Networks without Borders: Communication despite Disconnection

Gregory Kempe Norman C. Hutchinson
Department of Computer Science
University of British Columbia, Canada
{kempe, norm}@cs.ubc.ca

Abstract

Communication in the face of intermittent, short-lived and unreliable connectivity can be difficult when relying solely on Internet protocols which have an implicit assumption of well-connectedness. Furthermore, use of these protocols is impossible when there is no fully connected end-to-end path between hosts. We describe Euonym, an architecture that allows arbitrary intermediate hosts to be interposed between endpoints on-the-fly. These hosts provide routing, buffering and other support services to help relieve reliance on end-to-end paths and support communication in heterogeneous, intermittently-connected networks. We use the architecture to demonstrate use of regular TCP-based applications across long-term disconnections, highly heterogeneous links and networks without connected end-to-end paths.

1. Introduction

Some emerging networks and networks in remote or inaccessible areas lack the infrastructure that is taken for granted in the well-connected Internet [6, 10]. External connections are occasional and may be irregular or unreliable. These networks often lack sustained, connected end-to-end paths between hosts, particularly when there are multiple intermittent components on a path with no common window of availability. The reliance of Internet protocols on end-to-end paths makes their use problematic.

In these environments, it is important to reduce the impact of disruptions and make as much use of connections, when they arise, as possible. This includes relieving the dependence on connected end-to-end paths and augmenting Internet-style connectivity with more unconventional data transmission mechanisms, especially if they are the only feasible connection options.

We present *Euonym*,[1] an architecture that allows arbitrary intermediate hosts to be explicitly interposed on the path between two endpoints in order to support communication in heterogeneous, intermittently-connected networks and relieve the reliance on connected end-to-end paths. The architecture uses a layer of opaque *names* to persistently identify both end and intermediate hosts. By grouping names into *name stacks*, intermediate hosts can be chained together and invoked as necessary—even on the fly—in order to provide support services such as buffering and routing.

We demonstrate that intermediate hosts are a powerful tool in challenged networks: they can help maintain data flow across long-lived disconnections, bridge heterogeneous network elements and supplement regular Internet connections with novel forms of transport, such as data mules, without specific support from the end hosts. We show that regular, TCP-based applications can use the architecture to communicate in heterogeneous networks lacking connected end-to-end paths and with lengthy disconnections.

[1] *euonym*: a name well-suited to the person, place or thing named.

2. Intermittently-Connected Networks

We illustrate the work through an example inspired by the Wizzy Digital Courier project [10]. The project provides connectivity to outlying schools in South Africa by buffering web and email requests until off peak hours when telephone costs are low enough to make dialing into an ISP cost effective.

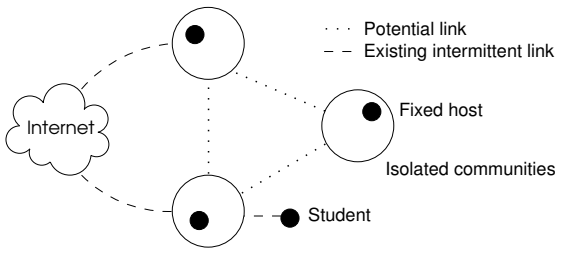

Figure 1. Student in a remote community with intermittent connections to other networks.

Consider a student doing research in an isolated community, Figure 1. The student is connected to the com-

Proceedings of the Advanced International Conference on Telecommunications and
International Conference on Internet and Web Applications and Services (AICT/ICIW 2006)
0-7695-2522-9/06 $20.00 © 2006 IEEE

munity's small LAN, centred at the local school. Long-distance phone call costs are such that the network is connected to the Internet only over the weekends using a dialup connection. The student collaborates with colleagues in nearby communities that have similar, or even entirely isolated, networks. Maintaining connections is difficult in these circumstances when relying upon applications dependent on Internet protocols and end-to-end paths, such as ftp, scp and ssh. When new network links are available, application-level connections must be re-established, remote applications restarted and file transfers resumed or restarted. In addition, despite being physically close, the local communities are forced to rely on a third party network (the Internet) to communicate.

