MFTP: A Clean-Slate Transport Protocol for the Information Centric MobilityFirst Network

Kai Su, Francesco Bronzino, K. K. Ramakrishnan[§] and Dipankar Raychaudhuri
WINLAB, Rutgers University, North Brunswick, NJ 08902, USA
[§]University of California, Riverside, CA 92521, USA
{kais, bronzino, ray}@winlab.rutgers.edu, [§]kk@cs.ucr.edu

ABSTRACT

This paper presents the design and evaluation of clean-slate transport layer protocols for the MobilityFirst (MF) future Internet architecture based on the concept of named objects. The MF architecture is a specific realization of the emerging class of Information Centric Networks (ICN) that are designed to support new modes of communication based on names of information objects rather than their network addresses or locators. ICN architectures including MF are characterized by the following distinctive features: (a) use of names to identify sources and sinks of information; (b) storage of information at routers within the network in order to support content caching and disconnection; (c) multicasting and anycasting as integral network services; and in the MF case (d) hop-by-hop reliability protocols between routers in the network. These properties have significant implications for transport layer protocol design since the current Internet transports (TCP and UDP) were designed for the end-to-end Internet principle which uses address based routing with minimal functionality (i.e. no storage or reliability mechanisms) within the network. Several use cases including web access, large file transfer, Machine-to-machine and multicast services are considered, leading to an identification of four basic functions needed to constitute a flexible transport protocol for ICN: (i) fragmentation and end-to-end re-sequencing; (ii) lightweight end-to-end error recovery with in-network transport proxies; (iii) optional flow and congestion control mechanisms; and (iv) scalable multicast delivery mechanisms. The design of the MobilityFirst transport protocol (MFTP) framework realizing these features in a modular and flexible manner is presented and discussed. The proposed MFTP protocol is then experimentally evaluated and compared with TCP/IP for a few representative scenarios including mobile data delivery, web content retrieval and disconnected/late binding service. The results show that significant performance gains can be achieved in each case.

ICN'15, September 30–October 2, 2015, San Francisco, CA, USA.
© 2015 ACM. ISBN 978-1-4503-3855-4/15/09 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810156.2810169.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

Transport protocol; future Internet architecture; Information Centric Networks; hop-by-hop transport; in-network storage; end-to-end reliability; flow control; congestion control;

1. INTRODUCTION

The TCP/IP architecture underpinning the current Internet is based on the end-to-end principle [1] of minimizing functionality in the network while handling service-specific requirements such as error and flow control at the endpoints. In addition, the current Internet architecture is based on the concept of routing between IP addresses requiring a static one-to-one association between hosts and network locators. While the Internet works well for traditional kinds of communication, increasing mobility levels, and emerging mobile content and Internet-of-Things (IoT) services have motivated consideration of clean-slate *Information Centric Network* (ICN) architectures [2, 3] which operate on names rather than addresses. Several distinct architectures for ICN have recently been proposed including MobilityFirst (MF) [4, 5], Named Data Network (NDN) [6], and XIA [7]. While there are differences in detail, all the proposed ICN protocols share some common design elements that need to be considered in the design of transport protocols to be used for end-to-end services. Specific characteristics of ICN include: (a) use of names to identify sources and sinks of information; (b) storage of information at routers within the network in order to support content caching and disconnection; (c) multicasting and anycasting as integral network services; and in the MF case (d) hop-by-hop reliability protocols between routers in the network. These properties have significant implications for transport protocol design since the current protocols, TCP and UDP, were designed based on the end-to-end Internet principle, which typically assumes end-to-end connectivity during a transfer and uses address based routing with minimal functionality (i.e., no storage or reliability mechanisms) within the network.

Consider first the implications of name-based routing on transport protocol design. Communication with named objects, whether content files, devices, groups of devices or more complex context-based groups is different from conventional TCP connections in the sense that an object may have multiple end-points because the object may be multihomed (i.e., multiple network interfaces to the same device) or multicast (to multiple devices, each with a different network interface) or multi-copy (i.e., multiple instances of the same information object can be found at different places in the network). This indicates that transport protocols need to be designed to provide appropriate service semantics for retrieving or delivering such named objects, for example, in multicast where the information object reaches all the named destinations or anycast where the object is fetched from the "nearest location". A second important property of ICN protocols is the fact that routers may store information objects such as content either for caching or for delay tolerant delivery. This implies the existence of in-network transport proxies which are in between the source and the destination, and the transport protocol should be designed to take advantage of the in-network copy to provide the desired service efficiently. For example, reliable delivery with an ICN transport would be able to utilize a copy of the information object stored at an intermediate router and avoid the need for end-to-end retransmission used in TCP. The third feature of ICN architectures is the fact that in-network storage can be associated with reliable hop-by-hop transmission of information objects between routers, thus alleviating the need for strong reliability mechanisms at the transport layer depending on the type of service desired.

In Section II, we consider the requirements for ICN transport in further detail and identify a set of core transport protocol functions needed to address an anticipated range of service requirements. These core transport protocol components are developed in further detail for a specific ICN architecture, MobilityFirst, and several examples of how these functions are integrated with the named-object network layer are given in Section III. The prototype of MFTP is discussed in Section IV. Finally, we provide a set of experimental results based on the prototype and compare its performance with conventional TCP/IP to the extent possible. The results demonstrate significant performance improvement for several example use-case scenarios.

Our contributions in designing and implementing transport protocols for an ICN architecture with explicit locators, such as MF, are twofold: (i) we examine a representative set of delivery service scenarios, and based on them, define the requirement space for transport protocols for any Information Centric architecture; (ii) with explicit locators in an ICN architecture, much richer end-to-end semantics, such as reliability delegation, and in-network retransmission, are enabled by integrating in-network transport services. We show that such features are conducive to supporting mobility, and flexibly and robustly supporting different delivery patterns. The proposed design is validated using an experimental prototype, with bulk (e.g., video) content and latency-sensitive web (text, image and video) content delivered over wireless networks to mobile clients. The general principles of our design for end-to-end transport, regardless of whether MF-like locators are used, can also be customized to work with and bring benefits to NDN as well, e.g., to apply per-hop error and congestion control to improve transmission efficiency, and employ router-proactive mechanisms to provide better and richer mobility support.

2. REQUIREMENTS FOR TRANSPORT LAYER SERVICE FOR ICN

We first consider four common service scenarios that arise in information dissemination. These are large file transfer, web content retrieval, M2M communication and multicast. Through systematic analysis of these use cases, we identify the set of transport layer features for an ICN environment to support each of these scenarios (see the summary of requirements in Table 1).

