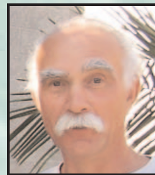
Institute for Software Research

University of California, Irvine

Streaming State Kinematics and Flow Engineering



Michael M. Gorlick
University of California, Irvine
mgorlick@acm.org

March 2006

ISR Technical Report # UCI-ISR-06-3

Institute for Software Research
ICS2 110
University of California, Irvine
Irvine, CA 92697-3455
www.isr.uci.edu

http://www.isr.uci.edu/tech-reports.html

Streaming State Kinematics and Flow Engineering

Michael M. Gorlick
Institute for Software Research
University of California, Irvine
Irvine, California 92697-3425
mgorlick@acm.org

ISR Technical Report **UCI-ISR-06-3**
March 2006

Abstract

We speculate that computational exchanges will evolve from short-lived, intermittent transactions to long-lived flows—repeated transfers of information, constrained in space, time, and domain, from one network point to another. Flows whose lifespans are measured in hours to months will be common and their delivery and manipulation will rank, as elements of infrastructure, equal in importance to other vital goods and services. Flows are an enduring architectural abstraction with roles in software reliability, incremental change, traffic management and shaping, and quality of service, as well as adaptations such as load balancing, device and network mobility, or service distribution. In this context we introduce the concept of *flow engineering*, the design, implementation, and deployment of flows with predictable behaviors. We introduce *streaming state kinematics*, a consideration of state transfer in the service of flow, and demonstrate its role in solving the problem of flow mobility—transplanting the endpoints of a flow from one network location to another while minimizing flow disruption.

Streaming State Kinematics and Flow Engineering

Michael M. Gorlick
Institute for Software Research
University of California, Irvine
Irvine, California 92697-3425
mgorlick@acm.org

ISR Technical Report **UCI-ISR-06-3**
March 2006

***Abstract*— We speculate that computational exchanges will evolve from short-lived, intermittent transactions to long-lived flows—repeated transfers of information, constrained in space, time, and domain, from one network point to another. Flows whose lifespans are measured in hours to months will be common and their delivery and manipulation will rank, as elements of infrastructure, equal in importance to other vital goods and services. Flows are an enduring architectural abstraction with roles in software reliability, incremental change, traffic management and shaping, and quality of service, as well as adaptations such as load balancing, device and network mobility, or service distribution. In this context we introduce the concept of *flow engineering*, the design, implementation, and deployment of flows with predictable behaviors. We introduce *streaming state kinematics*, a consideration of state transfer in the service of flow, and demonstrate its role in solving the problem of flow mobility—transplanting the endpoints of a flow from one network location to another while minimizing flow disruption.**

I. Introduction

Long-lived scientific and commercial computations in bioinformatics, materials sciences, sensor and telemetry processing, logistics, and weather prediction are often cast as medium- to large-grain data flow computations in which components, interconnected as a directed graph, consume and generate data elements that move from one component to another along the edges defined by the graph. Lifespans of hours to weeks are not uncommon for such computational structures and in some domains, such as logistics or global weather prediction, the computations are nonstop, running continously for months to years at a time over an enterprise-scale distributed system.

These applications are partially characterized by *flow*, the repeated transfer of information from one network service/host/attachment point to another. Flow appears in many domains and guises: streaming media (audio and video), distributed via HTTP with the assistance of browser plugins; RSS [18] feeds, employed by web sites to notify content consumers of a change or update in a site's content offerings; bulk transfers, within BitTorrent [5] peering structures; and level 3 flow routing, among IP network routers.

Given their prominence, number, and diversity, flows are worthy subjects of investigation in their own right and *flow engineering*—the design, implementation, and deployment of flows with predictable characteristics and behaviors—may be an important consideration in software systems. Flows are an enduring architectural abstraction with roles in software reliability, incremental change, traffic management and shaping, and quality of service, as well as adaptations such as load balancing, code mobility, or service distribution.

Flow engineering embraces the elements of:

- *Flow naming*, by which flows are identified, catalogued, organized, and searched
- *Flow establishment*, the acquisition, configuration, and initiation of flows
- *Flow modulation*, by which the content, structure, encoding, rate, and other flow parameters may be modified at any point in a flow's lifespan from establishment onward
- *Flows longevity*, with particular emphasis on long-lived flows whose lifespans may be minutes to months
- *Flow mobility*, which allows the endpoints of a flow to be transplanted from one network location to another while minimizing flow disruption
- *Flow transparency*, by which intermediaries may quickly apprehend and synchronize with a flow even if an intermediary is inserted into a flow post-establishment or the flow is repeatedly modulated
- *Flow integrity*, in the face of unreliable transport or flow mobility, to minimize content loss

Our contribution to flow engineering is *STREAK* (STREaming stAte Kinematics), a consideration of state transfer in the service of flow. We suggest that the challenges of flow engineering may be resolved, in part, by the orchestrated movement of state (kinematics) among flow participants. Explicit streaming state transfer from one point to another encourages *stateless* flow, a potent property for the mobility and performance of internet-scale flows. Further, state kinematics have important implications for flow interpretation, flow integrity, and flow transparency, particularly when dealing with real-time flows such as media or sensor telemetry.

We present the case for flow engineering against the backdrop of a classic problem in software systems, *dynamic adaptation consistent with architectural constraints*. As argued by Oreizy [14], self-adaptation is best served by closed-loop, architecture-driven processes that rely upon architectural models for system evolution, the consistent application of change over time, and system adaptation, the cycle of detecting changing circumstances and planning and deploy-

ing responsive modifications. Flow-based systems enjoy an intuitive and appealing architectural model (directed graphs) for which graph editing is the formal representation of one adaptative mechanism, the rearrangement of flows (directed edges) among system components (nodes). Flow engineering is the internet-scale reification of a rich set of responsive modifications that may be employed by self-adaptive systems.

In this context we consider flow mobility as required by an exogenous agent, namely the architecture-driven adaptive infrastructure sketched in [14], where components (nodes) have little or no say regarding the flows (edges) that connect one component to another in the large-scale flow graph. The decision of which flow to direct where and when is made by the adaptive infrastructure subject to architectural, domain, and component constraints. The mechanisms for flow mobility are wielded by an exogenous agent, standing "outside" of the flow graph, that may not have time to pause a component before detaching and reattaching a flow. We are particularly interested in long-running, long-lived, enterprise-critical applications for which termination, or even a momentary pause, is unacceptable yet for which adaptation, improvement, and modification in place is essential.

However, flow engineering speaks to a broader agenda. As will become evident, flow engineering, in the service of flow-aware applications, reaches far down into the lower layers of internet-scale infrastructures, including the transport layer. There is precedent for this phenomenon; for example, the demands of Representational State Transfer (REST), as exemplified by HTTP, put enormous strains on the transport layer, TCP, of the internet. Consequent efforts to improve TCP performance led to subtle but far-reaching amendments in TCP congestion control. Thus, an architectural requirement—reliable, stateless transactions—is reflected, obliquely by way of the extraordinary growth in HTTP traffic, in the detailed redesign of a fundamental transport protocol.