There are a number of opportunities to connect the communities using non-traditional means, such as inter-community commuter traffic, dedicated data mules and the postal network. We show how the student can take advantage of these options, make the most of the limited connectivity already available, and remove the reliance on third party networks, all by using intermediate hosts.

2.1. Supporting Intermittent Connectivity

The example above illustrates two main causes of difficulty when working with these networks:

- connection interruptions and a lack of connected end-to-end paths, and
- lack of infrastructure and infrastructure heterogeneity.

Our architecture tackles both of these problems. We next discuss disconnection tolerance and the reliance on end-to-end paths and how intermediate hosts can be used to support connectivity. Following that, we show how the same architecture naturally supports extending the network with novel transmission mechanisms and allows heterogeneous network elements to interoperate.

2.2. Disconnection Tolerance

Disconnections are common in these networks and may range in duration from a few seconds to hours, days or even longer. We address two types of disconnection:

- *Long-lived disconnections:* The network is predominantly disconnected and connections are brief and far between. When available, connections are readily usable (i.e. have reasonable bandwidth and delay) and provide connected end-to-end paths. Connection resumption must be supported and host identity persisted as networks and addresses are likely to change over such time spans. Applications should be aware of disconnections in order to provide user feedback.
- *Systemic disconnections:* Long-lived disconnections occur throughout the network and interfere with end-to-end paths. The reliance on end-to-end paths must be dropped and, because handling complex disconnections can exceed the abilities of the network and application, user input may be needed to determine usable paths or provide routing.

Traditionally, disconnections are considered fatal errors in the well-connected model of Internet-based networks. Instead, we consider them to be transient, non-fatal occurrences that can be tolerated and managed. We provide disconnection tolerance in a session layer between the application and the transport. It hides disconnections up to an application-specified duration, beyond which it interacts with the application and user to make further decisions. The use of a session layer, rather than modifying the transport or network, is important if we are to bridge different networks and their transport protocols, as we describe later.

2.3. End-to-End Paths and Intermediate Hosts

Hosts require an immediate end-to-end path when they must take all responsibility for the data that they send. In a well-connected network, this is a reasonable expectation. However, it becomes problematic when data reaches a disconnection in the network which the end host has insufficient information (or control) to work around. The hosts closer to a disconnection are often in a better position to handle it. Similarly, and more importantly, hosts closer to an intermittent connection are in a better position to make use of it than those separated from it by other disconnections. Our architecture allows these hosts to be explicitly included in a connection, allowing end hosts to take advantage of their position and services. By delegating responsibility for data to downstream hosts, end hosts can make use of connections and networks that they might not ordinarily have even been aware of.

The student in our example scenario can nominate a host in the community LAN as a temporary storage point for incoming data while he is unavailable. Upon his return to the network, the intermediate host forwards the buffered data on to him. Similarly, a host in the community can be interposed as a forwarding point for data going to other communities and the Internet, allowing pre-buffered data to be sent later when a connection is established, even if the student is no longer connected.

Intermediate hosts can provide a range of support services to address deficiencies in challenged networks and relax the dependence on connected end-to-end paths. These include

- *Routing:* Perform high-level routing decisions based on present and upcoming connection opportunities,

their duration, quality of service, cost etc.

- *Forwarding points:* Act on behalf of end hosts when they are unavailable, buffering data from upstream hosts and forwarding it on when a downstream connection becomes available.
- *Rendezvous points:* Provide a common contact point when a link can only be established from one side (e.g. because of a firewall); this lends itself naturally to a *put/get* communication model.
- *Message consolidation:* Group streams of data into a single stream to be transported *en masse*, important if a link is only cost effective above a certain utilisation level.

2.4. Heterogeneity

The Euonym architecture encourages the expansion of networks using novel transport mechanisms. Intermediate hosts can be used to sew different network types together without requiring knowledge of those networks at the end hosts. Any number of heterogeneous networks can be traversed provided proxies exist at the edges. They do not need to support Internet protocols such as TCP and IP or even have Internet-like characteristics, such as low latency and reasonably high throughput. The networks are free to use their own transport and addressing protocols within their borders. Moreover, applications and end hosts do not need to understand or support these intermediate networks or even be designed with their inclusion in mind.

