Application Protocol Design Considerations for a Mobile Internet

Jörg Ott
Helsinki University of Technology
Networking Laboratory
jo@netlab.tkk.fi

ABSTRACT

The Internet protocols were designed for a primarily "fixed" and relatively static network environment where communication links are stable and exhibit fairly uniform communication characteristics. Mobile wireless communication has fundamentally invalidated some of these assumptions, for (heterogeneous) wireless access networks and even more so for mobile ad-hoc networks (MANETs) formed between mobile users: from highly variable link characteristics to temporary disconnections to non-existing end-to-end paths. While many activities have focused on the link and network layer to provide seamless and ubiquitous connectivity for mobile users, thus mimicking the fixed Internet, and transport layer optimizations have addressed performance issues and connection persistence in wireless networks, application protocols have received rather little attention. However, the semantics of many of today's non-real-time applications are perfectly compatible with partly connected and disruptive mobile environments, it is just the protocol designs that are not. We identify issues with present application protocols and discuss requirements to make them workable in challenged mobile environments, leveraging Delay-tolerant Networking (DTN) as underlying communication paradigm and augmenting server-based operation by peer-to-peer communications.

1. INTRODUCTION

Communication in the Internet is increasingly dominated by nomadic and mobile usage scenarios: mobile nodes attach directly to (wireless) access networks and increasingly powerful mobile devices enable mobile ad-hoc networking, for interactions between users as well as for reaching infrastructure networks. However, the Internet protocols have been designed for rather static (typically fixed) networking environments with fairly stable end-to-end connectivity and uniform path characteristics (low bit error rates, RTT, and packet loss probability)—and so have been most traditional application protocols.

Modern mobile communication environments invalidate various of the fundamental assumptions: connectivity may appear and disappear unpredictably, may be short-lived or long-lasting, link and path characteristics may vary instantaneously, corruption-incurred losses may outweigh congestion losses, and not even an end-to-end path may exist, e.g., in sparse ad-hoc networks or with two peers experiencing alternating disconnections. For network access, a jungle of (wireless) service providers for different cellular, WLAN, and other link layer technologies peered with diffuse roaming agreements make obtaining connectivity technically challenging and economically risky for the user, at least, for the foreseeable future). Simple technical limitations (e.g., battery power) and isolated or remote locations may further add to connectivity loss. Finally, legal regulations or social conventions may entirely prohibit communication for certain periods of time or certain locations so that, even if communications opportunities existed, they could not be used.

In summary, mobile connectivity is uncertain and its quality may vary. In the past, numerous research efforts have tackled mobility support at virtually all layers and in-between from the link to a session layer between transport and application. Pros and cons can be found for mobility support at any of these layers—and they are in fact often complementary. All approaches have in common that they attempt to conceal the implications of mobility from the applications. However, since disconnections may be arbitrary, completely mimicking fixed Internet connectivity in the mobile domain is bound to fail, albeit mobility support at many layers may provide improvements.

We therefore characterize a mobile communication environment primarily by unstable and variable links and paths, the potential for frequent unexpected disconnections and unpredictable connection opportunities, and the potential non-existence of an end-to-end path. In short: our concern with mobility are *disconnections* and the *delays* that may result.[1]

Despite the shortcomings of mobility support only at the lower layers (which we discuss in section 2), application layer protocols have received only little attention. In this paper, we analyze design patterns of existing application protocols to identify which aspects of their designs make them less suitable for mobile operation. We focus on applications that are basically workable without permanent connectivity: this excludes interactive real-time applications (such as IP telephony and certain games) and we further restrict ourselves to applications that involve at least one human user. Based upon our analysis in section 3, we derive suggestions for the design of future mobile-enabled application protocols in section 4 and conclude this paper with a short discussion of future research directions in section 5.

MobiArch '06 San Francisco, California, USA

[1]We consider wireless characteristics of lesser importance (and poor performance can be subsumed under disconnections). Therefore, we do not discuss aspects of encoding efficiency on the link, (header) compression, and related issues.

