Lightweight, Dynamic and Programmable Virtual Private Networks

Rebecca Isaacs
Computer Laboratory
University of Cambridge
Cambridge, UK
Rebecca.Isaacs@cl.cam.ac.uk

Abstract— **A Virtual Private Network (VPN) that exists over a public network infrastructure like the internet is both cheaper and more flexible than a network comprising dedicated semi-permanent links such as leased-lines. In contrast to leased-line private networks, the topology of such a VPN can be altered on-the-fly, and its lightweight nature means that creation and modification can take place over very short timescales.**

In a programmable networking environment, such VPNs can be enhanced with fine-grained customer control right down to the level of the physical network resources, allowing a VPN to be employed for almost any conceivable network service. This paper examines some of the issues present in the provision of programmable VPNs. In particular, automated VPN "design" is considered, that is, how a VPN description can be translated to a set of real physical resources that meets customer requirements while also satisfying the goals of the VPN Service Provider (VSP). This problem—the distribution of resource allocations across network nodes in an optimal manner—has relevance for other approaches to VPN provision such as differentiated services in the internet [1].

The work described in this paper was carried out using a programmable networks infrastructure based on the switchlets mechanism [2]. It shows that automated VPN creation resulting in a guaranteed resource allocation is a feasible procedure that works well for both the VSP and for the customer that has requested a VPN. The problems inherent in dynamic VPN reconfiguration are also briefly explored together with the methods by which these might be addressed.

I. Introduction

Until recently, the management and control functionality of networks has been tightly coupled to the network hardware, leading to a "closed" and restrictive environment. The provision of new network services, signalling protocols and other software has been exclusively the domain of network equipment manufacturers, often, but not always, under the auspices of international standards bodies. The drawbacks of this approach are widely recognised:

- Although the use of standards ensures interoperability, the process of defining them is a slow and often highly political exercise. It is not unusual that by the time a standard is produced, technological developments have made it largely obsolete.
- Due to their monolithic nature, standards can be heavyweight and unwieldy, further stifling the impetus to develop and quickly deploy new network services.
- Mechanisms for introducing innovative network services are invariably very limited. If the demand has not already been anticipated, it is unlikely the network will be able to cater for the service in the optimal manner, if at all.
- Close integration of the network control software and the internal network elements makes upgrades and bug fixes difficult and costly.

Many illustrations of these shortcomings can be cited with respect to commonly deployed networks. Examples include the awkward integration of intelligent network services into the telephone network, and the requirement for some ten thousand lines of code on every switch for UNI/NNI signalling in ATM networks.

A. Programmable Networking

Programmable networking has been advocated as a means of addressing these issues. The main idea of this approach is that the network can be programmed, in other words its physical resources accessed and manipulated, through an open programmable interface. Such an interface allows control and management functionality to be devolved from the internal network hardware to external processors, hence separating the role of equipment vendors from that of software and service providers.

Potential advantages of programmable networking include the opening up of the network to third parties, the easy introduction of sophisticated and hitherto unanticipated network services, and significant speedups in the deployment of such services. However, in practice these benefits are hard to deliver and pose some challenging questions such as how programmable network interfaces should be defined, how much abstraction is called for, as well as performance, robustness and security issues. Many of these issues are under active consideration in the research community [3].

B. Virtual Private Networks

Virtual Private Networks (VPNs) that make use of existing (possibly public) network infrastructure are a way of procuring the equivalent of a private leased-line network that is relatively cheap and flexible. Permanently dedicated resources are not required, and the topology and size of the VPN can be altered as required. These VPNs are most commonly deployed on the internet, often to provide connectivity between corporate LANs, or to enable individual mobile users to access a private LAN. The establishment of internet VPNs does not require large overheads: tunnelling is used to provide routing and addressing, and security measures such as encryption of the VPN's traffic and authentication protocols prevent data tampering and unauthorised access.

Issues of network performance and guaranteed quality of service, as well as mechanisms for pricing and charging where appropriate (especially when multiple ISPs participate at the boundary of the VPN), have not as yet been fully addressed, although their importance is recognised. Current VPN product offerings either operate solely on IP networks, or over a small number of other protocols.

C. Programmable VPNs

The provision of VPNs in a programmable network environment takes the VPN concept a step further. It contains the means for a VPN to control not just its routing and addressing mechanisms, but any aspect of the underlying network resources that can be "programmed". Thus a VPN is not constrained to use a particular networking protocol.

Depending on the programmable interface used, the exposure of the network internals can guarantee VPN performance, and provide a low-level monitoring and charging mechanism for the VPN Service Provider (VSP). This scenario is possible when there is a means to isolate one VPN from another, so that each VPN can have exclusive control of and access to "its" resources. The switchlets concept [2] does exactly this by partitioning the physical resources and policing those partitions. An open control interface allows a VPN to access its partition without interference from any other VPN.

In general, resource partitioning combined with open network control has a number of benefits. Including:

- the removal of the need for a single instantiation of a prescribed control system (i.e. signalling mechanism or networking protocol)—each partition of the network can be controlled by a different system;
- the support for true multi-service networks where each type of service can operate in its appropriate environment, unaffected by other users of the network;
- the potential for a network operator to offer a *VPN Service*, in which VPNs comprising some subset of the available physical resource (in terms of both topology and capacity), can be acquired on demand and their allocated physical resources manipulated at will.

A VPN deployed in this environment can be extremely lightweight. Its existence could be as brief as a matter of minutes, and the network control software as minimal as required. An example is a VPN created purely for the duration of a video conference, running *service specific* control software tuned to support the requirements of video conferencing [4]. During the lifetime of such a VPN, its resource requirements might change, for example as a result of participants joining or leaving the conference, and this can be reflected in corresponding changes in its underlying physical topology and resource allocation.