Large file retrieval. A large file retrieval is abstracted as a *get*(content_name) socket call [8] in an ICN context. Clients inject a content request, independent of the content location, with a *get*() call, and the network will route the request to the location of a copy of the content. Then a flow with a large volume which carries the content is transferred reliably from the server to the requesting client. This is often referred as *anycast*.
Key TP functions required: Because of the large amount of data to be delivered, file transfer requires: (i) fragmentation and sequencing at the source, and reassembly at the sink; (ii) efficient usage of network resources with source rate control so as not to introduce congestion. Reliable delivery, flow control, and congestion control becomes more complicated when the destination is connected to the Internet wirelessly, and especially when it is mobile. For instance, a wireless connection is susceptible to fading, may introduce random losses, and can typically provide a lower transmission rate than the nominal rate. Further, the imbalance of rates at different segments of an end-to-end path makes it difficult to perform end-to-end control at high speeds, with small amounts of buffering, and to deal with transient disruptions. This problem can be alleviated by enabling additional in-network transport features, such as temporary storage for in-transit data (we call the en-route node with transport services a *transport proxy*).

Web content retrieval. In a web-browsing application, a sequence of content requests are sent by the client to the server. Each of the requests is for retrieving a constituent named object of a webpage. Two characteristics are inherent in web content retrieval: i) these requested objects are generally small in size, i.e. of tens or hundreds of kB; ii) user experience dictates that the objects must be received in a timely manner, preferably no more than several hundred milliseconds, thus making the transfer latency-sensitive.
Key TP functions required: End-to-end error and congestion recovery need to be provided, but in a lightweight manner, because any significant setup overhead is not amortized easily. Flow control is not required due of the limited amount of data transmitted, in order to avoid unnecessary overhead contributing to increased latency.

M2M communications. In Machine-to-machine (M2M) communications, sensor data is by nature idempotent. That is, if the PDU is lost (due to bit errors or congestion) or it is delayed beyond the limits of latency for the data, the transport layer need not attempt to reliably deliver that PDU. This transfer paradigm is captured in a *send*(dst_name, content_name) API with no explicit reliability preference.
Key TP functions required: In such cases, the transport layer could simply resort to stateless communication (e.g.,

Service scenarios	Fragmentation & resequencing	Reliable delivery	Lightweight transport	Flow/congestion control	In-network proxy
Large file retrieval	✓	✓		✓	✓
Web content retrieval	✓	✓		✓	
M2M communications	✓		✓		✓
Multicast	✓	✓	✓	✓	✓

Table 1: Transport requirements for different service scenarios

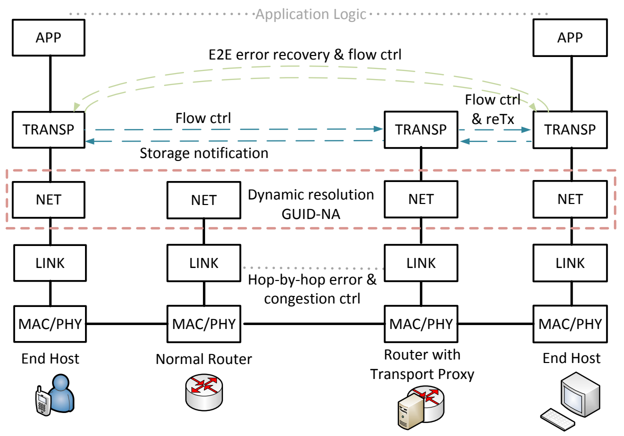


Figure 1: Protocol stack and transport layer functionalities

lightweight transport with no error recovery, and minimal flow and congestion control) to minimize overheads. Moreover, due to power constraints in devices, a sensor node may not be on all the time. End-to-end control is not always possible in this case and delegation of transport service guarantees, such as reliability, need to be made to other en-route nodes. Thus in-network proxy support is desired.

Multicast. A number of popular applications are based on multicast, such as group-based subscriptions (RSS), teleconferencing, online gaming, etc. In a name-based architecture, multicast can be realized with a *send* (dst_name, content_name) API with the dst_name referring to a group of individual endpoints names.
Key TP functions required: Guaranteeing 100% reliability in a multicast session is a well-known hard problem. To achieve reliable transport, the source relies on negative acknowledgement (NACK) from clients to initiate retransmissions. With the number of subscribers increasing, retransmission has to be implemented in an efficient manner such that the ACK-implosion (see [9]) is avoided. This may require aggregation of retransmission requests in the network, and retransmission from within the network. Thus in-network proxies are desired to handle such aggregation and storage of pieces of contents for retransmission.

3. MFTP DESIGN

MFTP is based on the four characteristics of different ICN proposals to support the analyzed requirements. Specifically, MFTP has been designed to operate on top of the MobilityFirst networking stack [10,11], while the principles may be more broadly applicable to other ICN frameworks. As described in [5], MobilityFirst is based on a clean separation of names and network addresses with a logically centralized but physically distributed global name resolution service (GNRS). The globally unique identifier (GUID) in MF is a flat public key identifier, i.e. a name, which can be used to represent any network attached object, including devices, people, groups, content, or context. Fig. 1 shows the major layers in the MF protocol stack and the role of the MFTP transport layer above the named-object GUID based network layer which is supported by the GNRS [11,12]. For additional details on MF, the reader is referred to [5,8,10,11].

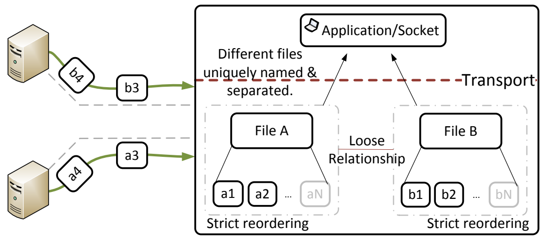


Figure 2: Illustration of named object's implication on fragmentation and sequencing. Transport layer fragments a content into large chunks. Sequential delivery is guaranteed for each content, but no strict ordering is maintained for chunks of different contents.

3.1 Fragmentation and re-sequencing

Typically in ICN, a data request is abstracted by an API, *get*(content_name). In NDN, such a request, called an Interest, with an associated relative sequence number, solicits one segment of a content. In MF, the requestor only sends one request for a piece of content; the server that handles the request then segments the content and assigns the segments a relative sequence number. In any case, sequence numbers are bound to the named content, rather than the two endpoints. This has significant implication for the hop-by-hop transfer and storage capability in ICN, as we shall see later. With content-centricity, such a sequencing scheme works naturally for anycast, multicast and multipath transfers. For example, in an anycast scenario, the forwarding plane decides where the content request should be handled. The transport layer is oblivious of the server location; rather, the transport's functionality of providing ordering and reliability can be fulfilled based on the knowledge of the data being delivered, using the content names and sequence numbers.