2.5. The End-to-End Argument

Intermediate hosts are not new in the Internet; middleboxes such as NAT devices and proxies provide address translation and performance enhancements, often by violating the end-to-end argument and fate sharing principle, to the detriment of the greater network [2]. We argue that because intermediate hosts are explicitly included in the connection, they do not violate these two principles. The end hosts are aware of the intermediate hosts, and the intermediate hosts only make use of data that is addressed to them. In no way are the intermediate hosts trying to deceive the end hosts or tamper with application-level data.

3. The Euonym Architecture

Euonym lies between the application and transport layer. We term a logical connection between source and destination end hosts a *flow*, and the transport level connections between all hosts *connections*. Flows span intermediate hosts whereas connections between successive pairs of hosts do not.

Flows are unidirectional and do not rely on bidirectional transport layer connections. A return flow is established independently of a forward flow and may use different intermediate hosts. This allows different services to be invoked if, for instance, the reverse flow is established a while after the initial flow and network connections have since changed.

3.1. Host and Service Identification

Every Euonym host has one or more *names*: flat (non-aggregable), globally unique identifiers, based on the endpoint identifiers described by Balakrishnan *et al.* [1]. A name is used to persistently identify a host and, when used outside of its network, to identify the routing, services and networks required to reach it (see Section 3.3).

Flexible name interpretation is an important aspect of the architecture. Names must have diverse uses without being encumbered by semantics such as location, administrative domain, network membership or host type. For instance, as challenged networks are often divided into isolated regions, it is tempting to identify a host by its region and network address. This simplifies routing but complicates management and limits flexibility, especially since hosts are likely to be mobile and regions ephemeral. Any aspect of a host that changes during its lifetime without affecting its identity should not require a change of name.

3.2. Name Stacks and Name Resolution

A name maps either to a transport-friendly *address* (e.g. an IP address and optional port number) or to a *name stack*: an ordered list of names and addresses. Name stacks are used to involve intermediate hosts in a flow and provide late binding and source- and destination-specified routing. Address formats are network-dependent and are not dictated by the architecture; a host using a certain address must simply be able to differentiate it from a name.

Name resolution is performed recursively until an address is found. Starting with a stack (a singleton name is first converted to a stack), the top entry is inspected without being removed; finding an address ends the process and the stack is returned, otherwise the name's mapping is looked up and pushed onto the stack, above the name itself. A name that maps to the empty stack is removed and the process continues. That is, it corresponds to the null service: reaching the host (or network) requires no work. A name with no mapping produces an error, and the resolution process always returns a stack, albeit possibly with only one item.

Lookup is done using a network-local naming service in which hosts have stored the appropriate mappings for their names. As names are flat, a DHT-like service is likely to be

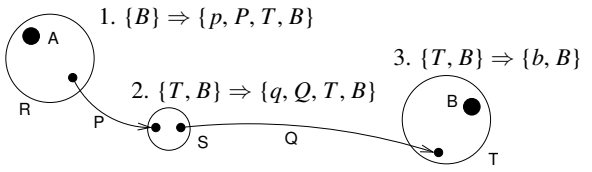

Figure 2. Flow establishment, late binding and name interpretation. Host X has address x.

favoured over a hierarchical one such as DNS. Intermittent and disconnected links form natural borders around well-connected hosts and divide the larger network into separate regions, each with their own naming service.

3.3. Flow Creation

We illustrate names and name stacks with an example (Figure 2). Host A in network R contacts host B in network T. To do so, it goes via network S using the services P and Q. Local name services map $B \Rightarrow \{T\}$ and $T \Rightarrow \{P\}$ in R, and $T \Rightarrow \{Q\}$ in S. Other host names map to their addresses (lowercase letters).

A recursively resolves B to $\{p, P, T, B\}$, pops the top entry (p) and connects to it. In the handshake, it sends its own name, the flow origin's name and the modified stack. P pops the first entry and verifies that it matches its own name, then moves the data and the stack into network S in some way (satellite, data mule etc.). In S, the receiving host follows the same process, resolving the stack—now $\{T, B\}$—to produce $\{q, Q, T, B\}$, popping the top and connecting to q. After Q moves the data into network T, the stack $\{T, B\}$ is resolved to $\{b, B\}$. When B is contacted, it receives $\{B\}$ and pops the top entry. The stack is now empty so B knows it is the end host of a flow with origin A.