This paper examines some of the issues involved in the provision of programmable VPNs, from the point of view of both the VSP and the VPN customer. These issues include:

- The tradeoff between expressiveness and simplicity for a VPN specification language that must cater for descriptions that range from the minimal to the comprehensive.
- The conflict between VSP goals of maximising resource usage, and customer demands for the best VPN to meet their needs.
- The theoretical intractability of finding an optimal VPN topology given these conflicting requirements.

The feasibility of automated VPN creation has been investigated by observing how VPN topology search performs with a naive implementation, and to what extent a simple heuristic improves performance. The scenario of a VSP using the switchlets infrastructure provides the context and the motivation for this work and is described in Section II. Section III proposes an approach to automated VPN design, encompassing a specification mechanism that reconciles the goals of the two parties to produce a cost function that is subsequently used to compute a VPN topology. Section IV presents the implementation experience, and further work concerning the problems inherent in dynamic reconfiguration of VPNs is discussed in Section V.

II. Context

A. Infrastructure

The infrastructure over which dynamic VPNs are provided is based on open signalling concepts. Control of the physical network is, as far as possible, devolved from the internal elements to general-purpose workstations. The functionality of the switch or router is encapsulated in an open control interface—typical operations available through such an interface include connection management, routing, alarm notification and the gathering of statistics. Examples of open switch control interfaces include GSMP [5] and VSI [6]. We use an interface developed in the Computer Laboratory called Ariel [7], [8].

Partitioning of the physical network is achieved by subdividing the resources of individual switches into one or more logically separate *switchlets*. Each of these is presented to its owning control system through an open switch control interface, which gives the illusion that the control system is managing an entire switch. This is the crucial feature that allows multiple control systems to co-exist on a single physical network, whilst also giving them fine-grained control of the resources they have been allocated.

A process called the *Divider* manages the creation, deletion and modification of switchlets. The Divider runs on a general-purpose workstation, ideally in a resource-controlled environment such as that provided by the Nemesis operating system [9]. Invocations on switchlet interfaces are policed on the control path by a Divider (one per switch) to ensure there is no interference between control systems. Fig. 1 shows multiple control systems sharing the resources of the physical network. Each control system makes invocations on the Ariel interface exported by the Divider for the corresponding resource allocation on that switch.

A full-blown virtual network can be constructed by acquiring switchlets in one or more network nodes. The *Network Builder* is responsible for the creation and deletion of virtual networks, and for allocating the resources required by those virtual networks. The topology of a virtual network need not map directly onto the underlying physical network. A single-link virtual network can span multiple physical links simply by reserving resources on all intervening nodes without exporting Ariel interfaces—such resource reservations are termed *tunnel switchlets*.

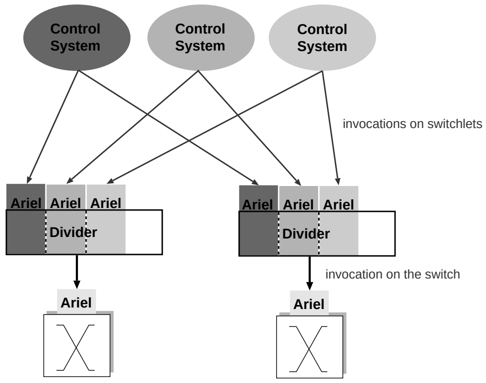


Fig. 1. Control systems operating simultaneously.

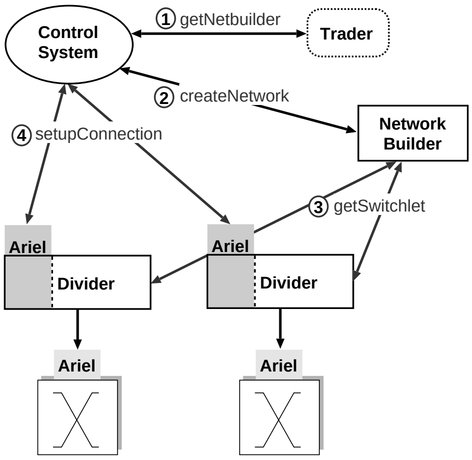


Fig. 2. Acquisition of a virtual network.

Communication between these architectural components takes place using a DPE, which runs over its own virtual network. A bootstrap virtual network is employed at start of day.

Fig. 2 illustrates the steps involved in the acquisition of a virtual network by a control system: a control system locates the Network Builder using a trader, and then asks it for a new network comprising some specified resources. The Network Builder in turn locates the Divider at each of the constituent nodes, and requests a new switchlet at each. After creating a switchlet, the Divider returns a new Ariel interface for that switchlet to the Network Builder. The switchlet interfaces are then passed back to the control system, which can then make invocations on switchlets directly and hence control its partition of the physical network resources.

The operation of control systems across possibly non-cooperating intervening networks is also addressed within the infrastructure [10]. Virtual networks can be set up that span multiple domains, with the result that a control system need not necessarily be aware of the underlying administrative boundaries. The infrastructure will take care of the tunnelling of traffic across other types of network, albeit with a potential loss of service guarantees.

The provision for multiple control systems to operate simultaneously means that no single system need be prescribed for all users (which is not to say that multiple instances of the same control system can not co-exist). Although for many users standard, general-purpose control systems will suffice, others are able to run service-specific control systems tailored to their individual needs, if they wish to do so.

An implementation of this control framework is currently operational on an ATM network consisting of 5 ATM switches, 5 host workstations and 7 audio/video codecs. Ongoing developments include novel and innovative control systems, federated virtual networks over wide-area links, investigation into pricing and charging mechanisms, dynamic resource reallocation and adaptive control systems, and control system interoperability. For a general overview and more information see [7] and [8].