On the sender side, the transport layer fragments the application data into large chunks[1], whose size can be negotiated by the two end-points based on a tradeoff between the

[1]we use "segments" and "chunks" interchangeably.

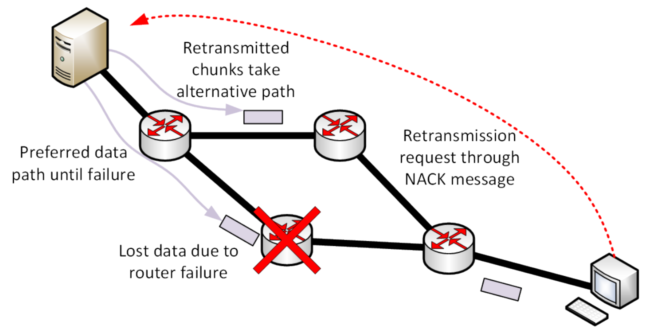

Retransmitted chunks take alternative path
Retransmission request through NACK message
Preferred data path until failure
Lost data due to router failure

Figure 3: End-to-end signaling to recover from in-network failure

overhead and the fair use of network resources across flows[2]. We allow the chunk size to go up to the order of megabytes. Note that the link layer breaks a chunk into packets to meet the link MTU requirement, but still logically maintains the semantics of a "chunk" at each hop. Figure 2 illustrates how the transport layer would support concurrent reception of multiple files. As shown in the figure, in-order delivery is strictly enforced among the chunks of a single transported file: transport will buffer out-of-order chunk arrivals. On the other hand, because each file has a unique name, retrieval of any file is fulfilled by a separate "flow". Thus there is no need for strict ordering of receiving files, say, based on the order of the requests, regardless of where the files originate.

3.2 Coordinated End-to-end error recovery and hop-by-hop reliable delivery

We use hop-by-hop reliable transfer to move each *chunk* from any node to its next hop, and use end-to-end reliability guarantees to ensure the entire *application data*, e.g. a file or content, is reliably delivered.

3.2.1 Per-hop reliability

In traditional transport protocols operating on an end-to-end basis such as TCP, loss (whether due to errors or congestion) or congestion at a link has to be detected after a feedback delay, possibly quite a few end-to-end RTTs. After the detection, recovery mechanisms, such as window reduction or retransmission, can incur an unduly large penalty to the flow. Also, due to queuing at routers, and heterogeneous transmission technologies employed along the route to destination, spurious, or premature, retransmissions are not uncommon [13]. A more efficient way to recover from congestion or error happening at a particular link is through link level mechanisms. This yields two benefits: i) congestion and errors can be detected and reacted upon more quickly; ii) reduces the possibility of spurious retransmissions. Hop-by-hop transfer maintains a per-hop reliability model: each chunk is only forwarded once it has been received reliably in its entirety from the previous node. This reliability model is suitable for ICN due in part to the fact that the segment of data being transferred is named; moreover, ICN routers can have storage capability, and can temporarily store the in-transit copy to provide delay-tolerant delivery, and also cache a copy to serve future requests. In NDN, each named data item is indeed transferred in a hop-by-hop manner: upon receiving such a data item, the router

[2]with MFTP, a "flow" is identified by a (source GUID, destination GUID) pair.

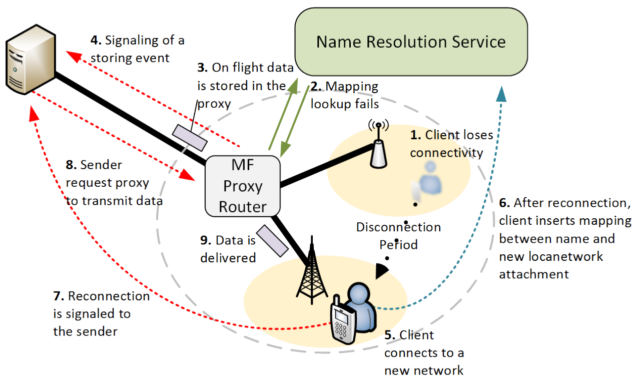

Name Resolution Service
4. Signaling of a storing event
3. On flight data is stored in the proxy
2. Mapping lookup fails
1. Client loses connectivity
MF Proxy Router
8. Sender request proxy to transmit data
9. Data is delivered
Disconnection Period
6. After reconnection, client inserts mapping between name and new locanetwork attachment
7. Reconnection is signaled to the sender
5. Client connects to a new network

Figure 4: Procedures involved to use in-network transport proxy to handle destination disconnection and retransmission: the proxy temporarily stores chunks when the destination disconnects, and transmits to the client when connectivity is restored as indicated by the name resolution service.

examines whether an Interest for the data has been received earlier, and whether it needs to cache the data.

MFTP integrates per-hop error recovery and congestion control whenever the problem can be resolved locally, and only invokes end-to-end mechanisms when it is absolutely necessary, e.g., a router fails and loses all the buffered data. On each hop, after every chunk that is transmitted, a corresponding control message called *CSYN* is used to explicitly request acknowledgement from downstream, which then replies with a bitmap of reception status for every packet in that chunk. The transmission for this chunk finishes if there is no loss, otherwise the lost packets of that chunk are retransmitted locally following the same procedure until all packets are received.

3.2.2 End-to-end reliability

Taking advantage of the hop-by-hop reliability of the network, we seek to have a parsimonious end-to-end mechanism that has minimal overhead (important in mobile wireless environments) while primarily aiming to recover from node and link failures. While per-hop recovery concerns about whether all the packets constituting a chunk are delivered to the next hop, end-to-end recovery strives to guarantee all the chunks of the application data are reliably received. The end-to-end error recovery mechanism is built to be *flexible* to accommodate application and sender needs (including *don't care, NACK, ACK*). With a Negative-ACK, i.e. NACK, the transport reduces end-to-end message overhead, and the receiver provides notification only when a chunk is not delivered over a conservatively long period of time (as a result of a failure that causes the reliable hop-by-hop mechanism to lose an acknowledged chunk as shown in Figure 3). It is only for short-sessions (e.g., single PDU delivery) and for latency-sensitive interactions that the sender would enable the use of an end-to-end ACK option. With idempotent data transmissions (e.g, sensor data which the transport layer sends and forgets), the sender may choose to use the *don't care* option.

