Contemporary Operating Systems are not ready for Peer Computing: Peer systems should be treated as second class citizens

Surendar Chandra, University of Notre Dame

Our work is attempting to bridge a critical incompatibility between the design canons of operating systems and peer computing systems. Operating systems were traditionally tasked with managing the system resources among the different schedulable entities (processes). Peer computing systems federate the *spare resources* available among a set of *independently owned and cooperating* peers for the common good [1]. As *independently owned* entities, the sharing is voluntary. The allocation of only the *spare resources* for the federation is also important; peer computing systems are not expected to debilitate the peer from performing other local tasks. Popular peer computing systems include wireless ad hoc networks and peer-to-peer (P2P) applications (e.g. Skype, Gnutella and BitTorrent). Kernel forwarding resources in ad hoc scenarios are not accounted to any entity [2] while resource requests from other peers applications are charged to the peer process. Note that distributed systems such as desktop clusters differ from peer computing systems in that the shared computing resources are not *independently owned* and belong to the same organizational entity. Their notion of sharing *spare resources* to other organizational entities is different from peer computing systems that share resources with (potentially) unknown entities.

The ownership difference has significant implications on the resource allocation mechanisms. Operating systems use authentication and authorization mechanisms as the gate-keeper to verify access rights to a user. Once admitted, priorities are used to order the resource requests and quotas are used to limit the resources allocated for the different processes. However, once a process is admitted, its resource requests are eventually honored. When resources are scarce, some processes might starve for brief durations. Similarly, prior work on energy conservation mechanisms delayed waking up certain devices; the corresponding processes waited longer to access these resources.

For constrained and non-renewable resources, this policy conflicts with the design philosophy of peer computing systems. Peer computing systems request *spare* resources from each peer for use by the global federation. However, the contemporary OS considers all schedulable entities to be first class citizens and all resource requests are honored. Hence, the resource requests from the peers can consume all the spare capacity of a particular peer. Though such a policy will provide good resource availability to the global federation, this policy can have a debilitating effect on the local peer. For example, in an operating system that manages its energy resources (such as in ECOsystem [3]) a energy constrained peer can completely drain its battery resources to service resource requests from authenticated peer applications (or at least to the extent of the cumulative energy resource quota assigned to the peer application). In our earlier work [2], we showed that kernel resources used for forwarding network packets in ad hoc networking scenarios can completely incapacitate the peer because kernel resource allocation for network forwarding are unaccounted and unmanaged in contemporary operating systems. In an contemporary operating system, peers are expected to be *nice* and limit their resource requests - the Operating System itself is not able to manage their resource requests.

We are developing a scheduling class called *less than best effort* to address this problem. Under resource constrained scenarios, resource requests from this class can either be denied or delayed indefinitely even if such a request from traditional scheduling classes can be allocated resources. Conceptually, under resource constrained scenarios, *less than best effort* allocates resources with lesser priority than for the *idle* task. For example, a energy constrained laptop might deny forwarding packets for Skype (potentially forcing the global Skype federation to deny Skype services for this peer) and leave the network idle. Similarly, a resource constrained peer might disallow personal file share requests until sufficient resources are available. When resource availability improves, resource requests from the *less than best effort* class behave in a fashion indistinguishable from traditional resource allocation classes. The challenge is in creating schedulable entities for peers with no local schedulable entity as well as in managing resources that were already allocated to peers in the *less than best effort* scheduling class (that continue to hold resources).

References

[1] R. Peterson and E. G. Sirer. Going beyond tit-for-tat: Designing peer-to-peer protocols for the common good. In *Workshop on Future Directions in Distributed Computing*, Bertinoro, Italy, June 2007.

[2] P. Xue and S. Chandra. Revisiting multimedia streaming in mobile ad hoc networks. In *ACM Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '06)*, Newport, RI, May 2006.

[3] H. Zeng, X. Fan, C. S. Ellis, A. R. Lebeck, and A. Vahdat. ECOSystem: Managing Energy as a First Class Operating System Resource. In *ASPLOS '02*, San Jose, CA, Oct. 2002.