2. RELATED WORK

Numerous research efforts have addressed providing *seamless connectivity* across different link layer technologies and service providers to keep users *always best connected*. They combine mobile IP for persistent endpoint identification, reachability with handover and roaming improvements (authentication, state transfer), and using multiple interfaces to minimize connectivity gaps, prevent packet loss, and reduce jitter. While such approaches clearly improve connectivity they are obviously not able to address disconnections due to coverage gaps, missing roaming agreements, or user policy (e.g., minimizing access charges).

Besides mitigating the impact of wireless links on end-to-end (TCP) performance [5], numerous extensions at the transport layer seek to shield connectivity changes from the applications to maintain their functioning. Such extensions may rely on mobile IP so that TCP connections do not break [11, 2] or explicitly signal IP address changes end-to-end [28]. Support for disconnections can also be achieved in combination with HIP [26] or by means of a shim (session) layer on top [25]. All of these approaches, however, require an occasional end-to-end path and have no means to prevent application protocols from timing out if, e.g., an outstanding protocol operation does not complete due to disconnection.

Protecting applications from disconnections may also be achieved by means of proxies—explicitly placed or transparently inserted—between the application peers (typically client and server) as in Fleetnet [3], Drive-thru Internet [19], and DHARMA [16], among others. This approach may also allow relaxing the need for end-to-end connectivity [19] as well as providing feedback to the mobile client application to prevent timeouts and inform the user about communication progress [8]. Again, at least an occasional "end-to-end" path from the client to the proxy is typically required and the interaction with application often remains cumbersome and security issues are usually not addressed.

All three areas above have in common that they follow a bottom-up approach mitigating one issue at a time. In all cases, they try to provide an *abstraction* to the higher layers that hides the particulars of the network. This abstraction becomes *leaky* as soon as, from a higher layer perspective, connectivity is lost for too long or the necessary end-to-end communication cannot be established at all. As a consequence, application protocols and programs need to become aware of and actively deal with delays and disconnections.

Delay-tolerant Networking (DTN) [7, 27] follows a different communication paradigm and exclusively relies on asynchronous message exchange without the need for an end-to-end path. Assuming appropriate routing mechanisms (such as [14]), this makes DTN perfectly suitable for communication in (sparse) MANETs or other challenged environments using dedicated application protocols. But DTN can also be used for interpersonal communication in MANETs [10, 12, 17] as well as for (mobile) Internet access [1, 21]. A prerequisite is that the application protocols have been designed for or can be adapted (e.g., by means of proxies) to work in a disconnected environment where sometimes only occasional message exchanges may be possible [18, 21, 15].

Surprisingly, application-specific support for nomadic and mobile usage has been very limited so far. Exceptions are dedicated application protocols developed for *disconnected operation* such as distributed file systems [13], calendars, or authoring and versioning systems (e.g., *cvs* or *subversion*) and the (limited) support for *offline* operation in web browsers and email clients, usually requiring manual control. Numerous web-based applications store session state on a server and are thus able to recover from disconnections to some degree. However, the arrival of interaction techniques such as AJAX [4] is creating new problems here as well.

In the following, we will assess the characteristics of application protocols commonly used in mobile environments in more detail to identify issues in and outline requirements for mobility-suitable application protocol design.

3. APPLICATION PROTOCOL ANATOMY

Looking at typical (mobile) usage scenarios for non-real-time applications, we can observe that numerous popular applications are well-suited for operation without (immediate) end-to-end connectivity [18]. Sending and receiving emails is asynchronous in the first place, the user mostly typing or reading so that connectivity is only needed during short periods of time; and in most cases emails may well be queued prior to actual transmission or reception (which may happen in mail servers anyway) [10, 23]. The same applies to interactions with discussion fora and newsgroups and may also hold for (not-so-instant) messaging and personal presence applications, depending on the desired update granularity. Calendar synchronisation is basically asynchronous (and calendar events are often mapped to email messages). Even access to web resources can tolerate delays provided that the user does [20, 21]: She usually alternates between reading and requesting new web pages, thus limiting the actual need for connectivity to short periods; tabbed browsing to request several resources in parallel (e.g., after a web search) further supports this idea [6], retrieving content in the background whenever possible.[2]

In principle, from a semantics perspective, all the above applications are workable in a temporarily disconnected environment and none of them requires an end-to-end path at any time as long as intermediary nodes (other mobile hosts or, as in email, infrastructure components) take the responsibility of forwarding the messages from the sender to the receiver. The degree of connectivity of a network then defines the user-perceived quality of service: how long does mail delivery or retrieval of a web page take?