3.3 In-network transport proxy

One of the challenges for conventional transport protocols is in dealing with the content delivery to mobile devices, where mobility results in intermittent connectivity and the end-to-end connection experiences frequent disruptions. If the transport protocol has to re-establish the connection, then the transfer has to re-start and any data already in transit in the network will have to be discarded. ICN's architecture inherently supports mobility and resolves connection disruptions in multiple ways. For instance, in NDN, each data is solicited by an Interest packet; in case a client moves before obtaining the requested data, it can re-issue an Interest packet for the same data, which will be delivered to the new location. In the MF architecture, the network can take on a more proactive role in re-initiating data transfers when connectivity is re-established.

To this end, we postulate having routers (or at least a subset of them) which provide in-network transport service such that the original source can delegate part of the end-to-end data transfer responsibility. The router, which we call an *in-network transport proxy*, would have substantial amounts of memory, e.g., several GB, to temporarily hold in-transit chunks when the destination is unreachable. This disruption may be due to: lack of connectivity to a mobile destination node, until connectivity is subsequently re-established; alternatively, in M2M communication, when a sensor node is only powered on intermittently, it may choose to deliver information chunks to the next hop and then power down.

The mechanisms implemented by such a node are shown in Figure 4: when faced with the impossibility of forwarding chunks with the information available at the network layer (i.e. the router detects that connectivity towards the destination of a chunk is disrupted), the router pushes up to the transport proxy layer the relative data chunks. Two reasons might generate this impossibility of forwarding chunks: (i) the destination does not have an active network address (NA) binding corresponding to its GUID entry in the GNRS; (ii) the chunk reaches the destination network given by its most recent binding, but either the destination has changed its point of attachment or it has disconnected from the network before the previous NA entry expires in the GNRS server. As a consequence, the link layer is not able to deliver the chunk despite several attempts, and corresponding CSYN timeouts. In these cases, the chunk is pushed up to the proxy layer to be temporarily stored. While this is similar to Delay-Tolerant Network protocols, the innovation here is the integration of these mechanisms with the support of dynamic mobility and ICN style named object services. Note that we differentiate the *storage* operation here from *buffering* and *caching*. They all involve the action of maintaining a copy of in-transit data. However, they differ in their final purpose. Buffering usually resolves the mismatch between ingress and egress rates, and caching is to serve future data requests more efficiently. Storage, considered here, is utilized to provide delay-tolerant delivery to deal with disconnection or to provide delegation.

We limit the amount of content that can be stored for a flow. Each (source GUID, destination GUID) pair is limited to have stored content up to a size S. When a chunk for a new flow arrives, the chunk will be stored directly if sufficient space is available for the new flow; otherwise, a chunk for the oldest flow is evicted to make room for the new chunk. In other words, an LRU policy is employed for chunk eviction from the storage. Therefore, the operations for storing a chunk, and for retrieving a chunk from the storage, can be implemented with O(1) complexity. When the chunk is stored, a timer is created to schedule future transmission. Further, a transport layer message, either *Store* or *Drop*, is transmitted back to the original source to notify it of the intermediate proxy storing or dropping the chunk. A stored chunk will be scheduled to retry a GNRS lookup to bind an updated NA to the destination GUID when its storage timer expires. The chunk will be pushed out if an NA is found, i.e., destination becomes connected again, otherwise it will be kept in storage. On the other hand, rescheduling of the chunks can also be initiated by the original source of a chunk. As is shown in Fig. 4, when the source receives a NACK message identifying a chunk as missing, if it is aware that the corresponding chunk originally destined to the requesting destination is stored in the network, based on a previously received *Store* message, it utilizes this in-network copy and initiates the retransmission from inside the network. This is done by the source sending a *Push* message to the in-network proxy to trigger retransmission.

Transport proxies also support content producer mobility by allowing the producer to delegate its end-to-end reliability guarantee to the proxy. For instance, a mobile client intending to upload a recently shot video can specify in the pushed data chunk that such a delegation is requested. Before forwarding the data chunk, the immobile access router (acting as a transport proxy) will save a copy of the chunk in order to respond to potential future NACKs.

3.4 Flow control and congestion control

With the hop-by-hop reliable delivery as a building block, MFTP uses a combination of per-hop back-pressure for congestion control and end-to-end window-based flow control.

3.4.1 Hop-by-hop congestion control

The hop-by-hop back-pressure scheme is built on top of a back-pressure buffer (of capacity B packets). As illustrated in Fig. 5, the back-pressure buffer essentially has all the chunks that are received from the network and are queued to be transmitted. In addition, between two adjacent routers on a link, the sender maintains a sending window W_{ostd}, i.e., number of outstanding packets, that is bounded by the receiver's advertised window, W_{ad}. As mentioned before, following the transmission of a chunk of data, a CSYN message is sent, which the downstream node then acknowledges with a CACK message. The receiver's advertised window is piggybacked in the CACK. The number of outstanding packets, W_{ostd}, is reduced based on the downstream node's acknowledgement. Also, whenever the router schedules to transmit on a particular outgoing interface, it attempts to transmit as many packets as W_{ad} allows. This greatly improves pipelining.

When the occupancy of back-pressure buffer reaches its capacity, the router blocks all incoming data chunks. Furthermore, it throttles the advertised window to all of its upstream nodes. This "congestion signal" eventually propagates back to the original traffic sources in a hop-by-hop manner, thus eventually limiting the traffic injected into the network.

3.4.2 End-to-end flow control

Hop-by-hop back-pressure is not sufficient to prevent the receiver's buffer from being overrun by the sender's data from an end-to-end perspective. Because MFTP does not require the receiving side to send frequent reception status

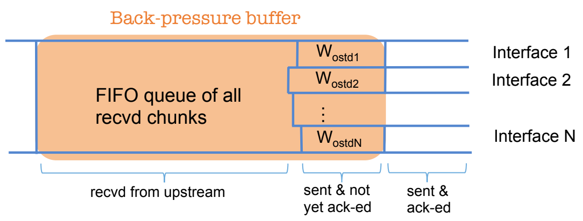


Figure 5: Back-pressure buffer and per-hop sending window.

update in the reverse path (it depends only on NACKs), the feedback from the receiver is both parsimonious and not timely for the sender to detect receiver buffer overflow. We therefore consider an explicit notification from the receiver. The sender starts at an initial end-to-end sending window W_e. For each window's worth of data chunks, the receiver then sends one window flow control message, to advise the sender to maintain, increase, or reduce the sending window to certain value based on the receiver's buffer occupancy. This message will be delivered reliably to the sender. Note that the sending window is also the atomic unit for the end-to-end NACK message, thus the NACK and flow control are fulfilled by a single message (if a NACK has to be sent, i.e., some chunks are lost). In the event that this special chunk is lost due to a node failure, a NACK timeout at the receiver would trigger the receiver to proactively notify the sender of the reception status (NACK) and receiver buffer status (flow control).