B. Related Work

Virtual networks within a programmable networking environment have been explored from two different angles. In *active networks* [11] programs can be inserted into the routers or switches and then executed on the messages passing through those nodes. This form of active network is the most extreme form of network programmability, but because the control and data paths are not distinguished it is difficult to provide hard guarantees for differentiation of services.

In contrast, the alternative approaches, which include the switchlets approach, partition network resources among virtual networks and provide some mechanism for independent operation of the virtual networks. Schemes such as Genesis [12] and VNRM [13] deploy a specific control system on a virtual network by instantiating objects that implement the desired control interface and protocol.

Using switchlets, no such built-in functionality is provided, but rather a handle onto a subset of the real, physical resources, which enables low-level and fine-grained manipulation of those resources. In consequence, a virtual network instantiated by means of a switchlet will be more lightweight and hence amenable to the automated VPN design that is the subject of this paper.

As an aside, direct manipulation of physical resources does not preclude the use of off-the-shelf control systems in a switchlets virtual network (indeed we run a cut-down version of IP on one of our virtual networks). It also does not mandate a particular resource abstraction for control, hence avoids needlessly restricting operations on the resource, or compromising efficiency, flexibility and performance.

Genesis addresses the automated deployment of virtual networks and their control systems through a process called spawning. This uses a network blueprint in the form of an executable profiling script which lists requirements for addressing, routing, management, topology, resources and so on. The profiling script is produced manually by a network architect, and the conflicting goals of VSP and virtual network customer that are the subject of this paper are not addressed. Although Genesis has not as yet been implemented, the implementation plan described in [12] intends to tackle some of these issues.

Ongoing work concerning resource management in VPNs includes the "hose model" [14], in which a performance abstraction for IP-based VPNs is proposed that characterises the desired performance characteristics of a VPN by an aggregated capacity figure. Extensive evaluation using trace driven simulations shows that considerable benefit is gained by statistically multiplexing traffic across the VPN as a whole—a technique that can also be employed in the context of this work where appropriate because of the hard partitioning of resources between VPNs.

Proposals for differentiated services in the internet [1] effectively partition resources in a public network in a similar way. However, mechanisms for dynamically shifting resources from one aggregate traffic classification to another have not as yet been defined, although the desirability of such behaviour is recognised.

C. A VPN Service Provider

The nature of the issues considered in this paper is motivated by the envisaged requirements of a VPN Service Provider (VSP) making use of the infrastructure to sell dynamic and flexible VPNs to a range of customers. A VSP will benefit from the potential for efficient use of network resources, multiplexing gains inside the network, and flexible and fast response times to customer demands. The VSP may also offer extra value over the standard VPN service, for example with configurable reliability for VPNs, advance VPN reservations or the facility for dynamically loadable customer code inside a switchlet (i.e. extremely close to the physical network itself). A flexible and easy-to-use environment is needed in order to be able to introduce new services as desired.

The customers of a dynamic VPN service will include not only corporate users who currently employ VPNs over the internet or leased lines for their private network, but also companies selling network services themselves. These might be telephone companies, ISPs, off-the-street users needing a network for a short time, perhaps for a distributed game or a broadcast lecture, or even designers and developers of other network services. All these customers will have different needs over different timescales—some VPNs will be semi-permanent, while others rapidly and frequently change their resource requirements, or even only persist for a matter of minutes.

To fully realise the potential of the technology in this scenario, the VSP must offer a responsive, reliable and flexible service. Administrative overheads must be minimised; in particular the automated creation and modification of VPNs is paramount, not only to cope with large volumes of such requests, but also to be able to respond to them quickly and to cater for the many different types of customer. The remainder of this paper considers VPN specification and realisation in this context.

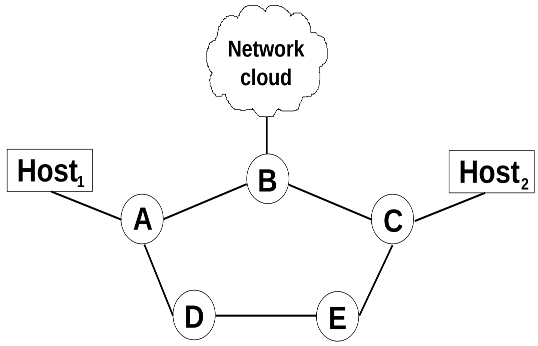


Fig. 3. Example network where preferred VPN topology from Host$_1$ to Host$_2$ is different for VSP and customer.

III. VPN Design

So far this paper has described programmable VPNs and their advantages, as well as reviewing the switchlets concept and the prototype implementation of a VPN Service that provided the motivation for this work. The deployment of such a service on a real-world network is an attractive prospect, but in order to cater to the demands created by short-lived, lightweight and dynamic VPNs, automation of their provisioning is essential. The challenges of doing so are now investigated, with the overall aim of showing that in spite of some inherent difficulties, the automated design and deployment of VPNs is a feasible procedure.

The goal in VPN design is to create a virtual network that conforms to the specification provided by the customer, whilst maximising subsequent resource availability for the VSP. In order to increase the likelihood of being able to satisfy future customer requirements, the VSP should endeavour to spread the load of bandwidth, label space and switchlet allocation. As an example, Fig. 3 shows a network topology where although a customer might prefer that their VPN from Host 1 to Host 2 passes through the single node B, the VSP will route the VPN through D and E in order to maximise the local resource availability for VPNs originating within the network cloud.

This section considers the two main steps of the design process—customer specification of the desired VPN, and a means of mapping that specification to a near optimal topology and resource allocation.

A. VPN Description

The characteristics of a VPN that a customer may specify in a creation or modification request include:

- (virtual) topology;
- performance characteristics such as bandwidth, delay, loss tolerance, size of label space etc;
- temporal attributes i.e. start time and duration;
- cost;
- built-in extras, such as redundancy;
- contractual issues—penalties etc.