Flow engineering hints at similar outcomes. We suggest that novel internet-scale architectures may, simply as a side-effect of considered and reasonable architectural choices, harbor profound and perhaps unexpected consequences for other (deeper) network structures and systems. Happily, the software engineering principles on which these internet-scale applications rely may be reapplied to good effect at lower levels elsewhere within the internet infrastructure. Flows, and their consequences, appear in many places, forms, and scales and are thereby a fit topic for foundational discussion and analysis.

The remainder of this paper is organized as follows. Section II presents flow as an architectural abstraction of software systems and explores the role of state kinematics in flow-dominated systems. Section III considers a narrow aspect of flow naming with the goal of sketching necessary and sufficient infrastructure for identifying individual flows as first-class network objects. Section IV turns to the general problem of flow mobility, applying STREAK to the design and implementation of flow endpoint transplantation. Finally, Section V summarizes our progress to date and sketches a program of investigation and experiment for validating STREAK.

II. Flow and Streaming State Kinematics

Flows appear repeatedly in networks and applications. Flow-aware networks [17], [15] seek to detect even short-lived flows (on the order of 20 packets or less), immediately establish flow state with only a single packet in flight, and thereafter route per flow rather than per packet. The claimed advantages include fair admission, traffic shaping, and guaranteed quality of service.

Flow contents, timescales, and rates vary widely. The aperiodic flow of an RSS feed may have an average frequency best measured on a time scale of days, but with a high burst bandwidth, while sensor-driven flows may have a periodic frequency of tens of Hertz (for example, 30 frames-per-second video) and a sustained bandwidth of hundreds of kilobits to megabits per second. The transmission and reception of a flow F may be decoupled—the transmitter of F is unaware of the destination(s) of F and a receiver of F is ignorant of its source(s). From this perspective, content-based networks such as Siena [6], intended for event distribution and notification, are flow middleware in which the content elements of the flow are events and both event producers and consumers are decoupled.

Flows possess distinct units of content, persist over time, exhibit aperiodic and periodic behavior, may appear and disappear, are not necessarily known in advance, and demonstrate 1-1, 1-n, m-1, and m-n relations among producers and consumers. Flow framing is a critical element of flow generation and interpretation and reflects many considerations, including rate control, the constraints and characteristics of network transport, and content semantics. For example, the framing structure of video codecs such as MPEG-4 or Ogg Theora [24], abstractly cast as a sequence

$$I_0, \delta_{0,1} \ldots, \delta_{0,m}, I_1, \delta_{1,1} \ldots, \delta_{1,m}, I_2, \ldots$$

where complete frames I alternate with a series of deltas δ to be applied in order against the immediately preceeding full frame, is a flow framing pattern that appears in many domains.

Flows are not necessarily frequent or periodic; for example, event distribution is marked by our inability to know in advance when an event may occur. On the other hand, periodic flow admits of a rich temporal semantics. The attendant calculation of flow integrity and reliability is a complex combination of flow periodicity, transport, and framing structure. Lower transport reliability may be tolerated in a high-rate flow such as streaming video where occassional missing content elements are ignored, interpolated, or reconstructed. Far more stringent requirements may be levied against a flow transporting infrequent but vital events, for example process control alarms or those events affecting human life or safety.

A. STREAK—Streaming State Kinematics

From a network perspective a flow is "a flight of datagrams, localized in time and space and having the same unique identifier" [15]. STREAK applies *state kinematics*—the representation, transfer, calculation, and inference of state—to the creation, maintenance, and manipulation of flows. STREAK is inspired by REST (Representational State Transfer), the

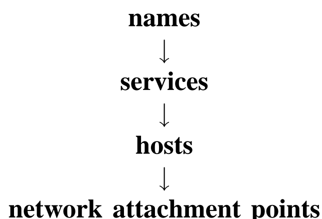
architectural style underpinning the modern web and, in the analyis of Fielding and Taylor [7], a pivotal contributor to the remarkable and explosive growth in web servers, clients, and services.

REST argues that independent, anarchic growth is best served when the responder is stateless and in which the state required to understand or prosecute the request/response transaction is fully contained (and repeated) in each and every request/response pair. Stateless endpoints are simpler to implement, exhibit excellent scaling, and may be more robust than their stateful counterparts. The apparent "wastefulness" of repeating the state vector in each and every transaction is outweighed by the substantial advantages.

Much of REST is exemplified by HTTP [8], the application-level protocol of the web. In particular, the headers and structure of HTTP are explicitly designed to minimize state maintenance at the endpoints and permit intermediaries such as a web proxy or cache to interpret and act upon the transaction. STREAK applies the lessons of REST to flows, offering mechanisms by which the transparency of both endpoint state and flow content may be economically maintained within the flow itself.

III. Flow Naming

Internet naming structures are exemplified by the Domain Name System (DNS), a hierarchical naming infrastructure that maps human-readable names such as `www.example.com` to IP addresses. In a prescient paper [19], Saltzer argued that this two-level mapping, from strings to network attachment points, was profoundly flawed and that a proper mapping contained at least four distinct levels:



where **names** represents one or more human-readable namespaces such as domain names (`sales.example.com`) or e-mail addresses (`jdoe@example.com`), **services** denotes network services such as time-of-day, RSS feeds, or web servers, **hosts** denotes individual computational engines that implement services, and **network attachment points** are low-level network addresses (such as IPv4 or IPv6 addresses). At each level the mappings are dynamic; for example, the same service s may map to different hosts at different times as a service provider upgrades a machine room or a mobile device (host) may map to distinct network attachment points over time as the physical location of the device changes and distinct wireless access points come in and out of range.

Balakrishnan and company [4] elaborate considerably on Saltzer's point and, proposing a small set of architectural principles for such naming structures, draw broadly on recent work to sketch concrete implementations for the bottom layers, **services** to **hosts** to **network attachment points**.

Setting aside the details of their proposal, the paper argues three vital points:

- The innovation of distributed hash tables (DHTs) permits the use of flat namespaces for services and hosts. Flat namespaces are semantic-free [25] and may be used to construct persistant namespaces for arbitrary higher-level objects
- Additional naming layers are required to shield applications from the underlying routing structure of the internet. Our inability to disentangle network naming from the domain- and topology-centric DNS discourages innovation and experimentation
- The interposition of services between network endpoints is a natural trend to be encouraged, *once* it is reset upon sound architectural footings. Network structures must permit intermediation but in a manner that preserves flexibility and transparency

To accomodate delegation and intermediation the mappings of [19] are generalized so that resolvers r at any level A mapping down to the next level B below may return finite sets of identifiers $i_1, i_2, \ldots, i_m$ where each $i_j \in A \cup B$. As supporting work illustrates, such as the semantic-free references of [25] and the Internet Indirection Infrastructure [23], [26], [12], the potential gains are enormous, including seamless support for physical and logical mobility from the service layer on down, principled intermediation, and service composition.