Small content transfers are not subject to such end-to-end flow control, mainly because the transfer will be complete even before the flow control notification can be generated. However, small content transfers are still regulated by per-hop congestion control.

3.4.3 *Alleviating head-of-line blocking due to hop-by-hop transfer*

A drawback inherent with hop-by-hop back pressure is the unfairness caused by head-of-line (HOL) blocking with FIFO queueing [14]. Consider a chunk at the head of the queue blocked from being transmitted by a back-pressure signal from the downstream node. This can prevent chunks behind it in the queue that is destined to a different destination that is not experiencing congestion. An alternative to having HOL blocking is to drop the chunk being back-pressured, but this has undesirable consequences of requiring retransmissions when a temporary buffering could overcome the short-term congestion. Theoretically, per-flow queuing solves this problem, but scheduling with per-flow queues is difficult to scale and is impractical with large numbers of flows. However, the in-network transport proxy provides some relief to this situation and alleviates the short-term unfairness. If a back-pressure signal is received for the chunk at the head of the sending queue, the transfers of chunks destined to other nodes will thus not be blocked because chunk at the head of the queue will be removed and pushed up to the transport proxy layer for temporary storage. The transport proxy will then attempt to transmit that chunk when the storage timer expires (or is dropped if the chunk is replaced in the storage buffer because of the eviction policy we described above).

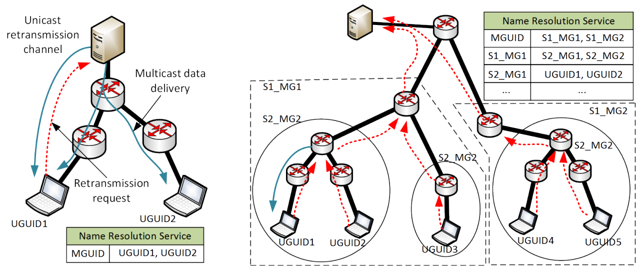


Figure 6: Multicast data delivery, small scale (left), large scale (right).

3.5 Multicast

Multicast is naturally supported by name based architectures. For instance, in NDN, data is forwarded to the requestor based on the receipt of the corresponding request: each router forwards the data on the interface(s) the request for the data was received on. Multicast is thus fulfilled by the stateful forwarding plane [6]. In MobilityFirst, a dynamically formed multicast group is explicitly identified by a globally unique identifier (GUID), which can be mapped into a set of individual clients' GUIDs or network addresses.

Depending on the scale of multicast group, multicast support varies. In the small scale case (show in the left side of Fig 6), during the transmission the source of the multicast data marks outgoing chunks with a multicast service identifier and selects as destination GUID the one identifying the multicast group. Multicast clients send NACK messages over a unicast channel and the multicast source can identify which multicast group a specific client belongs to. Further, the source aggregates retransmission requests for the transmitted chunks; it can, either employ multicast again for retransmission when the number of requestors exceeds a threshold; otherwise retransmitted data chunks can be sent using unicast destination GUIDs that identify the specific nodes that need the retransmitted data.

As the number of participants increases we can exploit in-network transport proxies to build multiple levels of *multicast group GUID to a set of GUIDs* mappings recursively. This scenario is shown on the right side of Figure 6. In order to limit potential explosion of unfulfilled requests reaching the original source, transport proxies can be instructed through proper chunk marking, to discard retransmission requests that exceed a number of traversed proxies without encountering the missing chunks. In a scenario where reliability is not demanded, the source just use the *don't care* option of the reliability preference.

4. IMPLEMENTATION

Our implementation of MFTP consists of two parts: end-system transport operations that are implemented on the MobilityFirst client stack, and an in-network transport proxy implemented as a pluggable module inside the MobilityFirst based Click router implementation [15].

Host Stack and API. The client host stack has been implemented on Linux as a user-level process built as an event-based data pipeline. Apart from the MF transport protocol, the stack contains a name-based network layer and a reliable link layer with large chunk transfer. Applications interface with the host stack through socket APIs that are available

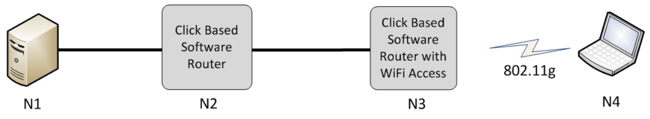

Click Based Software Router
Click Based Software Router with WiFi Access
802.11g
N1
N2
N3
N4

Figure 7: Experimental Setup

as a linkable library and include the primitives *send, recv*, and *get*, and a set of meta-operations. Examples of meta-operations include those to bind or *attach* a GUID to one or more NAs. By specifying the options field in the API call, an application is able to configure transport parameters such as the i) desired chunk size; ii) end-to-end reliability preference; iii) NACK timeout; iv) willingness to use in-network proxy.

Router. The MobilityFirst software router is implemented as a set of routing and forwarding elements using Click [16]. The router implements MFTP transport proxy layer, MF network layer including intra-domain routing and dynamic binding using GNRS, and hop-by-hop reliable transfer. The transport layer (proxy) interacts with the intra-domain route look up component: if a lookup does not yield a valid next hop, the chunk is pushed up to the transport proxy. The transport proxy at the router will hold the data chunk for some time and attempt to rebind the name with one or more network addresses. When rebinding is successful, the chunk is pushed back down to the routing layer for forwarding.

Timers. There are three types of timers used in our implementation: one for triggering the transmission of an end-to-end NACK message, one for storage, and another one for link layer retransmission. For guaranteeing end-to-end reliability, timers are indispensable because a node has to learn about a remote node's failure impacting the end-end path. Previous experience with TCP end-to-end timers have taught us that timers need to be set loosely so as to reduce number of false alarms [17,18], and not have a strict dependence of the transport protocol on timers for normal operations. In MFTP's design, this goal is achievable because: (i) different end-to-end service guarantees are dissected and each timer only handles a specific job; (ii) NACK timers and per-hop timers are associated with a chunk of data, rather than a single packet; (ii) the storage timer is only concerned about disconnection, and is thus decoupled from end-to-end latency and transferred data sizes, which could otherwise complicate timer settings.

5. CASE STUDIES AND EVALUATIONS

In this section, we present how MFTP can be used in several different service scenarios, and quantitatively compare it with the performance of conventional HTTP and IP based protocols.