This type of specification, which describes both the desired VPN and specifies the guarantees made by the VSP with respect to that VPN, is often known as a *service level agreement* (SLA). Traditional SLAs used by WAN service providers are contracts between customer and network service provider that detail the level of service agreed in terms of measurable parameters. The content of an SLA usually covers type of service, data rate and QoS issues as well as contractual matters such as charging and compensation in the event of non-compliance with the agreement.

The automated translation of a VPN specification to a set of physical resources requires a formalised notation that is sufficiently expressive to capture a range of requirements, but is not unnecessarily restrictive. As pointed out in [8], VPN descriptions may range from the ill-defined (eg, "a cheap network between A and B"), to the comprehensively specified. Furthermore, the difficulties of predicting network traffic characteristics from a given source are well known. Insisting on a precise specification when the user does not know or understand their requirements in such detail only leads to sub-optimal network usage, either because the source exceeds its usage parameters and loses data, or else because the network is over-provisioned and resources wasted.

A VPN may not necessarily be double-ended, in other words a VPN specification which describes the origin and characteristics of the traffic without explicitly stating its destination should be valid. The conversion to a complete specification should be able to take into account potential optimisations resulting from this single-endedness to multiplex where possible.

The exact nature of the notation adopted depends to some extent on the capabilities of the underlying network and the degree of specification that the VSP wishes to allow its customers. Greater freedom leads inevitably to more complexity in the system. In our experimental environment we allow a fairly limited choice of VPN parameters, namely participating nodes, size of label space, bandwidth and maximum hop count. Other characteristics that could easily be incorporated include delay, jitter, duration, start-time, redundancy and so on.

After an SLA is defined, the partial VPN specification it contains must be converted to a complete specification, that is, one that represents an instantiation of the specified VPN on the actual physical networks. This involves an augmentation of the incomplete description such that all of the necessary physical topology, together with the resources required on each node and link, is explicitly specified.

Once a complete description is arrived at, a VPN can easily be expressed as a set of switchlet specifications. The mapping of a customer-provided VPN description to a complete description is the most complex step in the process of creating a VPN. Two questions must be addressed:

1. Does the described VPN make sense in terms of the actual physical topology—is the description *feasible*?
2. How do we arrive at an arrangement of VPN topology which:

- satisfies the customer requirements,
- can be realised with available resources, and
- is optimal for the VSP?

Checking of feasibility is straightforward, as the VSP has global knowledge about the network topology. At this step it may also be able to refine the SLA so that it includes at least those nodes that *must* be present to meet the stated requirements. In contrast, the second question is quite difficult to answer, and is discussed at length in the next section.

B. VPN Routing

Determining an optimal route, or topology, of a VPN is a non-trivial problem. It can be expressed formally as follows:
The cost of a subgraph G' is determined by some function $f(G')$. Given a weighted graph G and a set of vertices in G denoted V, what is the cheapest way of forming a connected subgraph containing V according to the cost function f?

This problem is similar to that of finding a minimum Steiner Tree, where the goal is to connect a set of vertices in the graph by finding a minimum-weight spanning tree that can also use any of the remaining vertices. The difference is that the aim is not to find a minimum spanning tree, but to find the cheapest subgraph according to a cost function that will vary from one VSP to another, and may even be altered over time at the same VSP.

As an example consider the networks shown in Fig. 4. The requested VPN consists of the nodes A, B and C, and the edge weights have been calculated as shown, according to the VPN resource description, current resource availability and local policy. A costing function that favours the minimum-weight VPN topology will produce the subgraph highlighted in the left-hand network, whereas the subgraph of the middle network is produced if minimising the number of links is given greater priority. If full redundancy together with minimum-weight is required, the subgraph highlighted in the right-hand network will be the result.

It is clear from the problem description that once a partial VPN specification has been derived from the given SLA, there are then three stages in reaching a solution:

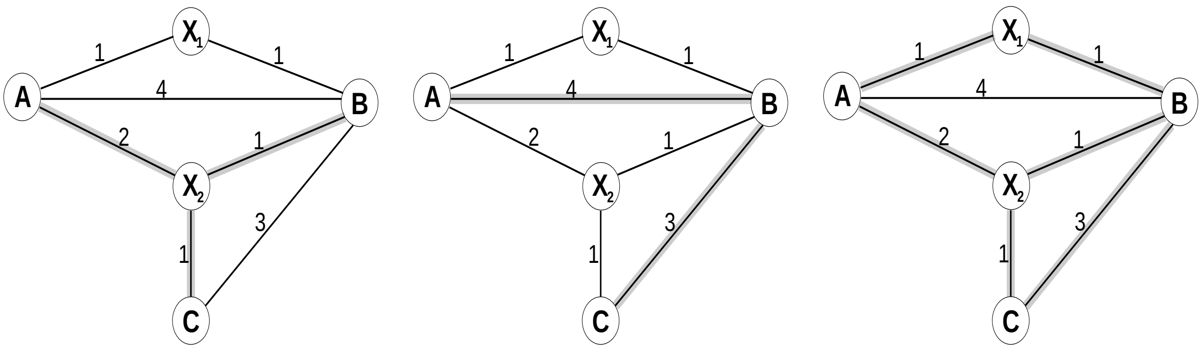
B.1 Assign edge weights

The weight assigned to any individual edge will depend on:
- availability of the resources specified in the VPN description (either explicitly or implicitly) as required for that link;
- any arbitrary cost associated with a link, as determined by local policy.

A link that is unable to provide the required resources is immediately assigned a weight of infinity and removed from further consideration.

B.2 Generate a cost function

The cost of a VPN is determined by a combination of customer requirements—a candidate VPN that doesn't meet the specifications of the SLA will have infinite cost—and the circumstances of the individual VSP. For example, in some cases bandwidth may "cost" more than label space (perhaps because



Minimum weight. Minimum number of links. Minimum weight with redundancy.