Downstream hosts are autonomous of upstream hosts. In particular, a flow's destination is unaware of all hosts between it and the flow's origin; intermediaries can be added and removed upstream as required without affecting it. There is no need to preserve stack information for the return flow since it is established independently.

3.4. Source-Specified Routing and Late Binding

Origin hosts can use source-specified routing to invoke services on outgoing flows. For example, to include service Q on a flow from A to B, the application at A specifies $\{Q, B\}$ as the flow destination. As a result, B is resolved by Q instead of A. Intermediate hosts can achieve the same effect by pushing names onto an outgoing flow's name stack before resolving it.

Hosts can use late binding to specify successive intermediate hosts and delay resolution of later names to distant points in the network. In Figure 2, for instance, B is only actually resolved to an address inside T. Prior to that, it merely identifies which network to find a route to. Similarly, T is repeatedly re-resolved at different points to identify services required to reach the network. Once there, T maps to the empty stack (the null service) and is removed, its function complete.

4. Implementation and Demonstrations

We have implemented a prototype of the Euonym architecture in Java. It provides a socket-oriented, stream-based API to the application, where names and name stacks substitute for host addresses and data is sent and received through regular Java stream objects. The implementation primarily uses TCP as its transport layer but also allows a stream to be written to a file and later read in and injected back into the network. Regular applications interact with the implementation using proxy services.

We next describe two experiments that use simulated link and host availability to demonstrate the efficacy of Euonym, in particular for regular TCP-based applications. In addition to these demonstrations, we have also used Euonym to maintain ssh connections on the order of weeks, despite lengthy network disruptions and address changes. We have also used it to swap to a rendezvous-style put/get paradigm mid-way through a connection. This is inherently nonreliant on a connected end-to-end path and useful when the upstream host is unable to contact the downstream host directly, possibly because ingress connections cannot be established directly (e.g. due to a firewall).

4.1. File Transfer

In this demonstration (Figure 3a) we transfer 116 340KB of data across two intermittent links. It shows use of the architecture by regular applications, tolerance of lengthy disconnections, interposition of arbitrary intermediate hosts mid-flow, and the use of intermediaries for data forwarding and to support two very different transport mechanisms: TCP and a simulated data mule.

Two isolated community networks, G and H, are connected to the Internet (I) by 16KBps intermittent links. G's link is up for 20 minutes every 30 minutes and H's link for 20 minutes every 40 minutes. Host A in G uses scp[2] to copy a file from B using P in the Internet as a forwarding point. Part of the transfer is performed by a high-bandwidth data courier moving directly from H to G.

All names map to host addresses, except: in G, $B \Rightarrow \{H\}$, $H \Rightarrow \{P\}$ and $G \Rightarrow \{\}$; in H, $A \Rightarrow \{G\}$, $G \Rightarrow \{P\}$

[2]We use scp with version 1.0 of the ssh protocol to eliminate the application-level ACK required every 16KB by version 2.0.

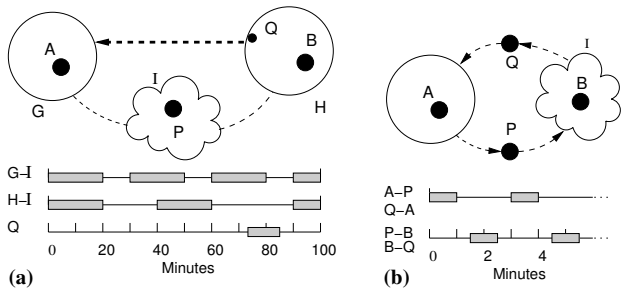

Figure 3. Network layout and link availability for the (a) file transfer and (b) web search demonstrations.