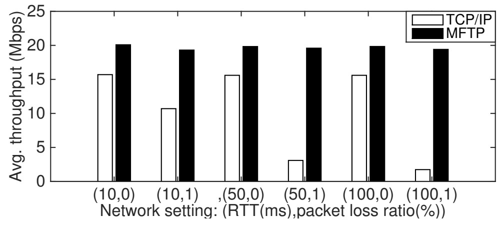
General experimental testbed setup. We use the ORBIT [19] wireless testbed for our experimental evaluation. Each machine in our experiment is equipped with Intel i7 2.93GHz processor and with 8GB RAM. We use Ubuntu 12.04 with Linux kernel version 3.2. In terms of networking capability, each node has one Gigabit-Ethernet interface and one WiFi interface using Atheros' ath5k wireless drivers. Physically all the nodes are connected to a single layer-2 switch; we use VLAN tags to create desired topology to isolate Ethernet traffic. For wireless traffic, we use 802.11g with the data rate fixed at 54Mbps. Access routers run hostapd [20] to operate as WiFi access points. We disable 802.11 authentication and use manual IP assignment (no DHCP), just to retain nearly the same amount of overhead with both MFTP and TCP for WiFi connection establishment. We considered a topology shown in Fig. 7, where a client, N_4 connects to a server N_1 through an access router N_3, which provides WiFi connectivity, and a regular router N_2.

Methodology. We evaluate three types of data delivery scenarios to compare MFTP with the current TCP/IP based architecture, in terms of the mechanisms employed, and their performance. We emulate the end-to-end RTT's of local, coast-to-coast and inter-continental communications, use the emulation tool *netem* [21] to add 10ms, 50ms, 100ms RTT between the two routers, respectively. To emulate loss in a controlled manner, we again use *netem* to introduce 1% loss. With MF, we run the MF Click router prototype (mentioned in section 4), and a local GNRS server on both N_2 and N_3. The MF client stack runs on N_1 and N_4. For specific use cases, we run corresponding applications that interface with the client stack through the MF API. In the case of TCP-based experiments, we run Click IP routers on node N_2 and N_3, rather than using Linux' default IP routing, just to eliminate processing time discrepancies of Click router compared with Linux routing (though in the experiments we found Click's overhead is negligible). TCP segmentation offloading is turned off as the basic Click IP router drops TCP packets with size larger than 1500 bytes. We enabled manual Ethernet header encapsulation on the Click IP router so no ARP message is triggered during routing. On the two end nodes, the default version of TCP on Ubuntu, TCP Cubic, is used, as it is the state-of-the-art TCP congestion control algorithm and performs better than other variants [22], e.g. TCP Westwood, including under wireless scenarios. We configured both nodes' TCP receiver buffer to be 2MB, so that it is not a bottleneck in a high delay-bandwidth path in any of the experiments.

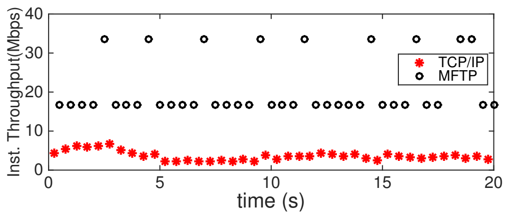
5.1 Large content delivery over wireless

We first look into a large volume data transfer experiment. A 400MB file is requested and transferred. A simple file retrieval application in MF is running on the two end nodes. In the case of TCP, we used iperf to generate a flow of equal size with the maximum packet payload size of 1400 bytes. We repeated this experiment for a number of network conditions: RTT being 10ms, 50ms, or 100ms, and loss on WiFi link being 0 or 1%, to explore their effect on both architectures' goodput (i.e., application throughput).

Fig. 8(a) shows the average throughput comparisons for the six different network settings. Both MFTP and TCP' throughputs are consistently high when there is no loss, despite varying the end-to-end latency. MFTP is slightly higher in throughput in the lossless cases. MFTP is significantly more robust in the presence of loss, e.g., the throughput degrades by only 10% when there is 1% residual loss, with all 3 RTT profiles. On the other hand, TCP throughput drops significantly when there is loss. For instance, with 50ms RTT, TCP throughput with loss drops to only a quarter of its throughput in the lossless case. Fig. 8(b) shows a



(a) Average throughput comparison for 6 different (RTT, loss rate) profiles.



(b) Instantaneous throughput (per 500ms) for 50ms RTT and 1% loss.

Figure 8: Throughput comparison. MFTP is robust in the presence of loss.

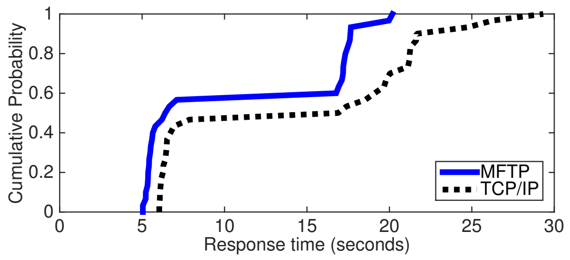
plot of instantaneous throughput (averaged per 500ms) for 50ms latency and 1% loss. MF's PDU is a chunk of data, and in every 500ms, it receives at least one chunk (1MB), even in the presence of loss. With TCP, throughput fluctuate around 5Mbps. This is because the end-to-end congestion window is throttled whenever loss is detected. This misinterpretation of loss unrelated to congestion unnecessarily penalizes the flow. With MFTP, loss is not considered a signal for congestion, thus the sending rate is not throttled; moreover, loss happening at the last hop is recovered locally. Note in this experiment, the client suppresses the NACK messages because all the data has been successfully received.

5.2 Transport proxy for disconnection

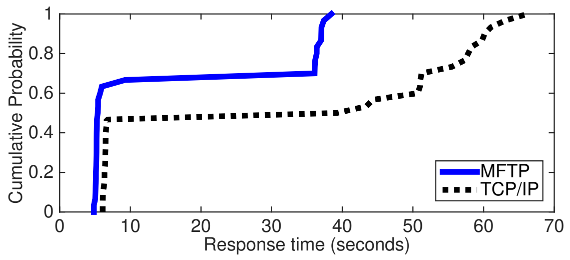
We evaluate the benefits of using in-network transport proxies for handling client disconnections in content retrieval. In the experiments with TCP, an application client issues one HTTP GET request to retrieve one file. For experiments with MFTP, in order to keep the modifications at the application end-hosts to a minimum, we developed an MF-HTTP-MF proxy whose main job is translating HTTP request and responses into MobilityFirst content requests and messages and vice-versa. We colocate 2 instances of these proxies[3] with the HTTP components of the system, i.e., on N1 and N4. We consider the same topology as above. The end-to-end RTT is set to be 50ms, and no loss is added so that difference in performance would not be incurred by having different mechanisms for error recovery. We use *netem* to introduce 100% loss intermittently, so as to emulate client disconnections. In the experiment, WiFi connectivity is on for 10 seconds; then is turned off for d seconds; then the connection is restored. During the first 10 seconds of connection, the client requests a 10MB file at a random time. The experiment is repeated 30 times for both MF and TCP. We compare the distribution of file retrieval response times between MFTP and TCP.

