.7)B X 8 RESULTS (LONGEST FIRST)



parc
A Xerox Company

CONCLUSION

- For very short names, there will be quantization differences due to 8, 4, 2, 1 byte CRC32C operations.
- For random N(7.3, 11.7) byte name components, see no difference in hashing performance.
- For the “All-at-once” strategy, there is very little difference in doing a large number of name components.
- The NDN 1,3,5 format is generally a little bit worse than the 2+2 format due to varint parsing for both the type and length.
- In revised results, still very little difference between ALU and 2+2 scheme.


Change, disruption, innovation
www.ccnx.org
parc
A Xerox Company


parc®
A Xerox Company
THANK YOU.