In theory, the structure outlined by Saltzer is sufficient to support host mobility but requires, in the extreme, that all network communication be conducted at the level of service identifiers and that the transport layer must resolve, for every single packet, the

$$\text{service} \rightarrow \text{host} \rightarrow \text{network attachment point}$$

mappings, as either endpoint may have moved in the time that it takes to transmit a single packet. While the expense of the mappings may be reduced by clever caching, it is patently impractical to recalculate these bindings per packet. Further, there are many common cases in which recalculation is either unnecessary or at worst infrequently demanded, including the obvious cases where both hosts are stationary or only one host is mobile but the transitions from one location (network attachment point) to another are sporadic.

Note, however, that the multilayered naming structure permits mobility (where by mobility we mean time-varying alteration in the binding from a denotation at one level to the next level below) *at all levels*. Just as DNS now permits rebinding names, `www.example.com`, from one IP address a to another a' (albeit with delays of at least minutes to hours for the rebinding to propagate network-wide), the Saltzer hierarchy offers like flexibility at each level. Thus "mobility" is possible at all levels:

- Service renaming, where a known service `www.example.com` is remapped from service identifier s to service identifier s'
- Rehosting, where a service identifer s is moved from one host h to another host h'
- Network mobility, where a host h is transferred from one

network attachment point a (given as an IP address) to another network attachment point a'

Network mobility may also be a consequence of physical mobility as a host, such as a personal wireless device, is moved from one geospatial location to another.

In this spirit we adopt the abstract naming infrastructure outlined in [4] as the starting point for flow naming. Informally, it helps to regard a flow as something akin to a network "garden hose" where the flow has an identity independent of the content moving through it and two distinguishable endpoints, the *source* and *sink* of the flow (the two ends of the "garden hose"). The endpoints may be attached and detached as need and circumstances dictate; it is this property we term *flow mobility*. As a matter of notation, for a given flow F we denote the flow source and sink by $\alpha(F)$ and $\beta(F)$ respectively. For each flow F there is a unique opaque service identifer s_F and s_F in turn resolves to an ordered pair $\langle s_{\alpha(F)}, s_{\beta(F)} \rangle$ where $s_{\alpha(F)}$ and $s_{\beta(F)}$ are the service identifiers of the flow endpoints (source and sink respectively), as illustrated in Figure 1.

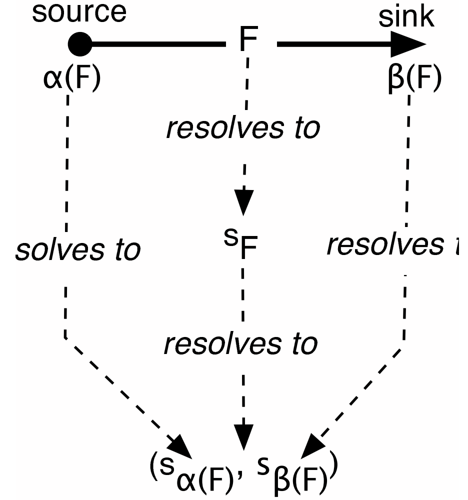


Fig. 1. Resolving flow F and its endpoints $\langle s_{\alpha(F)}, s_{\beta(F)} \rangle$.

This perspective treats a flow as something akin to a Reo channel [1] in which the flow is an object independent of any content moving "through" it or the attachments of the flow source and sink endpoints. Thus a flow, F, as given here, has multiple distinct binding states where 0, 1, or both endpoints have defined service $\rightarrow$ host bindings. Recalling that the flow naming structure sketched above permits three forms of mobility—service renaming, rehosting, and network mobility—let H be the partial mapping from service identifiers to hosts and N the partial mapping from hosts to network attachment points; thus, $N(H(s)) = a$ is the mapping (perhaps undefined) of service identifier s to network address a. These mappings are redefined over time as bindings come and go and to reflect their temporal variation we define $A(t, s) = N(t, H(t, s))$ where $H(t, s) = h$ is the binding of service identifier s to host identifier h at time t and $N(t, h) = a$ is the binding of h to network attachment point a at time t.

We define *flow mobility* as any rebinding over time of flow source and sink endpoints. More formally, for a given flow F with service identifier s_F, let $[t_0, t_1), t_0 < t_1$ be some finite time interval. Then F is mobile over that time interval iff $\exists t \in (t_0, t_1)$ such that $A(t, s_\alpha(F)) \neq A(t_0, s_\alpha(F))$ (flow source mobility) or $A(t, s_\beta(F)) \neq A(t_0, s_\beta(F))$ (flow sink mobility).

We now focus on the problem of maintaining flow integrity in the face of mobile, real-time flows where the timely transfer of information has semantic consequences (for example, satellite telemetry or sensor readings for process control). As shown in Section IV, the positioning and transfer of state (STREAK) is *the* critical consideration for a viable solution. Further, maintaining external, oracular, state in a Saltzer-like naming infrastructure is necessary, but not sufficient, for flow mobility; appealing to such a naming infrastructure for relief is a measure of last resort.

IV. Flow Mobility

We examine flow mobility in the context of unidirectional flows whose transfers are from producer to consumer, as

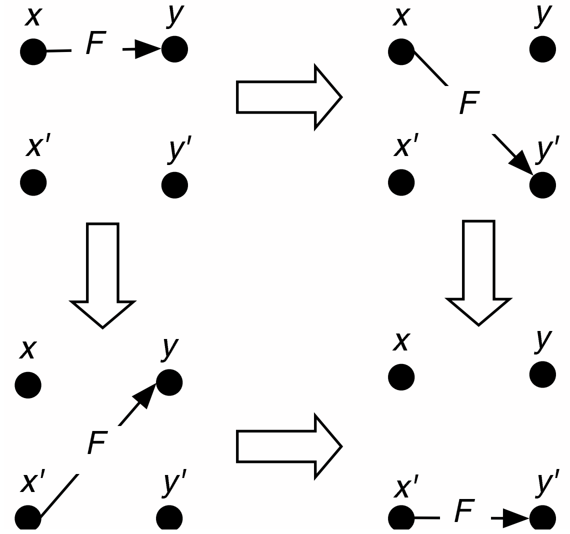


Fig. 2. Mobility of flow F among host sources x, x' and host sinks y, y'.

illustrated in Figure 2, which sketches the movement of flow endpoints, among sources, x and x', and sinks y and y'. Our perspective is the obverse of physical mobility in which a host, for example a laptop, moves from one network attachment point to another. Here, instead, the sources and sinks remain fixed while the flow endpoints are shifted from one network attachment point to another. As we shall show, the mechanisms described below are also easily capable of accomodating physical mobility in which one host (the source) remains fixed while the other (the sink) roams, while the cases of a roaming source or two endpoints roaming simultaneously requires an appeal to the multilayered flow naming infrastructure of Section III.

Three distinct base mechanisms may be applied to achieve flow mobility: connectionless transport, network indirection, or stateless connections. Connectionless transport is epitomized by UDP [16], an unreliable, datagram protocol. While UDP allows endpoints to shift and move, higher-order mechanisms

are required for informing a source of a change in the network location of a flow sink or for enforcing flow rates and congestion control—two vital concerns for real-time flows.

