BGPsec Scalability

Protocol Engineering meets
Software Engineering and Hardware Engineering

Experiments

- Take realistic absolute and relative state distribution numbers.
- The overall setup models a route server in a moderately sized IX.
- Instrumented implementation for performance measurement.
- No codepoint hijacks.
- Feeder side is precomputed ahead of time.
- Verification is performed prior to path selection.
- The results should not be generalized and interpreted outside of the experiment context.

- Number of prefixes and paths.
- Number of prefixes sharing the same path.
- Fanout ratio.
- Caching aspects.

Experiments

- BGP – 83 s.
- BGPsec – 2049 s.

Contemporary compute platforms

- Plenty of raw compute performance capacity
- Memory bandwidth and latency are limiting factors
- Vectorization
- Batching and caching
- Most important – contemporary platforms do not forgive lousy approaches to software engineering. Protocol engineering needs to take software and hardware specifics into account seriously.

```
void memcpy(char *a, char *b, size_t n) {
    while (n--)
        *a++ = *b++;
}
```

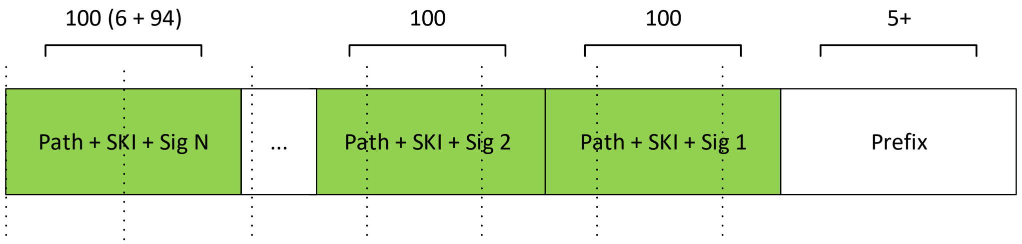
If you do this to your platform, do not expect that it will treat you friendly

BGPsec receive side processing

rx -> hash -> verify -> process prefix and path

SHA2 for hashing

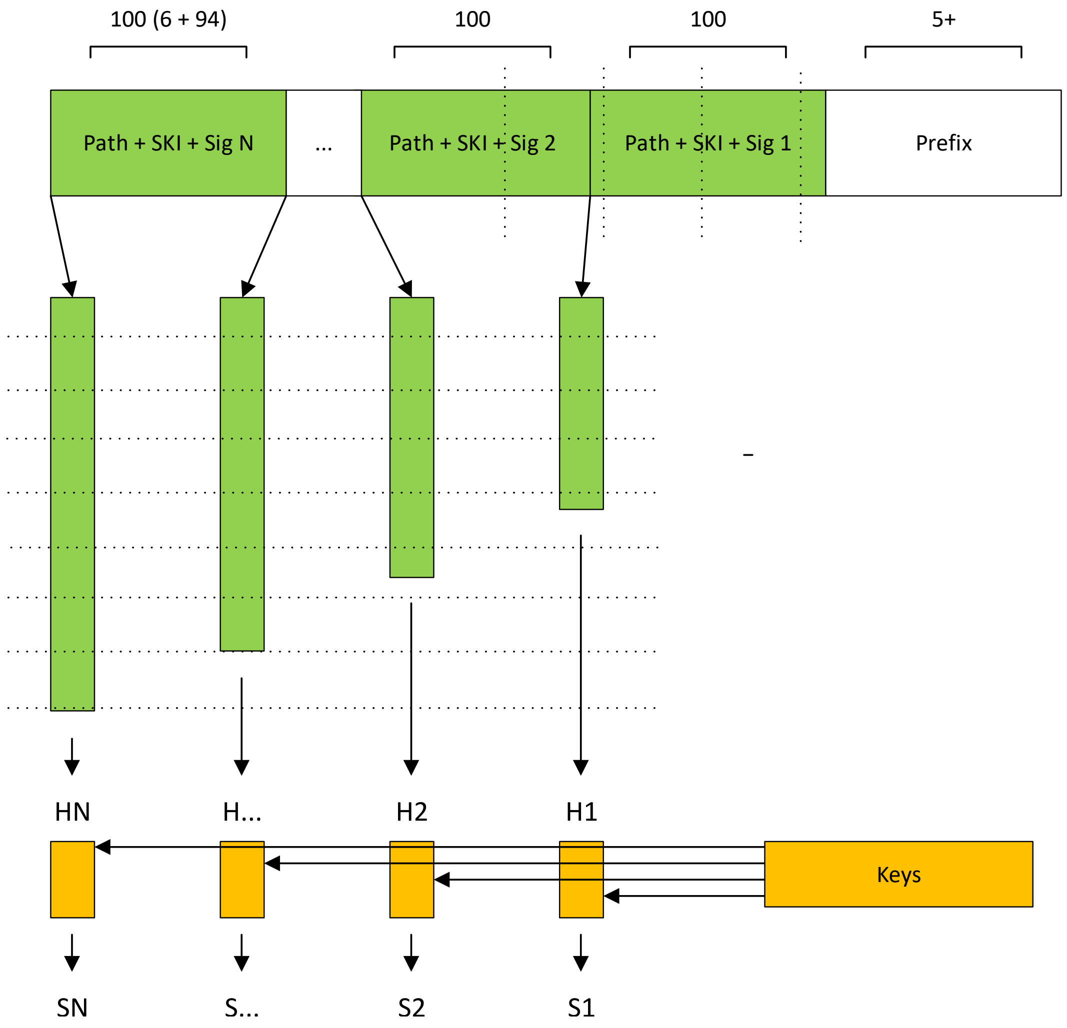
- Computationally inexpensive – but touches memory
- Operates on fixed size blocks with 4 byte base element granularity
- Vectorizes well, constrained by data layout