In practice, however, the designs of most Internet application protocols do not work without end-to-end paths as we will discuss in the following.

3.1 Actors: Endpoints and Intermediaries

We start by looking at the actors in application protocol interactions. These are the two[3] *endpoints* of the conversation and optionally one or more *intermediaries*. Whether an entity is considered an intermediary or an endpoint depends on the viewpoint. We take the perspective of the intended application semantics: for example, email messages are exchanged—end-to-end—between users, technically represented by their mail "clients", while the mail servers only facilitate end-to-end communication. In this sense, a mail server is an intermediary while an HTTP origin server is not.

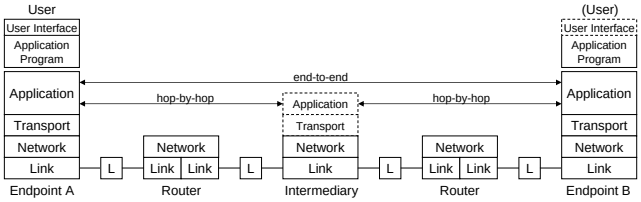

Figure 1: Application protocol actors and interactions

[2]Selected contents might also be pushed to the user's local web cache whenever opportunities exists.

[3]We do not consider group communication in this paper.

Figure 1 depicts two endpoints communicating via one intermediary which may operate at and across different layers. In some application protocols, intermediaries are mandatory (such as mail servers), in others they are optional (such as HTTP proxies). Endpoints may (have to) be aware of intermediaries and use them *knowingly*: this is the case for mail servers used as mandatory message store, SIP servers used for rendezvous purposes (both enabling user mobility), and may be the case for web caches and proxies used for performance improvement and possibly content adaptation. But endpoints may also use intermediaries *unknowingly*: examples are NATs and firewalls for "security", replicated application servers in content distribution networks (CDNs), transparently inserted web caches, and performance enhancing proxies.

Intermediaries *enable* communication, e.g., by providing well-defined rendezvous points, decoupling transmission and reception, and possibly even converting between different protocols and content representations. But they may also *hinder* communication due to unexpected protocol manipulation or blocking communications (firewalls, NATs, application-layer gateways) or impact the application protocol by breaking up end-to-end connections (as with HTTP proxies and end-to-end HTTP-over-TLS). Those meant to enable communication by serving as *mandatory* rendezvous point may also become (single) points of failure and prevent communication in challenged environments, e.g., if the peers reach one another in an ad-hoc network but not their allegedly well-connected servers in the fixed Internet. Intermediaries sometimes require user trust to reduce vulnerabilities to DoS or other attacks (like SMTP servers) and to protect user privacy (POP or IMAP servers).

Intermediaries typically cause problems with application protocols if the latter do not operate end-to-end by themselves but rather rely on reliability, security, or other support from lower layers—which reduces end-to-end to hop-by-hop semantics (see figure 1). This is important because intermediaries—application-specific as mail servers or generic as DTN bundle agents—will be essential for communication in a mobile environment where end-to-end paths do not always exist.

3.2 Actions: Application Operations and State

We have characterized application protocols with respect to their interaction behavior and derived their suitability for challenged environments [18]. One of the crucial observations is that, as they assume stable connectivity and short RTTs, many protocols are highly interactive ("chatty"). In the simplest case, a protocol may require many transactions to complete a single semantically meaningful protocol operation, such as retrieving a viewable web page consisting of many different resources. In more complex cases, the application may even require several interactions to perform a single transaction with shared state being created and advanced step by step, e.g., when logging into an IMAP or FTP server, submitting an email via SMTP, or performing authentication based upon a challenge-response mechanism like HTTP digest authentication.

In many such cases, the application protocol state is bound to the existence of the underlying transport (typically TCP) connection: if the transport connection disappears, the state is lost, too. If the identification of the peer instance is tied to the transport connection, a new transport cannot even be re-established to recover the state. Examples include mail protocols and FTP; a nice counterexample is the Session Initiation Protocol (SIP) that clearly separates application transactions and state from that of the underlying transport.