Network indirection comes in two forms. In the first, as a flow source (sink) shifts from one network location to another buffered forwarding agents are left behind to forward incoming flow traffic on to the predecessor (successor) location. Delays may quickly accumulate as traffic winds a serpentine path through a long series of past endpoint locations and flow reliability suffers as the number of transit points increases. Flow mobility may also fail to scale as ever-increasing quantities of host and network resources are consumed in forwarding long-lived flows whose endpoints have shifted repeatedly over time.

An alternative form of network indirection is exemplified by the Internet Indirection Infrastructure [23], which employs peering structures to efficiently and dynamically bind network-wide unique names to network attachment points (IP addresses). While conceptually attractive, it also requires hosts to act as transit points for network traffic routed through the indirection infrastructure.

The third mechanism, one which we adopt here, requires *stateless connections*. We first describe a stateless, connected, transport well-suited for flows and combine this mechanism with Saltzer-like naming infrastructure to achieve general, efficient, flow mobility that accomodates mobility in all of the senses implied by the Saltzer hierarchy. Our story begins with Aura and Nikander [2], [3] who observed that it was possible to transform a stateful network connection, one in which both endpoints retained state that determined the progress and rate of an exchange, to a *stateless* connection wherein only one endpoint maintained the state required by the connection. The transformation requires that the state information be attached to the messages exchanged between the endpoints where message authentication codes are applied to ensure the integrity of the state information and the connection. The authors suggested that servers employing stateless connections would exhibit far more graceful behavior under heavy connection loads since the amount of state required server-side was reduced to a small constant and invariant with respect to the number of ongoing connections.

This insight was revisited by Shieh, Myers, and Sirer [20], who reconstructed the complete TCP network stack using the techniques of [2], [3]—demonstrating throughput comparable to a stateful TCP stack and exactly the scaling performance predicted by Aura and Nikander. Both of these works are superb examples of streaming state kinematics in which state movement and state inference are orchestrated to create the state transfers emblematic of REST, but in the context of ongoing information exchange akin to flow. We apply these same techniques to another protocol, inspired by Cyclic UDP (CUDP) [21], [22], to create a stateless, connected transport suited for both high-performance flows, such as video or satellite telemetry, and for the seamless transplant of flow endpoints from one network attachment point to another.

A. CUDP—Cyclic User Datagram Protocol

CUDP is a best-effort transport protocol, layered atop UDP, with prioritized packet delivery, such that the probability of sucessful packet delivery is proportional to the packet's priority. Intended for multimedia transmissions such as video, CUDP transmission is clocked in *cycles*. The span of a cycle is a function of the flow periodicity, for example, real-time video at a rate of 30 frames-per-second has a cycle span of 33 milliseconds, the interval between successive frames in a 30 frames-per-second stream.

Abstractly, CUDP transmission is a sequence of frames $f_0, f_1, \ldots$. Each frame f_i is a finite sequence of messages $m_{i,0}, m_{i,1}, \ldots, m_{i,n_i}$ where the number n_i of messages in each frame is frame-dependent; that is, n_i may vary from frame to frame. Each message sequence $m_{i,0}, m_{i,1}, \ldots, m_{i,n_i}$ is organized in priority-order from highest priority ($m_{i,0}$) on down. A higher-level media layer determines the message ordering (equivalently priority). The rate of frame production dictates the cycle rate of CUDP and each cycle is devoted exclusively to the transmission of a single frame $f_i = m_{i,0}, m_{i,1}, \ldots, m_{i,n_i}$.

CUDP is structured to ensure that, within a given frame f_i, higher priority messages $m_{i,r}$ have a greater probability of arriving at the receiver than lower priority messages $m_{i,s}$, $r < s$. For the sake of simplicity, assume that each $m_{i,j}$ in frame f_i corresponds to a single UDP packet. CUDP transmits each $m_{i,j}$ in priority order from $m_{i,0}$ on down and marks each such $m_{i,j}$ as *sent* as it goes. Within any one cycle the sender makes successive passes over the message queue $m_{i,0}, m_{i,1}, \ldots, m_{i,n_i}$, starting at the head of the queue $m_{i,0}$ and transmitting any message $m_{i,j}$ not marked as *sent*.

The receiver regularly sends negative acknowledgements to the sender enumerating the messages $m_{i,j}$ missing in its reception of the ongoing cycle i. When a negative acknowledgement is received for a message $m_{i,j}$ in the transmission queue the sender erases the *sent* marker from $m_{i,j}$ thereby ensuring that $m_{i,j}$ will be retransmitted (in priority order) on the next pass through the transmission queue.

The details of CUDP are far more complicated than the bare outline above suggests. It is often the case that the path MTU (Maximum Transmission Unit) is substantially smaller than the size in bytes of some or all of the messages $m_{i,j}$ of frame f_i. Consequently CUDP (like so many other protocols) must decompose the individual messages $m_{i,j}$ into packets $p_{(i,j),k}$ such that no packet exceeds the MTU and inform the receiver of the details of reassembling the individual packets into the complete messages $m_{i,j}$ of each frame f_i. CUDP is also rate-controlled; to avoid network congestion and ensure timely delivery of packets the receiver must supply bandwidth estimates at regular intervals back to the sender.

CUDP regulates its bandwidth consumption by breaking each cycle (the cycle rate is equivalently the frame rate) into a series of *bursts* where each transmission burst sends no more data than the estimated carrying capacity of the path from sender to receiver. Each burst begins anew at the head of the message queue and it is this behavior, buried deep within the protocol, that ensures the fundamental property of priority

transmission.

While CUDP is comparatively straighforward in concept, there are many practical complications. The total size of frames may vary considerably from one frame to another and, for a given channel path and set of congestion conditions, there is no guarantee that there is enough time in a cycle to transmit an entire frame, even if all packets are delivered intact to the receiver. In other words, a new frame (cycle) f_i may begin before we have completed transmitting the prior frame (cycle) f_{i-1}, in which case the transmission of f_{i-1} must be abandoned in favor of the transmission of f_i. However, the priority delivery of CUDP addresses just that situation by deliberately resending missing packets in priority order. Even if all packets fail to arrive there is a high probability that the most important messages $m_{i-1,j}$ of frame f_{i-1} did arrive—raising the likelihood that the receiver can make some progress.

B. Stateless Flow

With CUDP serving as a technical backdrop we turn to the problem of *stateless flow*, that is, a flow in the spirit of the stateless connections described earlier [2], [3], [20] where all connection state is in the hands of the receiver. The semantics of unreliable media delivery are far richer than those of reliable transport. The variations include message importance within a frame, the reliability (per message) of delivery, delivery rate, and congestion control. DCCP, the Datagram Congestion Control Protocol [11], implements bidirectional, unicast connections of congestion-controlled, unreliable datagrams including mechanisms for negotiating a suitable congestion control algorithm. Stateless flows place the vast bulk of control in the hands of the receiver, thereby simultaneously achieving the goals of flow mobility (movement of flow endpoints) and fine-grained receiver control of reliability, rate, and congestion avoidance. We first sketch the protocol mechanics of stateless flow and then briefly illustrate the application of stateless flow to flow mobility.