Fig. 4. Three cost functions producing different optimal VPN topologies.

the VSP has very few customers), whereas in other networks the reverse will be true. The costing function should reflect the relative priorities in the local domain, and balance the requirements for a particular VPN with the overall needs of the VSP.

A cost function routine is automatically generated by parsing the SLA to derive explicit rules and incorporating these in a template that includes rules dictated by local policy. The details of this procedure will vary from one VSP to another, but there is always commonality in that the cost calculation routine used at any one time is (probably) unique to the individual VPN under consideration.

B.3 Find the best VPN topology

As stated above, a VPN's topology will be determined by a combination of the edge weights and the cost function, according to local policy.

Intuitively, an implementation that always finds the best subgraph (i.e. VPN topology) will be computationally expensive because of the large number of candidate subgraphs. If we consider a subgraph cost function that is simply the sum of the constituent edge weights, and if cycles are not permitted in the subgraph, then the problem becomes that of finding the minimum Steiner Tree. This is known to be NP-complete [15], and thus so is the optimal VPN topology problem described here.

Nevertheless, using a combination of sensible heuristics and careful engineering, an implementation can be produced that gives tolerable performance for a network of reasonable size. In the following section the brute-force solution is analysed in order to derive heuristics that give close to optimal solutions, as well as making the problem tractable within the desired bounds. These bounds are determined by deciding what "tolerable" performance is, and how large a "reasonable" size of network should be. Experimental observation supports the claim that the performance of the resulting algorithm is adequate.

IV. IMPLEMENTATION EXPERIENCE

This section examines in some detail the implementation of the search for an optimal VPN topology. By means of experimental results we compare the performance of the naive approach with that possible using a simple heuristic.

A. Algorithm

A brute force method of determining the optimal VPN topology is to construct every possible connected subgraph that satisfies the given VPN description, calculate the cost of each and then choose the cheapest. The pseudo-code of Fig. 5 describes an algorithm that takes this approach. For clarity, finer points of the implementation such as finding optimal topologies that incorporate redundant links are ignored.

Let $V^{'}$ denote the set of nodes in the VPN. For any $v \in V^{'}$, `single-hops[`v`]` contains all paths from v to each of the other nodes in $V^{'}$, that do not pass through any other node in $V^{'}$ and do not contain any cycles. The function `CALCULATE-COST` is generated as described in Section III-B.2.

The construction of `single-hops` is straightforward using a modified form of the Floyd-Warshall transitive closure algorithm to generate one possible set of paths between VPN nodes, and then taking the transitive closure of the resulting paths to obtain *all* possible paths. At this stage paths with edges that have infinite weight (i.e. cannot fulfill the resource requirements) are discarded.

The algorithm to find the minimum-cost subgraph containing $V^{'}$ is shown in Fig. 5. At line 7 whichever VPN node was added last to the subgraph is taken, and its paths to other VPN nodes looked up in `single-hops`. The procedure is then called recursively for each of these paths. Note that the resulting collection of paths will not necessarily be disjoint, but a subgraph is a candidate (i.e. costed) only if it contains all of the nodes in the VPN. At line 8 processing is also carried out to ensure that continuously cycling paths are not considered, and to guarantee termination within a reasonable time period, the search is abandoned after a certain (large) number of recursions.

```
MIN-SUBGRAPH(subgraph)
1   if all nodes V' in subgraph then
2      c = CALCULATE-COST(subgraph)
3         if c < min-cost then
4            min-cost = c
5            min-subgraph = subgraph
6   else
7      paths = single-hops[subgraph.last]
8      for each p in paths do
9         MIN-SUBGRAPH(subgraph + p)
```

Fig. 5. Brute-force algorithm to find a minimum-cost subgraph.

Let m be the number of edges in the graph G and let k denote the number of nodes in the VPN. The number of paths between any two nodes without cycles is at most 2^m, and for any node v there are paths to at most $k-1$ other nodes. Therefore the maximum number of iterations of the loop at line 8 is at most $(k-1)$, with each iteration recursing at most 2^m times. As expected, the brute-force algorithm is computationally intractable.

B. Heuristics

The intractability of this algorithm arises because the cost function is not monotonically increasing, i.e. the overall cost of a subgraph may reduce as it encompasses more of the VPN nodes (for example if redundancy is required by the customer). As stated above, if the cost function does happen to be cumulative, then the solution is the minimum spanning tree and can be found in polynomial time using any of the well known algorithms.

However, even with a non-monotonic cost function, heuristics can be applied to make the procedure practical for networks of reasonable size. At the potential expense of sub-optimal results, the search time can be speeded up by reducing the size of the search space. Two approaches are possible:

- abandon partial subgraphs that are thought likely not to be optimal in the future;
- order the search sequence to consider first those subgraphs that are most likely to be optimal and stop the search at the first hit.

The first approach can be implemented by choosing a reasonable cut-off point, and abandoning partial subgraphs that exceed this cost. This is obviously more likely to perform well if the cost function is 'almost' monotonic. Additionally, the cost function may contain cumulative aspects that can be considered in isolation. For example, partial subgraphs can be discarded on the basis of exceeding the desired hop count or end-to-end delay. However, a significant drawback with this approach is that the additional computation associated with calculating costs of partial subgraphs may dominate the running time and negate any improvements gained from the smaller number of subgraphs considered in total.

In contrast, the second technique will tend not to return as good results, especially with denser graphs, but will spend significantly less time calculating partial subgraph costs. Another key advantage is that if a subgraph that meets the customer's requirements (with the possible exception of a cost constraint) exists, then it will always be found. The first technique runs the risk of not finding a solution, due to having abandoned that subgraph earlier, whereas with ordered search if a solution exists then it is guaranteed to be found eventually.