Timeouts in application protocols are generally an issue because they have to strike the balance between trying to complete operations on one hand and preventing state explosion in servers as well as considering user patience on the other. The timeouts are defined by the specification or the programmer and are often independent of the networking environment, making the optimistic assumption of sufficiently small path RTTs to allow their interactions to complete. This is in contrast to, e.g., transport protocols that dynamically adjust timeout values to the observed path characteristics. But even if (exponential) backoff mechanisms and dynamic RTT calculations are applied (such as for retransmissions in SIP) they usually scale only to a limited extent (e.g., tolerating delays in the order of seconds to minutes and declaring failure afterwards).[4]

Application protocols legitimately rely on transport layer reliability and expect immediate positive or negative acknowledgements from their peers. This does no longer work as soon as intermediaries are involved which can usually provide only preliminary confirmations. For example, mail servers perform an immediate acceptance check for an incoming email and may instantly refuse it. If a server accepts a mail this has no meaning whatsoever concerning the ultimate success or failure of the mail delivery. There is no upper bound (timeout) for a positive confirmation but, at least, email supports end-to-end failure and delivery notifications.

Finally, application protocols use different mechanisms for state synchronization. Soft state protocols relying on regular state refreshes may suffer in environments where delivery delays cannot be estimated. But also hard state approaches may not work well if the application state of both endpoints needs to be closely coupled—and hard state may be left over after an application peer has long since disappeared. In either case, it may be impossible to differentiate permanent failure from temporary disruption. As many application protocol operations are not idempotent (exceptions include, e.g., some operations of NFS and HTTP), repeated invocation will cause repeated action (such as repeated delivery of an email). This has not been an issue with protocols operating well synchronized and being able to rely on actions being immediately confirmed or rejected—but this property may be lost in mobile environments.

3.3 Security

The previous considerations on end-to-end semantics also apply to security: as secure protocol design is almost a discipline of its own, many application protocols avoid the pitfalls by relying on security mechanisms readily available, typically by underlying protocols such as IPsec, SSH tunneling, and transport layer security (TLS). However, these only provide security for a single transport layer "hop" and are thus only useful in the absence of intermediaries (as with HTTPS) or if all intermediaries are trusted (as in some SIP scenarios). Since we require intermediaries but do not want to create single points of failure, we cannot depend on selected trusted intermediaries. Moreover, other users' mobile devices (which are generally not trustworthy) are also used for data forwarding so that hop-by-hop protection is clearly insufficient.[5] Security mechanisms of application protocols (particularly those not defining intermediaries) will require careful consideration.

Furthermore, present security protocols often require multiple message exchanges, e.g., for authentication, capability negotiation, and session key establishment. A simple example are challenge-response schemes (as in HTTP digest authentication that requires a four-way handshake for authentication only); a more complex one is the startup protocol of a TLS connection. The result is that messages cannot be sent asynchronously and processed without further

[4]Similarly, timeouts may be triggered in well-connected environments if current server (over)load delays processing requests and generating responses.

[5]A related issue arises with security mechanisms that assume that no man-in-the-middle can alter data in flight, such as Diffie-Hellman-based session key generation.

interaction between the peers. Limiting the number of exchanges to minimize the impact of delays also means that some security properties, such as perfect forward secrecy, are much harder to attain.

End-to-end security mechanisms such as S/MIME with public key cryptography would allow for such self-contained message protection by providing all the necessary credentials as part of a message. However, besides lacking global deployment, security mechanisms for validating an identity, e.g., when using a PKI to check a certificate, may require additional interactions with infrastructure components which may not be instantaneously possible.

3.4 Naming and Addressing

Applications use names and addresses for rendezvous and communication purposes and to identify their peers. The duality of IP addresses being both identifiers and locators and the associated issue that IP addresses may change due to mobility are well-known. More recent applications already rely on stable application layer identifiers instead (such as URIs) or use support from lower layer mechanisms such as HIP.