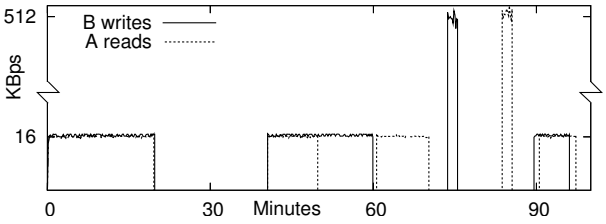

Figure 4. Average data rates during file transfer.

and $H \Rightarrow \{\}$. For example, A resolves B to $\{p, P, H, B\}$.

The average data rates for A (receiving) and B (sending) are shown in Figure 4. For the first 20 minutes both links are up and data flows from B to A, via P, at 16KBps. At 30 minutes, the *G-I* link is up but there is no data to send until 40 minutes when the *H-I* link comes up again. When the *G-I* link goes down again at 50 minutes, B continues and the data is buffered at P. At 60 minutes, when *H-I* goes down and *G-I* comes up, it is forwarded to A for the next 10 minutes. At 74 minutes another intermediary Q in network H is invoked (by mapping $G \Rightarrow \{Q\}$ in H) that reads 71 677KB of data from B at 512KBps and writes it to a file. Ten minutes later, the data is injected into network G and read by A. Finally, both links are up again at 90 minutes and the file transfer is completed at 98 minutes.

The transfer averages a rate of 19.8KBps over the entire 98 minutes. The two links are up simultaneously for only 40 minutes, and at 16KBps would regularly only be able to transfer 38 400KB (assuming that they resume the file transfer mid-way—not possible using scp—and ignoring the connection setup overhead). Using Euonym, they transfer over three times that, 116 340KB.

4.2. Web Search

In this demonstration (Figure 3b) we retrieve a collection of webpages across a series of intermittent links to show the use of round-trip dependent communication in the architecture, support for communication over TCP without a connected end-to-end path, and asymmetrical flows.

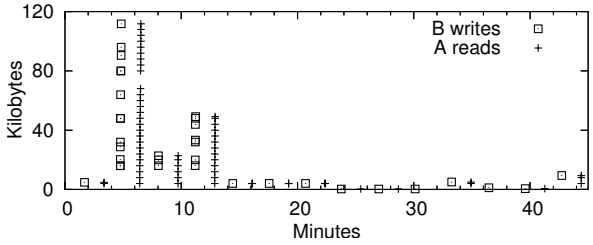

Figure 5. Data flow from B to A in the web search demonstration.

We use the Google API to search for the phrase "south africa" on `en.wikipedia.org` and retrieve the top three results, along with any embedded images, using an unmodified version of the wget tool. Host A issues all requests, including the Google search which uses HTTP-based RPC, through a Euonym-enabled HTTP proxy on B. The requests go from A to B via P and the responses from B to A via Q. Non-internet links are limited to 16KBps. Links are up for 1 minute and down for 2 minutes, staggered such that there is never a fully connected path between A and B in either direction. From A's perspective, B resolves to $\{P,B\}$ and from B's perspective, A resolves to $\{Q,A\}$. All other host names map to their addresses.

The flow of response data from B to A is shown in Figure 5. The first is the response to the Google search which arrives at A at 3 minutes. The next three responses contain the webpage HTML and are reasonably large. The subsequent data is for the embedded images. Every round trip takes approximately 3 minutes and, because wget retrieves items serially, progress is slow. Nevertheless, the pages and their images are successfully retrieved despite no connected end-to-end path and multiple disconnections.

A total of 217KB (15 requests) are retrieved in 47 minutes, an average rate of 79bps. This is significantly lower than the links' limit because of the dependence on lengthy, serial round trips. Even this speed is an improvement over ordinary circumstances in which no transfer at all is possible due to the lack of an end-to-end path.

We also performed the same task after pushing the retrieval logic to host B; A sends only the search string and required result count as its request. This reduces the transfer to a single round trip and makes far better use of available bandwidth. Retrieval begins immediately after the request is received by B and data is buffered at Q on the return flow. The transfer of 330KB takes 223 seconds (a change in the top three search results increased the amount of data transferred), an average throughput of 1.4KBps and an order of magnitude improvement in transfer time. Again, lacking an end-to-end path, no transfer would normally be possible at all.