P-256 for verification

- Computationally significantly expensive – but does not touch memory
- Vectorizes well, little data dependency
- Batching – ECDSA*

Vectorized SHA2 and P-256



Linear code block operating on different data sets in parallel

Hash multiple blocks in parallel
Sign/verify multiple hashes/signatures in parallel

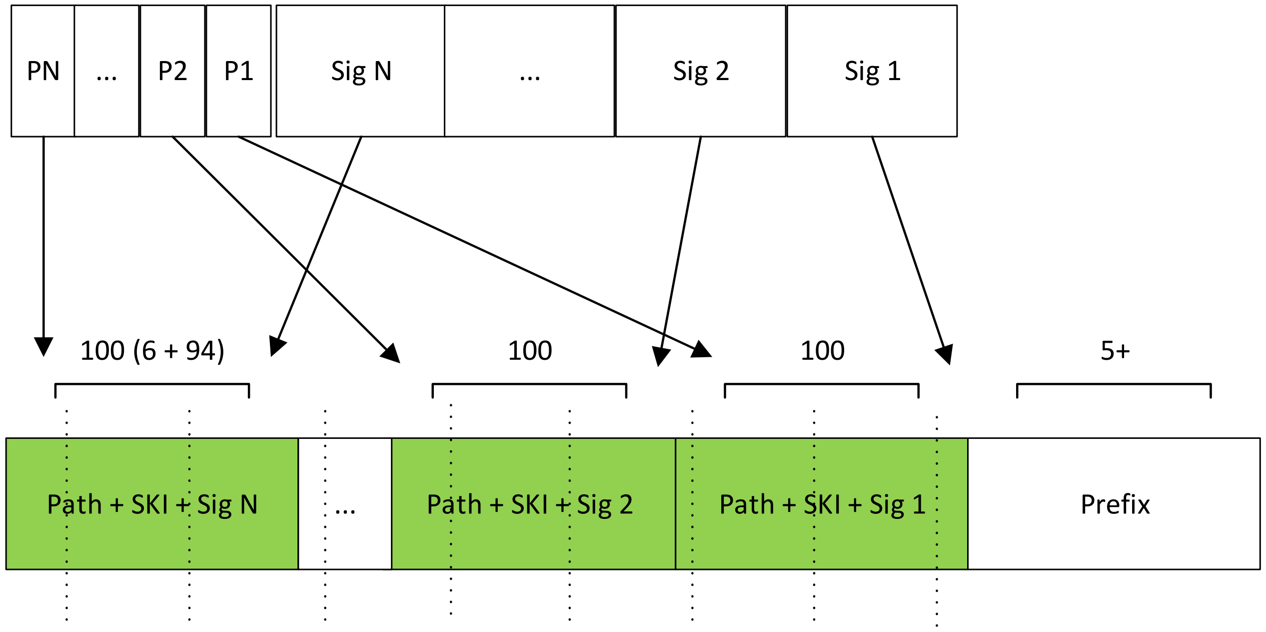
Vector lanes of fixed width

Gather operations place significant restrictions on data format

+20% latency results in +1500% throughput

If data structures allow.

Wire format impact



BGPsec wire format is incompatible with computation format.

Memory access is expensive

SHA2 latency is linearly proportional to block length

SHA2 operation width is 4 bytes

ECDSA signing is computationally expensive but constant, no memory access

ECDSA verification is even more computationally expensive but constant, no memory access

BGPsec transmit side processing

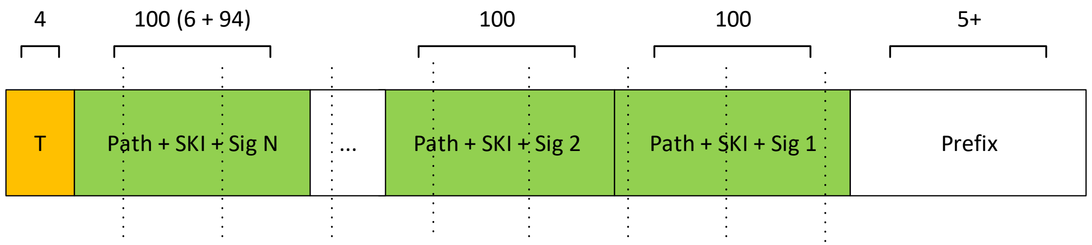
{Prefix, Path and signature elements, Target} -> hash -> sign -> tx

SHA2, same as for the receive side.

- Additional blocks need to be added, different layout for hashing and for wire encoding
- Target ASN position prevents caching

P-256 for signing

- Computationally expensive – but does not touch memory
- Vectorizes well



Experiments

- BGP – 83 s.
- BGPsec – 2049 s.
- BGPsec plus magic – 272 s.

Is BGPsec broken?

No.

As specified now, it is suboptimal and not aligned to contemporary hardware platform usage patterns.

What can be done then?

- BGPsec has some extensibility mechanisms inbuilt

- Protocol is versioned

- Algorithm identifiers could have different meaning in different versions
- Hashed block layout needs to be rearranged
- Wire format needs to be rearranged

Questions

- Can a smart compiler help here?
- Can a fashionable programming language help here?
- Vectorization availability?
- Memory system evolution trends?

Discussion

Do we care?