[3]These proxies are also used for experiments in section 5.3

In Fig. 9(a), all the transfers having a response time of less than 10 seconds are completed before the disconnection. For the transfers that experience the disconnection, MFTP has at least 3 seconds lower response time (at 60th percentile). With 30 seconds disconnection, as shown in Fig. 9(b), the difference in response time is about 15 seconds at 70th percentile. It is worthwhile to understand the difference in the approaches taken by TCP and MFTP to dealing with disconnection. With TCP, the sender retransmits, based on a timer whose timeout value increases exponentially when the disconnection persists. In MFTP, the chunk in-transit is stored at the in-network proxy. A network address and next-hop lookup, rather than retransmission, is triggered when the storage timer for that chunk expires. Thus the transport proxy takes advantages of the global name resolution service in MF to learn whether there is a network address binding update for a client, and retransmits only when client is connected. This results in fewer retransmission attempts and more accuracy in the knowledge of end-to-end connectivity. Fig. 9(a) and Fig. 9(b) together suggest that MFTP's reduction in response time is nearly proportional to the length of the disconnection.



(a) With 10s disconnection



(b) With 30s disconnection

Figure 9: CDF of response times.

5.2.1 *Comparison between network-proactive and receiver-driven approaches*

We perform another set of experiments, with two different settings of proxy's storage and client's NACK timers. The network-proactive approach was used in the previous experiments, where the NACK timers are set conservatively, and thus client relies on the network to re-deliver the data once the network connection is restored. In the receiver-driven approach, the transport proxy does not re-initiate the delivery by itself; the client sets the NACK timer aggressively and explicitly requests retransmission, and the original server will then remotely request the retransmission from the transport proxy where the data is stored (by sending a *Push* message). As shown in Fig. 10, similar to last set of experiments,

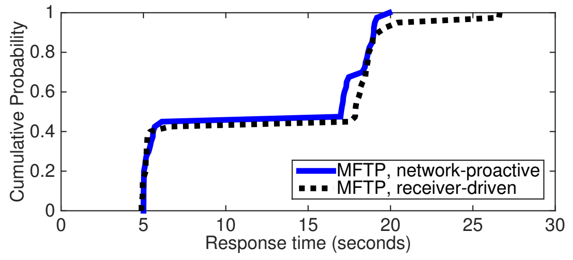


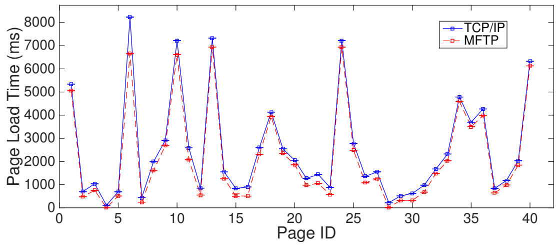
Figure 10: CDF of response times for network-proactive and receiver driven retransmissions, with 10s disconnection.

half of the transfers complete without experiencing disconnection. The long tail of the receiver-driven curve warrants a closer look: it corresponds to cases when a disconnection happens right after the client sends out the request. The content is transferred, but only a small portion gets delivered because client loses connectivity. The remaining data is temporarily stored at the proxy. The client has to wait for the NACK timeout to retrieve the data from the proxy. Thus, on the client side, whether it is an application or transport that is responsible to setting the end-to-end timer for a mobile client, a large number of characteristics, such as end-to-end path quality variations, disconnection interval, and content size, all collectively make estimating a reasonable timeout value difficult. On the other hand, retransmission from inside the network by the transport proxy only concerns itself with the connection/disconnection events. This improves performance, and more importantly, provides better manageability of end-to-end timers in mobile scenarios.

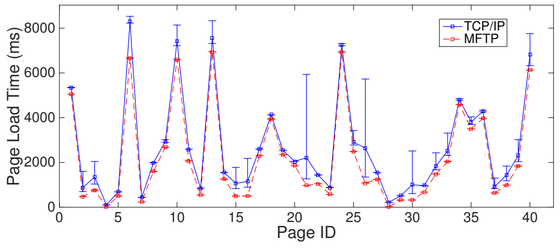
5.3 Web content retrieval

Web content retrieval is also evaluated. We use the same topology as described before to compare MFTP and TCP's performance. In addition to the routers, we run an Apache server (version 2.2.22) on node N_1, and a web browser emulator on node N_4 which requests webpages. We reuse the browser emulator, *epload*, presented in [23]. We also download the dataset introduced in [23] which consists of the real webpage objects of the 200 most accessed websites recorded by Alexa [24] in 2013. Among these we randomly select 40 pages and place them on N_1 to be hosted by the Apache server. In each run of the experiment, the browser emulator opens up 6 concurrent TCP connections (default settings in most browsers [23]) and sequentially request the 40 webpages, using HTTP 1.1 (also by default). For both MFTP and TCP, we performed 5 runs of the experiments with end-to-end RTT of 50ms, and 0 loss or 1% loss.

Fig. 11 are the plots for average page load times (PLT), i.e. the time between emitting the first HTTP request to reception of the last byte of last object, for the experiments with 50ms RTT. Page load time with MFTP is consistently lower than TCP. In the case of no loss, when there is a smaller amount of data to be transferred, e.g. page 21, 22, and 23, with TCP the PLT is about 30% higher than with MFTP. The difference in PLT can be attributed to several features of MFTP: (1) MFTP is connectionless, and thus there is no overhead due to setting up a connection; (2) TCP identifies different requested objects by differences in sequence numbers of that connection, while MFTP differentiates each requested object by a unique name, therefore HOL blocking (happens when multiple concurrent HTTP requests are fulfilled by a single TCP connection [25]) does not occur with MFTP; (3) each TCP connection "slow-starts", whereas with MFTP, short transfers, such as retrieving web objects, are not subject to flow control and are regulated only by per-hop back-pressure based congestion control, which allows sender to transmit at full rate as long as no congestion signal. As can be seen in Fig. 11(b), loss introduces a great amount of variability with TCP. For instance, for page 21, the minimum PLT is around 1500ms with TCP, but the maximum is 6000ms, which is several orders of magnitude higher than with MFTP. For all the pages, MFTP maintains minimal variability in terms of page load time.