Service protocols such as DNS resolve names into IP addresses. Clients contact DNS infrastructure servers, possibly in multiple iterations. While the servers are important for rendezvous purposes, they may inhibit communication if they are not reachable. Furthermore, DNS maintains static address bindings and is thus only useful for fixed servers.

For direct communication between mobile users, other (application-specific) mechanisms are applied to update a mobile node's address frequently (such as dynamic DNS or SIP registrations). Again, temporary non-reachability may limit the usefulness of these protocols in mobile environments as long as fixed infrastructure is required. Distributed location protocols (e.g., server-less registrations in peer-to-peer SIP using DHTs) may eliminate the need for an infrastructure, however, their applicability to highly dynamic and disconnected environments remains to be proven.

3.5 Application Programs and User Interfaces

Application protocols provide services to users; the application offering a service abstracts from the underlying communication processes. As discussed for transport protocols above, this abstraction becomes leaky as soon as communication does not proceed as expected. This expectation is governed by application timeouts and ultimately by the user's patience to wait for an operation to complete. Application timeouts may try to estimate user patience, but in particular they are used to determine whether a requested service is unavailable. With mobility, unavailability becomes hard to distinguish from temporary unreachability—and while a user may want an immediate notification that a request will definitely fail because of a broken peer application, she may be more delay-tolerant knowing that the result will finally be delivered (rather than having to manually poll the service by issuing the same request over and over again). Internet applications and their user interfaces often deal badly with temporary failures; they quickly generate error messages to the user, even if it would be possible to keep trying. Better distinction between different types of failures is missing and user controls rarely exist. Mail clients at least support regularly polling their servers and do not give up if they do not succeed; web browsers may become tolerable when proxies are involved. But these are rather exceptions and they still provide little feedback.

4. PROTOCOL REQUIREMENTS FOR MOBILE OPERATION

We assume that applications can and shall operate under various fixed and mobile conditions using wired and wireless links, with infrastructure-based connectivity and in ad-hoc environments. We propose using DTN with asynchronous message passing as the most flexible communication substrate to deal with the aforementioned challenges of mobility—after all, disconnections and delays may render mobile communication asynchronous from an application perspective anyway.

In the following subsections, based upon the shortcomings identified above, we present first a number of technical suggestions for application protocol design and finally discuss some soft factors that deserve attention. Such general considerations are obviously difficult—and incomplete—by nature, details may differ between applications, and we are not able to present satisfactory solutions to realize all the suggestions. Yet they provide a set of data points in the design space of future mobile application protocols from one particular viewpoint.

4.1 The Role of Intermediaries

Intermediaries will play a crucial role in establishing connectivity by means of asynchronous message forwarding. To avoid adding single point(s) of failure, one set of intermediaries—i.e., DTN routers located in mobile nodes and infrastructure components —should operate independent of particular applications and focus on message forwarding.

Further application-specific intermediaries may exist, e.g., for gatewaying purposes or as a message store. However, an application protocol should not depend on them in message exchanges to avoid that, e.g, remote fixed components (that may be temporarily unreachable) inhibit communication between colocated mobile peers. Instead, application protocols should be designed for direct end-to-end operation, e.g., avoiding asymmetric protocols as used for email transmission and access. Rendezvous mechanisms and decoupling of sending and receiving actions—the most common uses for intermediaries—can already be provided by the distributed DTN substrate and need not be delegated to a dedicated component. Application-specific intermediaries (such as mail "servers") may still provide valuable services (e.g., as message store) and collect (backup) copies of messages for later synchronization; but they should not be in the critical path of message delivery.

Application protocols should explicitly support controlled interaction with known or unknown intermediaries by providing protocol-independent hints about their intentions (e.g., the requested resource) so that supportive functions (e.g., caching) can be implemented in arbitrary nodes, in a generic fashion as well as by means of application-specific modules where available [22]. This also requires a clear separation of the *communication transaction* portion (e.g., an HTTP GET response) of a protocol message from the *substance* portion (e.g., the resource carried in the response) conveyed as part of a transaction so that the origins of the message and the contents may be different and may be operated on (e.g., cached) and secured (e.g., authenticated, integrity protected) independently.[6]

4.2 True End-to-End Semantics

The end-to-end principle suggests that protocol functions can be best (if not only) provided where the necessary context information is, i.e., in the applications [24]. This applies particularly to applications in mobile communication environments where end-to-end paths may not exist so that intermediaries are needed. This calls for applications maintaining state entirely independently of lower layers and also modifying state alone by explicit end-to-end operations within the application protocol, using explicit positive and negative

[6]Unlike HTTP, where attributes about a contained resource are carried in HTTP headers making caching of resources authenticated in their entirety difficult.

confirmations where appropriate to achieve end-to-end reliability.