Flow generators create frames f where each frame comprises one or more messages $m_{f,0}, m_{f,1}, \ldots, m_{f,r-1}$. We assume that the frame messages $m_{f,i}$ are ordered by decreasing priority where $m_{f,0}$ is the highest-priority (most important) message of frame f and $m_{f,r-1}$ is the lowest-priority (least important). Finally, each message $m_{f,i}$ is broken into one or more payloads $p_{m_{f,i},0}, p_{m_{f,i},1}, \ldots, p_{m_{f,i},s-1}$. All payloads p (of all messages of all frames) are a known fixed size N (on the order of a few kilobytes) with the possible exception of the last payload of a message which may be $< N$.

Each generator mantains a finite queue q of *current* frames, say the five most recent frames of a real-time video stream in temporal order The contents (and length) of the queue will change over time as dictated by the semantics of the generator; for example, the queue of a generator managing a 15 frames-per-second video stream will change less frequently than that of a generator managing a 30 frames-per-second video stream.

Stateless flows are transported via UDP. We assume that a flow generator g maintains a known UDP port for its flow. Flow consumers may request a *frame map* from generator g that enumerates the details of the enqueued frames. A frame map contains:

- The version number of the flow queue, an unsigned integer that is incremented whenever a frame f is either added to, or removed from, q
- The *short-term delta*, the magnitude of the change in the version number over the previous second
- The *long-term delta*, an ordered pair $\langle n, \delta \rangle$ where δ is the magnitude of the change in the version number over the past 2^n seconds
- The length of the flow queue q, that is, the total number of frames in q
- For each frame f in q
  - The sequence number s_f assigned to f
  - The total number of messages in f
  - The amount of time (in milliseconds) f has resided in q
  - For each message $m_i \in f$, the number of payloads required for transmiting m_i

With a frame map in hand a flow consumer c selects just those messages from the set of available frames in which it has an interest. Each payload GET contains a set of payload ranges enumerated per frame per message. Let frame f be contained in a recent frame map of generator g. A single payload range for f contains:

- The sequence number s_f of frame f
- A message identifier $m_{f,i}$ for a message $m_{f,i} \in f$
- A payload range $[\alpha, \alpha + \beta]$ where $\beta \geq 0$ and payloads $p_\alpha, p_{\alpha+1}, \ldots, p_{\alpha+\beta}$ are contiguous payloads of message $m_{f,i}$

The payload GET requested by consumer c of generator g contains one or more payload ranges for one or more frames f.

Consumer c administers rate and congestion control by modulating the frequency and breadth of its requests, tracking responses to payload GET requests (including packet loss), and measuring the round-trip response time to simple "heartbeat" pings against generator g. The consumer may implement whatever congestion control it deems appropriate such as the AIMD (Additive Increase Multiplicative Decrease) of TCP or TFRC (TCP Friendly Rate Control), an equation-based form of congestion control that responds to congestion more smoothly than AIMD but cooperates fairly with TCP over the long term. Consumer c can estimate the impact of any payload GET request since, knowing the payload size, it can estimate the total number of bytes required to satisfy the request. The generator g treats each GET as a burst request, transmitting every payload enumerated in the GET. The consumer may implement the equivalent of a sliding window algorithm (used by TCP) by carefully staging GET requests and permitting multiple data packets to be in simultaneous transit, making more efficient use of network bandwidth.

One substantial complication is the ever-changing contents of the frame queue q maintained by generator g—requiring a consumer c to periodically query g for updates. For example, a 30 frames-per-second video stream will update a frame queue q every 33 milliseconds. Consumers use the short- and

long-term deltas of the version number, reported in a frame map, to estimate the volatility of a generator's frame queue and thereby adjust the frequency of frame map requests. To improve estimation of the sampling rate by the consumer the frame queue version number is piggybacked atop the frame payloads transmitted by g to c. Tracking the rate of change in the queue version number allows the consumer to refine its estimate of queue volatility and determine the sampling rate (equivalently the request rate) for frame maps.

Since the frame queue contents change over time, a consumer's GET request may contain references to a frame f no longer in the queue. In this case the generator simply ignores those particular payload requests as a protection against denial of service attacks. However, for a legitimate consumer, the lack of response is ambiguous, as it may either be interpreted as packet loss or as an unanticipated change in the composition of the frame queue. Even having the most recent queue version number in hand is inadequate, since the monotonic increase in version numbers only quantifies the total number of additions and deletions and not the details of the changes. The proper strategy varies with the significance of the frame, the consumer's tolerance for missing information, and the rate of update at the generator. To address the problem we permit a GET request to carry a flag requesting that a fresh frame map be transmitted at the end of burst. This is functionally equivalent to the consumer issuing a GET request followed immediately by a frame map request but eliminates the overhead of explicitly constructing and transmitting the frame map request.

The consumer can also compensate for frame payloads lost in transmission by rerequesting the missing payloads in successive GET requests. Since frame and payload request order is entirely at the discretion of the consumer the consumer can selectively request frame payloads in order of decreasing importance—concentrating on the most significant payloads of the frame and thereby emulating the priority delivery behavior of CUDP. Each payload is tagged with a frame/message/payload identifier that allows the consumer to reconstruct the messages $m_{f,i}$ of a frame f and pinpoint any holes. In this manner the consumer determines the reliability (loss rate) of the flow and adapts its behavior. For example, in a bandwidth-constrained flow a consumer may choose to subsample the frames via selective GETs in a manner that is flow- or consumer-dependent.

One danger of stateless flows is ill-behaved or buggy consumers that issue GET requests without due regard for the bandwidth state of the network. To guard against this threat a modest amount (tens of bytes) of generator state s_g, encrypted with a secret key known only to the generator, accompanies each frame map and payload response. That encrypted state s_g is used by the generator to maintain generator-side estimates of the bandwidth state and set fair (by the standards of the generator) consumption limits, ignoring those portions of GET requests that exceeed generator-set consumption limits. The consumer is obligated to return its latest copy of s_g with each successive GET. Since s_g is encrypted with a secret key known only to the generator the consumer can not manipulate s_g to its advantage. Trickles [20], a stateless implementation of TCP, employs the same technique to prevent the stateful TCP endpoint from corrupting the state (network continuation) of the stateless endpoint.

C. Implementing Flow Mobility

We now consider the problem of moving the endpoints of a unidirectional flow in which datagrams are flowing from a generator g to a consumer c. We assume a framework of exogenous control of flow establishment and interconnection; that is, the creation of a flow from g to c is undertaken by an authoritative agent whose span of control includes g and c. A similar assumption is made by Reo [1], a channel-based coordination model for component composition. A partial implementation of Reo channels (the rough equivalent of our flows) is described in [10]. There, channel mobility relies on a breadcrumb trail of buffers and TCP connections as endpoints are shifted from one network location to another. The intermediary TCP connections and buffers, increasing in number with each endpoint shift, introduce delay and increase the likelihood of channel failure. While this mechanism guarantees reliable transfer, its suitability for high-performance real-time flows such as video or streaming audio is questionable.