Thus with the ordered search heuristic the VSP may lose out on fulfilling its overall goals, and the customer may be penalised on VPN price, but both gain substantially from search speed increases. Of course in practice the VSP can subsequently mitigate any losses by modifying the cost function appropriately. The results in the following section support this analysis, and demonstrate that by incorporating the ordered search heuristic, automated VPN design is a feasible procedure.

C. Experimentation

This section presents experimental results to back up the assertion that in spite of its computational complexity, the process of automated VPN creation is practical on a reasonably large network.

For ease of implementation, an interpreted scripting language was used, hence the CPU usage timings give an idea of relative improvements rather than demonstrating what can be achieved absolutely. A well-engineered solution written with the appropriate tools would perform much better. The network topologies were generated using the TIERS random network topology simulator [16], with a small range of edge densities reflecting the sparsity generally found in real-life networks. The networks are assumed to be under the administrative control of a single VSP and are accordingly no larger than 25 nodes.

Because the topologies are produced randomly no two runs give the same results and there are occasional large fluctuations for relatively large and dense networks. Notwithstanding this, some care has been taken to ensure that the results included represent typical executions, and all data points are average values over 10 runs.

The graph in Fig. 6 shows how badly the brute-force algorithm actually performs in practice. It indicates the extreme deterioration in performance as both the graph size and the proportional VPN size increase. Once the VPN covers more than about 30% of the network, the search time increases dramatically.

Problems are also caused for brute-force searching by increases in graph density, and this is shown by the graph in Fig. 7. This graph shows the search time for a VPN covering a fixed proportion of the physical network—60%, with average node degree increasing from 1 to 4. Search times increase substantially once the graph size exceeds 10 nodes. Clearly the brute-force approach is not adequate for any network of reasonable size.

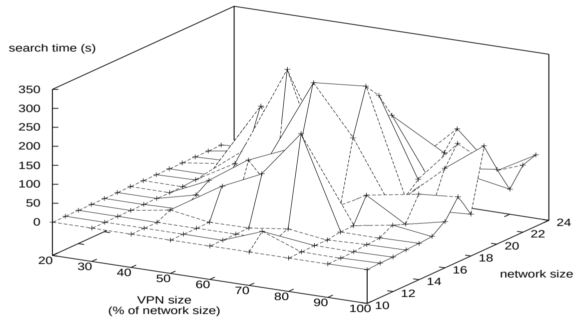


Fig. 6. Performance of brute-force search on sparse networks.

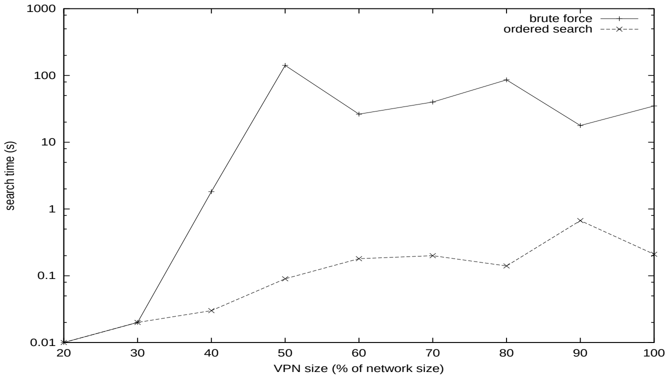


Fig. 8. Brute-force vs ordered search on a 20-node sparse network.

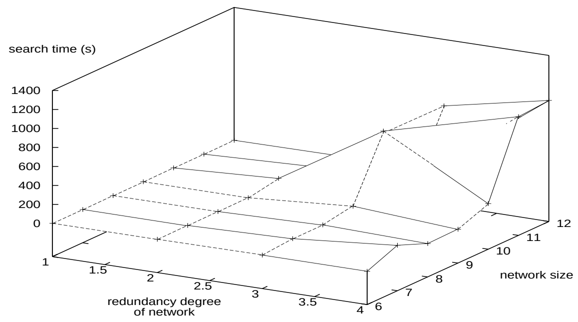


Fig. 7. Performance of brute-force search for VPN size 60% of network size.

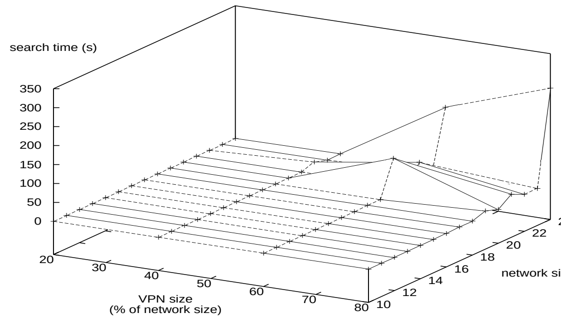


Fig. 9. Performance of the ordered search heuristic on sparse networks.

The next experiment examines the gains that can be made using the ordered search heuristic as well as the corresponding cost penalty. The heuristic is incorporated in the pseudo-code algorithm of Fig. 5 by ordering the `paths` list iterated over at line 8 according to the result of applying the cost function to each one. Recall that these are acyclic single-hop paths between 2 nodes of the VPN, therefore the overheads of the cost calculations are minimal.

The graph in Fig. 8 compares the search times of the brute-force algorithm and the ordered search algorithm for a medium-sized graph of 20 nodes. It shows the expected marked improvement in performance using the heuristic. Note that the y-axis is log scale.

Finally, the first two experiments run using brute-force searching are repeated with the ordered search heuristic. To facilitate comparison, the results are plotted with the same Z-axis range which confirms that the large humps present in the graphs of Figs. 6 and 7 only appear here for much bigger networks. As expected, ordered search performs much better in both cases.