This also requires end-to-end security mechanisms at the application layer which must be robust against untrusted entities on the path capable of inserting, modifying, and deleting messages at random. With the presence of random intermediaries, message contents needs to *self-protecting* end-to-end as well, i.e., any private information needs to be encrypted within a message and not rely on lower layer mechanisms.

4.3 Aspects of Protocol Operation

Application protocols must be prepared to operate asynchronously and to deal with RTTs in the order of minutes or more. Therefore, they must not depend on frequent interactions but use *self-contained messages* that carry all the necessary information for processing—by their remote peers as well as in intermediaries.[7] For all interactions, protocol timers should be flexible to deal with highly variable RTTs.

Since messages may be replicated and several copies may arrive at the destination (possibly with minutes or hours in between), protocol operations should ideally be idempotent and, if necessary, explicitly indicate the initial state they wish to act upon as some precondition so that the recipient can filter out outdated ones. To minimize the need for end-to-end interactions, the concept of state predicates may be expanded to more complex operation flows described by metadata (as minimally present in conditional operations in HTTP) and even to mobile code. This would allow, e.g., messages to include alternative request treatments in the order of preference or even numerous operations and alternative processing paths for later operations depending on the outcome of earlier ones. In effect, this would support maximizing independent message handling, applicable to endpoints as well as intermediaries [6].

Applications should not use potentially ephemeral lower layer (e.g., IP) addresses to identify endpoints or intermediaries but rather persistent identifiers (such as URIs) and also allow distinguishing between different application instances. To avoid relying on a (potentially unreachable) address resolution infrastructure, the identifier-based routing capabilities of DTN with *late binding* should be exploited so that the address resolution can be deferred by the mobile node to the routing substrate. Where translations between names, addresses, and identifiers or other location functions are needed, infrastructureless operation (which may be application- specific) may be preferable—which may leave trust issues to be solved.

4.4 Some Security Considerations

As noted above, end-to-end security mechanisms are needed and transaction and content ("substance") protection should be independent. Furthermore, the limited interactivity implies that challenge-response mechanisms for authentication or n-way handshakes to prevent DoS attacks may not be as easily applicable. Instead, some variant of public key cryptography may be needed to authenticate the device (e.g., based upon cryptographic host identities as in HIP) and/or the user (e.g., using certificates). However, it may not be possible to instantaneously validate newly received certificates so that additional plausibility checks may need to be performed by the application, paired with caching of self-signed certificates (ideally, after an initial contact in a trustworthy environment).[8] Depending on the level of authentication achieved, the amount of resources allocated to process a message may vary (and, e.g, grow over time as trust increases over repeated contacts). End-to-end encryption may benefit from HIP as well as from identity-based cryptography.

[7]With the capability of DTNs to convey messages of arbitrary size, MTU size limitations are no longer an issue.

[8]Certificate revocation will also need to be addressed.

Finally, an important design consideration (not just for application protocols) are denial-of-service and other attacks. With fewer interactions, the computational complexity of operations to validate a single message may increase. Furthermore, when variants of flooding or probabilistic routing are used—not unusual for mobile ad-hoc networks—messages may be disseminated widely so that some "on-path" attacks become easier: malicious nodes may easily learn about potential targets from observing other messages and then flood the victim.

4.5 Soft Factors

Designing application protocols for challenged mobile environments implies increased robustness to all kinds of failures which may also be advantageous in usually well-connected environments. The downside of an asynchronous design, however, may be a lack of (perceived) responsiveness or performance while connected (compared to traditional protocols fulfilling the same function). Since a user must not be required to know (or even configure) whether she is well connected or not, any application protocol designed for a mobile environment should not perform noticeably worse in a connected environment compared to its predecessor. From the user interface perspective, this may imply preserving the basic look and feel and the basic interaction paradigms for the user and gradually adding further means for user awareness about disconnections and operations in progress as well as, ultimately, features to control delay-tolerant operation.