Flow mobility addresses both logical and physical mobility. Adaptive systems, responding to either performance stress or outright failure, may wish to transfer function from one network location to another, even within a confine as small as a machine room. The ability to redirect a flow to a "warm" backup can be a valuable mechanism for reliability and fault-tolerance. Adaptation, or plasticity, eases the evolution of systems. For example, redirecting flows may allow system implementors to transparently modify flow-based computations by seamlessly inserting new components mid-flow. In some cases the transfer of flow is an event that is anticipated and scheduled, for example, the installation or upgrade of components within a flow graph. In other cases, the transfer is anticipated but unscheduled, say the transfer of load to maintain adequate responsivity during peak periods. Finally, the transfer may be sudden, as in the unanticipated transfer of flow to recover from a catastrophic hardware failure.

Physical mobility, such as the movement of a wireless digital device among a set of wireless hot spots, requires that a flow follow the device as its network attachment point changes. Particularly challenging is sustaining flow when both the source and sink of the flow change their respective network attachment points simultaneously, as may arise in an exchange among two or more mobile devices. Given these scenarios, we consider flow mobility from three perspectives, the movement of a flow sink, the movement of a flow source, and the simultaneous movement of both source and sink.

D. Movement of Flow Sinks

Stateless flows eliminate the necessity for intermediate buffers and forwarding agents (as in Reo) since all connection (flow) state is maintained at the consumer c. Let s_c denote that consumer-side state. Generator g is indifferent to the number of independent consumers since it does nothing more than maintain an ongoing queue q of frames and respond

to consumer requests in the form of frame maps and GETs. What little state is required by g to guard against "bandwidth bandits" is returned to it with each individual GET request from each consumer. Moving the sink of a flow from one consumer c to another consumer c' requires only that the flow state maintained by c be expeditiously transferred in its entirety to c'. Once accomplished, c', with respect to the flow, may resume exactly where c left off.

However, this is not to say that the transition is entirely smooth. At the time of the shift in flow from c to c' frame payloads transmitted by g to c may still be in flight through the network. Those payloads are discarded or ignored by c. Nonetheless, c', armed with the flow state s_c, can request that precisely those payloads indicated by s_c as missing be retransmitted—by issuing an appropriate GET request to g just as c would have done had those payloads not been delivered due to network congestion. Maintaining all of the flow state at the sink permits the successor sink c' to recover gracefully. However, time permitting, we can delay the shift in flow, allowing the packets already in flight from g to c to arrive safely at c before moving s_c to c'. This may reduce the resumption time of c' and reduce network load since c' need not request the retransmission of packets that have already drained to c.

Since the receiver c holds all of the state relevant to the flow an authority over c can determine, with comparative ease, the degree to which outstanding packets may still be in flight and the expected wait time for those packets. With sufficient domain knowledge, that authority may also judge the consequences of executing the transfer of a flow sink immediately versus waiting for the network connection to drain. Here stateless flow allows the exogeneous authority to choose from a broad range of behaviors, varying the speed, precision, or transparency of the transfer as needs and circumstances demand. Finally, an exogeneous authority may halt outright or quiesce the flow generator, restart the flow consumer at its new network attachment point, and restablish from scratch the flow from generator to consumer. However, it is easy to imagine cases in which the exogenous agent does not hold sufficient authority to control the generator, say a purchased, commercial feed. The generator, because it serves many distinct customers simultaneously, may not be halted, even momentarily, to suit the demands of a single consumer. Stateless flow decouples the consumer from the generator and permits flexible peering relationships where there is no single overarching span of control.

The recovery is also robust even if the state transfer from c to c' is delayed for a long period of time relative to the frame production rate of g. Imagine that we detach the sink end of the flow and just let it "dangle." Since stateless flows are receiver-driven, no payloads are transmitted unless explicitly requested (via GETs) and the flow quickly quiesces. At the very worst the flow state s_c will become stale since the frame map maintained by g will diverge over time from the frame map snapshot contained within s_c. When the flow sink is finally reattached c' can adapt in one of two ways. First, using the timestamps and measures of frame volatility maintained in state s_c (now in the possession of c') c' can make an educated estimate of the discrepancy between the frame state as known by c' and the true frame state residing in g. If the drift is thought to be modest then c' can optimistically restart the flow at, or near, the point of disruption. At the very worst, c' will quickly discover (within a few round-trip times) that the state gap is too large and will resynchronize, via a frame map request, with g. In short, in the optimistic case, c' will resynchronize without payload loss and at worst fully recover but at the price of a gap in the frame flow from g. Alternatively, c' assuming the worst, immediately requests a fresh frame map from g and, upon computing the differences between the old state s_c and the fresh state from g, simply resynchronizes as best it can. In the pessimistic case c' may loose the opportunity to fully recover since in the time taken by c' to reconstruct fresh state g may, under the pressure of a fixed production rate, discard just those frames that c' requires for a seamless restart.

E. Movement of Flow Sources

While flow sink mobility is straightforward flow source mobility is far more constrained. In the case in which generator g is a member of a pool of like generators $g_0, g_1, \ldots$ where the g_i share state, including the secret key employed by g to encrypt the generator-side state for ensuring fair bandwidth consumption, then the source endpoint of the flow is moved to a new generator g' by informing the consumer c to redirect all future requests to g' rather than g. Even if the state of g and g' differ to some extent, for example the frame map of g' lags that of g slightly, then c can still resynchronize with g' following the redirection. Just this modest capability alone is useful for load balancing, reliability, and fault tolerance. In the very worst case the flow can be terminated altogether and a fresh flow created to serve c from some more distant, but semantically related, g'. For example, if c is consuming a real-time video stream generated by g then, modulo application constraints, it may be appropriate to transfer the flow source to another camera g' nearby that views the same scene as g but from a different location and perspective.

For those cases in which the movement of a flow source can be anticipated and the successor source g' is known in advance the flow source can piggyback a REDIRECT on the next consumer-issued GET or frame map request. A REDIRECT informs the consumer c that all future requests for this flow should be redirected to the specified network location. However, given that the flow is implemented atop the unreliable delivery of UDP, generator g may remain "in place," waiting over a period of several round-trip times for all traffic from c to subside. If further traffic from c continues to arrive g may be forced to repeatedly reissue the REDIRECT and wait, eventually aborting the exchanges after a finite timeout.

An alternative mechanism is to employ an indirection infrastructure such as *i3* [23], a rendezvous-like abstraction in which forwarding agents (in this case a Chord peering infrastructure) associate opaque indentifiers with network locations and forward any packet sent to an identifier onto the given network location. *i3* can be used, absent any of the stateless mechanisms described here, to achieve both source and sink

mobility but at the price of requiring each packet to pass through an intermediary forwarding agent. Nonetheless, combining stateless flows with *i3*-style indirection is suggestive as it offers an important optimization—eliminating the overhead of indirection altogther in the common case in which the source is static and the sink is mobile.