(a) 50ms RTT, no loss



(b) 50ms RTT, 1% residual loss

Figure 11: Page Load Times (min, average, and max of 5 runs) for 40 different webpages.

6. RELATED WORK

Future Internet architectures (FIA): A number of clean-slate information-centric network architecture designs [4,6,7, 26] have been proposed recently to address challenges faced by today's IP network. They differ from each other in how they realize name-based service: while NDN [6] proposes a name-based routing approach in which packets are forwarded directly based on name; some other architectures (like MobilityFirst [4], XIA [7] and HIP [26]) place object names outside of the routing plane and uses a name resolution service to translate names to addresses.

Transport protocols for FIA: There have been a number of works on transport protocols for FIA, e.g. NDN [6], and XIA [7]. In NDN, a receiver asks for content by issuing a group of interest packets, i.e. requests; the corresponding data chunk is returned by the network in response to each interest. The NDN community has looked at how the transport layer can be adapted to such a interest-data interactive and multi-source/multi-path content-transfer pattern. Most of these works propose that the receiver maintains a Interest window and controls the issuing rate of interest packets (i.e., ICTP [27]), while others proposes a hop-by-hop Interest shaping scheme at each router (i.e., HR-ICP [28]). To support multi-source/multi-path transfer, CHoPCoP [29] proposes to utilize explicit congestion signaling from network to effectively notify the receiver about network conditions. In [30], a transport protocol, Tapa, for XIA architecture [7] is introduced which proposes to manage end-to-end delivery through segment-by-segment control. We share some of the techniques with these schemes here for the MFTP design. In addition, we investigate a broader set of ICN transport requirements which are derived from a collection of data delivery service scenarios. This allows MFTP to flexibly support different applications ranging from receiver-initiated retrieval, to sender-initiated publish, and from throughput-sensitive large file transfer, to latency-constrained short transfers of web objects.

7. CONCLUSIONS

This paper presents the design of a clean-slate transport layer protocol for the MobilityFirst future Internet architecture. The proposed transport layer protocol, called MFTP, is based on an understanding of the key requirements of name-based Information Centric Networks. These requirements include the use of names rather than addresses for routing, in-network storage, hop-by-hop reliability and multicasting as a basic service. Several core transport protocol components responsive to the above requirements were identified and discussed in the context of the MobilityFirst protocol stack. A proof-of-concept experimental validation has been developed and used to demonstrate feasibility and significantly improved performance relative to conventional TCP/IP for several use cases including large file transfer, web access and late binding/delay tolerant services.

8. ACKNOWLEDGMENTS

This research was supported by NSF CISE Future Internet Architecture (FIA) grants CNS-1040735 and CNS-1345295. We are grateful to Ivan Seskar for his guidance and support on the ORBIT experiments. We would also like to thank Chao Han and Feixiong Zhang for their help in the implementation and evaluation of the prototype, respectively.

9. REFERENCES

[1] J. H. Saltzer et al. End-to-end arguments in system design. *ACM TOCS*, 1984.

[2] B. Ahlgren et al. Design considerations for a network of information. In *ACM CoNEXT*, 2008.

[3] A. Ghodsi et al. Information-centric networking: Seeing the forest for the trees. In *ACM HotNets*. ACM, 2011.

[4] MobilityFirst Project. `http://mobilityfirst.winlab.rutgers.edu/`.

[5] D. Raychaudhuri et al. Mobilityfirst: a robust and trustworthy mobility-centric architecture for the future internet. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2012.

[6] L. Zhang et al. Named data networking. *SIGCOMM Comput. Commun. Rev.*, 44(3):66–73, July 2014.

[7] D. Han et al. Xia: Efficient support for evolvable internetworking. In *USENIX NSDI*, 2012.

[8] F. Bronzino et al. Network service abstractions for a mobility-centric future internet architecture. In *MobiArch*. ACM, 2013.

[9] A. Erramilli and R. P. Singh. A reliable and efficient multicast for broadband broadcast networks. In *ACM Workshop on Frontiers in Computer Communications Technology*, 1988.

[10] S. C. Nelson et al. Gstar: Generalized storage-aware routing for mobilityfirst in the future mobile internet. In *MobiArch*. ACM, 2011.

[11] T. Vu et al. Dmap: A shared hosting scheme for dynamic identifier to locator mappings in the global internet. In *IEEE ICDCS*, June 2012.

[12] A. Sharma et al. A global name service for a highly mobile internetwork. In *ACM SIGCOMM*, 2014.

[13] S. Mukherjee et al. Evaluating opportunistic delivery of large content with tcp over wifi in i2v communication. *IEEE LANMAN*, 2014.

[14] Mario Gerla and Leonard Kleinrock. Flow control: A comparative survey. *IEEE Transactions on Communications*, 1980.

[15] F. Bronzino et al. Experiences with testbed evaluation of the mobilityfirst future internet architecture. In *EuCNC*, 2015.

[16] E. Kohler et al. The click modular router. *ACM Transactions on Computer Systems*, 2000.

[17] L. Zhang. Why tcp timers don't work well. In *ACM SIGCOMM*, 1986.

[18] I. Psaras and V. Tsaoussidis. Why tcp timers (still) don't work well. *Computer Networks*, 2007.

[19] D. Raychaudhuri et al. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In *Wireless Communications and Networking Conference*. IEEE, 2005.

[20] hostapd. `http://wireless.kernel.org/en/users/Documentation/hostapd`.

[21] netem: network emulation tool. `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`.

[22] M. Li et al. Block-switched networks: A new paradigm for wireless transport. In *USENIX NSDI*, 2009.

[23] X. Wang et al. How speedy is spdy. In *USENIX NSDI*, 2014.

[24] Alexa: the top 500 sites on the web. `http://www.alexa.com/topsites`.

[25] J. Erman et al. Towards a spdy'ier mobile web? In *ACM CoNEXT*, 2013.

[26] R. Moskowitz et al. Host identity protocol. *RFC 5201, April*, 2008.

[27] S. Salsano et al. Transport-layer issues in information centric networks. In *ACM ICN*, 2012.

[28] G. Carofiglio et al. Joint hop-by-hop and receiver-friven interest control protocol for content-centric networks. In *ACM ICN*, 2012.

[29] F. Zhang et al. A transport protocol for content-centric networking with explicit congestion control. In *IEEE ICCCN*, 2014.

[30] Fahad R. Dogar and Peter Steenkiste. Architecting for edge diversity: Supporting rich services over an unbundled transport. In *ACM CoNEXT*, 2012.