V. Further Work

So far the issue of automated VPN creation has been addressed without regard to possible reconfigurations of the VPN in the future. In fact, the ability to alter the topology and resource allocation of a VPN on-the-fly is one of the main advantages of the VPN Service described in this paper. Some scenarios where dynamic reconfiguration might be used include:

- A service-specific control system tailored to a particular multicast application, for example video-conferencing, where changes to group membership may require switchlets to be created at additional nodes, or even for nodes to be removed from the VPN. Similarly, end-user requests for improvements in video quality may be met by increasing the bandwidth allocation of the VPN on the relevant links.
- A control system for a VPN supporting mobility that modifies its topology in response to the movement of wireless devices—holding on to resources only at adjacent base stations, and discarding those that are "far away" from the current location.
- An efficient IP-on-ATM control system where the size of the label space in the underlying switchlets is altered rapidly in response to the creation or termination of traffic flows.

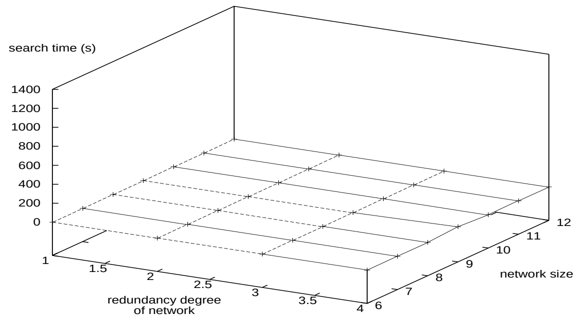


Fig. 10. Performance of the ordered search heuristic for VPN size 60% of network size.

VPNs that reconfigure as and when required can make more efficient use of network resources and are more flexible than statically pre-configured networks. The risk for the VPN owner is that a reconfiguration request may be rejected by the VSP, perhaps because of lack of resources, with the resulting loss for the VPN owner of efficiency, flexibility and possibly even the ability to provide a service to their own customers. The appropriate charging mechanisms to reflect this trade-off are the responsibility of the VSP.

Automated reconfiguration should be built in to the system for the same reasons that automated creation and deletion are necessary, as discussed in Section II-C. Reconfiguration can involve changes to either the VPN topology, or to its resource allocation, or both. The problem considered here concerns just alterations to VPN topology, which may of course result in changes to the overall VPN resource allocation as well. The details of in-place reconfiguration of resources at a particular switch are currently the subject of ongoing research.

A. Analogy With Dynamic Multicast Routing

The problems of dynamic VPN topology reconfiguration parallel those found in dynamic multicast routing (but not for datagram networks). The issues, which are extensively discussed in the literature (see, for example [17]) involve compromises between optimal placement of a node joining the multicast session and the corresponding deterioration of the tree as a whole.

As with VPN creation, the initiation of a multicast session generally involves a computation of the optimal topology. The topology of multicast trees, which are single source to multiple receivers, can be influenced by many different constraints. These include scalability, QoS requirements such as end-to-end delay bounds, efficiency requirements and algorithm complexity considerations. When a new destination is introduced into the multicast tree, naive (computationally cheap) placement of the new node can result in deterioration of the quality of the tree—with some chance of making the initial effort expended in calculating an optimal tree a waste of time. On the other hand, constant reconfigurations of the topology to maintain an optimal, or close to optimal, tree are also expensive, and can disrupt traffic to existing members of the multicast group. Most solutions adopt a compromise where changes to the tree are kept as localised as possible, and periodically the entire tree is re-routed to try and maintain a close to optimal topology.

Some examples of current research in this area include [18] and [19]. In [18], an algorithm is presented that maintains a good (but not optimal) multicast tree (in terms of minimising the sum of the edge weights) without undue computational cost using Kruskal's shortest-path algorithm. Rearrangements are triggered after a certain amount of deterioration in the quality of the tree—determined by counting the number of changes in a vicinity—but enough state is maintained to be able to confine the necessary rearrangements to localised regions. In contrast, in [19] the quality of the multicast tree is assessed according to whether it meets delay variation constraints. However there is a similar emphasis on minimising the effects of leave and join operations on the tree as a whole.

B. Discussion

Although it has a lot in common with the multicast problem, dynamic reconfiguration in the context of VPNs in the switchlets environment has some important differences.

On the one hand, the allocation of physical network resources directly to a VPN owner means that topological rearrangements of the VPN at the whim of the VSP are not necessarily going to suit the way the customer is using their portion of the network. On the other, the difficulties can be eased by the fact that the customer may be able to explicitly identify "sensitive" regions of their network that should not be modified, and nodes where disruption is tolerated. Any threshold of deterioration (which will roughly translate to cost to the customer) can be chosen on a per-VPN basis, and to take this to an extreme, the mechanism by which the topology is updated can also be specified by the VPN customer. In effect this gives the VPN owner, i.e. the person paying for the network, a great deal of control, allowing them to tailor the dynamic reconfiguration behaviour as appropriate.

However, it must also be expected that many VPN customers will not willing or able to specify such details. A minimal VPN specification such as "Give me a network containing nodes A, B and C" which at some point in the future is modified by "Add node D to my network" is a perfectly valid one. The question is whether this customer would be happy with a less than optimal topology where D is simply joined to A, B and C, or whether a rearrangement of all the links in the VPN at this point would be acceptable. In general it seems unlikely that the latter option would be preferred, and indeed should a customer require this they could always request a new network containing A, B, C and D at the appropriate time. On the other side of the coin, the VSP may have expended much effort in calculating the original network topology, and this effort may subsequently be rendered useless by the changes requested by the customer, especially if these changes occur frequently. It would be in the interests of the VSP to have some sort of characterisation of the "dynamicity" of a VPN at the time of creation. A VPN that is likely to change a lot over its lifetime can be given a topology that is sub-optimal in terms of cost but is more resilient and gives a cheaper overall topology with plentiful changes.