For real-time flows the price exacted by *i3* may be too high to bear. Fortunately, Saltzer-like naming infrastructure can be combined with stateless flows to achieve arbitrary source endpoint mobility. When source g is moved from one network address a to another a' the mapping $H(g)$ changes accordingly as the naming infrastructure is updated to reflect the rebinding. When the move occurs consumer c will notice (within a few round-trip times) the lack of response from g. At that point c, knowing the host identifer of g, consults the naming infrastructure, obtains the successor address a', and redirects its stateless requests to network address a'. Again, since all of the flow state resides with c it can pick the flow at the point in time in which g shifted its network attachment point from a to a'. We suspect that, for those flows with infrequent changes in source, the occassional overhead of reverting to the naming infrastructure for the most recent network attachment binding of g will be considerably less, over the lifespan of the flow, than the costs of repeated indirection required by an *i3*-like infrastructure. An interesting open question is the relative costs of these two approaches as a function of the frequency of change in a flow source. No doubt there is some crossing point at which the cost of *i3* indirection is less than the cost of repeatedly consulting a Saltzer-like naming infrastructure.

F. Simultaneous Movement of a Flow Source and Sink

Maintaining flow integrity during the simultaneous movement of a flow's source and sink is a daunting challenge. To illustrate, a full-color, 30 frames-per-second, 320×240 pixels, MPEG-4 encoded, video stream v, transmitted from a mobile wireless device equipped with a video camera to another mobile wireless device, requires a sustained bandwidth on the order of 200 kbps. A longhaul internet path, say from Los Angeles to Boston (4193 kilometers), has a one-way, flight-of-light time of at least 20 milliseconds and a minimum round-trip time of 40 milliseconds.[1] Video stream v has a cycle rate of 30 Hz and an interframe periodicity p of 33 milliseconds, a span of time well below the theoretical best round-trip time of our Los Angeles/Boston path.

At outlined in the previous section the overhead of detecting the movement of the flow source is at least $nr+q$, $n > 1$ where r is the round-trip time between source and sink and q is the time required to query the "Saltzer" naming infrastructure and receive an authoritative reply. That delay may easily amount to many multiples of the video frame periodicity (33 milliseconds/frame), and consequently the sink may be forced to drop frames (by adjusting its GET requests) as compensation. While stateless flows give the receiver the tools to accomodate flow interruptions in a manner sensitive to the semantics of the flow, minimizing interruption due to source mobility is still worthwhile.

One reasonable prospect, accounted for in the protocol supporting stateless flows, is a REDIRECT from the receiver as it simultaneously responds to a GET and manages the mechanics of trading its present network attachment point for another. Designing mobile devices to juggle two wireless network connections as one fades and another strengthens eases the difficulty of tracking the source, smooths handoff, and may reduce interruptions in flow.

Since stateless connections are receiver-driven, the round-trip time r must be significantly less then the interframe periodicity p. An alternate, but more general view, states that the control granularity (in frames) of the receiver is at best r/p. For our example longhaul path between Los Angeles and Boston the control granularity is bounded below at $40/33 = 1.12$ frames and, in practice, is much worse, approximately $90/33 = 2.73$ frames. The tension between frame rate and round-trip time is an inescapable law of physics for stateless connections and effective receiver control is possible only if the control granularity is $\leq 0.5 = r/p$. Therefore, the solution of detecting flow source movement by waiting on a flow "failure" is practical only if the ratio r/p is substantially less than 1.

How, then, under the circumstances of $r/p > 0.5$, can the receiver be informed, in a more timely manner, of changes in network location of the transmitter? *By treating the naming infrastructure as if it were a source of real-time telemetry, namely, the network location of the source as it changes over time.* Thus the receiver employs the *identical* flow mechanisms with respect to the naming infrastructure as it does to the video stream. Let F_v be the video flow and F_n be the "change of network attachment point" flow. We may safely assume that the naming infrastructure is comparatively fixed and static; hence the roving video receiver, by way of stateless flow, is easily capable of transparently managing shifts in flow F_n that are a consequence of the receiver's movement about the network. Further, in terms of round-trip time the naming infrastructure may well be far "closer" to the receiver than the video source and the provisioning for the naming infrastructure is likely to be more generous and robust than that allocated to the video flow F_v.

Solving the problem of simultaneous movement of both source and sink demands, in the extreme case, as illustrated in Figure 3, two simultaneous stateless flows, one devoted to the stream of interest and the other devoted to tracking the network movements of the source. For both flows the sink has complete control and adjusts frame selection, transfer, and rate as it wishes. For example, if the sink detects a variation in rate of network movements of the source it may vary its frequency of GETs correspondingly. It may even, using ancillary information, anticipate the next "destination" of the source and preemptively redirect the flow.

[1] The one-way, flight-of-light time in seconds is given by $d/0.7c$ where d is the distance in kilometers, c is the speed of light in kilometers/second (3×10^5) and 0.7 is the approximate velocity factor of light in a fiber optic strand. The estimated delay is a lower bound as it does not account for routing and switching delays in any internet infrastructure. Informal measurements in February 2006 using `ping`, a popular and well known internet tool, gives actual round-trip times on the order of 80–100 milliseconds between Los Angeles and Boston.

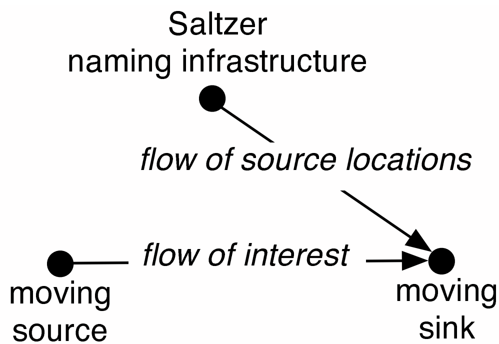


Fig. 3. Flows structure when both source and sink are mobile.

V. Summary

Streaming state kinematics (STREAK) strives to recast flow, repeated unidirectional transfers constrained in space and time, as *stateless flow* where all flow state resides in the receiver. STREAK achieves:

- *Transparency*. All flow state is explicitly reproduced in each flow transfer, permitting any observer or intermediary to apprehend flow condition, progress, and content.
- *Flexibility*. Flow receivers select exactly that portion of the flow that they require and no more—adjusting their uptake as semantics and circumstances demand. Even flow reliability is receiver-controlled for, modulo the frame generation rate, receivers may implement any degree of reliability they require from unreliable best-effort to guaranteed delivery.
- *Sink mobility*. As all flow state is receiver-controlled, receiver physical mobility is trivial to implement and modest state transfer allows a secondary receiver to seamlessly pick up a flow at exactly the point at which the primary receiver left off.
- *Source mobility*. For low-rate flows (where $r/p < 0.5$) the receiver may detect source movement unaided in enough time to redirect the flow to the new source location. For high-rate flows ($r/p > 0.5$) a multilevel naming infrastructure may be reconstituted as a static flow source to advise a sink on source movements for the sake of flow redirection.
- *Simultaneous mobility*. The techniques for sink and source mobility may be combined to accomodate the case where both source and sink change network locations contemporaneously.