Furthermore, the deployment barrier for individuals must be low. Usage incentives (i.e., personal gain) should not be depend on an already large adoption. Basically, the capability of DTNs to operate in sparse environments may support communication even in a small user community. Additionally, any new application protocol design must also offer a migration path for user applications (clients, servers, peers) and intermediaries that allows for gradual introduction. With proper intermediary support built into a protocol, designing appropriate gateway functions to support the transition should be straightforward (even if many gateways may have to be trusted to translate, e.g., authentication functions, so that some points of failure will remain).

Finally, as deployment also implies at least some support by users, vendors, operators, and various other stakeholders, the relevant economic and political aspects need to be considered [9]. For example, with ad-hoc networking, users may be concerned about others intruding their mobile devices (which carry lots of important personal data today) and may wonder why they should share their CPU, memory, and battery capacity with others so that incentives are needed (e.g., based upon reciprocity). Also, service providers should see an opportunity (rather than just a threat) for their operations which, in turn, may influence the willingness of device vendors to support these new applications—in addition to the latter's motivation of providing richer functionality in their handsets.[9]

5. CONCLUSION

In this paper, we have discussed protocol design aspects for mobility support exclusively based upon asynchronous communication. To a certain extent, these are mostly about robust application protocol design where mobility essentially may increase the failure probability significantly and allows for shortcuts (e.g., peer-to-peer) that were not needed before.

DTN-based mobility raises one major question: will the performance of purely asynchronous operation be acceptable in (mo-

[9]For example, we observe a slowly increasing support for IEEE 802.11 WLAN in mobile phones.

bile and) well-connected environments. Given that today's protocols are quite inefficient in their interaction behavior and in the use of network resources (and that some 20+ years of experience have gone into the development of today's Internet protocol stacks), there is no reason why well-tuned asynchronous protocol implementations should not achieve similar performance. Nevertheless, some application protocols may exhibit different behavior to the user: for example, with bundled transmission of web pages, their content may no longer be displayed incrementally as the individual resources arrive but rather all-at-once. In effect, the user loses the option to take the next action while retrieval is still in progress (such as selecting the next link to follow or aborting retrieval). To maintain this user experience, optimizations such as some form of incremental delivery may be worth considering (when the user is seemingly well-connected).

This highlights one aspect of the general issue of redesigning application protocols: entities need to be more proactive and clearly communicate all of their intentions at once rather than iteratively interacting with a peer. This may lead to increased complexity in application protocols—which may employ metadata or even mobile code to express themselves—so care must be taken to avoid overengineering.

Relying on a DTN substrate also means further assessing its ability to efficiently deal with user mobility and its "reliability" of message delivery—which may influence design choices for application protocols. In particular, the behavior of DTNs under stress ("congestion") is also yet to be investigated as are suitable "feedback loops" for applications. The aforementioned explicit signaling in application protocols may also cover their interaction with a DTN layer, e.g., by means of routing and message handling hints.

Finally, security—particularly authentication of other nodes and authorization of local resource utilization—to prevent DoS attacks, spamming, and other security threats, will be crucial for large scale deployment. Mechanisms need to be designed that can operate without frequent end-to-end interactions and without an always available infrastructure.

6. REFERENCES

[1] Mindstream project. http://mindstream.watsmore.net/, 2006.

[2] A. Baig, M. Hassan, and L. Libman. Prediction-based Recovery from Link Outages in On-Board Mobile Communication Networks. In *Proceeding of IEEE Globecom 2004*, December 2004.

[3] M. Bechler, W. J. Franz, and L. Wolf. Mobile Internet Access in FleetNet. In *13. Fachtagung Kommunikation in verteilten Systemen, Leipzig, Germany*, April 2003.

[4] O. Bergmann and C. Bormann. AJAX: Frische Ansätze für das Web-Design. SPC TEIA Lehrbuch Verlag, 2005.

