Datagram Forwarding Considered Harmful

Micah Beck, University of Tennessee
mbeck@utk.edu
14 January, 2024

In 1968 letter to Communications of the ACM [https://dl.acm.org/doi/10.1145/362929.362947] entitled “Go to Statement Considered Harmful”, Computer Science pioneer Edsger Dijkstra argued against the excessive use of go to statements in programming. Dijkstra gave a very specific reason for this judgment:

The unbridled use of the go to statement has an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress. Usually, people take into account as well the value of some well chosen variables, but this is out of the question because it is relative to the progress that the meaning of these values is to be understood.

The “coordinates” that Djikstra referred to were the procedure call stack and the values of nested loop indices. These coordinates define the environment within which invariant program assertions can be reasoned about either informally or formally (e.g. using Hoare Program Logic). The unstructured use of go to statements has the effect of expanding the set of possible “coordinates” which can exist at the target statement. This increases the complexity of reasoning about program assertions and potentially weakens such invariants.

We can think of a datagram as a mobile process, whose “code” is its destination address and whose purpose is to move its payload to that destination. Each forwarding node that such a datagram encounters “executes” it by reading and modifying certain fields in its header. Header fields thus play the role of mutable and read-only variables of the datagram-as-process. The payload is also largely unchanged during forwarding except for reversible operations such as fragmentation may be applied in transit.

If we consider a datagram to be a mobile process, then it makes sense to ask what invariants are true at any point in its “execution”, meaning its path of transmission and forwarding operations through the network? It is tempting to say that nothing is known except the information expressed in the header or carried in the payload. However, there is also a context, analogous to the “coordinates” described by Dijsktra, which are necessary for its correct and safe incorporation into the state of the receiving program. These coordinates are defined by what is known about the sender. Although we will not do so here, “known” can be formalized in a modal “logic of knowledge”.

In a very tightly controlled closed system, very strong assertions can be made about the sender of a packet (which is the local form of a datagram). Examples are words and blocks sent across memory and I/O busses in computer systems and between physically adjacent neighbors in wired networks. It is common to use such internal communication in a manner that reflects very strong knowledge (or assumptions) of the correct and nonmalicious nature of the sender and the reliability of the local network.

In the wide area, an edge router that forwards datagrams outward from a local area network may have fairly strong knowledge of assertions about the senders. This reflects the constraints imposed by their topological “coordinates”. However, a router that receives datagrams from multiple edge networks has an expanded set of possible sources. This increases the complexity of reasoning about the senders and potentially weakens knowledge assertions about them. A router in the network core has a hugely expanded set of possible sources, and almost no usable knowledge about the senders, except what can be inferred from unreliable header information.

The impact of this lack of knowledge about datagram senders is twofold. The first is that it greatly unburdens routers from the need to maintain the correctness of invariant knowledge, making the job of forwarding datagrams much easier. The result has been the ability for the Internet to be deployed and operated globally and at low cost. The second has been that end-to-end protocols must be used to try and establish knowledge of the sender by the receiver. The success of this effort has been limited, as is evident in the current state of global cybersecurity and the growth of distributed Cloud networks which have much stronger internal coherence.

In the 1968 CACM letter, Dijstra also argued that while loops are superfluous because of the availability of recursion. However, he did not seek to exclude while loops “for reasons of realism”. He took a similarly realistic position on the use of go to statements:

The go to statement as it stands is just too primitive; it is too much an invitation to make a mess of one’s program. One can regard and appreciate the clauses considered as bridling its use. I do not claim that the clauses mentioned are exhaustive in the sense that they will satisfy all needs, but whatever clauses are suggested (e.g. abortion clauses) they should satisfy the requirement that a programmer independent coordinate system can be maintained to describe the process in a helpful and manageable way.

Arguably, datagram routing as it stands is also just too primitive; it is too much an invitation to make a mess of one’s network. Dijkstra looked to structured programming as a means of restricting and controlling the use of unstructured branching. This has largely occurred in modern programming languages, which have greatly reduced the need for unstructured branching and its use.

Perhaps we need to look to forms of structured networking as a means of restricting and controlling the use of unstructured routing. This has occurred de facto in the deployment of Network Address Translation (NAT) in edge routers, which imposes a more hierarchical structure on wide area communication patterns.

The question of what kinds of structure might serve the purposes of a wide enough class of applications to justify the burden it might impose on the implementation of the network is a difficult one. Many argued that NAT would “kill network scalability”, but that has not happened. However the haphazard way in which it was introduced has arguably had a deleterious effect on the generality of the network. This was due in no small measure to the push-back on it from the Internet Engineering Task Force and network research community, which fought the standardization of NAT until it was globally deployed anyway.

The creation of a more structured wide area network would be a massive, intellectually challenging task. Efforts such as Content Centric Networking have foundered on the paradox that the end-to-end design of the current Internet makes the introduction of such structure at the Internetworking Layer, or Layer 3 of the Internet protocol stack is essentially impossible. Perhaps it is time to reevaluate the use of unstructured routing as the only means of implementing processes in the wide area.