These results permit *flow mobility*, one important aspect of flow engineering—the design, implementation and deployment of flows with known characteristics and behaviors. To our knowledge the application of stateless flow to resolve general problems in flow mobility is novel.

Many questions of flow engineering remain unanswered. We mention but a few here. The protocol for stateless flow, while described here, has yet to be implemented and its actual behavior is unknown. While we anticipate that its performance will be adequate for many real-time flows its limitations must be established by simulation and experiment. Implementing stateless flow and measuring its performance is a high priority. The technical details of state transfer from one sink to another ("rehosting" in the terminology of Section III), clearly achievable in theory, remain to be defined and demonstrated. Flow integrity is itself a rich topic and it is clear that higher-level protocols, perhaps modelled on RESTful principles, can contribute to rapid recovery following the loss of information as a consequence of time constraints, unreliable transfer, or source movement. This is an important adjunct to flow mobility that we intend to explore in the near future. Finally, while not addressed here, reliable flows for non-real-time applications are certainly also possible, as Trickles [20], a stateless implementation of TCP, can serve as the reliable, guaranteed delivery equivalent of stateless flows. All of the techniques employed to obtain mobile stateless flows may be reapplied with Trickles, without change, to implement mobile reliable flows.

Flow engineering also requires a sophisticated naming infrastructure. In particular, a complete solution to flow mobility demands a Saltzer-like [4], [19] multilevel layering of names. We treat this naming infrastructure as a flow source of location telemetry, subject to stateless flow, to extract in real-time the network trail of content sources as they move from one network attachment point to another. The recursive application of flow engineering to resolve network location as an adjunct to transparency under mobility hints at other applications. We suggest that many other fundamental elements of network infrastructure besides naming may be profitably treated as flow sources for the purposes of network monitoring, management, and diagnosis.

Finally, we put forth flow engineering as a subdiscipline of software engineering in its right. Flows, in myriad forms, appear in a broad range of applications and domains, both scientific and commercial. Irrespective of the contribution of streaming state kinematics, flows deserve treatment as important software constructions that may be manipulated to specific ends. Starting with flows one may be able to generalize data flow structures such as Weaves [9] to internet-scale assemblages or establish viable, large-scale distribution infrastructures for telemetry, multimedia, and commercial transactions. We see a bright future for the study and application of flows in the context of software engineering.

VI. Acknowledgements

I am indebted to Eric Dashofy, Deborah Shands, and Stephan Schwab for formative discussions on the problem of flow mobility and for their review of an early, partial draft of this report.

This work was supported in part by NSF Grant #0438996.

References

[1] Farhad Arbab, "Reo: A Channel-based Coordination Model for Component Composition," Mathematical Structures in Computer Science, 14(3), pp. 329–366, June 2004.

[2] T. Aura and P. Nikander, "Stateless connections," Research Report A46, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland, 27 pages, May 1997.

[3] T. Aura and P. Nikander, "Stateless connections," Proceedings of the First International Conference on Information and Communication Security (November 11 - 14, 1997), Y. Han, T. Okamoto, and S. Qing, Eds. Lecture Notes In Computer Science, vol. 1334. Springer-Verlag, London, 87-97, 1997.

[4] Hari Balakrishnan et al., "A Layered Naming Architecture for the Internet," Proceedings of the ACM SIGCOMM Conference, Portland, Oregon, August 2004.

[5] http://www.bittorrent.org

[6] A. Carzaniga, D.S. Rosenblum and A.L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," ACM Transactions on Computer Systems, 19(3), August 2001, pages 332–383.

[7] Roy T. Fielding and Richard N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology*, 2(2), May 2002, pp. 115-150.

[8] Roy Fielding et al., "Hypertext Transfer Protocol—HTTP/1.1," RFC 2616, The Internet Society, June, 1999.

[9] Michael Gorlick and Rami Razouk, "Using Weaves for Software Construction and Analysis," Proceedings of the 13th International Conference on Software Engineering, Austin, Texas, May 1991.

[10] Juan Guillen-Scholten, "MoCha: A Model for Distributed Mobile Channels," Masters Thesis, CWI, Amsterdam, The Netherlands, May 2001, available as CWI Technical Report SEN-E0418, 2004.

[11] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol," Internet Engineering Task Force, Internet Draft, December 2005, 99 pages.

[12] Karthik Lakshminarayanan, Ion Stoica and Klaus Wehrle, "Support for Service Composition in i3," ACM Multimedia (Position paper), New York, New York, 2004.

[13] David L. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis," RFC 1305, Internet Engineering Task Force, March 1992.

[14] P. Oreizy, M. Gorlick, R. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, A. Wolf, "An Architecture-Based Approach to Self-Adaptive Software," IEEE Intelligent Systems, 14(3), May/June 1999.

[15] S. Oueslati and J. Roberts, "A New Direction for Quality of Service: Flow-Aware Networking," Proceedings of the Conference on Next Generation Internet Neworks, Rome, Italy, April 2005.

[16] Jon Postel, "User Datagram Protocol," RFC 768, Internet Engineering Task Force, August 1980.

[17] Lawrence G. Roberts, "The Next Generation of IP-Flow Routing," SSGRR International Conference, L'Aquila, Italy, July 2003.

[18] RSS 2.0 Specification, http://blogs.law.harvard.edu/tech/rss.

[19] J. Saltzer, "On the naming and binding of network destinations," In P. Ravasio et al., editor, Local Computer Networks, pages 311–317, North-Holland Publishing Company, Amsterdam, 1982. Reprinted as RFC 1498, Aug 1993.

[20] Alan Shieh, Andrew C. Myers, and Emin G. Sirer, "Trickles: A Stateless Network Stack for Improved Scalability, Resilence, and Flexibility," Proceedings of Symposium on Networked Systems Design and Implementation, Boston, Massachusetts, May 2005.

[21] Brian C. Smith, "Implementation Techniques for Continuous Media Systems and Applications," PhD Thesis, University of California, Berkeley, September 1994.

[22] Brian C. Smith, "Cyclic-UDP: A Priority Best-Effort Protocol," Computer Science Department, Cornell University.

[23] Ion Stoica et al., "Internet Indirection Infrastructure," Proceedings of the ACM SIGCOMM Conference, Pittsburgh, Pennsylvania, August, 2002.

[24] "Theora I Specification," Xiph.org Foundation, May 6, 2005.

[25] Michael Walfish, Hari Balakrishnan and Scott Shenker, "Untangling the Web from DNS," Proceedings of the Symposium on Network Systems Design and Implementation, San Francisco, California, March 2004.

[26] Shelley Q. Zhuang, Kevin Lai, Ion Stoica, Randy H. Katz and Scott Shenker, "Host Mobility using an Internet Indirection Infrastructure," First International Conference on Mobile Systems, Applications, and Services (ACM/USENIX Mobisys), May 2003.