[5] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135, June 2001.

[6] C. Bormann, D. Kutscher, and J. Ott. Disruption Tolerance: The Near End. Presentation at the Dagstuhl Seminar on Disruption Tolerant Networking, April 2005.

[7] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H.Weiss. Delay-Tolerant Network Architecture. Internet Draft draft-irtf-dtnrg-arch-05, Work in progress, March 2006.

[8] H. Chang, C. Tait, N. Cohen, M. Shapiro, S. Mastrianni, R. Floyd, B. Housel, and D. Lindquist. Web Browsing in a Wireless Environment: Disconnected and Asynchronous Operation in ARTour Web Express. In *Proceedings of ACM Mobicom 97, Budapest, Hungary*, pages 260–269, 1997.

[9] D. D. Clark, K. R. Sollins, J. Wroclawksi, and R. Braden. Tussle in Cyberspace: Defining Tomorrow's Internet. In *Proceedings of ACM SIGCOMM*, August 2002.

[10] A. Doria, M. Uden, and D. P. Pandey. Providing connectivity to the saami nomadic community. In *Proceedings of the 2nd International Conference on Open Collaborative Design for Sustainable Development, Bangalore, India*, December 2002.

[11] T. Goff, J. Moronski, and D. Phatak. Freeze-TCP: A True End-to-end TCP Enhancement Mechanism for Mobile Environments. In *Proceedings of IEEE Infocom*, 2000.

[12] Haggle project website. http://www.haggleproject.org/, 2006.

[13] J. J. Kistler and M. Satyanarayanan. Disconncted Operation in the Coda File System. In *ACM Transactions on Computer Systems*, volume 10, February 1992.

[14] A. Lindgren and A. Doria. Probabilistic Routing Protocol for Intermittently Connected Networks. Internet Draft draft-lindgren-dtnrg-prophet-02, Work in Progress, March 2006.

[15] M. Lukac, L. Girod, and D. Estrin. Disruption Tolerant Shell. In *ACM SIGCOMM Workshop on Challenged Networks (CHANTS)*, September 2006.

[16] Y. Mao, B. Knutsson, H. Lu, and J. Smith. DHARMA: Distributed Home Agent for Robust Mobile Access. In *Proceedings of the IEEE Infocom, Miami*, March 2005.

[17] O. Mukhtar and J. Ott. Backup and Bypass: Introducing DTN-based Ad-hoc Networking to Mobile Phones. In *Proceedings of RealMAN 2006*, May 2006.

[18] J. Ott and D. Kutscher. Why Seamless? Towards Exploiting WLAN-based Intermittent Connectivity on the Road. In *Proceedings of the TERENA Networking Conference, TNC 2004, Rhodes*, June 2004.

[19] J. Ott and D. Kutscher. A Disconnection-Tolerant Transport for Drive-thru Internet Environments. In *Proceedings of IEEE Infocom, Miami*, March 2005.

[20] J. Ott and D. Kutscher. Applying DTN to Mobile Internet Access: An Experiment with HTTP. Technical Report TR-TZI-050701, Universität Bremen, July 2005.

[21] J. Ott and D. Kutscher. Bundling the Web: HTTP over DTN. In *Proceedings of WNEPT 2006*, August 2006.

[22] J. Ott and M. J. Pitkänen. Application-aware DTN Routing. Submitted for publication, 2006.

[23] A. Pentland, R. Fletcher, and A. Hasson. DakNet: Rethinking Connectivity in Developing Nations. *IEEE Computer*, 37(1):78–83, January 2004.

[24] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[25] J. Salz, A. C. Snoeren, and H. Balakrishnan. TESLA: A Transparent, Extensible Session Layer Architecture for End-to-end Network Services. In *4th Usenix Symposium on Internet Technologies and Systems*, March 2003.

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

[27] K. Scott and S. Burleigh. Bundle Protocol Specification. Internet Draft draft-irtf-dtnrg-bundle-spec-05, Work in progress, May 2006.

[28] A. C. Snoeren and H. Balakrishnan. An End-to-End Approach to Host Mobility. In *Proceedings of Mobicom*, 2000.