In summary, the following aspects relating to reconfiguration can be specified by the VPN customer:

- the expected rate of membership modification;
- tolerance of disruption to the VPN as a whole;
- tolerance of disruption at particular nodes;
- any preferred means of rearrangement;
- thresholds for triggering rearrangement;
- amount prepared to pay.

A combination of these factors could be used to influence the choice of initial routing. A VPN that underestimates its degree of dynamicity will initially pay an extra cost for unanticipated reorganisation overheads. However it is also possible in this situation that the system could slowly adapt to the observed behaviour of a misbehaving VPN, and successively produce less optimal but more resilient topologies.

The nature of a VPN topology that is resilient to change, in terms of maintaining its overall cost as its membership alters, will vary according to the cost function in use by the VSP. Algorithms proposed for dynamic multicast routing, such as ARIES [18], could be adapted to operate in this environment according to the chosen cost function. Investigation into this aspect of the provision of dynamic VPNs is continuing.

VI. Conclusion

The automation of VPN design and deployment has many advantages for both VSP and customer. It allows a customer to obtain a VPN extremely quickly, while still being able to explicitly specify the VPN's performance and other characteristics. The administrative overheads for the VSP are greatly reduced, and network usage efficiency is enhanced. With the incorporation of charging mechanisms correlated against computed VPN cost, the administrative burden can easily be further minimised.

This paper has shown that in spite of computational complexities it is feasible to automate VPN topology generation in accordance with both customer requirements for the VPN itself and VSP goals for the network as a whole. A VPN service level agreement, together with current resource allocation and global policy, can be incorporated in a cost function that is then used to determine the optimal physical topology of a VPN that satisfies the specification. The generation of this topology is made computationally tractable by means of an ordered search heuristic. Experiments have demonstrated that even within a non-optimised environment, a VPN can generally be determined in under 2 minutes on a sparse, reasonably sized network of up to about 25 nodes.

A programmable network infrastructure, such as that facilitated by switchlets, opens up the way in which a VPN can be exploited. As well as a flexible and dynamic network topology and resource allocation, there are no built-in restrictions on the control system, network protocol or end-user applications using that network. This "open" environment, with lightweight and dynamic VPNs, is a practical realisation of "networks-on-demand".

References

[1] S.Blake, D.Black, M.Carlson, E.Davies, Z.Wang, and W.Weiss. An architecture for differentiated services. RFC 2475, December 1998.

[2] Jacobus E. van der Merwe and Ian Leslie. Switchlets and dynamic virtual ATM networks. In Aurel Lazar, Roberto Saracco, and Rolf Stadler, editors, *Integrated Network Management V*, pages 355–368. IFIP & IEEE, Chapman & Hall, May 1997.

[3] Opensig working group. Details at `http://comet.columbia.edu/opensig/`.

[4] Jacobus E. van der Merwe and Ian M. Leslie. Service specific control architectures for ATM. *IEEE Journal on Selected Areas in Communication*, 16(3):424–436, April 1998.

[5] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching, T. Lyon, and G. Minshall. Ipsilon's General Switch Management Protocol specification version 2.0. RFC 2297, March 1998.

[6] William P. Buckley. Virtual Switch Interface (VSI) implementation agreement. Available from `http://www.msforum.org/`, November 1998.

[7] Sean Rooney, Jacobus E. van der Merwe, Simon A. Crosby, and Ian M. Leslie. The Tempest: A framework for safe, resource-assured programmable networks. *IEEE Communications Magazine*, 36(10):42–53, October 1998.

[8] Jacobus E. van der Merwe, Sean Rooney, Ian Leslie, and Simon Crosby. The Tempest—a practical framework for network programmability. *IEEE Network Magazine*, 12(3):20–28, May 1998.

[9] Ian Leslie, Derek McAuley, Richard Black, Timothy Roscoe, Paul Barham, David Evers, Robin Fairbairns, and Eoin Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas in Communication*, 14(7):1280–1297, September 1996.

[10] Herbert Bos. Application-specific policies: Beyond the domain boundaries. In Morris Sloman, Subrata Mazumdar, and Emil Lupu, editors, *Integrated Network Management VI*, pages 827–840, Boston, May 1999. IFIP & IEEE, Chapman & Hall.

[11] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[12] Andrew T. Campbell, Michael E. Kounavis, Daniel A. Villela, John B. Vicente, Hermann G. De Meer, Kazuho Miki, and Kalai S. Kalaichelvan. Spawning networks. *IEEE Network Magazine*, 13(4):16–29, July/August 1999.

[13] Andrew Do-Sung Jun and Alberto Leon-Garcia. Virtual network resources management: A divide-and-conquer approach for the control of future networks. In *Proceedings of the IEEE Global Telecommunications Conference (Globecom 98)*, Sydney, Australia, 1998.

[14] N.G. Duffield, Pawan Goyal, Albert Greenberg, Partho Mishra, K.K. Ramakrishnan, and Jacobus E. Van der Merwe. A flexible model for resource management in virtual private networks. *Computer Communication Review*, 29(4):95–108, October 1999. Proceedings of SIGCOMM September 1999.

[15] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[16] Matthew B. Doar. A better model for generating test networks. In *Proceedings of the IEEE Global Telecommunications Conference (Globecom 96)*, pages 86–93, London, UK, November 1996. Source code available from `ftp://ftp.nexen.com/pub/papers/tiers1.2.tar.gz`.

[17] Matthew Doar and Ian Leslie. How bad is naive multicast routing? In *IEEE INFOCOM'93*, pages 82–89, San Francisco, USA, April 1993.

[18] Fred Bauer and Anujan Varma. ARIES: A rearrangeable inexpensive edge-based on-line Steiner algorithm. *IEEE Journal on Selected Areas in Communication*, 15(3):382–397, April 1998.

[19] George N. Rouskas and Ilia Baldine. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal on Selected Areas in Communication*, 15(3):346–356, April 1998.