5. Related Work

The Wizzy Digital Courier Project [10] provides intermittent connectivity to isolated schools in South Africa by buffering web and email requests until off-peak hours when a dialup Internet connection is cost-effective. This requires application-specific proxies and does not provide a general communication framework. The Sámi Network Connectivity Project [6] has a similar goal of linking isolated communities of nomadic reindeer herders in northern Scandinavia. They require connections to each other and the Internet, but lack reliable wired, wireless and satellite links. By using more unconventional transport mechanisms, like commuters and the postal network, communities could be connected using existing infrastructure. In this vein, Randolph *et al*. [9] argue that postal networks are often highly developed and reliable, even in isolated areas, and taking advantage of them can provide inexpensive, high-bandwidth connectivity. Doing so in a generic way without requiring extensive modifications to the applications and end hosts is desirable.

The Delay Tolerant Network [4] architecture promotes communication in intermittently-connected networks using a store-and-forward model with a datagram-style bundle abstraction and fixed routing hosts at network boundaries. The architecture incurs the management overhead of a two-level naming hierarchy and does not allow inclusion of arbitrary support hosts. In contrast, Euonym's persistent names give control to both end and intermediate hosts, and can be semantically overloaded without requiring complex region, network or service hierarchies and namespaces.

Separation of host identity and address has more uses and implications than we explore here. The Host Identity Protocol (HIP) [5] investigates it further and uses cryptographic keys to mitigate the security concerns that separation entails, an approach that could be used in Euonym. Schütz *et al.* [7] supplement HIP with a modified, disconnection-tolerant version of TCP to provide mobility support in the Internet. The Internet Indirection Infrastructure [8] uses name stacks to support multicast and anycast in a DHT-based overlay network which provides rendezvous-style communication, separating sender and receiver. In both projects, however, there remains a reliance on end-to-end paths and a well-connected network. Similarly, while previous work has investigated disconnection tolerance either through a session layer [3] or by modifying TCP [11], the focus is on the Internet and not environments with systemic disconnection.

6. Conclusion

This paper describes Euonym, an architecture that uses persistent names to identify hosts and services and to include intermediate helper hosts in a connection. Euonym promotes application-transparent, Internet-style connectivity in intermittently-connected networks by interposing arbitrary intermediate hosts in the flow of data between two endpoints. Through the use of name stacks, we demonstrate how regular TCP-based applications can be made to work across networks that are disconnection-prone, exhibit intermittent connectivity, are highly heterogeneous, and never provide a connected end-to-end path between two hosts.

References

[1] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A layered naming architecture for the internet. In *ACM SIGCOMM*, pages 343–352. ACM Press, Sept. 2004.

[2] B. Carpenter and S. Brim. Middleboxes: Taxonomy and issues. RFC 3234, Feb. 2002.

[3] R. Ekwall, P. Urbán, and A. Schiper. Robust TCP connections for fault tolerant computing. In *Ninth International Conference on Parallel and Distributed Systems (ICPADS'02)*, Chung-li, Taiwan, Dec. 2002. IEEE.

[4] K. Fall. A delay-tolerant network architecture for challenged internets. In *ACM SIGCOMM*. ACM Press, Aug. 2003.

[5] R. Moskowitz and P. Nikander. Host Identity Protocol. Work in progress, Internet Draft, 2005. `http://www.ietf.org/internet-drafts/draft-ietf-hip-arch-03.txt`.

[6] Sámi Network Connectivity (SNC) Project, Aug. 2005. `http://www.snc.sapmi.net/`.

[7] S. Schütz, L. Eggert, S. Schmid, and M. Brunner. Protocol enhancements for intermittently connected hosts. *SIGCOMM Computer Communication Review*, 35(3):5–18, 2005.

[8] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *ACM SIGCOMM*, Aug. 2002.

[9] R. Y. Wang, S. Sobti, N. Garg, E. Ziskind, J. Lai, and A. Krishnamurthy. Turning the postal system into a generic digital communication mechanism. *SIGCOMM Computer Communication Review*, 34(4):159–166, 2004.

[10] Wizzy Digital Courier, Sept. 2005. `http://www.wizzy.org.za/`.

[11] V. C. Zandy and B. P. Miller. Reliable network connections. In *ACM MobiCom*, pages 95–106. ACM Press, Sept. 2002.