

BBN Technical Memorandum No. 1222

Justifications for Various Sprocket and Spanner Design Decisions

Job No. 11822000

September 27, 1999

Prepared for:

Defense Advanced Research Projects Agency
Computer Systems and Technology Office
3701 North Fairfax Drive
Arlington, VA 22203-1714

Prepared by:

GTE
10 Moulton Street
Cambridge, MA 02138

Justifications for Various Sprocket and Spanner Design Decisions

Beverly I. Schwartz

What are Sprocket and Spanner?

Sprocket and Spanner are computer languages created by BBN Corporation for the Smart Packets project. Sprocket is a C-like high level language; Spanner is a CISC-style stack based assembly language with high level types.

Why did we choose to create Sprocket and Spanner rather than using an

existing language?" The Smart Packets project requires that programs fit in a single, non-fragmented packet. Because ethernet is so common in today's LAN's, the Smart Packets project is looking at an MTU of 1500 bytes. Header and authentication information also must fit into the packet, leaving about 1000 bytes for Smart Packets programs. No existing high or low level computer language provides for the compactness requirements of the Smart Packets environment. Sprocket and Spanner were created closely together. Sprocket provides the ease of a high level programming language similar to C, something that is familiar to most programmers. Spanner provides an extremely compact assembly language for those bold enough to play with assembly who want direct control over how many bytes are used in their programs. After a bit of research, we felt it would be easier to write a compiler for a simple language like Sprocket than to fit a Spanner backend on an existing compiler.

Why does Sprocket look so C-like?

Most computer languages boil down to the same sorts of constructs - variables, operations, flow control, functions, etc. Sprocket is no different. We felt that it didn't matter what the syntax looked like as long as the basic functionality of a high level language was there. C is ubiquitous; most programmers are familiar with it. Since the creator of Sprocket is most familiar with C/C++ and has a bias against LISP-like languages, and because the C grammar is readily available and easily modified, C seemed like a good starting point.

Why does Sprocket use similar concepts to C, but use different words like

array and persistent?" Any place where Sprocket differed from C/C++, an entirely new word or symbol should be used to highlight the difference, so that programming errors wouldn't be made from equivalent syntax with different semantics.

Why does Sprocket and Spanner have so many complex types? Can't these be

represented by records? What happens when new network services come along that have types that are not represented by Sprocket and Spanner?" The complex data types are offered so that common tasks can be represented in a very compact

manner. Records provide a general way to represent more complex types, but both the representation of the record and operations on fields in the record will take up a significant amount of encoding space. When new network services come about, a user can represent what s/he needs using records. If the network service becomes ubiquitous, then a new type and primitives can be created, allowing for more compact representation for the new service.

Why are Spanner variables and structures represented by a discontinuous set of numbers?" Compactness is key in Spanner encoding. Having one byte for variables would allow for only 256 variables; this would not be enough variables. Having two bytes for variables would cause a lot of unnecessary bloat. So, both are used. If the top two bits of the first byte are on, then the variable is a two byte number; otherwise it is a one byte number. This allows compactness in variable representation for the vast majority of Spanner programs, but also allows the programmer to have over 10,000 variables if needed.

Why are Spanner instructions fixed size?

Variable length instructions may provide the ability to have more compact encodings. However, for this to be truly effective, the relative frequency of use of Spanner instructions would have to be known, so that those most frequently used can have the shortest instructions. At this time, this information is not known. Variable length instructions would add complexity to the virtual machine. It is not clear that the added complexity would provide any benefit. After some actual experience with Spanner it becomes evident that a variable instruction length would provide some benefit, then a new version of the Spanner encoding can be created.

Why are Spanner instructions 16 bits?

16 bits provides for 65536 instructions which will allow for a sufficiently large instruction set. 8 bits would provide for only 256 instructions, which would not be enough. Since most architectures work on byte boundaries, having an instruction between 8 and 16 bits is not practical.

Why does the Spanner virtual machine use a stack rather than registers?

A stack is very simple. Implementing a register machine and a compiler for a register machine would be far more complex than for a stack machine. Register machines provide a big win in computational efficiency for hardware that uses pipelining. Since this is a virtual machine completely implemented in software, pipelining is a non-issue. A well-designed register machine with a finely tuned compiler may provide a more compact representation than a simple stack machine, although a significant improvement is unlikely. Given Smart Packet's tight schedule, completely changing direction in our design to something that would be more complex that probably would not provide an improvement in compactness did not seem worthwhile.

Why does Spanner have variable declarations?

Spanner does automatic promotion of types. In order to do this correctly, Spanner needs to know what types it is dealing with.

Why does Spanner do automatic promotion?

Originally, Spanner was designed to have strong typing. 16 bit instructions would not be sufficient to encode Spanner's various instructions with type information embedded. In addition, it would require programs to have explicit casts any time different types were mixed. This takes precious encoding space.

Automatic promotion happens as a space saving measure. Yet to make automatic promotion work correctly, variable declarations are needed which take space! Why take this approach?" Variable declarations happen once, and that information is known for the rest of the program. If the program uses different variable types, then many casts might have to be done to perform operations.

Why does Spanner have structures?

Because Sprocket has structures. And because, other than declarations and casts, instructions do not carry type information. If a field in a Sprocket structure is being used, somehow its type information has to be conveyed. The most compact way to represent fields in Spanner is to have a dot operator that operates on the structure and its field. The cost for making access to structure fields cheap is having to define the structure before using it.

Why do some Spanner instructions have push and np (no push) options?

Spanner was designed to work closely with Sprocket. Some operations in Sprocket, like equals or post-auto-increment, usually are not used in another operation, so the result should not be pushed on the stack. However, sometimes equals are chained together, and pushing the result on the stack for all of the equals in the chain except the last one, is desirable. This allows for greater flexibility while keeping the encoding compact.

The default for each operation is what seems to be the most frequent use, so some instructions default to no push and other default to push. Although it is not documented, if `-np` or `-pu` is added to any instruction, Spanner will do the right thing. The default form is included with each instruction.

Why are arrays limited to 3 dimensions?

Sprocket is limited to 3 dimensions because Spanner is. Spanner is limited to 3 dimensions because of encoding considerations. 3 is the maximum number that can be encoded with 2 bits. Some array operations may take as many arguments as there are array dimensions. For each argument, 2 bits are needed to indicate whether the argument is a variable, literal or comes from the stack. For 3 dimensions, 6 bits will be needed for argument information, about as much space as can be spared for this type of information.

Why are arrays 0 based in Spanner?

Because they are in Sprocket. If array accesses in Spanner were different from Sprocket, compilation would be a nightmare. Sprocket arrays are 0 based because Sprocket is based on C, and C arrays are 0 based.

CONTENTS